XML SCHEMA MATCHING

BALANCING EFFICIENCY AND EFFECTIVENESS BY MEANS OF CLUSTERING

Marko Smiljanić

PhD dissertation committee:

Promotor
Prof. dr. Willem Jonker

Assistant-promotor
Dr. ir. Maurice van Keulen

Members
Prof. dr. Peter Apers, Universiteit Twente, The Netherlands
Prof. dr. Paolo Atzeni, Università Roma Tre, Italy
Prof. dr. Geert-Jan Houben, Vrije Universiteit Brussel, Belgium
Prof. dr. Franciska de Jong, Universiteit Twente, The Netherlands
Prof. dr. Martin Kersten, Centrum voor Wiskunde en Informatica, The Netherlands

CTIT Ph.D. Thesis Series No. 06-84
Center for Telematics and Information Technology (CTIT)
P.O. Box 217 - 7500 AE Enschede - The Netherlands

SIKS Dissertation Series No. 2006-07
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

ISBN: 90-365-2347-8
ISSN: 1381-3617 (CTIT Ph.D. Thesis Series No. 06-84)
Cover design: Vesna Smiljanić
Cover photos: André Nutbean Photography (www.pbase.com/kspec)
Printed by: Wöhrmann Print Service, Zutphen, The Netherlands

XML SCHEMA MATCHING

BALANCING EFFICIENCY AND EFFECTIVENESS BY MEANS OF CLUSTERING

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
under the authority of the rector magnificus,
prof.dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Friday, April 21, 2006 at 13.15

by

Marko Smiljanić

born on April 3, 1971
in Niš, Serbia

This dissertation is approved by:

Prof. dr. Willem Jonker (promotor)
Dr. Maurice van Keulen (assistant-promotor)

to life, wife, and daughters

Your children are not your children.
They are the sons and daughters of Life's longing for itself.
They come through you but not from you,
And though they are with you, yet they belong not to you.
You may give them your love but not your thoughts.
For they have their own thoughts.
You may house their bodies but not their souls,
For their souls dwell in the house of tomorrow, which you cannot visit,
not even in your dreams.
You may strive to be like them, but seek not to make them like you.
For life goes not backward nor tarries with yesterday.
You are the bows from which your children as living arrows are sent forth.
The archer sees the mark upon the path of the infinite,
and He bends you with His might that His arrows may go swift and far.
Let your bending in the archer's hand be for gladness;
For even as He loves the arrow that flies, so He loves also the bow that is stable.

On children / O deci

Kahlil Gibran [37]

Vaša deca nisu vaša deca.
Oni su sinovi i kćeri Života koji žudi za sobom.
Oni dolaze kroz vas ali ne od vas.
I mada su sa vama, vama ne pripadaju.
Možete im dati vašu ljubav ali ne i vaše misli.
Jer oni imaju sopstvene misli.
Možete udomiti njihova tela ali ne i njihove duše,
Jer njihove duše borave u kući sutrašnjice, do koje vi ne možete,
čak ni u snovima.
Možete se truditi da budete kao oni, ali ne tražite da oni budu kao vi.
Jer život ne ide unazad, niti stoji.
Vi ste lukovi sa kojih se vaša deca, kao žive strele, šalju unapred.
Strelac vidi znak na stazi večnosti,
i savija vas svojom moći da bi njegova strela otišla brzo i daleko.
Neka vaše savijanje u strelčevoj ruci bude radost.
Jer kao što voli strelu koja leti, On voli i luk koji je postojan.

Acknowledgments

To all the bows who make me fly,
to all the arrows I send ahead.

Thank you.[1][2][3][4]

Svim lukovima koji čine da letim,
svim strelama koje šaljem unapred.

Hvala Vam.[1][2][3][4]

[1]**science:** Willem Jonker, Maurice van Keulen, Peter Apers, Henk Blanken, Maarten Fokkinga, Ling Feng, Paolo Atzeni, Geert-Jan Houben, Franciska de Jong, Martin Kersten, Herman Balsters, Roel Wieringa, Slobodanka Djordjević-Kajan, Vojkan Mihajlović, Henning Rode, Milan Petković, Henk Ernst Blok, Djoerd Hiemstra, Jan Flokstra, Nirvana Meratnia, Arthur van Bunningen, Sandra Westhoff, Suse Engbers, Joeri van Ruth, Ander de Keijzer, Sander Evers, Ana Ivanović, Jelena Marinčić, Novica Zarvić, Wijnand Derks, Nicolas Anciaux, Pavel Serdyukov, Rick van Rein, Klaas Sikkel, Pascal van Eck, Rob van de Weg, Zlatko Zlatev, Maya Daneva, Dulce Pumareja, Andreas Wombacher, Manfred Reichert, Remco de Vos, Dejan Rančić, Leonid Stoimenov, Dragan Stojanović

[2]**family:** Marina, Marija, Vesna, Zlata, Svetislav *Smiljanić*, Petar, Nikola, Nataša, Zvonimir *Georgijev*, Ivana, Dragica *Marinković*, Kees Jr., Andjelka, Kees *Thijssen*, families Zmrzlikar, families Kuharič, families Mlosavljević, families Georgijev, families Gajić, families Jerinić, families Smilianitch

[3]**friends:** Sofka Trajčevska, Vojkan Mihajlović, Duško Jovanović, family Jovanović, family Petković, family Potić, family Kuzmanović, family Golo, family Stančić, family Mihajlović, family Ognjanović, family Erić, family Stojanović, families Veličković, family Živković (Enschede), family Živković (Den Haag), Tanja Djekić, Zoran Živković, Dejan Radivojević, Bojan Orlić, Ljubomir Manola, Nataša Jovanović, Tanja Topalović, Katarina Babić, Bart Wentink, Dragan Knežević

[4]**other:** all people who are in my mind but not in this list.

Contents

Acknowledgments **ii**

1 Introduction **1**
1.1 Finding information on the Internet 1
1.2 Information representation on the Internet 3
1.3 Tools for finding information on the Internet 5
1.3.1 Web directories 6
1.3.2 Web search engines 8
1.3.3 Web portals - mediators 9
1.3.4 Free querying of heterogeneous structured sources 10
1.3.5 Personal schema based querying 13
1.4 Schema matching in personal schema based querying 14
1.5 Research questions, contributions, and thesis organization 16

2 Understanding the schema matching problem **19**
2.1 Introduction 19
2.2 State of the art in schema matching 20
2.2.1 Schema matching and its applications 20
2.2.2 Mainstream schema matching systems 23
2.2.3 Other schema matching approaches 31
2.3 Formalizing the XML schema matching problem 32
2.3.1 What is a semantic schema matching problem? 33
2.3.2 Approximating semantic XML schema matching 35
2.3.3 Formal specification of the problem 39
2.3.4 The benefits of using COP framework 42
2.3.5 Other formalization efforts 42
2.4 Conclusion 43

3 Using clustering to increase the efficiency of schema matching **45**
3.1 Introduction 45
3.2 Improving the efficiency of schema matching 46
3.2.1 Efficiency improvements based on pre-computation 47
3.2.2 Efficiency improvements based on pruning of computations 48
3.3 Clustered schema matching 49
3.3.1 Introducing clustering into the schema matching architecture 49

3.3.2 Clustering criteria 54
3.3.3 Clustering transforms the schema matching problem 56
3.4 Choosing the clustering algorithms 58
3.4.1 Overview of clustering algorithms 58
3.4.2 K-means clustering algorithm 62
3.5 Related research 63
3.6 Conclusion 64

4 Computing effectiveness bounds 67
4.1 Introduction 67
4.2 Standard validation approaches 68
4.2.1 Validation of efficiency 68
4.2.2 Validation of effectiveness 69
4.2.3 Validation of effectiveness in large scale problems 72
4.3 Introduction to effectiveness bounds approach 74
4.4 Computation of effectiveness bounds 78
4.4.1 Best and worst-case analysis 78
4.4.2 Computing the best/worst case *P/R* curve 81
4.4.3 Examples of P/R curve bounds 85
4.4.4 Using the *P/R* curve of a random system for lower bound . 88
4.5 Interpolation in effectiveness bounds computation 89
4.5.1 Using an interpolated *P/R* curve as input 89
4.5.2 Sub-increment level bounds 90
4.6 Conclusion 91

5 Clustered schema matching in tree-based repositories 93
5.1 Introduction 93
5.2 Overview of the experimental approach 94
5.2.1 Experimental workflow 94
5.2.2 Definition of an experiment 96
5.2.3 Element matching 97
5.2.4 Mappings combining 98
5.2.5 Clustering 100
5.2.6 Validation of clustered schema matching 101
5.3 Clustered schema matching algorithms in Bellflower 104
5.3.1 Element matcher 104
5.3.2 Mappings combiner 105
5.3.3 Clusterer 109
5.4 Experimental repositories 113
5.4.1 Harvesting the Internet to build schema repositories 113
5.4.2 Properties of the experimental repositories 114
5.5 The influence of element matching 116

5.6 Setting up the clustering algorithm . . . 119
5.6.1 Initialization . . . 119
5.6.2 Centroid . . . 125
5.6.3 Reclustering . . . 127
5.6.4 Convergence criteria . . . 133
5.7 Effectiveness and efficiency of clustered schema matching . . . 136
5.8 Correlation between clustering and schema matching . . . 143
5.9 Scalability of the clustered schema matching technique . . . 145
5.9.1 Revisiting the complexity of clustered schema matching . . . 146
5.9.2 Scalability discussion . . . 147
5.10 Summary . . . 149

6 Conclusion **151**
6.1 Introduction . . . 151
6.2 Understanding the schema matching problem . . . 152
6.3 Improving the efficiency of schema matching systems . . . 153
6.4 Validating effectiveness . . . 155

A The experimental framework **157**
A.1 Introduction . . . 157
A.2 Experimental framework . . . 157
A.3 Architecture of Bellflower . . . 159

Bibliography **163**

SIKS Dissertation Series **171**

Summary **179**

Samenvatting **181**

Chapter 1

Introduction

1.1 Finding information on the Internet

There are 32 million active websites on the Internet [63] accessed by one billion Internet users [16]. In such a colossal system one challenge prevails. *How to quickly find the exact information one is looking for?*

What is information?
There exists no universal definition of *information* [7, 30, 86]. However, in information science it has been widely accepted that *information = data + meaning. Data* is considered to be raw and meaningless, such as "alternating current" or "1891". On the other hand, the following sentence conveys *information*: "In 1891, in Telluride–Colorado, Nikola Tesla and George Westinghouse set up the first commercial alternating current power plant." In information, data is given meaning by means of *relational connections*, such as the relation between "alternating current" and "1891" in the sentence given above. While raw data cannot answer any questions, information provides answers to questions such as *where*, *what*, *who*, and *when*.

Why find information?
Humans look for information in order to refresh, complete, or extend the information which they already posses. From a broader perspective, information finding is part of the learning process through which humans enlarge their *knowledge* and *wisdom* [7]. From a more concrete perspective, humans look for information when trying to find specific *data*, such as a date in "*When* was the first commercial alternating current power plant built?", or when trying to discover a relation between some data as in "*what relates* the alternating current and the year 1891?"

Internet as the superset of all the information sources
Information comes in variety of forms. Speech and music, video and animation, text and hypertext, vibration of the force-feedback joystick, and even smell and taste produced by computer controlled generators, they all carry information. Many businesses only exist to organize and distribute information: libraries, publishing companies, telephone companies, radio and television broadcasters. Vari-

ous technologies are used to store and exchange information, such as paper, radio, television, CDs, and finally the Internet. Internet supports all the information forms that can be digitized, and implements most of the legacy conventions for information exchange, such as mail, libraries, and television broadcast, and furthermore, constantly introduces new ones. *Blogs*, *SPAM*, *Peer-to-peer file sharing*, *chat rooms*, *free telephone*, *on-line universities*, and publicly edited encyclopedias *Wiki*-s[1], are some of the Internet-induced inventions. Though just a decade old, Internet outgrew all predecessor technologies and is nowadays the biggest and the most important information source on Earth.

Problems with information finding on the Internet

The already large volume of information on the Internet is constantly growing and changing. *Information finding* in such an environment is only possible with the use of information finding tools. The search engine, as the most popular information finding tool, has come to be indispensable for Internet users. Though very useful, current information finding tools are not perfect and still have many problems to solve. These problems fall in two broad categories.

First, *technical* or *syntactic* problems. The syntactic problems are related to *data* rather than to *meaning*. These problems address the low level issues of data management such as data representation, data storage, and data exchange protocols, to name a few. To a large extent, technical problems have been solved through the use of mature technologies, such as databases [28] for storing and querying large volumes of data, and through the use of common Internet protocols [71] for seamless data exchange on the Internet. Thanks to these technologies, today's Internet users are mostly unaware of technical problems.

The second category of problems are the *semantic* problems. These are related to the *meaning* of *data*. Semantic problems occur when there is a disagreement about the meaning, interpretation, or the intended use of some data [77] and are caused by the fact that the computer system which manipulates the information cannot really understand the meaning of the information. This phenomenon is also known as the *semantic gap*, referring to the large gap between how computers and humans *understand* information. Internet users are well aware that semantic problems exist. For example, a search engine returns thousands of documents for every query, and it is up to the user to analyze and understand which of these documents, if any, properly answers the question. Ideally, a search engine would return only one document or answer: the correct one.

In trying to solve the semantic problems on the Internet, science pursues two complementary research directions: (1) it develops smarter tools, which try to capture and use the meaning of information, and mimic human reasoning when answering user queries, (2) it investigates ways to represent (i.e., store) and orga-

[1] see WikipediA at `www.wikipedia.org`

nize information in such a way that not only data, but also the meaning of data, becomes explicit and machine readable.

To illustrate these approaches, the following sections briefly survey *information representations* (Sec. 1.2) and *information finding tools* (Sec. 1.3) on the Internet. Sec. 1.3.5 proposes *personal schema based querying* as a new way for gathering useful information from the XML portion of the Internet.

1.2 Information representation on the Internet

HTML

The year 1990, when Tim Berners-Lee wrote the first ever Web page, marks the beginning of the Internet's rapid expansion. Wide acceptance of the World Wide Web service induced an exponential growth of the volume of information available on the Internet.

Information on the Web resides within simple text documents enriched with elements of the *Hypertext Markup Language (HTML)* [93]. The HTML provides a fixed set of annotations most of which specify how to display the content of the Web document to the user. For example, the HTML line: `<i>Nikola Tesla</i> was born on <b>July 10, 1856</b>`, displays as: *Nikola Tesla* was born on **July 10, 1856**. With this approach the information within the Web page is understandable only to humans. HTML does not encode the meaning of data. HTML does, however, support the creation of links between HTML documents, i.e., *hypertext*, which enables large-grain information structuring. Still, data in the HTML documents is said to be *unstructured* and unsuitable for automatic processing. Consequently, compared to plain text documents, HTML brings moderate benefit to information finding tools.

XML

HTML quickly became a bottleneck in the effort to store and manage huge volume of data on the Internet. In May 1996, Jon Bosak became the leader of the group responsible for adopting SGML for the use on the Internet; *Extensible Markup Language (XML)* was created and became a standard W3C recommendation in February 1998. XML was designed to enable the creation of *structured* Web documents. XML allows authors of documents to use an arbitrary hierarchy of tags to annotate information, for example: `<scientist><name>Nikola Tesla</name> <dob>1856-07-10</dob></scientist>`. Unlike HTML, the XML tags, e.g., `<scientist>`, `<name>` and `<dob>`, do not give instructions on how to present the information: they simply describe and structured data. XML documents are further processed as needed. For example, to display information, technologies such as XSLT or XQuery [93], transform an XML document to a desired HTML document.

XML brought new possibilities for information finding. Information is represented in a structured way and information finding tools can exploit the explicit relationships that exist in the XML documents, and more precisely answer queries such as *find names of scientists* or *what is the date-of-birth of Nikola Tesla*. Though XML enriches the representation of data, the main semantic problems remain. Machines still do not understand what information means, only that there exist certain relations between data. Furthermore, these relations are not consistent over the Internet: there exists no universal convention on how to use XML to model information, and XML authors use different XML structures to model the same concepts, e.g. <`car`> and <`automobile`>. Consequently, XML documents on the Internet are highly heterogeneous. To partially overcome this problem, smaller communities develop and adopt standardized *XML schemas* to which XML documents have to adhere.

RDF, OWL (Semantic Web)

In 1998, Tim Berners-Lee conceives the Semantic Web: an information representation technique geared towards enabling fully automatic reasoning over the represented information. The core components of the technique are the *Resource Description Framework* (RDF) and the *Web Ontology Language* (OWL). RDF models information as a set of *subject-predicate-object* triples, where *predicate* is a directed relation between two resources: the *subject* and the *object*. For example, the following set of statements models information on Nikola Tesla.

```
man2987 has-name "Nikola Tesla"
man2987 has-vocation Scientist
man2987 was-born-on 10/07/1856
```

RDF models can be encoded in different ways. For use on the Internet, RDF models are encoded using XML.

To avoid the modeling heterogeneity in Semantic Web, RDF must use *predicates*, *subjects*, and *objects* which are defined in some vocabulary. In the Semantic Web such vocabularies are called *ontologies*, and OWL is the language used to specify the ontologies. Ontologies define classes, relations, functions, axioms, and instances [20] that can be used to represent information in the Semantic Web. Apart from being used as vocabularies, ontologies are also used to perform inference-based reasoning.

In an ideal Semantic Web, the semantic gap would not exist. In information finding, a user would be able to express his information need in terms of a known ontology, and the information finding system would use simple matching and inference, to generate the exact answer. However, the creation of ontologies is a huge challenge, and similar to XML schemas, the standardization of ontologies is confined to smaller communities. For this reason the Semantic Web is nowadays perceived with both optimism and skepticism. Some believe that the obstacles in

creating stable and widely accepted ontologies are too big to ever overcome [10]. In smaller communities, the Semantic Web is expected to have brighter future.

▷ research focus

The three information representation techniques, HTML, XML, and RDF with OWL, currently coexist on the Internet and will remain so in the foreseeable future. However, the research in this thesis solely considers XML as a format for representing information. The simplicity and the maturity of XML technologies, and its wide acceptance in both commercial and scientific communities, motivates this choice. Furthermore, as both XML and the Semantic Web confront the problem of semantic heterogeneity, which in turn induces problems considered in this thesis, we believe that ideas presented in this thesis can be exploited in both the XML and the Semantic Web contexts.

1.3 Tools for finding information on the Internet

Over the years, several information finding tools have established themselves as an inseparable part of the Internet. Still, scientific research and commercial enterprises constantly look for ways to improve or deliver new information finding tools. This section first discusses the three most common information finding tools: *directories*, *search engines*, and *Web portals*, and then takes a look at research efforts for delivering more powerful information finding in XML environments.

Phases of the information finding process

The information finding process, used by tools mentioned above, is interactive. When seen from the perspective of the user, three interaction phases can be distinguished (see Fig. 1.1).

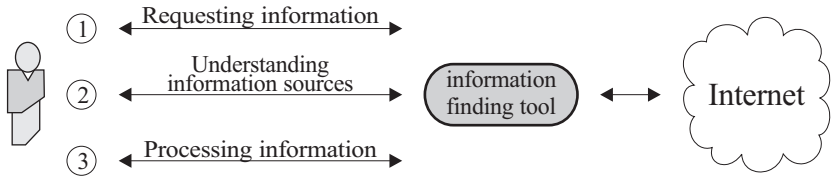


Figure 1.1 *Three interaction phases in the information finding process.*

(1) *Requesting information* is the phase in which a user expresses his *information need* by submitting a *query* to the information finding tool. After receiving the query, the task of the tool is to search the Internet in order to find the *information source* which contains the requested information.

(2) The *understanding information sources* phase occurs when the information finding tool has finished the search and has discovered multiple sources with information which, in the tool's "opinion", corresponds to the requested information. Due to the semantic gap, the information finding tool cannot reach a definitive understanding on the adequacy of the information sources it has discovered. Therefore, the user is asked to assess the proposed sources to reach this understanding by himself. Upon making up his mind, the user is expected to inform the tool of his decision by pointing to one or more information sources that he finds relevant.

(3) *Information processing* is the last phase. Information sources usually contain a large amount of information. For example, the whole biography of an actor is returned while the user only wanted to find the birthday of the actor. The information processing phase is thus the phase in which the user analyzes the information source to find information or data which fulfills his specific information need.

Different information finding tools provide different support to the user in these three phases, and differently organize the information finding workflow. For example, with search engines users often iterate over phases (1) and (2) several times before finding the desired information source. In some cases, the information processing phase (3) can in fact be a whole new information finding process.

1.3.1 Web directories

A Web directory organizes websites in a hierarchical catalog, much alike how a library catalog organizes books. The Web directory catalog comprises many semantic categories such as *Science* and *Music* which are sibling categories, or *Stringed Instruments* and *Guitar* which are parent and child categories. Each website in the directory is assigned to one or more such categories.

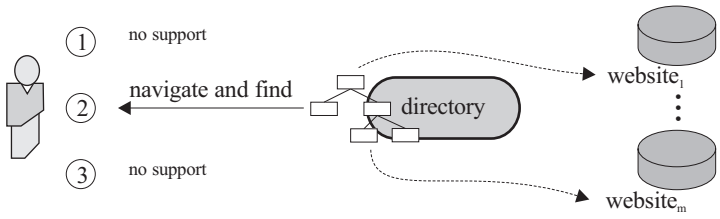


Figure 1.2 *Using a directory to find information.*

A Web directory is passive system. It enables the user to browse through the hierarchy of categories but does not provide any kind of active help. Fig. 1.2 illustrates the three phases of the directory supported information finding. Phase (1) does not exits; the user does not submit a query. Instead, the user proceeds

directly to phase (2) – information source understanding. In this phase the user is expected to find an information source which corresponds to his information need. The user navigates through the category hierarchy, understands the semantics of the categories and how these map to his information need, and eventually finds a website which seems to have the wanted information. The selected website can comprise just one page of text, or thousands of pages with multimedia content. Consequently, the information processing phase (3) can be very demanding and can require large additional efforts from the user. This phase is, however, not supported by a Web directory.

The creation and maintenance of a Web directory faces two semantic problems:

- the design of the category hierarchy,
- the assignment of websites to categories,

Category hierarchies are built by humans. For example, the most widely accepted classification for libraries is the *Dewey Decimal Classification* [65] proposed by Melvil Dewey in 1876. Dewey classification is not static but evolves under the control of more than 50000 libraries organized in the *OCLC Online Computer Library Center*. The most popular Web directories use exactly the same approach – they are developed and maintained within ongoing projects, such as *Open Directory Project*[2] and *Zeal*[3] . These involve hundreds of thousands volunteer Internet users, each maintaining a small part of the directory hierarchy. In such directories, humans are fully responsible for bridging the semantic gap: first, humans manually create the hierarchy of categories and classify the information sources, and second, humans manually search for a desired information source within the directory.

The size and the dynamic of the Internet information sources makes it very hard to keep the directories updated even by using broad human editorship. Hence the research in automatic classification of web content [76]. Automatic classifiers are mostly based on machine learning techniques; after going through a learning phase based on an initial set of manually classified documents, classifiers can autonomously categorize new documents.

Recently, directories and automatic classifiers are being combined with other information finding techniques. For example, the answers delivered by a search engine query can be presented in the form of a directory structure instead of a plain list [14]. This helps the user to better understand the semantics of the documents returned by the search engine.

[2]see Open Directory Project at `www.dmoz.org`
[3]see Zeal at `www.zeal.com`

1.3.2 Web search engines

The best known tool for finding any kind of information on the Internet is the *search engine*. A search engine is a kind of *Information retrieval (IR)* system [72] which sees the Internet as a document collection. In an IR system, the user expresses his information need with a query, and the IR system provides the answer by pointing to the documents which are the most relevant to the user's query.

Information finding with a search engine is illustrated in Fig. 1.3. In phase (1), the user expresses his information need with an information query k. The query is usually a list of words which are likely to appear together in a desired document, say "`book author Dickens`." The task of the search engine is to check every document doc_i, $i = 1 \ldots m$ on the Internet for the existence of the words given in the query. The search engine returns a list of document addresses, i.e., URIs, of documents which contain the words in the query.

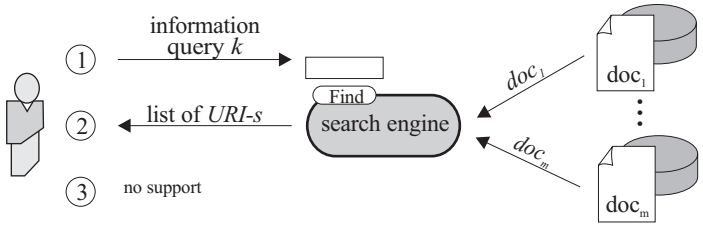


Figure 1.3 *Using a search engine to find information.*

Due to the size of the Internet and low expressiveness of the query, a search engine returns thousands of documents which match the query. To distinguish between these, a search engine ranks the documents using various heuristics. For example, documents get higher rank if the query words are close to each other and in the right order within the document, or if the document is often referenced on the Internet. The ranking algorithm is the "intelligent" part of the search engine as it tries to "understand" which of the documents are the most relevant for the query. Unfortunately, a search engine usually returns many answers which match the query and rank high, but do not in fact match the user's actual interests – a manifestation of the semantic gap.

For this reason, in phase (2), the user performs a semantic assessment of the proposed URIs. To help the user in understanding the proposed documents, a search engine shows excerpts form the documents making it possible for the user to see the context in which the words of the query appear. When the user identifies a document that matches his expectation, the information finding process moves on to phase (3). A search engine, in general, does not provide support for information processing, and the user processes the document either manually, e.g., by reading the document, or semi-automatically, e.g., by using regular expression matchers to find desired data.

1.3.3 Web portals - mediators

Internet can also be seen as a large distributed database [66, 77, 80] (as opposed to a document collection) populated with structured data, such as relational tables [28] or XML documents [93]. Each structured data source is characterized by a *schema* which defines how the data is structured. The strict adherence to a schema, makes it possible to effectively and efficiently query the data sources using formal query languages such as SQL or XPath/XQuery. Unfortunately, large distributed database systems such as the Internet, are not immune to the semantic gap problems. Due to the large number and heterogeneity of the data sources, it becomes hard to know the exact structure (schema) and the "meaning" of each data source. Consequently, it becomes hard for users to query distributed data sources. A common solution for this problem is provided by *Web portals*, in the database community commonly known as *mediators*. A mediator integrates data from several distributed data sources, exposing the data through a single *mediated view*. The user of the mediated view is unaware of the distributed nature of data and the intricate details of each individual data source. The user only sees the mediated view and expresses his queries in terms of the mediated view.

Users of the Internet regularly use mediated views to find information. For example, shopping portals provide a mediated view over information about products, where the product information originates from many different websites.

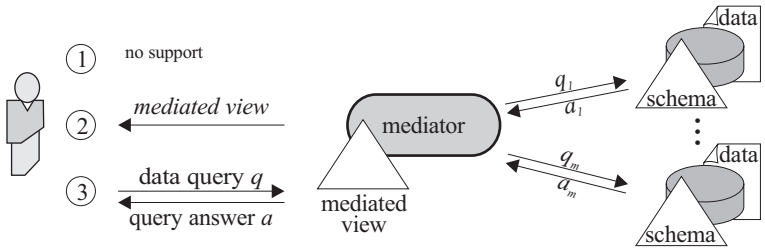


Figure 1.4 *Using a mediator (i.e., portal) to find information.*

Fig. 1.4 illustrates the information finding process based on a mediator. A mediator provides access to information related to one specific domain, for example, information about home electronic products. Hence, the user cannot freely express his information need, but is confined to the type of the information offered by a mediator. Consequently, phase (1) is not supported: the user cannot ask for different kind of information. Instead, the user only needs to the understand the semantics of the mediated view in phase (2). The strong point of the mediator is the capability to provide very specific answers in the information processing phase (3). The user can specify a formal data query, such as: find plasma TVs with screen larger than 30 inches, and the mediator will provide a specific answer.

Seen from the user's perspective, the structure of the mediated view is predefined and unchangeable. This constrains the user's freedom in the information search. However, the user can get very precise and specific information without having to worry about the semantic gap. Indeed, most of the burden for bridging the semantic gap falls on the designer of the mediated view – another human. The designer's task is to create the mediated view and to select the distributed data sources from which the actual information will be gathered. In this, the hardest task that the designer performs is the reconciliation of the information from distributed data sources with the structure of the mediated view.

A mediator fully automates the query answering process: the mediator rewrites the user's query q into queries on the real data sources $\mathit{rewrite}(q) \rightarrow q_1, \ldots, q_m$, (see Fig. 1.4). When executed, these queries provide answers $a_1, \ldots, a_m$ which a mediator integrates into a single answer $\mathit{integrate}(a_1, \ldots, a_m) \rightarrow a$. The integrated answer a is what the users sees. Alternatively, mediators can avoid on-demand querying and answer rewriting by maintaining local reconciled replicas of distributed data.

1.3.4 Free querying of heterogeneous structured sources

Fig. 1.5 shows one way to compare the information finding tools presented so far. In the figure, the x-axis shows the *querying freedom*: Web directories and search engines allow users to freely look or ask for any kind of information, regardless of the content or the way in which the information is structured. At the same time, Web portals always *constrain* querying to a certain domain and allow only certain type of queries. The y-axis indicates if the structure of the information source is

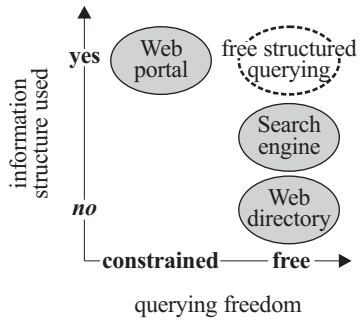


Figure 1.5 *The querying freedom and the use of structure for different information finding tools.*

exploited in information finding. For directories the structure of information plays no role, even if working with structured data sources. The user has no means of using the information structure to improve the search. With search engines,

structure, when available, might be used in a very restricted way. For example, search engines can give more importance to documents in which search terms belong to the same HTML element. Also, a user can be offered simple control over the structure of the Web documents, e.g., to search for terms appearing in the title section only. Finally, Web portals fully exploit the structure of information sources and enable users to ask very specific queries, but no free querying.

The number of XML data sources on the Internet is increasing, and the Internet converges from unstructured toward a more structured state. Current information finding techniques are not well suited for searching over large volumes of XML data. Web portals, as the only popular tool that fully exploits the structure of data, are not well suited for environments with large number of rapidly changing XML data sources. Currently, there are no widely accepted information finding systems which offer users to perform free querying over heterogeneous structured sources on the Internet: this empty region in Fig. 1.5 still has to be filled. When deployed, the benefits of such systems will be great. Users will use structured queries to more precisely specify their information needs; tools will find more relevant and very specific information.

Science is actively looking for ways to build systems for free querying of XML data (though not always for the Internet scale). The problem to overcome is a *semantic* one: users do not known the structure of the data in the information sources and the structure of the user's query is not guaranteed to match the structure of the information sources. And yet, systems are expected to provide *meaningful* answers. Systems have to find information that matches the meaning of the query.

In current research, two approaches are investigated for this problem: (1) the user is responsible for taking into account the possible mismatch between his query and the structure of data, and (2) this is the responsibility of the system. In some approaches, the division is not strong. Note, tools used for free querying of structured data on the Internet can be considered both as *information finding* and *querying* tools. Consequently, we use both terms to describe the purpose of these systems.

User responsible for representation mismatches

With this approach, XML query languages are extended with operators and functions which enable users to define relaxed match conditions for document parts for which they do not know the structure or content. For example, XIRQL [33] extends the XQL query language (language similar to XPath) with `#inode` to specify the elements for which the name is not known. Also an operator `$contains$` can be used to check if the context element contains a certain word. This operator uses stemming for approximate string comparison. The XXL system [87] uses the *semantic similarity operator* $\sim$ which is applicable to both elements and the content. For each answer, the actual similarity between the query and the answer is

quantified and used to perform relevance ranking of answers. Another example is the NEXI language [90]: a simplified XPath query language with an added `about` clause. The NEXI query `//article[(about(.//section,'science')]` asks for an article which has a part *similar to* `section`, where this part contains information about something similar to `science`. NEXI is widely used in the XML retrieval systems participating in INEX (INitiative for the Evaluation of XML Retrieval[4]), and each concrete system offers different implementations of the `about` clause. Another interesting approach is the *Meaningful Lowest Common Ancestor Structure*, i.e., `mlcas()` predicate, which can be added to XQuery [53]. A user forms an XQuery over unknown XML data using the descendant access step for various elements expected to exist in the data, e.g., <`name`> and <`address`>. The user then uses the `mlcas()` predicate to check if the retrieved pairs of XML elements <`name`> and <`address`>, when discovered in the same document, both belong to the same meaningful structure (i.e., have a meaningful common ancestor).

System responsible for representation mismatches
The second group of techniques allows users to ask queries using unmodified query languages, and to freely select the terms and the structure of the query. XSEarch [15], allows users to ask non-structured keyword-based query. The system uses the structure of the actual XML documents to determine if the search terms are connected in a meaningful way within the XML document. The strength of the connection is used to rank the answers. Another group of systems performs *query relaxation* [2, 3, 75]. In these systems users are allowed to ask structured queries. Such queries, when evaluated in a strict way, often return no answers as no information source fully matches the structure of the query. To solve this problem, these systems use a set of rules to transform the user's query into a more relaxed one, for example by substituting the child (`/`) XPath axis with the descendant (`//`) axis. The relaxed query has better chances of resulting in a non-empty answer set. If not, further relaxation is performed. An approach based on schema matching is encountered in relational databases. *Semantic query processing* allows the user to ask a query over a database without having to know the database schema [51, 70]. The user only specifies the SELECT and WHERE part of the SQL query using concepts which he is familiar with. It is up to the evaluation system to generate the FROM parts of the query by matching the users query against the schema of the database.

At this point, we introduce our technique for structured querying of unknown XML data: *personal schema based querying*. The technique belong to the second group of techniques, i.e., the systems takes the responsibility for resolving representation mismatches, and is similar to the idea of semantic query processing. The following section describes the technique.

[4] see INEX at `inex.is.informatik.uni-duisburg.de`

1.3.5 Personal schema based querying

We envision an information finding system which allows a user to define his own mediated view called *personal schema*. Through a personal schema, the user expresses his current information need and expectation with respect to the structure of the desired information. If the user is, for example, interested in books, he might define a personal schema similar to the one given in Fig. 1.6a. This per-

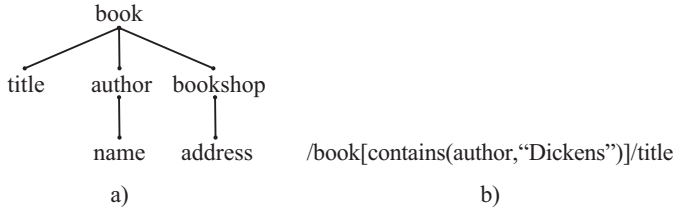


Figure 1.6 *a) personal schema, b) personal query.*

sonal schema embodies the user's private model of book-related information, and may be entirely different from the structure of the actual book-related information in the Internet, of which the user does not need to have any knowledge. In personal schema based querying, the user is also allowed to ask queries on his personal schema, such as the XPath query given in Fig. 1.6b. These queries are called *personal queries*. We call a system which implements personal schema based querying, a *personal schema based query answering system (PSQ)*. The information finding phases of a PSQ are illustrated in Fig. 1.7.

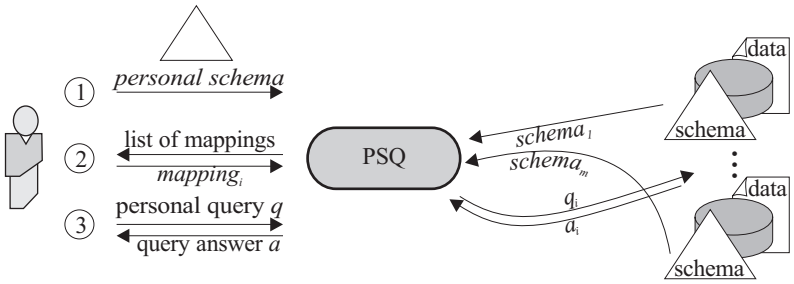


Figure 1.7 *Using PSQ to find information on the XML Web.*

In phase (1), the user submits his personal schema. The responsibility of the PSQ is to find data sources on the Internet, the schemas of which match, as good as possible, the user's personal schema. Just as a search engine finds many URIs for a given query, the PSQ finds many *schema mappings*. Each schema mapping is one way of relating elements of the personal schema with semantically

similar elements in the Internet schemas. In phase (2) the discovered mappings are presented to the user. The user assesses the proposed mappings in order to check if these are semantically sound, i.e., the user checks that the meaning of the personal schema and the meaning of the corresponding schema on the Internet are indeed equal, or at least similar enough. If so, the user decides to use a specific mapping, e.g., $mapping_i$ in Fig. 1.7, to retrieve the actual information. In the information processing phase (3), the PSQ allows the user to ask an arbitrary personal query q for which the user gets answer a.

▷ research focus

The research in this thesis is solely built around the idea of personal schema based querying. However, the idea of personal schema based querying is not entirely new and shares many properties and problems with other techniques mentioned above. In that respect we believe that this thesis addresses problems of interest to the broader area of structured querying of distributed XML information sources.

1.4 Schema matching in personal schema based querying

When implemented, a PSQ must have at least two parts: a *schema matcher* and a *query evaluator* (see Fig. 1.8).

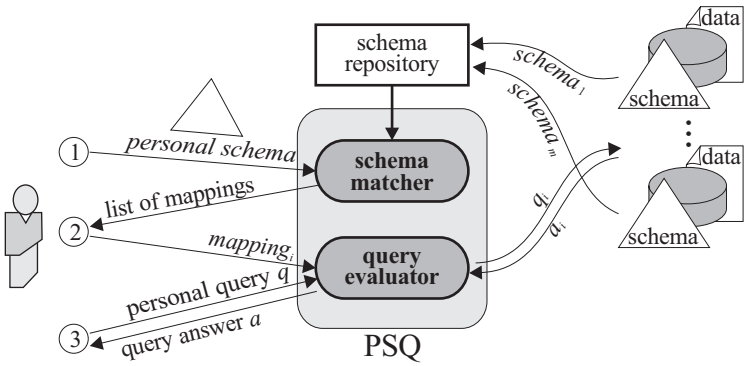


Figure 1.8 *Two components of a PSQ.*

The schema matcher is responsible for matching the personal schema, supplied by the user, against the schemas of the Internet, which are presumably stored in a large *schema repository*. The schema matcher delivers a list of possible mappings between the two.

The following example illustrates a schema matching problem. Fig. 1.9a shows a very simple personal schema of a user interested in countries and names of their

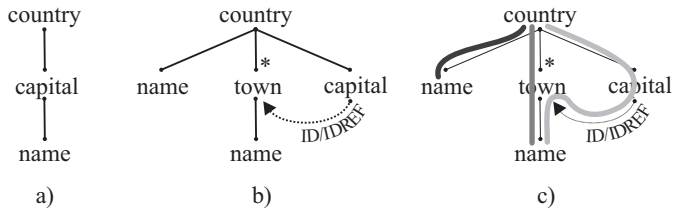


Figure 1.9 *Simple schema matching example.*

capital cities. Fig. 1.9b shows a very small repository. In the repository, name of the country is given, along with names of towns, and a capital city which is specified using referential integrity constraint (ID/IDREF [93]). When matching the given personal schema against this repository, the schema matcher discovers three possible mappings shown in Fig. 1.9c. To decide which of the mappings is semantically the most similar to the personal schema, the schema matcher uses different heuristics and computational techniques to produce the similarity indexes for each considered mapping. In the case given in Fig. 1.9, the schema matcher should compute the highest similarity index for the mapping between the personal schema and the repository path `country/capital/town/name` (the brightest path in Fig. 1.9c).

Schema matching is a known problem explored in the context of many different applications, for which different solutions have been proposed (see Sec. 2.2). Though extensively researched, schema matching has so far not been seen as a tool for on-line usage by non-expert users. On the contrary, matching systems were designed to be used off-line by domain experts. Furthermore, in research, experimental systems were validated using small scale matching problems in which the efficiency was not an issue. The efficiency of schema matching in large scale environments, such as PSQ, has not been sufficiently investigated and is an open challenge.

The second component of the PSQ is a distributed query evaluator (see Fig. 1.8). With the schema mapping known, the query evaluator rewrites the personal query q into a query over a concrete data source q_i, and optionally, transforms back the answer a_i into the structure corresponding to that of the personal schema. Distributed query evaluation based on mediated views has been extensively researched [48, 80, 79]. Though not considered in the context of PSQ, the use of operators for approximate querying, such as the `about` clause in the NEXI language [90], could require the modification of classical techniques for distributed querying.

▷ research focus

In this thesis we solely address the problems related to the *schema matcher* component of a PSQ, and in particular the *efficiency of schema matching*. Recent

schema matching techniques report considerable effectiveness [21], however, the efficiency aspects of these techniques are largely ignored. But the efficiency of schema matching in a PSQ cannot be ignored: in a PSQ the efficiency of schema matching is equally important as the effectiveness. Today's Internet users are accustomed to sub-second response times of a search engine. Without an efficient and effective schema matching algorithm, it is not likely that a system like a PSQ would be considered useful. The same challenge has been recognized in other related domains, e.g., in industrial-strength matching systems [9]. For these reasons we see the efficiency of schema matching, while retaining the effectiveness, as one of the top challenges in building a PSQ, and we place it in the focus of our research (see Fig. 1.10).

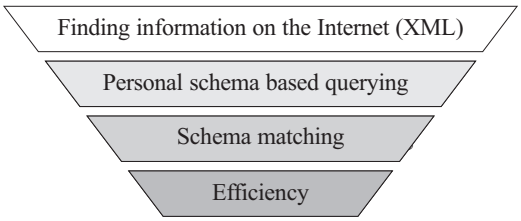


Figure 1.10 *Research focus in this thesis.*

1.5 Research questions, contributions, and thesis organization

This thesis comprises three parts focusing on three different problems. The problems and the chapters in which these are discussed are:

- Understanding the schema matching problem (chapter 2).
- Improving the efficiency of schema matching (chapters 3 and 5).
- Validating the performance (chapter 4).

The research questions and the contributions related to each of the three problems are introduced below. Apart form these, the thesis also presents conclusions in chapter 6, and the description of the experimental system Bellflower in appendix A.

Understanding the schema matching problem[5]

The best way to reach full understanding of a problem is by making a complete, possibly formal, problem specification. This thesis proposes a formal specification for the problem of *semantic XML schema matching*. Semantic schema matching

[5]this part of the research has been published in [82]

has been extensively researched, and many matching systems have been developed (see Sec. 2.2). However, formal specifications of problems being solved by these systems do not exist, or are partial. The thesis analyzes the problem of semantic schema matching, identifies its main components and makes a formal specification based on the *constraint optimization problem (COP)* formalism [82]. The benefit of formalizing a schema matching problem as a COP is that a schema matching problem can now be regarded as a combinatorial optimization problem with constraints. The thesis shows that finding good mappings for a schema matching problem amounts to finding good solutions for a COP problem. COP problems have been extensively investigated and many techniques for efficient solving have been proposed [58, 60]. Branch and bound, clustering methods, simulated annealing, tabu search, to name a few, can be investigated and adopted to schema matching problems represented as COPs.

Improving the efficiency of schema matching[6]

The COP formalism reveals that the schema matching problem in personal schema matching has a polynomial complexity in size of the schema repository. Consequently, naive schema matching implementations which enumerate the complete problem search space, are not applicable in large scale environments.

The techniques used to efficiently solve constraint optimization problems all try to reduce the number of candidate solutions, i.e., valuations, which are generated and tested before the best solutions are found. The techniques try to prune, as soon as possible, the areas of the search space in which there is no chance of finding a good solution.

This thesis proposes, in chapter 3, an efficiency improvement technique based on *clustering* [84]. The technique is inspired by an observation that for a certain personal schema, good mappings are often found in limited areas within the schema repository. Clustering is used as a tool to quickly identify such areas and then restrict the expensive schema matching algorithms to these areas instead of searching through the whole repository. This technique, called *clustered schema matching*, adds a clustering step to an existing schema matching system.

In chapter 5, the thesis describes one concrete implementation of the clustered schema matching technique: the prototype system Bellflower. Bellflower implements a simple non-clustered schema matching technique, and adds a clustering step to build a clustered schema matching system. Various experiments are performed to analyze the properties of the technique, such as the trade-off between efficiency and effectiveness. Bellflower experiments led to identification of several conceptual and low-level problems for some of which the thesis proposes solutions.

The clustered schema matching technique is a complex technique based on interoperability of multiple algorithms. Consequently, the performance of the tech-

[6] this part of the research has been published in [84]

nique depends on many different choices and parameters. The thesis investigates clustered schema matching in a holistic way and analyzes how its numerous components interact and influence the final performance. The depth in which each of the components is individually investigated, however, is limited.

Validating the performance[7]

The performance of a schema matching system consists of efficiency and effectiveness. In this thesis both the efficiency and the effectiveness of a clustered schema matching system are observed by comparing it to a non-clustered schema matching system. Validation is based on the prototype system Bellflower.

The *efficiency*, which expresses how much faster one system performs than the other, is in Bellflower measured by means of timers and counters. The validation of the *effectiveness* is a harder problem. Effectiveness is usually expressed through *precision* and *recall*, the computation of which requires significant human effort. This makes the effectiveness validation in large scale schema matching very expensive and slow. To sufficiently overcome this problem, the thesis uses a validation approach which exploits the fact that the clustered schema matching technique is an extension of an existing non-clustered matching system. The validation approach compares sets of resulting mappings delivered by the clustered and the non-clustered systems to make conclusions about the effectiveness of the clustered schema matching system. In the thesis, this validation approach is used to compare different variants of the clustered schema matching technique, and to observe the positive and negative changes in effectiveness caused by changes in certain parameters or conditions.

Chapter 4 shows that this validation approach indeed relates to classical precision and recall measures. The chapter describes a mathematical method which, under some constraints, computes the exact effectiveness bounds, i.e., the worst and the best case precision and recall of the clustered schema matching system [83]. Though this technique does not calculate the actual effectiveness, the lower effectiveness bound guarantees the worst case effectiveness. Also, the width of the bounded area, which varies over different clustering techniques, provides more information on the potential real effectiveness of the system. Nevertheless, this computational method, apart from its clear theoretical value, still has to show its practical utility.

[7] This part of the research has been published in [83]

Chapter 2

Understanding the schema matching problem

2.1 Introduction

Prior to solving a problem, one must understand the problem he is trying to solve. This common knowledge has driven the research presented in this chapter.

Personal schema based querying, our motivating application introduced in chapter 1, depends on a component called *schema matcher*. The schema matcher is a software component which tries to discover corresponding elements in two schemas, that is, tries to discover schema elements which model the same real-world object or concept. We say that schema matcher solves the *schema matching problem*.

Schema matching has been extensively researched, and many matching systems have been developed (see Sec. 2.2). These systems make use of various heuristics to detect corresponding schema elements. Schema matching research mostly focuses on the effectiveness of schema matching, i.e., on how well schema matching systems recognize corresponding schema elements. On the other hand, not enough research has been done on formal foundations of the schema matching problem (see Sec. 2.3.5).

In this thesis, our goal is to improve the efficiency of schema matching. To be able to devise efficient algorithms for solving the schema matching problem, we need a profound understanding of all the problem components which contribute to the problem's complexity. The best way to acquire such understanding is by making a complete formal specification of the schema matching problem. For this, we did not find a suitable formalism in the existing schema matching research, and have resorted to introducing our own formalizing approach. This chapter introduces the approach. We emphasize that this chapter does not try to provide new ways to solve the schema matching problem, but rather introduces a complete, formal, and declarative way to specify a problem that needs to be solved.

The chapter has two parts. In the first part, we introduce schema matching and its applications, and we bring a generic and a specific view on several prominent schema matching systems. We need this knowledge to extrapolate the main

components of a schema matching problem. In the second part of the chapter, we introduce a formalization approach which accommodates, in a formal way, all the schema matching problem components. Please note that the approach primarily targets the schema matching problem encountered in the context of personal schema based querying, in which a single personal XML schema is matched against a large XML schema repository.

2.2 State of the art in schema matching

2.2.1 Schema matching and its applications

A schema, i.e., *database schema*, is a *model* of a *universe of discourse*. It is a formal declarative model of how to represent, within a database, a set of real-world objects. Schemas are represented using dedicated declarative languages. There exist a large number of different schema languages: different data models make use of different schema languages, but also, more than one schema language can exist for the same data model. For example, the *relational data model* [28] most commonly uses the *Data Definition Language (DDL)* and the XML data model makes use of the *DTD* language or the *XML Schema Language* [93]. Schema languages provide various features which can be used to build schemas. XML schema languages differ in the expressiveness, and the "ease of use" they offer to the schema designer [50]. We list several features found in different XML schema languages:

- *Data types*: schema languages provide built-in data types, ways for declaring user-defined data types by means of restriction and extension, and ways to handle *null* values,
- *Elements*: schema languages provide techniques to declare simple and complex elements, element cardinality constraints (min and max occurrence), default values, ordering, choices among elements, and conditional element existence,
- *Integrity constraints*: schema languages allow the declaration of *unique* elements (no duplicates allowed), *key* elements and *foreign key* elements,
- *Miscellaneous features*: schema languages provide functionality such as internal *documentation* or *version* information.

There exist no formal and universal rules on how to use the schema language features to create models of the universe of discourse. Consequently, schema design is a subjective and creative activity heavily influenced by the experience of a human designer. As a result, schemas differ significantly, even if modeling the same real-world object.

Fig. 2.1 shows tree representations of two example XML schemas. Both schemas model the same real-world concept – *library*, and yet, being designed by two

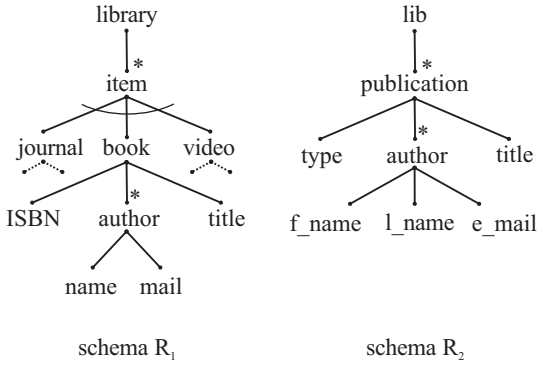


Figure 2.1 *Two XML schemas modeling a library.*

designers, the schemas differ significantly:

- schema R_1 uses the <`library`> element and schema R_2 uses the element with abbreviated name <`lib`> to depict the same concept,
- schema R_1 represents a book with an instance of a <`book`> element, schema R_2 represents a book with an instance of a <`publication`> element in which the sub element <`type`> has the value "`book`",
- schema R_1 contains the <`ISBN`> element, which does not exist in schema R_2
- schema R_1 represents the author's name with one XML element <`name`>, schema R_2 represents the author's name with two XML elements <`f_name`> and <`l_name`>,
- it is not clear whether the element <`mail`> in schema R_1 represents a postal address or an e-mail address; it is clear that in schema R_2 element <`e_mail`> represents an e-mail address.

In distributed applications, there is often a need to exchange data between such *semantically similar*, yet *syntactically different* schemas. These applications must find a way to know which elements, coming form different schemas, represent the same real-world concept. We list few such applications.

- *Web-portal* (discussed in Sec. 1.3.3) provides a mediated schema through which a user accesses multiple distributed data sources. Distributed data sources have heterogeneous schemas different from the mediated view. Web-portals must know which elements in the source schemas correspond to which elements in the mediated view.
- *Data warehouse* stores data using a local data warehouse schema. The actual data coming from various heterogeneous sources has to be transformed to match the warehouse schema.
- *B2B message exchange* requires that an XML message generated by *Com-*

pany A is translated into an XML format understood by *Company B*.

- *Web-services* are characterized by their input, output, and additional metadata. To automatically match the consumer of the service with a service provider, or to be able to compose multiple services to provide a new one, an adequate parameter matching must be performed.
- *System migration* often involves the migration of data from a legacy database into a new database system. This migration can involve a change of the schema or even a change of the data model. Alternatively, wrappers can be built to enable the use of legacy databases through new interfaces. In both cases, finding correspondences between schemas is essential.
- *Personal schema based querying* described in Sec. 1.3.5, needs to find correspondences between the elements of a personal schema and elements of schemas available on the Internet.

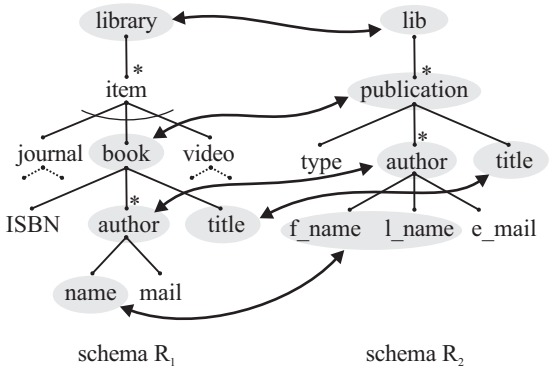


Figure 2.2 *One possible schema mapping.*

The activity which tries to discover semantic correspondences between the components of two schemas is commonly called *schema matching*. Given a schema R_1 with a set of schema components $R_1 = \{n_1, \ldots, n_p\}$ and a schema $R_2 = \{n'_1, \ldots, n'_q\}$, the goal of schema matching is to discover *component mappings*, i.e., pairs of matching schema components n and n' such that n and n' represent the same real-world concept. We denote a *component mapping* with $n \mapsto n'$, where n is the *mapped component* and n' is the *mapping component*. When put together, component mappings define a *schema mapping* $a = \{n \mapsto n' \mid n \in R_1, n' \in R_2\}$. Note that we use the term *component* instead of the term *element*, to indicate that schema matching can look for correspondences not only between XML elements but also between a more complex schema constructions. However, when mapping an element to an element, terminology will change accordingly to *element mapping* for $n \mapsto n'$, where n is the *mapped element* and n' is the *mapping element*.

A possible schema mapping a_1 between schemas in R_1 and R_2 is illustrated in Fig. 2.2, where $a_1 = \{$<library>$\mapsto$<lib>,<book>$\mapsto$<publication>,<author> $\mapsto$<author>, <title>$\mapsto$<title>, <name>$\mapsto q($<f_name>, <l_name>$)\}$, where q is some function, e.g., a query.

Schema mappings are not necessarily correct: for example, a human observer might argue, that in the previous matching example, the component mapping <book> $\mapsto$ <publication> could be replaced with <item> $\mapsto$ <publication>. This kind of uncertainty must be accounted for and expected in schema matching. In general, schema matching produces multiple schema mappings with different levels of correctness.

Schema matching can be preformed manually. A human can observe schemas, reason about the semantic similarity of schema components, and type in (or "mouse in") the discovered correspondences. Manual schema matching is, however, expensive, hard, and errorprone. It cannot support today's large and dynamic Internet-based information exchange. Therefore, efforts are vested in the development of automated schema matching systems. Though it is impossible to fully automate schema matching [17], even semi-automated approaches bring important improvements to the schema matching process.

Before introducing the systems for automated schema matching, we need to point to another problem which is very related to schema matching, but should not be confused with it; the problem of *query discovery*. While schema matching tries to discover semantic correspondence between schema components, query discovery aims at building precise queries which can be used to transform data from one schema into another. Query discovery can be seen as a step coming after and complementing the schema matching step. In our previous example, once the component mapping <name> $\mapsto q($<f_name>, <l_name>$)$ is discovered by a schema matcher, a query discoverer could be used to generate the actually query q. For example, q can contain the following XQuery fragment:

<name> { $a/f_name/text(), " ", $a/l_name/text() } </name>

This XQuery transforms the author's name as given in schema R_2 into author's name as given in schema R_1 by concatenating the first and the last name into a single string. Clio is an example of systems for semi-automatic query discovery [61].

In our research we focus exclusively on the schema matching problem involving XML schemas. We assume that schemas are specified using the *XML schema language* [93].

2.2.2 Mainstream schema matching systems

Schema matching research has delivered a number of automated schema matching systems. These systems rely on the expectation that the similarity of the representations, i.e., the *syntactic similarity* of schema components, indicates the *semantic*

similarity of real-world objects modeled with these components. Matching systems use heuristics, also called *hints*, clues, or evidence, to compute the semantic similarity of schema components. Hints, in general, are not very accurate. For example, similarity of element names can correctly indicate semantic similarity when comparing elements <auto> and <automobile>, but can be misleading in the comparison of <star> and <start>. The general approach in schema matching is to use multiple hints. This improves the probability that for each matching situation at least some hints will correctly detect semantic similarity. Schema matching can also be thought of as the reverse-engineering of the schema design process. In order to be successful, a schema matching system should be able to reverse-engineer many different design approaches.

We now describe several well known approaches to schema matching. In particular we consider five schema matching system: Similarity Flooding [59], Cupid [57], COMA [22], LSD [24], and iMap [19]. Though different in many ways, these systems can be described through a common architecture. The architecture is illustrated in Fig 2.3.

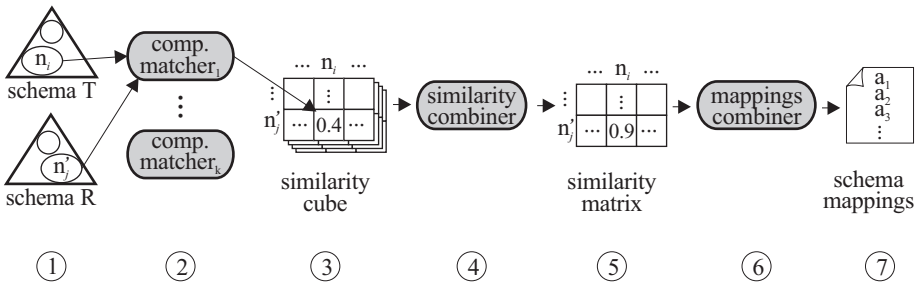


Figure 2.3 *Schema matching system's architecture.*

The main input into the schema matching system are schemas T and R (1) ((1) reads "see part (1) in the figure"). Schemas comprise a number of components, i.e., $T = \{n_1, \ldots, n_p\}$ and $R = \{n'_1, \ldots, n'_q\}$. Schema matching should detect semantic correspondences between these components. Matching starts with the computation of multiple *similarity indexes* for each pair of schema components $(n, n') \in T \times R$. The similarity is computed using various hints with each hint implemented as a single *component matcher* (2). E.g., one matcher might compare element names, another compares data types, and so forth. The result of the first step is one *similarity matrix* per component matcher. Together, these matrices form what the authors of COMA call *similarity cube* (3). The dimensions of the cube are $k \times p \times q$. The figure shows that $matcher_1$ computed 0.4 similarity index for components n_i and n'_j. In the following step, the *similarity combiner* (4) is used to compute a single similarity index for each pair of components. These indexes are computed by combining the similarity indexes computed by different component

matchers. The result is a *similarity matrix* (5). The figure illustrates that the final similarity of the components n_i and n'_j is 0.9. In the final step, *mappings combiner* (6) selects multiple *component mappings* to build a *schema mapping* and compute the similarity index for the schema mapping. Mappings combiner creates several schema mappings and orders them according to the computed similarity index. The final result is therefore an ordered list of schema mappings (7).

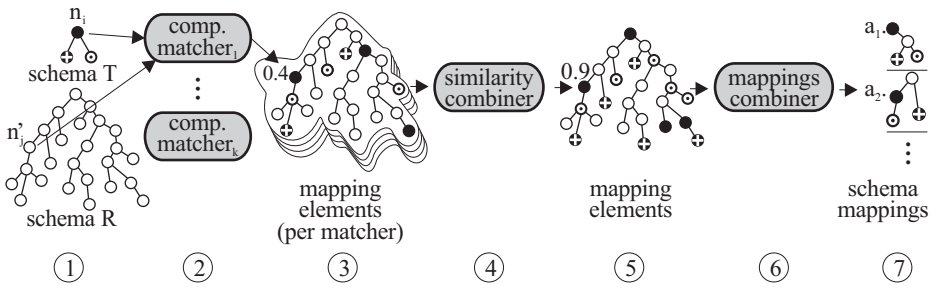


Figure 2.4 *Schema matching in personal schema based querying.*

Fig. 2.4 shows the same architecture in a slightly different way. The results of individual matching phases are visualized using schemas, rather than matrices, providing in that way new insight into the schema matching process. Schemas T and R in the figure are deliberately disproportionate to mimic a schema matching problem which would typically be encountered in personal schema based querying (PSQ, see Sec. 1.3.5). In PSQ, the personal schema T is relatively small, say, with at most ten elements, but the schemas in the repository R can be very large and can comprise thousands of nodes. Note that the repository R is normally a collection of schemas. For brevity, in the figure, only one schema is shown, i.e., schema R. In here we match elements, as opposed to matching components.

As already explained, in the first step component matchers (2) cross compare all schema elements. If $matcher_1$ computes that the similarity of elements n_i and n'_j is larger than 0, we say that node n'_j becomes a mapping element for element n_i. In this way, each $matcher_i$ produces a *set of mapping elements* $\mathcal{M}_n^{matcher_i}$ for each personal schema node n. Different matchers produce different sets of mapping elements. In the figure, the three black nodes (3) are mapping elements for the black personal schema node n_i (1). The task of the similarity combiner (4) is to compute a single similarity index for each node in the repository. This results in a single set of mapping elements for every personal schema node (5). Finally, mappings combiner (6) creates many different schema mappings by selecting one mapping element for each node in T.

We provide below more details on how the five concrete matching systems implement components of the schema matching architecture.

Component matcher

A component matcher is an implementation of a component matching hint. To make a distinction between matchers, i.e., hints, two aspects of hints can be observed: first aspect considers the *component properties* which are used by the hint to compute the component similarity, and second, the *formula* or the *algorithm* used to perform the actual similarity computation. For simplicity, in the sequel, we assume the schema component to be a single XML element.

We first observe the component properties which are used to compute the similarity. These can be classified in several ways [25, 57, 70]:

- *atomic vs. structural* properties
- *schema vs. instance based* properties
- *auxiliary* properties

Atomic properties are local to the components being compared. Examples include *name* of an XML element, *data type* for simple type elements, *comments* attached to the element. The *structural* properties, on the other hand, are extracted from the context within which the component resides: these include the *number of child or sibling elements*, *depth of the element* within the schema, *properties of the path* leading from the root to the element in concern, *path cardinalities*, and *referential and identity constraints*.

Schema based properties are properties that reside in the schema itself. *Instance based* properties on the other hand, reside in the data instances of the schema. For example, in relational databases, the attribute names and the attribute data types are schema base properties, while the values stored in the databases for a particular attribute, represent the instance based properties for that element.

Auxiliary sources can be used to gather additional component properties. For example, lexical databases such as WordNet[1] can be used to discover synonyms of words. Matching system also use previous matching results and user feedback as useful auxiliary sources.

The other aspect of hints is the *formula* or the *algorithm* used to compute the similarity index. In this aspect two main categories exist:

- *rule based approaches vs. machine-learning approaches*

Rule based approaches use computational techniques to compare component properties and to produce the similarity index. For example, the names of XML elements can be compared using the Levenshtein distance algorithm (i.e., the edit distance) to compute the similarity index [22]. In the *machine-learning approaches*, neural networks or Naive Bayes classifiers are used to classify schema component into semantically equivalent classes. Machine learning approaches require that the learners are first trained using a set of manually prepared matching examples for which the desired outcome is known.

[1] see WordNet a lexical database at `wordnet.princeton.edu`

We now discuss component matchers used in the five systems. Similarity Flooding [59] first uses a *StringMatch operator* to compute the initial node name similarity. The StringMatch operator compares common prefixes and suffixes in words. This is an atomic, local, and schema based hint. The output of this computation is used in the next hint, the iterative fixed point computation in a *similarity propagation graph*. This hint exploits the intuition that the similarity of two nodes also depends on the similarity of their neighbors. This is a structural hint as it exploits the structural context of nodes.

Cupid's [57] hints, i.e., component matchers, utilize both atomic and the structural properties. The atomic properties are used by a *linguistic matcher* which exploits name and data type information. The linguistic matching goes through three phases:

- *normalization*: element names are subjected to *tokenization* (e.g., "Shipping-Agent" is split into "Shipping" and "Agent"), *expansion* which identifies abbreviations (e.g., "Qty" is expanded to "Quantity"), and *elimination* which removes articles and prepositions. This phase makes use of a thesaurus as an auxiliary source to resolve abbreviations and acronyms.
- *categorization*: element names are clustered based on the element data type. For example, elements of type *date* are separated from the elements of type *float*.
- *comparison*: names belonging to the same category are compared and the linguistic similarity index $lsim \in [0, 1]$ is computed. This step utilizes the thesaurus to acquire synonyms of the words.

Cupid also uses structural properties. The *TreeMatch operator* computes *ssim* as the similarity of the context within which the elements appear in schemas.

COMA [22], a system for combining schema matching approaches, utilizes a large number of hints. COMA classifies hints as being *simple* or *hybrid*. Simple hints exploit single element property and a single formula for similarity computation. Hybrid matchers, on the other hand, combine simple hints or other hybrid hints to compute similarity index. Examples of simple hints include *affix* which is identical to SimilarityFlooding's StringMatch operator, *n-gram* which finds common substrings in node names, and *EditDistance*, *Soundex*, *Synonym*, *DataType*, and *user feedback*. An example of a hybrid hint is the *NamePath* hint. This hint first creates *hierarchical names* for the compared nodes: hierarchical name is created by concatenating the names of all the nodes encountered between the schema root and the target node, e.g., in Fig. 2.1, the hierarchical name of element <book> is "libraryitembook". Hierarchical names are then compared using other simple hints for comparing names. The NamePath hint combines and exploits both atomic and structural properties. Another hybrid hint is *TypeName* which combines name comparison simple hints and DataType hint.

LSD [23] uses learners to match schema elements. Matching is modeled as a classification process in which the mediated schema elements define classes, and the source schema elements need to be classified. Different types of learners, such as Naive Bayes, or Whirl, can be used to build component matchers. These can be trained, using both schema and instance based properties: schema based properties include element names, proximity of elements; instance based properties include data ranges, representation formats, word and value frequencies, and text length. Additionally, more specific recognizers are developed to handle country names, postal codes, and phone numbers.

iMap [19] aims at detecting complex mappings between schemas. It uses specific component matchers, called searchers. The *Numeric searcher* assesses the similarity of complex numeric fields such as `width` and `length` in one schema, with the `area` element in another schema. Data instances, domain knowledge, previous matching results, and other external sources are used to suggest that `width` $\times$ `length` $\mapsto$ `area`. Other searchers include the *searcher of categorical data*, *mismatch searcher* which detects that element content in one schema matches element names in another, *unit conversion searches* and *date searchers*.

Similarity combiner

The similarity combiner computes a single similarity index for every pair of compared schema components, by combining similarity indexes which were computed by component matchers. This combining is not trivial. Each of the component matchers could have produced correct of incorrect similarity index, i.e., false negatives or false positives. Automated schema matching depends on the expectation that the use of many component matchers will marginalize such errors of the individual matchers. It is up to the similarity combiner to realize this expectation.

Cupid, combines linguistic similarity *lsim* and the structural similarity *ssim* in a weighted sum of the two: $wsum = \alpha \times wsim + (1 - \alpha) \times ssim$, with the coefficient α determining the relative importance of the two hints.

COMA investigates four combining strategies: *Max*, *Min*, *Weighted*, and *Average*. The *Max* strategy assumes that no false positives are produces by component matchers, and that the maximum similarity index indicates the level of the actual element similarity. *Weighted* approach, equal to the one used by Cupid, assignees weights to each of the component matchers based on their expected importance. The tuning of these weights, is however, a difficult problem.

To combine the results of component matchers, LSD uses a *meta-learner* which decides how much each of the matchers should be trusted. Meta-learner is trained using the input from the component matchers on one side, and the manually specified expected similarity indexes. By observing the mismatch between the two, the meta-learner learns how much weight each of the base matchers should be given. LSD implements the meta-learner using the machine learning technique called *stacking*.

Similarity Flooding approach does not have a similarity combiner as it uses only one component matcher which produces one similarity matrix. iMap relies on techniques described above.

Mapping selector

The similarity matrix (⑤ in Fig 2.3) is the result of the cross-comparison of all schema T elements with all schema R elements. An excerpt from the matrix is shown in Fig. 2.5: n_1 and n_2 components are mapped to both n'_1 and n'_2 components. The tasks of the *mapping selector* is select one mapping for each component in schema T and to form, in that way, a full *schema mapping*. This is a combinatorial problem, and the mapping selector creates a number of different schema mappings, which can be ranked based on some adopted criteria.

Based on element mappings represented in Fig. 2.5, different schema mappings can be formed, for example (assuming that no other nodes exist in schemas) $a_1 = \{n_1 \mapsto n'_1, n_2 \mapsto n'_2\}$ and $a_2 = \{n_1 \mapsto n'_2, n_2 \mapsto n'_1\}$. Depending on the selection criteria, some schema mappings rank higher that others.

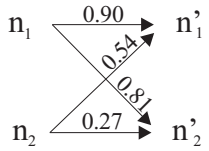


Figure 2.5 *An excerpt form the similarity matrix (adopted from [59]).*

Similarity Flooding considers several strategies for assigning scores to alternative schema mappings. These find their base in the *theory of matching in bipartite graphs*. As an intuition, source components are thought to be men, while target components are women. One of the strategies is the *stable marriage* strategy, where the goal is to have such mappings that no two pairs $x \mapsto x'$ and $y \mapsto y'$ exist where x prefers y' over x' and y' prefers x over y. In the example, a_1 is a stable marriage while a_2 is not. The alternative approach is the maximization of the total partner satisfaction. For a_1 this is $0.90 + 0.27 = 1.17$, for a_2 this is $0.81 + 0.54 = 1.35$. Now, a_2 wins over a_1. Similarity Flooding also utilizes another approach based on *perfectionist egalitarian polygamy*, which allows for polygamous marriages, i.e., $1 : n$ mappings.

COMA proposes three mapping selection strategies: *MaxN*, *MaxDelta*, and *Threshold*, as well as the combination of these techniques. *Max1* (*MaxN* with $N = 1$), for example, select mappings for each element in schema A by taking the schema B element with the largest similarity index, and then by repeating the procedure for the remaining elements. With $N > 1$, each schema mapping offers multiple mapping choices for each element. To compute the similarity index of a schema mapping, COMA utilizes two strategies: the *average* and the *dice* strategy.

The first one averages on similarity indexes, and the second computes the ratio of matched elements over the total number elements involved in matching. In experimentation, the average computation proved to be superior over dice.

Simple selection of the schema component mappings can create schema mappings which violate domain constraints. LSD applies a *constraint handler* in the mapping selection phase, to ensure the correctness of the produced schema mappings. Domain constraints are specified beforehand by a domain expert and are defined in terms of a mediated schema. For example, a constraint might state: if element a is a match for the mediated schema element `HOUSE-ID` then element a must be a *key* element in the target schema. Constraint handlers are used in iMap as well.

The complexity of schema matching

Schema matching systems experience exponential complexity in respect to the size of the schemas being matched. We show complexities of different processing steps in the schema matching architecture (see Fig. 2.3) assuming that schemas have a tree structure.

The k *component matchers* (2) cross-compare $|T|$ schema components with $|R|$ schema components resulting in $O(k \cdot |T| \cdot |R|)$ matcher invocations. The individual component matchers can also be very complex, possibly dependent on the size of the schema.

The similarity combiner (4) visits every value in the $k \times |T| \times |R|$ similarity cube, and computes $O(|T| \cdot |R|)$ similarity values.

Let $\mathcal{M}_n$ be the set of mapping components for each component n in the personal schema T. And let mappings combiner generates schema mappings by selecting one mapping component $n' \in \mathcal{M}_n$ for each $n \in T$. The number of possible schema mapping is thus $O(|\mathcal{M}_n|^{|T|})$. This is also the size of the search space within which mappings combiner looks for best schema mappings.

The complexity also varies depending on how is *a component* in schema matching defined. In most XML schema matching systems, a component is an XML element or a constellation of XML elements. Systems which match one XML element in the source schema to one XML element in target schema are said to perform $1:1$ matching. In practice, however, $1:p$ and $p:q$ mappings[2] are quite common and should be accounted for in schema matching. This however, introduces additional complexity explosion. When looking for $p:q$ element mappings, where p and q are some fixed integer numbers, schema matcher compares groups of p elements in schema T with groups of q elements in R. The number of such groups in T is ${}_{|T|}C_p$ (combinations of p elements in the set of $|T|$ elements) and ${}_{|R|}C_q$ in R. These numbers of element groups determine the dimensions of the similarity matrices. The size of the similarity matrix is thus $O({}_{|T|}C_p \cdot {}_{|R|}C_q)$ for $p:q$ mapping, as opposed to $O(|T| \cdot |R|)$ for $1:1$ element mapping.

[2] we use $p:q$ in place of more common $m:n$ notation to avoid other notation conflicts

Most of the schema matching systems represent schemas as trees. This is a simplification of what is the true structure of schemas – a graph. The penalty of this simplification is the reduced applicability of the tree-based schema matching systems. They cannot match graph structured schemas, such as recursive schemas or schemas with ID/IDREF links (e.g., ID/IDREF in Fig. 1.9b). The benefits, however, come through reduced complexity of tree-based algorithms. In trees, only one path exists between any two nodes, meaning that the context within which a node, i.e., component, appears is unique. In graphs, the number of paths between some nodes may be larger than one, consequently placing some elements in multiple contexts. Schema matching would have to consider all the contexts for each element, increasing in that way the number of computations.

2.2.3 Other schema matching approaches

To the large extent, schema matching research is directed toward improving individual phases in the architecture of a schema matching system (see Fig. 2.3). The broader research area is presented and well structured in few surveys [25, 70, 78]. Below we describe one recent schema matching approach which extends the presented schema matching architecture with the concepts similar to these of data mining.

Corpus based schema matching

The *corpus based*, also called *holistic*, schema matching approach improves the matching effectiveness by exploring the properties of not only two schemas being compared, but rather the whole corpus of related schemas. The intuition behind this approach is that schemas are often designed in a similar way, and consequently share common properties. The expectation is that design patterns can be discovered and used in schema matching. Several techniques are reported.

The simplest technique uses corpus as an auxiliary source of features in the mainstream schema matching. Corpus of schemas is used to augment, i.e., extend the properties of schema elements which are being compared [55]. Augmentation works as follows. When comparing elements n and n' from schemas T and R:

- corpus elements similar to n are discovered using some *component matchers*,
- the properties of discovered corpus elements are used to augment the properties of the element n,
- the augmented n is then compared with element n' using all *component matchers*,
- the rest of the matching process continues normally.

The corpus can also be used to discover schema patterns [55, 40]. One way to discover these patterns is by forming clusters of similar corpus elements. These clusters can then be interpreted as *concepts*. Next, the most common relations

between the concepts are observed. For example, frequent existence of certain *attribute concepts* within a specific *table concept* (in relational databases), or cooccurrence of *table concepts* within the same schemas, or recurring order of concepts within schemas. This knowledge can be used in schema matching.

If the hypothesis is assumed that all the schemas in one corpus are derived from a common hypothetical schema, then schemas in the corpus can be seen as derived instances of the common hypothetical schema. Schema matching, in such a case, can be treated as hidden model discovery [40]. Statistical methods can be used to evaluate how well a common proposed model corresponds with the schemas in the corpus.

Finally, we need to mention a research field which shares and extends the approaches used in schema matching – the semantic matching of ontologies [64].

2.3 Formalizing the XML schema matching problem

To understand and handle the complexity of the schema matching problem and to be able to devise an efficient algorithm to solve the matching problem, a *formal problem specification* must be acquired. To the best of our knowledge, none of the existing schema matching system was built on the basis of a complete formal specification of the schema matching problem. In the following sections, we focus on understanding, modeling, and formalizing the XML schema matching problem. We ask and provide answers for the following three questions:

- What is a semantic schema matching problem?
- How to approximate the semantic schema matching problem so that computers can be used to solve it?
- What is a good formalism for specifying the approximated semantic schema matching problem?

Note that we use the term *semantic*, in semantic schema matching, to emphasize the fact that schema matching looks for pairs of schema components which *mean* the same, as if the matching is done by humans. Also, we place our research in the context of our motivating application: the personal schema based querying (PSQ) described in Sec. 1.3.5. So, though generic, our formalization approach was developed with the PSQ scenario in mind. In PSQ a small personal schema is matched against a large *schema repository*. An example of a personal schema and a fragment of the targeted schema repository are given in Fig. 2.6. These are used in this section as a running example. This section is based on the work published in [82].

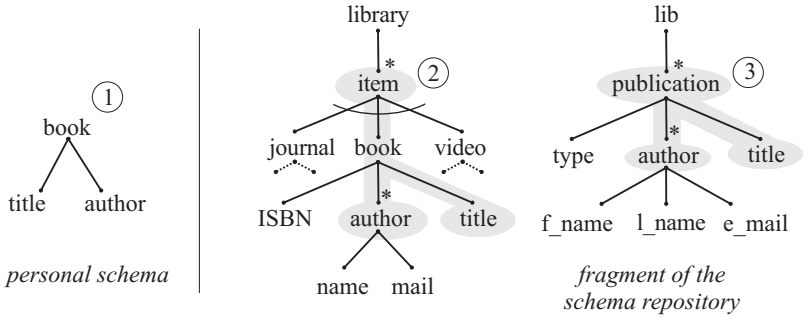


Figure 2.6 *Personal schema and a fragment of schema repository.*

2.3.1 What is a semantic schema matching problem?

In order to understand the semantic schema matching problem we first introduce a model of a *generic matching problem* and then extend this model to create a model of the *semantic schema matching problem.*

Generic matching problem

In a generic matching problem, as illustrated in Fig. 2.7, a *template object* T is matched against a set of *target objects* $R = \{\tau_1, \ldots, \tau_k\}$. If a template object relates to a target object through some *desired relation*, i.e., $T \approx \tau_i$, it is said that they match and $T \mapsto \tau_i$ is called a *mapping.* The *solution* of a matching problem is a set of mappings which satisfy the desired relation. We further model the desired relation through two components: the *predicate function* $C : \{T\} \times R \rightarrow \{True, False\}$, also called the *required relation*, and the *objective function* $\Delta : \{T\} \times R \rightarrow [0, 1]$.

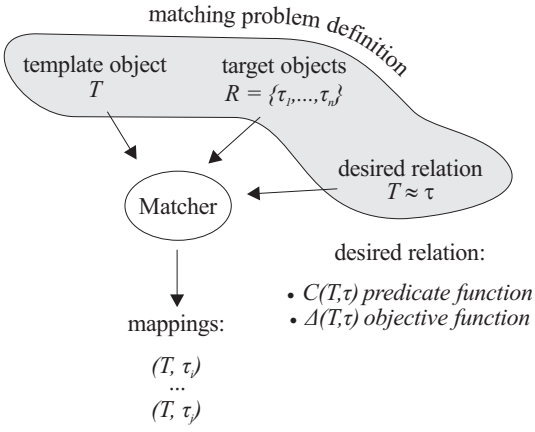


Figure 2.7 *Components of a generic matching problem.*

For example, let the template object T be a constant value, e.g., 45.23, and let the set of target objects be a set of constants $R = \{40.33, \texttt{"text"}, 21/07/2005, 44\}$. The desired relation is the similarity of values. The predicate function C models mandatory requirements. In this example C states that two elements are similar only they have compatible data types, i.e., $C(T, \tau) = \textit{datatype}(T)$ *compatible with* $\textit{datatype}(\tau)$. This predicate filters out mappings $45.23 \mapsto$ "`text`" and $45.23 \mapsto 21/07/2005$. The objective function, on the other hand, defines the non-predicate part of the relation, e.g., the degree of similarity of values: $\Delta(T, \tau) = |\textit{value}(T) - \textit{value}(\tau)|$. When used to rank the mappings, this objective function would give preference to mapping $45.23 \mapsto 44$ over $45.23 \mapsto 40.33$ because $\Delta(45.23, 44) < \Delta(45.23, 40.33)$.

To sum up, a generic matching problem is defined if the following four components are defined:

- template object T,
- set of target objects $R = \{\tau_1, \ldots, \tau_k\}$,
- predicate function $C(T, \tau)$, part of the desired relation,
- objective function $\Delta(T, \tau)$, part of the desired relation.

Semantic matching problem

A semantic matching problem differs from the generic matching problem in that objects are matched based on their semantics. Semantics is commonly defined as the *meaning* of data. Therefore, in semantic matching the template and the target objects are *meanings* rather than data. The *desired semantic relation* is consequently a relation between meanings.

For example, in a semantic matching problem a person is looking for a book similar to Verne's book "20,000 Leagues Under the Sea." The person is matching its mental perception of Verne's book against a set of mental impressions about target books, e.g., books in his personal library. In this problem, the desired semantic relation is *similarity of mental impressions about books.*

The development of a precise generic model of semantic matching, i.e., the model of what is really happening in human mind, is beyond our expertise and interests, so we resort to a pragmatic approach. We shall use the generic schema matching problem to model the semantic matching problem.

Our model of semantic matching therefore, also has four components:

- the semantic template object,
- set of semantic target objects,
- semantic predicate function,
- semantic objective function.

Parts of the desired semantic relation that must necessarily be satisfied are captured within the semantic predicate function. For example, a person is looking for a book; it must be true that the target object is what this person thinks is

a book. The semantic objective function models the human's ability to establish ranking among semantic mappings. E.g., the person will think that book A is more similar to Verne's book than book B if book A has a more similar topic; the person's opinion on topic similarity ranks the books.

The semantics, i.e., the ability to generate meanings about objects and to reason about these meanings, is the privilege of humans. In our understanding, building a computer system that performs true semantic matching would amount to building a complete artificial human, which is in principle impossible. In practice computer systems only *approximate* semantic matching [23]. Given our four component model of semantic matching , we claim that a semantic schema matching problem can be solved by a computer, only if all four components of the problem are adequately approximated. In the next section we elaborate on these approximation in the context of the XML schema matching.

2.3.2 Approximating semantic XML schema matching

This section describes the approximation of all components of the model of semantic matching problem. Each semantic component is approximated with a syntactic counterpart ($\rightsquigarrow$ reads *approximated with*):

$$\begin{aligned} \text{semantic template object} &\rightsquigarrow \text{syntactic template object} \\ \text{semantic target object} &\rightsquigarrow \text{syntactic target object} \\ \text{semantic predicate function} &\rightsquigarrow \text{syntactic predicate function} \\ \text{semantic objective function} &\rightsquigarrow \text{syntactic objective function} \end{aligned}$$

The process of approximating is a design process in which semantics is modeled with syntactic constructions. As such, it is a subjective process burdened with trade-off decisions. In this section we describe approximations which are tailored to meet the needs of XML schema matching in personal schema based querying. Other schema matching systems can decide to approximate the problem components differently. Nevertheless, the common point of all systems is that these semantic components are approximated.

Approximating the semantic template and target objects

The process which approximates the semantic template/target object with a syntactic counterpart is the *schema design process*. Fig. 2.8 shows a schema designer who approximates his *understanding of a concept book* with an XML schema. In this case, the XML schema is the *syntactic template/target object*.

A schema matching system is, in principle, not responsible for how the semantic template/target objects are approximated. The matching system only requires syntactic template/target objects as input. In practice, the schema matching system faces another problem with XML schemas: the problem of heterogeneity of

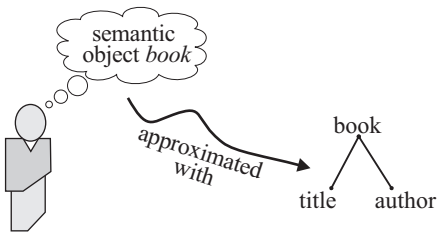


Figure 2.8 *Schema design approximates semantic objects with XML schemas.*

languages and schema features used by schema designers. To tame this syntactic diversity, schema matching systems often use a generic, unified, data model. Different schema representations are then captured within this model. For XML schema matching, we use *directed graphs* enriched with *node and edge properties*; a model similar to ones used in other schema matching systems.

Definition 1 A directed graph G with properties is a 4-tuple $G = (N, E, I, H)$ where

- $N = \{n_1, n_2, ..., n_i\}$ is a nonempty finite set of *nodes*,
- $E = \{e_1, e_2, ..., e_j\}$ is a finite set of *edges*,
- $I : E \rightarrow N \times N$ is an *incidence function* that associates each edge with a pair of nodes to which the edge is *incident*. For $e \in E$, we write $I(e) = (u, v)$ where u is called the *source node* and v the *target node*.
- $H : \{N \cup E\} \times \mathcal{P} \rightarrow \mathcal{V}$ is a property set function where $\mathcal{P}$ is a set of properties, and $\mathcal{V}$ is a set of values including the null value. For $n \in \{N \cup E\}$, $\pi \in \mathcal{P}$, and $v \in \mathcal{V}$ we write $\pi(n) = v$ (e.g., $name(n_1) =$ *"book"*).

A *walk* p in a directed graph is any alternating sequence of nodes and edges, i.e., $p = (n_1, e_1, n_2, e_2, \ldots, e_{k-1}, n_k)$ such that an edge in p is incident to its neighboring nodes. Nodes n_1 and n_k are said to be the *source* and the *target* nodes of the walk p. We treat the terms *path* and *walk* as synonyms.

A graph G' is a *partial subgraph* of graph G (i.e., $G' \sqsubset G$), if G' can be constructed by removing edges and nodes from graph G.

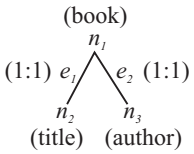


Figure 2.9 *Model of personal schema.*

The book personal schema from Fig. 2.6 is modeled as shown in Fig. 2.9.

- $N = \{n_1, n_2, n_3\}$
- $E = \{e_1, e_2\}$
- $I(e_1) = (n_1, n_2),\ I(e_2) = (n_1, n_3)$
- $name(n_1) =$ *"book"*, $name(n_2) =$ *"title"*, $minOccurs(e_1) = 1,\ \ldots$

We call a graph that models an XML schema a *schema-graph*. In schema-graphs, XML schema components are represented with either a *node*, an *edge*, or a node's or an edge's property. For example, relationships defined in XML schema by means of *id/idref* pairs, or similar mechanisms, are modeled as edges that we call *explicit edges*. More details on how we represent an XML schema using a directed graph can be found in [81]. Schema-graphs are a complete representation of an *XML schema*, i.e., the one can be converted into the other and vice versa without loss of information.

Approximating the semantic predicate function

In semantic matching, the desired semantic relation is a relation between the meanings of a template and a target object. As shown in above, schema-graphs are used to approximate these meanings. Consequently, the desired semantic relation will be approximated as a relation between schema-graphs.

We approximate the semantic predicate function with a predicate function $C(T, \tau) \in \{true, false\}$, where T, τ are the template and the target schema-graphs. C is usually built as a conjugation of several predicates c_i:

$$C(T, \tau) = \bigwedge_{i=1}^{k} c_i(T, \tau)$$

Predicates c_i specify various aspects of the relation between the template and the target schema-graph that must be true in a mapping. What follows is a discussion of predicates that we use in defining the schema matching approach.

Our schema repository comprises many XML schemas, or rather schema-graphs, collected from the Internet. Such a repository can be treated in two different ways: as a set of independent schema-graphs, or as one large schema-graph. This distinction influences the definition of the target object in a matching task. The matching task is either:

1. for a template schema T, *find the most similar target schema-graph* τ_i in the repository $R = \{\tau_1, \ldots, \tau_k\}$. The output of this matching approach is a list of concrete schemas from R, namely the ones most similar to T, or

2. for a template schema T, *find the most similar partial subgraphs* τ_i in R (i.e., $\tau_i \sqsubset R$, see Def. 1). The output of this matching approach is a list of subgraphs of the repository schema-graph R. Such subgraphs can in general be composed of nodes and edges from different concrete schema-

graphs participating in R, assuming that there exist an edge connecting the two graphs; for example, an edge modeling the referential integrity between elements of two schemas.

In our research, we adopt the second matching goal. This allows a personal schema to be matched to a target schema obtained by joining fragments of several distinct schemas, or to be matched to a fragment of only one schema. A predicate function $c_1(T, \tau) := (\tau \sqsubset R)$, where R is the schema repository, can be used to specify this matching goal. Predicate c_1 is very loose and does not consider the properties of the personal schema. For example, in the case of matching the book personal schema in Fig. 2.6, the c_1 predicate would be satisfied for any subgraph of the repository, including the individual nodes. This seldom makes any sense. More predicates must be added. The following set of predicates defines a schema mapping as a $1:1$ mapping.

Definition 2 The target schema-graph $\tau = (N_\tau, E_\tau, I_\tau, H_\tau)$, where $\tau \sqsubset R$ for repository R, forms a $1:1$ mapping with a template schema-graph $T = (N_T, E_T, I_T, H_T)$ if the following set of predicates is satisfied.

1. $\forall n \in N_T,\ \exists_1 n' \in N_\tau$ *such that* $n \mapsto n'$
   E.g., in Fig. 2.6 element <item> (2) is the mapping node for for the node <book> (1), i.e., <book> $\mapsto$ <item>.
2. $\forall e \in E_T,\ I_T(e) = (n_i, n_j),\ n_i \mapsto n'_i,\ n_j \mapsto n'_j$
   $\exists_1 p' \in$ *paths*(τ), $I_\tau(p') = (n'_i, n'_j)$ *such that* $e \mapsto p'$
   where *paths*(τ) is a set of all paths in τ.
   E.g., in Fig. 2.6 the path <item> − <book> − <title> (2) is the match path for the edge <book> − <title> (1). This *edge-to-path* mapping rule is a practical simplification of a more general *path-to-path* rule.
3. schemas T and τ form a schema mapping $T \mapsto \tau$ if they satisfy the conditions 1 and 2.

A new predicate can be defined as $c_2(T, \tau) := \tau \mapsto T$.

The first point in Def. 2 restricts a component mappings to what is commonly known as "1 to 1" element mapping [70]. Most of the existing schema matching systems are designed to discover "1 to 1" mappings [25], and in our research, we investigate clustered schema matching in the context of such systems. A different definition of a mapping is needed to accommodate $1:p$ and $p:q$ element mappings.

The second point in Def. 2 is rarely considered in related schema matching systems. This is because most of the systems model schemas as trees, and this point is needed only when modeling schemas as graphs; in graphs there can exist more than one path between two nodes, and to define a mapping it must be clear which of the paths is used.

A matching system can use additional predicates. For example, a schema matching system which does not handle cyclic data structures should include the predicate $c_3(T, \tau) := (T \textit{ is non-cyclic}) \wedge (\tau \textit{ is non-cyclic})$. A predicate can impose the requirement that the matching elements belong to the same cluster of elements, as it is the case in Cupid [57](see component matcher in Sec. 2.2.2). Also, LSD [23] uses a number of *hard constraints* based on domain knowledge, to filter out invalid mappings. An example of such constraint was presented in Sec. 2.2.2 when describing the implementations of the mapping selector.

Approximating the semantic objective function
The semantic objective function is approximated with an objective function $\Delta(T, \tau) \in [0, 1]$, where T is a template schema-graph, τ is a target schema-graph taken from the repository R. $\Delta(T, \tau)$ is treated as *undefined* if the predicate function $C(T, \tau) = \textit{false}$. The objective function is built using heuristics, i.e., hints, in the ways that were described in Sec. 2.2.2. In particular, similarity indexes are first computed by *component matchers*, these indexes are then combined by *similarity combiner*, to be again combined with *mappings combiner*. The objective function is a complex composition of formulas or computational algorithms, each of which exploits different heuristic for estimating the similarity between two schemas components or schemas as a whole. The objective function is a precise specification of this complex similarity computation.

In this section, we have shown that existing schema matching systems use approximations to declare the schema matching problem in a way that can be used in automated schema matching systems. We have identified four declarative components which need to be approximated. In our research, we came to understand that these components are almost identical to components of a known class of problems called *constraint optimization problems (COP)* [5, 58]. In next section, we show one way to specify a schema matching problem in the COP framework. We also discuss benefits that schema matching can draw from being represented as a COP.

2.3.3 Formal specification of the problem

In this section, we first describe the formalism for representing constraint optimization problems. We then show one way to specify semantic XML schema matching problem using this formalism.

Constraint optimization problems
Constraint programming (i.e., CP) is a generic framework for problem description and solving [5, 58]. CP separates the declarative and operational aspects of problem solving. CP defines different classes of problems, of which we solely focus on the declarative aspects of *constraint optimization problems*.

Definition 3 A *constraint optimization problem* (i.e., COP) P is a 4-tuple $P = (X, D, C, \Delta)$ where

- $X = (x_1, \ldots, x_n)$ is a list of *variables*,
- $D = (D_1, \ldots, D_n)$ is a list of finite *domains*, such that variable x_i takes values from domain D_i. D is called the *search space* for problem P.
- $C = \{c_1, \ldots, c_k\}$ is a set of *constraints*, where $c_i : D \rightarrow \{true, false\}$ are predicates over one or more variables in X, written as $c_i(X)$.
- $\Delta : D \rightarrow \mathbb{R}$ is a the *objective function* assigning a numerical quality value to a solution (*solution* is defined below).

COP is defined in terms of *variables* X, taking values from search space D. A complete variable assignment is called *valuation*, written as Θ. Θ is a vector in D, thus assigning a value in D_i to each variable x_i, $i = 1, n$. A valuation Θ for which *constraints* $C(X)$ hold, i.e., $C(\Theta) = true$, is called a *solution*. The quality of a solution is determined by the value of the objective function, i.e., $\Delta(\Theta)$.

A partial valuation $\tilde{\Theta}$ is an incomplete valuation of X, where only the variables in a partial variable list $\tilde{X}$ (i.e., $\tilde{X} \sqsubset X$) have been assigned a value.

Semantic XML schema matching as COP

This section presents one possible way for specifying an XML schema matching problem as COP. The approach is based on rules which are part of the Def. 2. To support the explanation in this section, we will use the book personal schema, and the schema repository shown in Fig. 2.6, as well as the personal schema-graph given in Fig. 2.9.

Definition 4 A semantic XML schema matching problem with a template schema-graph T, a repository $R = (N_R, E_R, I_R, H_R)$ of target schema-graphs τ_i, where $\tau_i \sqsubset R$, a predicate function $C(T, \tau)$, and an objective function $\Delta(T, \tau)$ is formalized as a constraint optimization problem $P = (X, D, C, \Delta)$. The following four rules construct P in a stepwise manner.

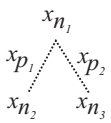


Figure 2.10 *Target object variables.*

Rule-1
For a template schema-graph $T = (N_T, E_T, I_T, H_T)$, the *repository of target schema-graphs* R is formalized in a COP problem P, as follows:

1. for each node $n_i \in N_T$, a node variable x_{n_i} and a domain N_R are added to P (see Def. 2 rule 1),
2. for each edge $e_i \in E_T$, a path variable x_{p_i} and a domain $\mathcal{L}_R$ are added to P, where $\mathcal{L}_R$ is the set of all paths in repository R (see Def. 2 rule 2),
3. for each edge $e_k \in E_T$, where $I_T(e_k) = (n_i, n_j)$, a constraint $ic_k(X) := n_i \mapsto \mathit{source}(x_{p_i}) \wedge n_j \mapsto \mathit{target}(x_{p_i})$ is added to P. We denote the conjunction of all such constraints as $IC(X)$ – the incidence constraint. This constraint ensures that target paths are connected in the same way as template edges are connected.

In the book example, P is, so far, defined as

$$\begin{aligned} X &= (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2}) \\ D &= (N_R, N_R, N_R, \mathcal{L}_R, \mathcal{L}_R) \\ C &= \{IC(X)\} \\ \Delta(X) &\quad \textit{not yet defined} \end{aligned}$$

Fig. 2.10 illustrates *Rule-1*; node and edge variables are assigned based on the shape of the book template schema-graph.

Rule-2
The predicate function $C(T, \tau)$ is formalized in P by adding a constraint $C(X)$ – a constraint function constructed from a predicate $C(T, \tau)$ in a straightforward manner; node and path variables found in X replace τ, nodes and edges of T (e.g., n_1, n_2, n_3, e_1, and e_2 in the book example) appear as constants in $C(X)$.

For example, `c4(X) := (datatype(`n_2`) = datatype(`x_{n_2}`))` is a constraint responsible for ensuring that the mapped element `‘‘title’’` and its mapping elements x_{n_2} have the same data type.

Rule-3
The objective function $\Delta(T, \tau)$ is formalized in P using the objective function $\Delta(X)$ – a function constructed from $\Delta(T, \tau)$ in a straightforward manner (see Rule-2).

Rule-4
Template schema-graph T is constant in P, i.e., for two different schema-graphs T_1 and T_2, two different COP problems must be defined. As indicated in *Rule-2*, and *Rule-3*, T is represented through constants used in $C(X)$ and $\Delta(X)$.

For the schema matching problem, the specification of P is now complete.

$$\begin{aligned} X &= (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2}) \\ D &= (N_R, N_R, N_R, \mathcal{L}_R, \mathcal{L}_R) \\ C &= \{IC(X), C(X)\} \\ \Delta(X) &\quad \textit{defined as explained in} \\ &\quad \textit{Sec. 2.3.2 "Approximating the semantic objective function"} \end{aligned}$$

2.3.4 The benefits of using COP framework

The benefit of formalizing a schema matching problem as a COP is that a schema matching problem can now be regarded as a combinatorial optimization problem with constraints. COP problems have been largely investigated and many techniques for efficient solving have been proposed [60, 58]. Branch and bound, clustering methods, simulated annealing, tabu search, to name a few, can be investigated and adopted to schema matching problems represented as COPs. Important issues that influence the efficiency, e.g., variable ordering, value ordering, constraint simplification are also discussed in the COP framework. It is in the domain of COP that we have found suggestions for what algorithms to actually use in our research to efficiently solve the schema matching problem.

2.3.5 Other formalization efforts

Semantic heterogeneity problem in distributed databases attracts significant attention of the scientific community [43]. Various tools, including schema matching tools, have been proposed to cope with this problem . Nevertheless, formal bases of the semantic problems remain underdeveloped and need further attention [25].

Avigdor Gal et al. have utilized the *fuzzy framework* to model a schema matching problem, and have used this framework to define the *monotonicity* as a desired property of a schema matching problem [35]. Fuzzy relations are used to model the uncertainty present in schema matching: a fuzzy set is formed out of component mappings $n \mapsto n'$, and the computed similarity of components $\mu^{n \mapsto n'}$ determines the membership degree of the $n \mapsto n'$ mapping in a fuzzy relation. The similarity, i.e., confidence, of the whole schema mapping a is computed by combining the element similarities with an aggregate fuzzy functions, $\mu^a = h(\mu^{n \mapsto n'} | n \mapsto n' \in a)$.

The authors use this framework to define the *monotonicity* property as desired property for a schema matching problem. Monotonicity ensures that the mapping confidence produced by a schema matching system coincides with the discrepancy between the mapping a_i generated by the system and the ideal mapping $\bar{a}$ built by a human expert. The discrepancy between the ideal mapping $\bar{a}$ and an arbitrary mapping a_i is formalized as the number of mismatching attributes schema components i_{a_i}. A schema matching approach is said to be monotonic if for any mappings a_i, a_j such that $i_{a_i} < j_{a_j}$ the switch from a_i to a_j increases the schema mapping's

similarity. It is shown that strict monotonicity is hard to achieve by a schema matching system, hence a looser requirement is defined: statistical monotonicity.

Madhavan et al. describe a formal framework for developing languages which can be used to represent mappings [56]. The framework has the following properties: it allows the creation of languages for a direct mapping between different schema languages, such as mappings between XML schemas and relational tables. A specific instance of this framework is therefore a mapping representation language targeting one concrete source and one concrete target language. The framework allows for a third schema, the so called helper model, to be used in the situations when the concepts in two schemas are similar, but cannot be directly mapped. For example, schema A contains students from *University A*, and schema B contains students from *University B*, these are mapped using a helper model which contains the more generic concept *University Student*. Within the framework authors define three properties that a mapping should have:

- query answerability: it should be possible to rewrite the query over one model into the query over the other model,
- mapping inference: it must be possible to show that two mappings are equal or whether a mapping is minimal,
- mapping composition: the ability to form transitive mappings.

The authors show that finding mappings with these properties is an NP-complete problem.

For our formalization approach, we found inspiration in the work of Bergholz [8]. Bergholz formalized the problem of querying semi-structured data as a constraint satisfaction problem. In his work he considers the querying of XML data with languages such as XPath [93]. The framework does not support ranking of the results.

References to a few other works on formalization can be found in a survey by AnHai Doan [25].

2.4 Conclusion

This chapter describes the problem of schema matching and delivers an approach for formal specification of the semantic XML schema matching problem. The formalization starts with a model of semantic matching which is then mapped onto a syntactic domain by means of approximations. These approximations enable automatic solving of the schema matching problem. Further, the constraint optimization framework is identified as a suitable framework for capturing all the declarative syntactic components of the problem. With this formalism, the goal of this chapter is achieved: a clear and unambiguous understanding of the schema matching problem is reached.

Chapter 3

Using clustering to increase the efficiency of schema matching

3.1 Introduction

Personal schema based querying (PSQ, see Sec. 1.3.5) uses schema matching to find mappings between the personal schema and the schemas of the Internet. A schema matcher in PSQ must reach interactive speeds to satisfy an impatient user; efficiency is an imperative. Even though efficiency has been recognized as an important issue in the development of schema matchers [29, 9], so far, it has been treated as a secondary issue in schema matching research. Efficiency of schema matching is largely an open challenge.

In this chapter we contribute to the research of the schema matching efficiency. We propose a *clustered schema matching* technique – a technique for improving the efficiency of schema matching by means of clustering. Clustering divides schema elements into clusters, reducing in that way the workload for a part of the schema matching process. Clustered schema matching can be implemented by adding a clustering step to existing schema matching systems. Furthermore, most of the other optimization techniques are orthogonal to clustering, and can be used in combination. This makes the technique broadly applicable.

This chapter has the following structure. Sec. 3.2 introduces general ways for improving efficiency of schema matching systems. The section also provides examples of specific techniques which are, or can be used in schema matching systems. Sec. 3.3 introduces the clustered schema matching technique. Sec. 3.4 discusses the selection of a clustering algorithm for the application in clustered schema matching. Related research is surveyed at the end of the chapter, in Sec. 3.5

3.2 Improving the efficiency of schema matching

We define the *efficiency* of a schema matching system as the time the system needs to solve a schema matching problem (see Sec. 2.3 for a formal representation of schema matching problem). Improving efficiency of schema matching, therefore, means shortening the time.

In principle, efficiency of any computer system can be improved in (at least) two ways. During the runtime, computer system executes large number of computations, and efficiency can be improved by means of:

- *Pre-computation*, in which case, part of the computations is performed beforehand, i.e., off-line. The results are stored and reused during runtime.
- *Pruning of computations*, in which case, at runtime, different techniques are used to reduce, i.e., optimize, the number of computations that are to be executed.

Improved efficiency often comes with a trade-off in the effectiveness of the system. That is, the improved efficiency can have negative impact on the results produced by the system. In the case of schema matching systems, we identify two properties which characterize this impact. We say that efficiency improvements can be:

- *mapping-preserving* or *non mapping-preserving*, and
- *similarity-preserving* or *non similarity-preserving*.

We define these terms as follows. Let S_1 be a schema matching system. For a schema matching problem P, system S_1 produces a set of schema mappings A_1. Let S_2 be a schema matching system built by improving the efficiency of system S_1. For problem P system S_2 produces a set of schema mappings A_2.

Definition 5 Schema matching system S_2 is a *mapping-preserving* improvement of system S_1 if for every schema matching problem P, both S_1 and S_2 produce identical sets of schema mappings, i.e., $A_1 = A_2$.

Non mapping-preserving efficiency improvement comes with a cost of not finding some schema mappings, which are otherwise discovered by the original system. In many applications, including personal schema based querying, this is an acceptable trade-off, as long as the highly ranking mappings are discovered with the improved system.

Definition 6 Schema matching system S_2 is a *similarity-preserving* improvement of system S_1 if for every schema matching problem P, and for every schema mapping $T \mapsto \tau \in A_1 \cap A_2$ both systems S_1 and S_2 compute the same value of the objective function, i.e., $\Delta_{S_1}(T, \tau) = \Delta_{S_2}(T, \tau)$.

In *non similarity-preserving* systems, in the best case schema mappings change the similarity indexes but preserve the relative order. In such case, the improvement is acceptable, because the order of the proposed schema mappings is of most

importance to the user. However, in general case, the non similarity-preserving improvements should be used with care because they can change the ordering of discovered schema mappings, in which case they actually change the effectiveness of the system in an unpredictable way.

In an ideal case, the improved system S_2 is both *similarity-preserving* and *mapping-preserving* improvement of S_1. This, however, is seldom the case. We now describe several concrete efficiency improving techniques.

3.2.1 Efficiency improvements based on pre-computation

Node labeling techniques

Computation of structural properties in trees and graphs is very expensive. For example, finding the shortest path between two nodes, or checking if two nodes are in a ancestor/descendant relationship requires navigation and multiple access to data structures. A technique known as *node labeling* can be used to significantly speed up the computation of such properties [1, 36, 46, 47]. Node labeling assigns labels to nodes of a tree/graph, such that it is possible to compute the relation of two nodes using only labels, thus without the need to navigate the structure. In node labeling, three performance aspects need to be considered:

- time needed to compute the labels,
- space required to store the labels,
- time needed to compute the properties using labels.

Of these aspects, the third directly influences the efficiency of schema mapping, as it is frequently used at runtime. Some node labeling algorithms, perform this property computation in constant time [1].

Labeling techniques can generally be divided into *precise* and *approximate*. Precise labeling techniques compute property value, e.g., path length between two nodes, precisely just as when computed with other slower techniques. In the approximate labeling techniques, the value of a property is only estimated. For example, label based computation only guarantees that the path length between the nodes lies within certain boundaries [36].

The use of precise labeling techniques in schema matching is both *similarity-preserving* and *mapping-preserving* and can be used in the computation of the objective function. Approximate labeling techniques, on the other hand should not be used in the objective function. It can, however, be used to support other optimization techniques.

Indexing

In applications which handle large amounts of data, it is imperative to quickly access stored data. To shorten access times, various indexing and compression techniques can be used [31, 92, 18, 62]. Elaboration of these techniques is beyond the scope of this thesis.

3.2.2 Efficiency improvements based on pruning of computations

Complex problems are solved by finding good solutions within large search spaces of possible solutions. For example, a naive schema matching system S_1 could solve the schema problem P in an exhaustive generate-and-test way. System would enumerate all possible schema mappings, and compute the value of the objective function for all. Of these, most schema mappings $T \mapsto \tau$ will rank very low, and will not be of any interest due to small value of the objective function $\Delta(T, \tau)$. Consequently, large efficiency improvements can be acquired if system S_1 generates and tests only schema mappings which have the potential to rank high, and to discard the other mappings as soon as possible. Below, we present few pruning approaches in concrete applications.

String comparison with q-grams

String comparison is an important operation in schema matching. Levenshtein distance, commonly known as edit distance, is often used to compute string similarity. Luis Gravano et al., show that it is possible to use q-grams, i.e., short substrings of length q, to quickly compute if two string have the edit distance within certain threshold k [38]. Authors show that comparison of q-grams can be performed very fast using standard database operations, if strings are stored in the database as q-grams. The technique for computation of edit distance on a large set of string pairs has two steps. First step quickly compares q-grams of the string using native database operations. This step filters out all string pairs with edit distance larger than k. In the second step, the precise edit distance is computed for all strings pairs which were not filtered out. The computation of the edit distance is costly, but is performed on set of string pairs which is, due to filtering, much smaller. Experiments showed 20 times faster execution with the use of q-grams compared to direct computation of edit distance on all string pairs. This technique does not change the final result of string comparison, and if used in schema matching, it would be both similarity-preserving and mapping-preserving. Bernstein et al., report a similar technique for string comparison which stores and retrieves q-grams from a hash table [9].

Heuristic algorithms

As shown in chapter 2 schema matching is a constraint optimization problem. Various heuristic algorithms can be used to solve optimization problems. Some of the well known algorithms are *greedy algorithm*, *dynamic programming*, *branch and bound*, and A^* [60]. These algorithms improve efficiency by pruning parts of the search space. They use different heuristics to predict or compute that no good solutions exist in these parts of the search space.

In schema matching, the search space is a set of all possible schema mappings. Schema mappings, can be built by incrementally adding element mappings until a schema mapping is formed. Using heuristics, a search algorithm can determine whether the partial schema mapping has the potential to deliver a good mapping when completed. If the answer is positive, partial mapping is built further by adding another element mapping, otherwise the partial mapping is discarded, i.e., search space is pruned.

Some schema matching systems use heuristic algorithms. LSD uses A* algorithm to find the optimal combination of element mappings ([23], pg. 40). *iMap* uses the beam search algorithm to limit the number of element mappings which are, at the same time, considered as potential members of the final schema mapping [19]. Authors claim that the use of effective pruning techniques are essential in managing the complexity of the schema matching problem.

Depending on the algorithm and the precision of the heuristics, schema matching system improvements based on these techniques can be both (non) mapping-preserving and (non) similarity-preserving. The beam search algorithm, for example, is non mapping-preserving, as it can "miss" some schema mappings.

Schema partitioning

Schema partitioning can be used to reduce the workload of schema matchers. Rahm et. al, propose and demonstrate that XML schema fragmentation can lead to efficiency improvements [4, 29]. Schema fragments can be formed off-line, by exploiting syntactic substructures such as complex types or groups. Schema matching is performed in two steps. First, fragments as a whole are compare to other fragments, computing in that way the *fragment similarity*. In the second step, the detailed and more expensive matching is performed only between the fragments with a large fragment similarity. In some aspects, this approach is very similar to our *clustered schema matching* approach. The main difference lies in the way in which fragments are formed.

3.3 Clustered schema matching

3.3.1 Introducing clustering into the schema matching architecture

In chapter 2 we have introduced the architecture of a typical schema matching system (see Fig. 2.4). We now show how clustering can be added to this architecture with the benefit of improved efficiency. In explanation, we reuse the schema matching problem illustrated in Fig. 2.4. In this problem, the personal schema T has three nodes $N_T = \{n_1, n_2, n_3\}$ (see Fig. 3.1).

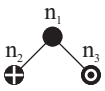


Figure 3.1 *Personal schema T.*

Fig. 3.2 shows the last phase of matching, and the last component in the original schema matching architecture, i.e., the mappings combiner (6). Mappings combiner combines mapping elements (5) in order to generate schema mappings $a_1, a_2, \ldots$ (7). Mappings combiner computes the value of the objective function Δ for each schema mapping, and ranks schema mappings accordingly.

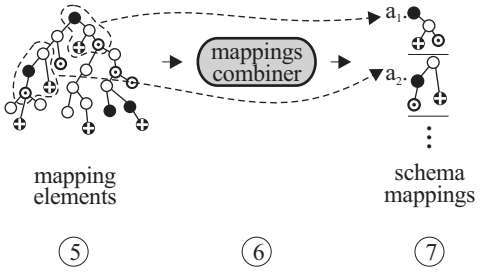


Figure 3.2 *Mappings combiner creates schema mappings.*

Fig. 3.2 (5) illustrates sets of mapping elements for each personal schema node using black, cross, and dot nodes. Sets of mapping elements have following cardinalities: $|\mathcal{M}_{n_1}| = 4$ (black nodes), $|\mathcal{M}_{n_2}| = 5$ (cross nodes), and $|\mathcal{M}_{n_3}| = 5$ (dot nodes). These sets define the set of all mapping elements $\mathcal{M} = \mathcal{M}_{n_1} \cup \mathcal{M}_{n_2} \cup \mathcal{M}_{n_3}$, where $|\mathcal{M}| = 14$. Mappings combiner can create $|\mathcal{M}_{n_1}| \cdot |\mathcal{M}_{n_2}| \cdot |\mathcal{M}_{n_3}| = 4 \cdot 5 \cdot 5 = 100$ different schema mappings.

In a real system, most of these 100 schema mappings would be incorrect and of no value to the user. On the other side, we have observed that, if a good schema mapping $a = T \mapsto \tau$ exists, the mapping schema τ will not be spread over the repository, but will be located in rather small region of the repository. Following this observation, we came to propose the clustered schema matching technique.

In a nutshell, clustered schema matching identifies regions in the repository schema R which are likely to comprise good mapping schemas τ for a certain personal schema T. The mappings combiner then looks for mappings only within these regions, instead of searching through the repository as whole. This reduces the workload and improves the efficiency of schema matching.

In search for an algorithm which can quickly generate such regions we have resorted to clustering. Hence, we call such regions *clusters* and we call the technique *clustered schema matching*. The choice and the tuning of the clustering algorithm

is a very important issue in clustered schema matching. However, the discussion of these details is deferred to coming sections and chapters. For now, we shall treat the clustering algorithm as a black box.

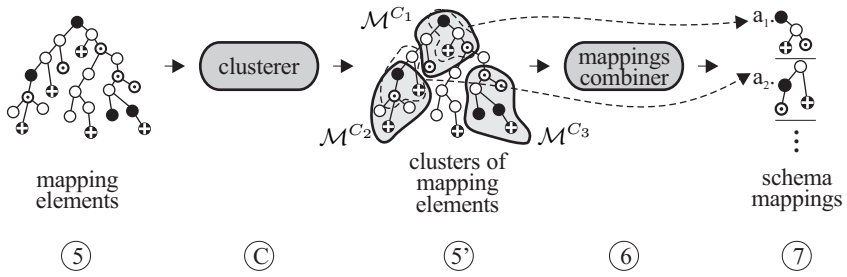


Figure 3.3 *Introducing clustered schema matching.*

The clustered schema matching technique is built by adding clustering to an existing schema matching system. Fig. 3.3 shows the place of clustering in the "standard" schema matching architecture. The clustering step (C) is introduced after the generation of mapping elements. Clusterer divides all mapping elements from $\mathcal{M}$ into clusters, e.g., into $\mathcal{M}^{C_1}$, $\mathcal{M}^{C_2}$, and $\mathcal{M}^{C_3}$ shown in the figure (5'). Each cluster C_i captures mapping elements of various personal schema nodes n_j. We write $\mathcal{M}^{C_i}_{n_j} = \mathcal{M}^{C_i} \cap \mathcal{M}_{n_j}$. When counting the patterned nodes in the figure which belong to the cluster C_1, we find that $|\mathcal{M}^{C_1}_{n_1}| = 1$ (black nodes), $|\mathcal{M}^{C_1}_{n_2}| = 1$ (cross nodes), $|\mathcal{M}^{C_1}_{n_3}| = 2$ (dot nodes). The mappings combiner considers schema mappings within each cluster separately. In the cluster C_1 the number of possible schema mappings is computed as $|\mathcal{M}^{C_1}_{n_1}| \cdot |\mathcal{M}^{C_1}_{n_2}| \cdot |\mathcal{M}^{C_1}_{n_3}| = 1 \cdot 1 \cdot 2 = 2$. In the clusters C_2 and C_3 the number of possible mapping is 2 and 4. This means, that instead of testing 100 possible schema mappings in the non-clustered schema matching, the mappings combiner has to test only $2 + 2 + 4 = 8$ mappings when clustered schema matching is used.

In this example, clusters were formed in such a way that both top ranked mappings a_1 and a_2 are fully contained within individual clusters, and are thus discovered by the mappings combiner.

Clusters partition the mapping elements into clusters, consequently reducing the size of the search space for the mappings combiner and improving the efficiency. In the non-clustered case the mappings combiner has to traverse the search space of the size $O(|\mathcal{M}_n|^{|N_T|})$, where $|\mathcal{M}_n|$ is the number of mapping elements for a personal schema node n, and $|N_T|$ is the number of nodes in the personal schema T (mappings combiner's complexity was first introduced in Sec. 2.2.2). It is reasonable to assume that the size of the repository R determines size of sets of mapping

elements $\mathcal{M}_{n_i}$, i.e., $O(|N_R|) = O(|\mathcal{M}_n|)$. Based on this we write

$$O(|\mathcal{M}_n|^{|N_T|}) = O(|N_R|^{|N_T|}),$$

i.e., in the non-clustered case, the search space of the mappings combiner shows polynomial dependency in respect to the size of the repository R. Clustering reduces this complexity as follows.

In the clustered case, the nodes in $\mathcal{M}_n$ are partitioned into k clusters, with each cluster ideally containing $\frac{|\mathcal{M}_n|}{k} = \mathtt{M}$ elements from $\mathcal{M}_n$. Mapping combiner considers each cluster independently. Consequently, the search space for the mappings combiner is now computed as

$$O(k \cdot \left(\frac{|\mathcal{M}_n|}{k}\right)^{|N_T|}) = O(k^{-(|N_T|-1)} \cdot |\mathcal{M}_n|^{|N_T|}) = O(k^{-(|N_T|-1)} \cdot |N_R|^{|N_T|}).$$

If the number of clusters k is kept constant, the complexity of the search space for the mappings combiner remains polynomial in respect to the size of the repository R and is merely reduced by a constant factor $k^{(|N_T|-1)}$. However, the clustering algorithm does not keep the number of clusters k constant, but instead, it keeps the ratio $\mathtt{M} = \frac{|\mathcal{M}_n|}{k}$ constant by creating more clusters in larger repository schemas. In such a case, it is true that $O(k) = O(|\mathcal{M}_n|) = O(|N_R|)$, and the search space size becomes

$$O(k \cdot \left(\frac{|\mathcal{M}_n|}{k}\right)^{|N_T|}) = O(|\mathcal{M}_n| \cdot \mathtt{M}^{|N_T|}) = O(|N_R| \cdot \mathtt{M}^{|N_T|})$$

Because $\mathtt{M}$ is constant, the search space size is no longer polynomial, but a linear, in respect to size the repository schema R. The complexity and scalability of the whole technique is further discussed in Sec. 5.9.

The clustered schema matching, as introduced above, improves the efficiency of schema matching by reducing only the workload, i.e., the search space, for the mappings combiner. The mappings combiner, is however, only one part of the schema matching architecture. Other components, namely component matchers, consume significant part of time. Therefore, the question of how much the efficiency can be improved with clustering can only be answered by knowing exactly what hints are computed before clustering, that is, by component matchers, and what hints are computed after cluster, that is, by mappings combiner.

In the existing schema matching systems (see in Sec. 2.2.2) the component matchers do most of the work. This means that clustering would bring limited improvement in the overall schema matching architecture. This, however is caused by the fact that the architecture of the current schema matching systems is a rather naive one, and pays no attention to efficiency. In well tuned schema matching system, it can be expected that several computational phases would exist, at least

a clear separation of the computation of non-structural and structural hints. It is to expect, that in such a system, clustering phase could come earlier, and that structural hints would be computed after clustering, that is, on a reduced set of potential schema mappings. This would bring more efficiency improvement. We can conclude that in the worst case scenario, clustering improves only the efficiency of mappings combiner, but at the same time, it is reasonable to expect that much of the computation performed in the component matchers, can actually be moved to the mappings combiner.

Another promise of clustered schema matching

The main task of clustered schema matching is to improve efficiency of schema matching: clusters reduce the search space, which brings efficiency improvement. There is, however, another aspect of clusters; clusters as semantic objects. Namely, clusters are expected to comprise good schema mappings, i.e., schema mappings which rank high.But not all clusters are equal in that respect. Some clusters contain mappings which rank higher than the mappings in other clusters. In schema matching, clusters which deliver top ranked mappings are more important than the clusters which comprise mappings which rank low. Of course, before the actual schema matching is done, it is not known which clusters comprise the top mappings. The last statement, however, does not have to be true. If it would be possible to compute the likelihood that a cluster contains good schema mappings, without actually generating the mappings, this *semantic quality of a cluster* could be used to decide which clusters to send first to the mappings combiner. This would shorten the time needed to generate first good mappings. This hypothesis, however, is not explored in this thesis.

Problems with clustered schema matching

Clustered schema matching reduces the workload for the mappings combiner. This reduction, however, comes with an effectiveness cost. In an ideal case, clusters comprise all the good mappings τ, i.e., mapping for which for which $\Delta(T, \tau)$ is large. In practice, clusters are not ideal, and they cut-out some good mappings by shredding mappings over several clusters. Clustered schema matching is therefore a *non mapping-preserving* improvement. Clustered schema matching offers a trade-off: the more clusters the more efficient schema matching, but the higher the chances of loosing some valuable schema mappings. Such a trade-off is acceptable in many applications. Nevertheless, clustering must try to preserve as much good mappings as possible, where by a good mapping we presume a mapping which are ranked high by the objective function. It is desirable to loose only the mappings which rank low.

Due to the loss of mappings, clustered schema matching is a *non mapping-preserving* efficiency improvement. It is, however, *similarity-preserving* since in

a general case, clustering does not change the value of the objective function computed for a mapping. I.e., a schema mapping $T \mapsto \tau_1$ keeps the value of the objective function $\Delta(T, \tau_1)$, regardless of the usage of clustering.

When strictly following Def. 2, a cluster can produce a schema mapping, only if it has all the necessary mapping elements: at least one mapping element for each personal schema element. Such clusters are called *useful clusters*. The chances are that some clusters will not have all the needed elements. These clusters do not contain any schema mappings. To overcome this limitation, the definition of a schema mapping should be extended with a notion of *partial schema mapping*. This would enable the discovery of partial mappings in *non-useful* clusters. Such partial mappings might, nevertheless, be valuable to the user. This is future research.

3.3.2 Clustering criteria

Having described the main idea of clustered schema matching, next question is what criteria to use to make clusters. Clustering is expected to form clusters in such a way that these comprise highly ranked schema mappings. Knowing that the objective function determines the rank of the schema mapping, it is only logical to base the clustering criteria on the objective function. In other words, the objective function determines the way in which clusters should be formed. Having in mind that the objective function is based on complex heuristics, we can expect that the clustering criteria should use the same or similar heuristics.

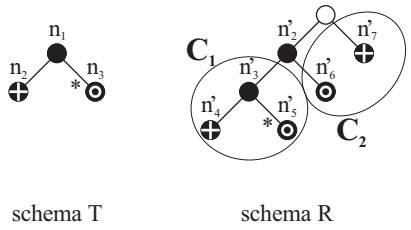


Figure 3.4 *Illustrating clustering criteria.*

To illustrate this fact, we use a simple clustering dilemma. Fig. 3.4 shows a personal schema T and a repository schema R. The component matchers have already finished their work, and the mapping elements are determined as indicated in the figure: $\mathcal{M}_{n_1} = \{n'_2, n'_3\}$, $\mathcal{M}_{n_2} = \{n'_4, n'_7\}$, $\mathcal{M}_{n_3} = \{n'_5, n'_6\}$. The figure shows the state at which all but one mapping element are already clustered. Clusters are $C_1 = \{n'_3, n'_4, n'_5\}$ and $C_2 = \{n'_6, n'_7\}$, and the element n'_2 still needs to join one of the clusters. The clustering criteria, determined by the objective function, decides which cluster it will join.

We now observe how heuristics are used in the objective function, and how these influence clustering criteria. Fig. 3.4 is used throughout the discussion. Note that the effectiveness of various heuristics is not discussed. We will validate the effectiveness of one concrete clustered schema matching technique in chapter 5.

Similarity index of mapping elements

All mapping elements have an assigned similarity index computed by the component matchers and the similarity combiner. Let $\Delta_{cm}(n_1, n_2') = 0.8$. The value of the similarity index is an important factor in determining the similarity index of the whole schema mapping. Clustering can therefore try to maximize the average similarity index of the elements in the clusters. Let elements in cluster C_1 have element similarities $0.5, 0.6, 0.7$, and in C_2 similarities $0.2, 0.9$. Clustering criteria decide to join node n_2' the cluster for which:

- *resulting average similarity is largest.* Average similarity for C_1 is 0.6 and grows to 0.65 with the addition of n_2'. Average similarity for C_2 is 0.55 and grows to 0.63 with the addition of n_2'. Node n_2' joins C_1 due to $0.65 > 0.63$.
- *improvement of the average similarity is largest.* The improvement of average similarity for cluster C_1 is $0.65 - 0.6 = 0.05$, for C_2 it is $0.63 - 0.55 = 0.08$. Node n_2' joins cluster C_2 due to $0.08 > 0.05$.

Path length

The distance between the nodes in the mapping schema τ takes part in determining the similarity index for a schema mapping $T \mapsto \tau$. Clustering criteria which exploits this fact, can be formed as follows.

Mapping node joins the cluster in which other *relevant nodes* are closer. Mapping node n_2' is a mapping node for n_1, i.e., $n_2' \in \mathcal{M}_{n_1}$. In T node n_1 is connected to nodes n_2 and n_3, and therefore, the relevant nodes for n_2' are nodes in $\mathcal{M}_{n_2}$ and $\mathcal{M}_{n_3}$. The distance of the node n_2' to the relevant nodes in C_1 is $\mathit{path\_len}(n_2', n_4') = 2$, and $\mathit{path\_len}(n_2', n_5') = 2$. In C_2 these distances are $\mathit{path\_len}(n_2', n_7') = 2$, and $\mathit{path\_len}(n_2', n_6') = 1$. Node n_2' joins C_2 due to shorter paths to relevant nodes.

Cardinality

Relative cardinality of schema elements is a useful feature in the objective function. For example, in personal schema T, the Kleene star specifies that n_1 can contain none-or-more elements n_3. This means, that n_2' being a mapping element for n_1 prefers members of $\mathcal{M}_{n_3}$ with which it has a *none-or-more* cardinality. In the example, the cardinality between nodes n_2' and n_6' is *exactly one*, while the cardinality between nodes n_2' and n_5' is *none-or-more*. This should be implemented in the clustering criteria to make cluster C_1 preferable cluster for n_2'.

Relation
In a reasoning similar to that of the cardinality, we can say that since n_1 and n_2 are in a *parent-child* relationship, node n'_2 prefers the node n'_4 with which it has *ancestor-descendant* relationship, over the node n'_7 with which n'_2 has *sibling* relationship. Clustering criteria should use this heuristics to make cluster C_1 a cluster of choice for n'_2.

Minimum spanning tree (replacement for path length)
The suggested clustering criteria discussed so far closely follows the heuristics used in the objective function. However, the computation of these criteria during the clustering can be quite expensive. Therefore clustering criteria can use a different simpler to compute heuristics, which is still in line with the heuristics used in the objective function. For example, the path length heuristics, as formulated above, takes into account the relevance of nodes, and needs to computes the distances to all relevant nodes. This is computationally demanding, and this criteria can replaced with the minimum spanning tree heuristics: clustering criteria can be the minimization of the minimum spanning trees (MST) connecting elements in each cluster. In the example, clusters C_1 have MST of size 2, and C_2 have MST of size 3. When considering the addition of the node n'_2, this criteria favors the C_2 because the MST size would not change in that case, while by adding n'_2 to C_1, the MST size of C_1 grows from 2 to 3. Fast incremental algorithms exist for the computation of the minimum spanning tree [32, 42].

These were some examples of clustering criteria which are based on the heuristics used in the objective function. Other heuristics can be used as well, but this time aiming on improving certain aspects of clustering not directly related to objective function. We give one example.

Number of useful clusters
If the node n'_2 joins the C_1, cluster C_2 will be a non useful cluster, i.e., cluster which cannot deliver any mappings. A clustering criteria can be formulated to increase the overall number of useful clusters, and to assign node n'_2 to cluster C_2.

A number of other questions needs to be answered before having a working clustered schema matching systems. For example, what clustering algorithm to use (see Sec. 3.4), and how to implement the selected clustering criteria? Finally, do the clustering algorithm and the selected criteria provide the expected benefits?

3.3.3 Clustering transforms the schema matching problem

This section discusses the relation between the formal specification of the schema matching problem (given in chapter 2) and the *non-clustered* and *clustered schema matching*. The section shows that the formation of sets of mapping elements $\mathcal{M}$ and the use of clustering, can be seen as schema matching problem transformation.

For illustration, this section uses the schema matching problem given in Fig. 2.6 (page 33). That problem is formalized as follows (see Sec. 2.3.3).

$$\begin{aligned} X &= (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2}) \\ D &= (N_R, N_R, N_R, \mathcal{L}_R, \mathcal{L}_R) \\ C &= \{IC(X), C(X)\} \\ \Delta(X) &= \textit{definition of the objective function} \end{aligned}$$

where x_{n_i} is a node variable, x_{p_i} is a path variable, N_R is a set of nodes in the repository, $\mathcal{L}_R$ is a set of all possible paths in the repository, C is a set of constrains, and finally, Δ is the objective function. This specification is purely declarative and gives not suggestion on how to build a schema matching system.

On transformation by non-clustered schema matching

The first two processing steps in the schema matching architecture, i.e., *element matching* and *similarity combining*, form the sets of mapping elements $\mathcal{M}_n$, for each node n in the personal schema T. These sets are formed by cross comparing elements of T and R. This transform the schema matching problem. The domain N_R of each node variable x_{n_i} is reduced to $\mathcal{M}_{n_i}$. The transformed problem is:

$$\begin{aligned} X &= (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2}) \\ D_{\mathcal{M}} &= (\mathcal{M}_{n_1}, \mathcal{M}_{n_2}, \mathcal{M}_{n_3}, \mathcal{L}_R, \mathcal{L}_R) \\ C &= \{IC(X), C(X)\}, \textit{(partially evaluated)} \\ \Delta(X) &= \textit{definition of the objective function(partially evaluated)} \end{aligned}$$

The third processing step, i.e., the *mappings combiner*, solves this transformed problem.

On transformation by clustered schema matching

Clustering also transforms the schema matching problem. Clustering is added at the point at which sets of mapping elements $\mathcal{M}$ are already formed, so the transformation given above also applies to the clustered case. Clustering further transforms the problem as follows. Clustering divides the sets of mapping elements into clusters: each $\mathcal{M}_{n_i}$ set gets divided over k clusters, i.e., $\mathcal{M}_{n_i}$ is split into sets $\mathcal{M}_{n_i}^{C_j}, j = 1, k$. Having in mind that clusters are independently considered by the mappings combiner, each cluster $C_i, i = 1, k$ defines an independent schema matching problem:

$$
\begin{aligned}
X &= (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2}) \\
D_{\mathcal{M}}^{C_i} &= (\mathcal{M}_{n_1}^{C_i}, \mathcal{M}_{n_2}^{C_i}, \mathcal{M}_{n_3}^{C_i}, \mathcal{L}_R, \mathcal{L}_R) \\
C &= \{IC(X), C(X)\}, \textit{(partially evaluated)} \\
\Delta(X) &= \textit{definition of the objective function(partially evaluated)}
\end{aligned}
$$

The fact that clustering divides one schema matching problem into a set of k independent problems opens some interesting issues. An idea for further research is to use, as a form of optimization, different solving approaches to solve the schema matching problem in different clusters.

3.4 Choosing the clustering algorithms

Amongst large number of known clustering algorithms, we need to select one suitable for the application in clustered schema matching. In choosing, we need to consider the *effectiveness* and the *efficiency* of the clustering algorithm.

In clustered schema matching, we define the *effectiveness* of the clustering algorithm as the capability of the algorithm to create clusters which comprise highly ranked schema mappings. In other words, clustering algorithm is effective if it successfully discovers areas in repository schemas which yield highly ranked mappings. The *efficiency*, on the other hand, is the time the clustering algorithm needs to form clusters. In clustered schema matching, clustering must be efficient, or else efficiency improvements induced by clustering could become insignificant compared to the clustering overhead. Clustered schema matching cannot be efficient unless the clustering algorithm is efficient.

Of the two listed aspects, only the efficiency aspect can be used in selecting the clustering algorithm for the clustered schema matching. There exist no certain way of predicting how effective any of the clustering algorithms will be in clustered schema matching. Furthermore, it has been shown that different clustering algorithms yield similar effectiveness in schema integration applications [96]. The only way to know the exact effectiveness of the clustering algorithm is to implement it and measure its performance. Consequently, at this stage, efficiency is the only criterion for choosing the clustering algorithm.

3.4.1 Overview of clustering algorithms

Clustering is the unsupervised arrangement of elements into groups of elements, i.e., clusters [44]. Elements within the same cluster share some common property while the elements from different clusters do not. In clustered schema matching, the common property shared by elements in a cluster is the potential of the elements to jointly deliver valuable schema mappings.

Clustering should not be confused with classification. In classification, element is assigned to one of predefined classes. By comparing properties of the element with the properties of classes a decision is made to assign the element to the most similar class. In clustering, however, classes are not known beforehand. In fact, clustering can be used to discover classes.

Most of the clustering algorithms are built around the following concepts:

- element,
- distance measure, and
- cluster abstraction

Element is the entity which is being clustered. Elements are described with one or more properties which characterize the elements. *Distance measure* is usually defined on pairs of elements. Distance measure computes how close, i.e., how similar, the elements are in respect to one or more properties. For example, Euclidean distance measure can be used to cluster points on a geographical map. *Cluster abstraction*, or cluster representation, is used in some clustering algorithms to simplify and make efficient the way in which clusters are managed. Instead of always representing a cluster as a set of elements, cluster abstraction can be used instead. For example, a cluster of points on a map can be represented with a single point, a *centroid*, which is the middle point of centroid. The use of centroid makes clustering more efficient, and also makes it easier for humans to interpret the results of clustering.

All clustering algorithms are divided in two groups:

- *hierarchical algorithms*, and
- *partitional algorithms*.

Hierarchical algorithms

Hierarchical algorithms do not form one set of clusters, but rather several cluster sets. This can be best explained with the use of a *dendrogram* – a tree structure which illustrates the result of hierarchical clustering. Fig. 3.5 shows five points A, B, C, D, E in the Euclidean space which are to be clustered based on their distance. In this case, a hierarchical clustering algorithm produces a dendrogram as shown in the figure. To from one specific set of clusters from this dendrogram, a distance (similarity) threshold must first be selected (see the y axis in the dendrogram). This threshold is in fact, the maximum allowed distance between the elements in the cluster (the interpretation of the threshold depends on the variant of the hierarchical algorithm used). For a small distance threshold, e.g., *level 0* every element belongs to a different cluster (see *level 0* clusters). With the increasing distance threshold some elements will belong to the same cluster, e.g., points A and B, and points C and D at *level 2*. With an infinite threshold, i.e., *level 4*, all elements belong to a single cluster.

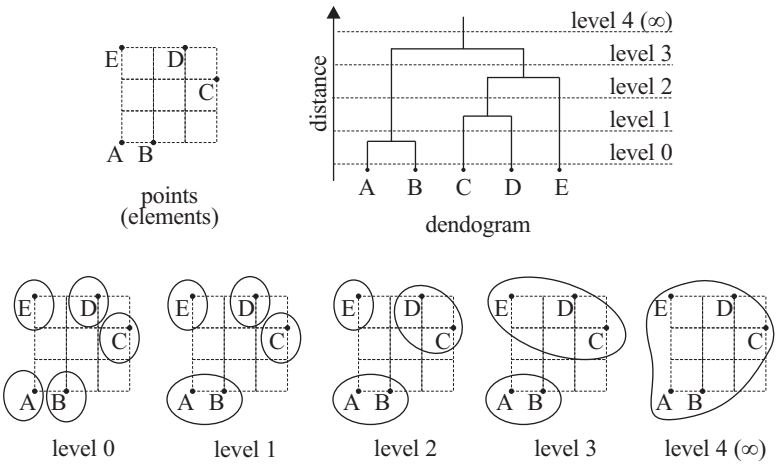


Figure 3.5 *Hierarchical clustering.*

Hierarchical clustering algorithms are well suited for problem in which elements naturally form hierarchical structures, such as animal, or plant hierarchies [26]. Unfortunately, the computational complexity of hierarchical clustering algorithms makes them inapplicable in large data sets. Element hierarchy, i.e., dendrogram, is crated with $O(l \cdot n^2)$ time complexity where l is the number of dendrogram levels and n the number of elements.

Partitional algorithms

Partitional algorithms, unlike hierarchical algorithms, create only one set of clusters. We illustrate partitional clustering approach with a simple example. Fig. 3.6 shows a set of five points which need to be clustered. Say, we need to form two clusters. The first step in clustering is the selection of initial centroids, for example by random placement. The figure shows centroids as X marks (see *centroid seed* in the figure). Centroids are used as the points around which the clusters are formed. Clusters are formed by assigning every point to the nearest centroid. The figure illustrates two different selection for centroids which lead to the creation of different clusters. In the *case 1* clusters are $C_1 = \{A, E\}$ and $C_2 = \{B, C, D\}$, in the *case 2* $C_1 = \{A, B\}$ and $C_2 = \{C, D, E\}$.

Compared with hierarchical algorithms, partitional algorithms are more efficient as they create clusters in a single pass over the set of elements. However, the problem these algorithm face is the need to know beforehand the number of clusters, and to correctly estimate where to form the clusters. As shown in the previous example, different selections of centroids lead to different clusters. Partitional algorithms are often create clusters in an iterative process which ends when

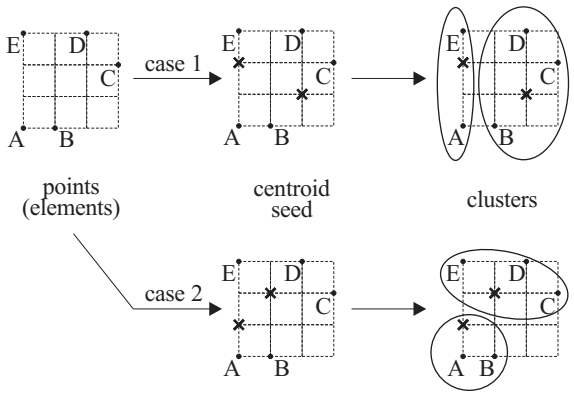


Figure 3.6 *Partitional clustering.*

a certain clustering criteria is met.

There exist a large number of different algorithms in both the hierarchical and partitional directions. These clustering algorithms differ in many aspects. We mention few for illustration.

- *Agglomerative vs. divisive.* Agglomerative clustering algorithms can build clusters by joining small clusters into larger, divisive algorithms split large clusters into smaller ones. In both cases, clustering stops when some clustering criteria is met.
- *Hard vs. fuzzy.* In hard clustering, an element can belong to one cluster only. In fuzzy clustering, elements can belong to several clusters with different degrees of membership.
- *Deterministic vs. stochastic.* Deterministic clustering algorithms come with a guarantee that the formed clusters are the best possible clusters for a given distance measure and the clustering criteria. Stochastic algorithms find local optima clustering solutions.

For more detail on different clustering algorithms refer to [44], and [26] pg. 125.

Following our need to have an efficient clustering algorithm we have decided to use one of the partitional algorithms. The hierarchical algorithms have an exponential complexity which is prohibitive for the use in clustered schema matching. In particular, we decided to use the k-means algorithm, which is the most typical representative of the partitional algorithms. The algorithm is introduced in detail in the next section.

3.4.2 K-means clustering algorithm

Algorithm 1 is the k-means algorithm. Note that the *reclustering* step (line 10) does not exist in the standard k-means algorithm, but will be used in clustered schema matching.

The k-means algorithm uses concepts *element*, *cluster*, *centroid*, and *distance measure*. Elements are entities which are being clustered. In clustered schema matching, elements are in fact *mapping elements* created in the first part of the schema matching process. *Clusters* are thus collections of mapping elements. Each cluster is represented with a *centroid*, an entity which can be computed once the cluster's membership is known. Finally, the *distance measure* is a measure which calculates the distance between a mapping element and a centroid.

The algorithm starts with the initialization of centroids (line 1). The initialization is used to "seed" the centroids around which the clusters will be formed. The initialization determines two things: first, the number of clusters to be created, and second, the locations around which the clusters will be formed.

Algorithm 1 K-means clustering algorithm.

```
 1: initialize centroids
 2: repeat
 3:    for each mapping element do
 4:       for each centroid do
 5:          compute distance(mapping element,centroid)
 6:       end for
 7:       assign mapping element to nearest centroid
 8:    end for
 9:    compute new centroids for all clusters
10:    perform reclustering
11: until convergence criterion is met
```

The iterative part of the algorithm, starts with a nested loop (lines 3 to 8) in which the nearest centroid is determined for each mapping element. In clustered schema matching, distance measure (line 5) computes how important is each element, in relation to some cluster, for the formation of a good schema mappings. A mapping element becomes a member of the nearest centroid's cluster (line 7). After the clusters have been formed, new centroids are calculated (line 9). These new centroids represent clusters in the next iteration. For reasons discussed in Sec. 5.6.3, we have introduced a reclustering step (line 10). This step is used to perform additional modifications of clusters. The iterations continue until a *convergence criterion* is met (line 11).

The complexity of the k-means clustering algorithm is $O(k \cdot i \cdot |\mathcal{M}|)$, where k is the number of clusters being formed, i is the number of iterations, and $\mathcal{M}$ is the set of all mapping elements in the repository schema.

The given algorithm is only a skeleton. For an experimental quantitative analysis all the sub-algorithms of the k-means algorithm, such as the initialization algorithm, the distance measure, and computation of centroids, have to be implemented. These, however, can only be designed once that the exact properties of the schema matching system, which is being extended with clustering, are known. As a part of our research, we develop a schema matching prototype called Bellflower, which implements a non-clustered schema matching system, and a clustering algorithm used to make a clustered schema matching system. Chapter 5 introduces Bellflower, and discusses in depth the implementation of a complete k-means algorithm for the application in Bellflower.

3.5 Related research

Schema matching research largely focuses on the effectiveness of matching systems, with the efficiency of these systems being a secondary concern. Recently, efficiency of schema matching began to attract attention. Bernstein et al., underline that the efficiency of schema matching must be considered and improved in order to build an industrial strength schema matching system [9]. The authors claim that the only way to improve schema matching efficiency is by considering schema matching operations at a higher level, that is at the level of graphs, rather than at the level of individual elements, because simple cross comparison of n individual schema elements always has $O(n^2)$. At a higher level, different techniques and heuristics can be used to reduce the amount of work. Clearly, our clustered schema matching, was conceived on the bases of the same observations; clustered schema matching focuses at improving the efficiency of schema matching, and does so by manipulating elements within a graph (or a tree) with clustering being aware of the mutual relations between the elements in the graph.

A noticeable similarity exists between our work and the work of Rahm et al., on matching large XML schemas [29, 4]. We have introduced their partitioning method and how it relates to clustered schema matching in Sec. 3.2.2.

Other that these two recent works, we found almost no research which focuses on the techniques to improve the efficiency of schema matching systems in large scale environments. The area needs more attention.

Clustering techniques have been widely used to mine large data collections ([26], pg. 125). Similar techniques can be used in large scale schema matching, or in the integration of Web data.

For example, clustering is widely used to discover regularities in Web data. Athena Vakali et al. survey the Web data clustering techniques [91]. *Clustering of users session data* and *clustering of web documents* are marked as the most prominent uses of clustering. User session data clustering is used in applications such as on-line monitoring of user behavior, performance analysis, or in detecting

traffic problems. User sessions are clustered by comparing Weblogs, of individual user's session data. Clustering of web documents, on the other hand, is used to identify *Web communities*. Communities are detected by clustering web pages based on their linking. Such techniques enable Web crawlers and search engines, to focus on particular thematic domains.

Clustering is used in the context of information retrieval (IR) to cluster document both prior or after querying. For example, Tombros et al. demonstrate the potential of different clustering techniques to improve the effectiveness of IR [89]. The relevance of this research lies in the fact that schema matching is a form of information retrieval. Furthermore, schema matching shares heuristics with IR, such linguistic hints for the computation of similarity of strings.

In the context of XML data, clustering is used to create summarizations of XML documents: XCluster Synopses [68] are created by clustering XML elements on both their structure and content.

Clustering has also been used in schema matching. Two distinct approaches can be identified: in first, *schemas are clustered* into groups of schemas, in second, schema *elements are clustered* into groups of elements. Clustering of schemas is performed as a part of schema integration. Clustering first identifies clusters of related schemas, schema integration is then performed per cluster [27, 41, 52]. The other approach, i.e., clustering of elements, is used to discover groups of semantically related elements [67, 85, 94, 96, 12]. It is interesting to see that our clustered schema matching technique bares similarity with both approaches. Clustered schema matching perform clustering of elements, but it does not create clusters of semantically similar elements. Clustered schema matching creates groups of elements which have the potential to form a good mapping schema τ; this makes clustered schema matching similar to techniques which cluster schemas.

Finally, while working with clustered schema matching we drew knowledge from more distant, but related fields. In particular, clustering of objects on a spatial network, provided some insight on the problems related to clustering of nodes in a graph [95].

3.6 Conclusion

Schema matching, being a complex process based on heuristics, offers multiple opportunities for efficiency improvement. After analyzing the approaches for efficiency improvement applicable to schema marching, this chapter proposes the *clustered schema matching*: a technique for improving the efficiency of schema matching by means of clustering. The technique adds a clustering step to the regular, non-clustered schema matching system. Clustering partitions the repository schema and reduces the problem search space. This improves efficiency. The specific choice of the clustering algorithm is also made: the K-means clustering

algorithm is proposed, due to its non-exponential complexity and simplicity, for the initial implementation of the clustering step. The discussion of several sub-algorithms necessary for a full implementation of the clustered schema matching technique is deferred to chapter 5, where the quantitative analysis of the technique also takes place.

Chapter 4

Computing effectiveness bounds

4.1 Introduction

The performance of a schema matching system is defined by its

- *effectiveness*, which represents the capacity of the matching system to deliver semantically correct mappings, and
- *efficiency*, which specifies the amount of resources (e.g., time, main memory, electricity) the system needs to operate.

Effectiveness is either determined analytically, i.e., by *verification*, or experimentally, i.e., by *validation*. Verification of effectiveness uses formal methods, such as model checking [45], to acquire a proof that a system performs correctly. Effectiveness verification can only be used on systems which can be sufficiently formally described. In systems which lack formally verifiable models effectiveness must be determined by validation, i.e., through experiments. Such is the case with schema matching systems. The effectiveness of a schema matching system depends on heuristics used to detect correct mappings. The correctness of these heuristics cannot be modeled, and formal verification is inapplicable: effectiveness can only be determined through validation. The most common approach for effectiveness validation is to use human inspection to determine the set of correct results for given problem. These results are then compared with the results delivered by the system [21]. For reliable validation, the system should be tested using a large set of test cases. Still, validation gives no guaranties that the system will deliver the measured effectiveness in every possible case.

For efficiency, the use of analytical tools is possible in some cases. For example, *cost models* can be used to compute the exact number of CPU cycles needed to complete a CPU microcode operation. For algorithms, *big–O notation* is commonly used to express the complexity, i.e., cost model. For example, "space complexity $O(n^2)$" means that the algorithm needs storage space which is quadratic in size comparing to the size (n) of some input. We have used big–O notation in chapters 2 and 3 to specify the complexity of schema matching and clustering algorithms. In systems which use optimization algorithms (e.g., A* [23] and

beam search [19] used in schema matching systems), cost models are not reliable. Performance of most optimization algorithms depends on the properties of input data: the cost model can, with certainty, only compute the worst case efficiency. However, in practice these algorithms show much higher efficiency. Therefore, efficiency can be determined more accurately using validation, that is, by experiments and run-time monitoring or resource consumption. For these reasons, we also use validation to determine the efficiency of our system.

So far, schema matching research has largely focused on improving the effectiveness, i.e., on improving correct discovery of schema mappings. In this thesis, however, we investigate a clustered schema matching technique which is primarily aimed at increasing the efficiency of schema matching. Consequently, we are more interested in measuring the efficiency gains. At the same time, the technique that we propose is non mapping-preserving, i.e., it alters the result set and consequently changes the effectiveness of the system. This makes the effectiveness measurement in our research a necessity. While it is relatively easy to measure the efficiency of the system, in measuring the effectiveness we encountered a serious challenge: the need to invest an overwhelming amount of human effort in manual problem solving. To get around this severe obstacle we have devised the *effectiveness bounds* validation technique. To the largest part, this chapter is used to describe this technique.

In the first part of this chapter, we briefly describe standard validation techniques in the context of large scale schema matching, with references to related validation approaches. We then describe in detail the effectiveness bounds technique and how it is used to make judgements about the effectiveness of the clustered schema matching technique.

4.2 Standard validation approaches

4.2.1 Validation of efficiency

Efficiency of a system quantifies the consumption of resources during the run-time of the system. Depending on the purpose of the system, different resources might be of interest. Usually, *time* is the main resource: the efficiency shows how quick a system completes a given task. Next important resource is the *storage space*, i.e., main memory or disk space. Some algorithms need extensive amounts of storage space for preserving intermediate results during the run-time. Other kinds of resources include electricity consumption, human working hours, cost of using leased resources such as web-services or network.

In schema matching research, efficiency has mostly been ignored and most work does not report on any efficiency validation. In this thesis, however, efficiency is in the focus and it exclusively considers time consumption. In particular we are

interested in expressing the efficiency improvement acquired by introducing the clustered schema matching technique into existing schema matching systems.

Measuring time

The usual way of measuring the execution time of programs is by placing timers within the code. The run-time overhead induced by code modification is in most cases very small compared to the overall execution time, and is thus acceptable. In complex prototypes, i.e., research systems, it is often the case that some parts of the system are implemented in a naive, thus suboptimal, way. These parts can have a negative impact on the overall performance of the prototype and can make time measurements inaccurate. This problem can be handled in several ways, for example: (1) by applying time measurement only on relevant, e.g., optimized parts, of the algorithm, (2) by using counters to determine how many times an operation is invoked, instead of measuring the execution time, (3) by using time measurements of systems S_1 and S_2 not to deliver an absolute efficiency, but rather to determine the relative efficiency improvement ($\frac{t_{S_2}}{t_{S_1}}$) of system S_2 over system S_1.

Due to the complexity of our clustered schema matching prototype, we use all three techniques for observing the efficiency. More details of efficiency measurements are given in chapter 5.

4.2.2 Validation of effectiveness

The effectiveness of a schema matching system depends on the completeness and the correctness of mappings discovered by the system. The correctness of a schema mapping can only be assessed by a human, and consequently, the effectiveness of a schema matching system can only be determined by means of human inspection. In this respect, schema matching systems are equivalent to other semantic retrieval systems, such as *text document* and *multimedia* retrieval systems. In these, only a human knows if a document, photo, audio or video clip, returned by the retrieval system, is a meaningful answers for a given query.

Precision and recall

The usual measures for reporting effectiveness of semantic retrieval systems are *precision* and *recall*. These measures are regularly used for schema matching systems [21]. In principle, to acquire the precision and recall measures, the process illustrated in Fig. 4.1 must be executed.

First, a test collection must exist. In schema matching, the test collection is usually a set of real-world schema matching problems. In our research, each problem P is a problem of matching one personal schema against a large schema repository. Formal specification for problem P was discussed in chapter 2. For each

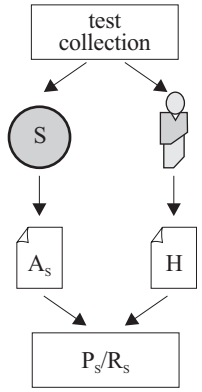


Figure 4.1 *Computing the precision and recall for system S.*

problem P, the *search space*, i.e., set of all possible schema mappings, $SS = \{a_1, \ldots, a_n\}$ is known. In the problem formalization (see Def. 4 page 40) the search space is defined as a set of domains D.

Next, the system S (see Fig. 4.1) solves the problem P by investigating the search space SS. From the search space, the system selects a number of schema mappings which rank high according to the used objective function. In that way the system creates an answer set $A_S = \{a_{i_1}, \ldots, a_{i_m}\}$, where $A_S \subseteq SS$. The system "expects" that these mappings are correct. Independently, a human evaluator (see Fig. 4.1) solves the same problem manually. The evaluator selects only the semantically correct schema mappings, creating in that way a set of correct answers $H = \{a_{j_1}, \ldots, a_{j_k}\}$, where $H \subseteq SS$. The human evaluator inspects the whole search space SS and selects *all* and *only* correct mappings. To partly overcome the problems of this expensive and error-prone activity and to even out subjective human decisions, it is common to involve many human evaluators in the validation of a test collection.

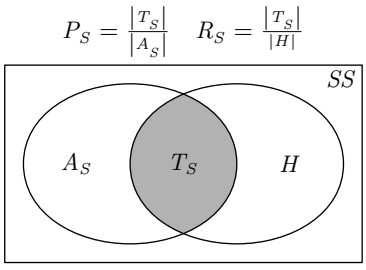


Figure 4.2 *Precision and recall.*

With sets A_S and H available, it is possible to compute the precision and the recall for system S as shown in Fig. 4.2. By knowing H, which comprises all correct mappings, it is possible to distinguishing between correct and incorrect answers in A_S. Let $T_S = H \cap A_S$ be the set of correct answers discovered by the system S. These are called *true positives*. Precision and recall are then defined as in Fig. 4.2. Precision $P_S = \frac{|T_S|}{|A_S|}$ is the percentage of correct answers among the discovered answers. Recall $R_S = \frac{|T_S|}{|H|}$ is the percentage of correct answers found by the system. It is common to report an average precision and recall computed over a number of different test cases.

Precision recall curve (P/R curve)
In schema matching systems, the criterion whether or not a certain mapping will become an answer is not absolute. Rather, each mapping has a certain degree of computed quality by which it is ranked. Quality of the mappings is determined by the *objective function* $\Delta : SS \rightarrow \mathbb{R}$. In this chapter, we assume that the objective function computes how different two schemas are, i.e., if $\Delta(a_1) < \Delta(a_2)$, a_1 is said to be a *better* mapping, i.e., it is higher ranked. Since users are only interested in the most relevant mappings, we define a *threshold* δ which defines the largest allowable difference between two schemas which define a mapping. The *answer set* A_S^δ is the set of mappings for which holds that $\forall a \in A_S^\delta \bullet \Delta(a) \leq \delta$. A system S is called *exhaustive* if it returns all possible mappings for a certain threshold, i.e., $A_S^\delta = \{a \in SS \mid \Delta(a) \leq \delta\}$. Consequently, $\delta_1 \leq \delta_2 \Rightarrow A_S^{\delta_1} \subseteq A_S^{\delta_2}$ (see Figure 4.3). Therefore, by increasing the threshold, we can increase the number of answers S produces. Note that we do not exclude a situation where $\Delta(a_1) = \Delta(a_2)$ in which S is indecisive.

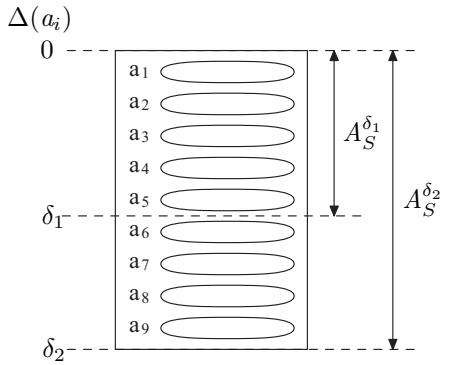


Figure 4.3 *Thresholds in an answer set.*

We now want to observe the changes in precision and recall with the changing threshold δ. By increasing the threshold, the number of answers grows, and so

does the recall. However, the natural behavior of a schema matching system, or to this end, the natural behavior of any semantic retrieval system, is to loose precision with rising recall; when producing more answers, the chance of delivering incorrect answers, i.e., answers outside H, increases. The characteristics of the precision/recall trade-off is captured by a P/R curve (see Fig. 4.5). The intended way of constructing a P/R curve is by determining the precision for the fixed recall levels $0, 0.1, \ldots, 1$. Since it is hard to find the right parameters for obtaining these exact recall levels, the P/R curve is often constructed by varying the threshold and then measuring precision and recall. We call this a *measured P/R curve*. See Figure 4.4 for an illustration of such a curve.

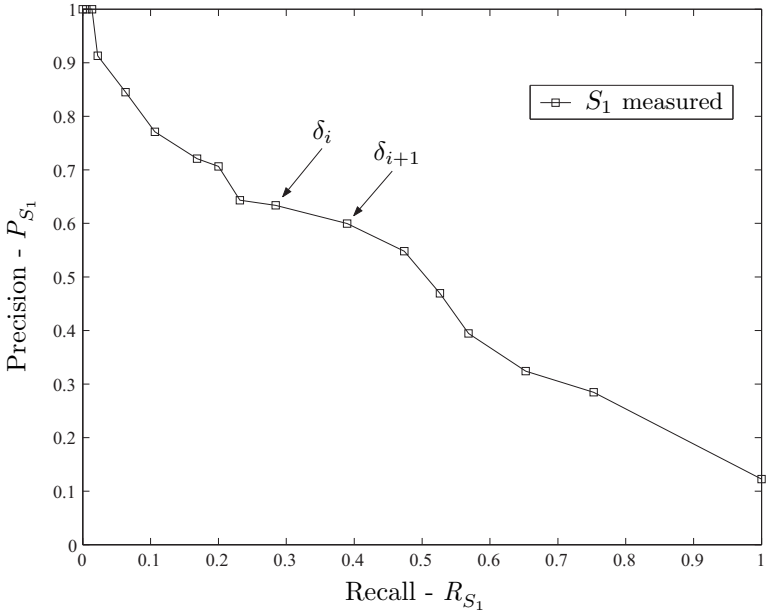


Figure 4.4 *Measured P/R curve.*

Figure 4.5 shows an interpolated P/R curve constructed from the measured one in Figure 4.4. This 11-point P/R curve is constructed from the measured P/R curve by interpolating the precision at the 11 recall levels. Different interpolation methods can be used.

4.2.3 Validation of effectiveness in large scale problems

Recent works show large-scale, high-performance schema matching systems are gaining more attention [9, 29]. At the same time, the availability of large and properly constructed test collections is rather limited in the schema matching

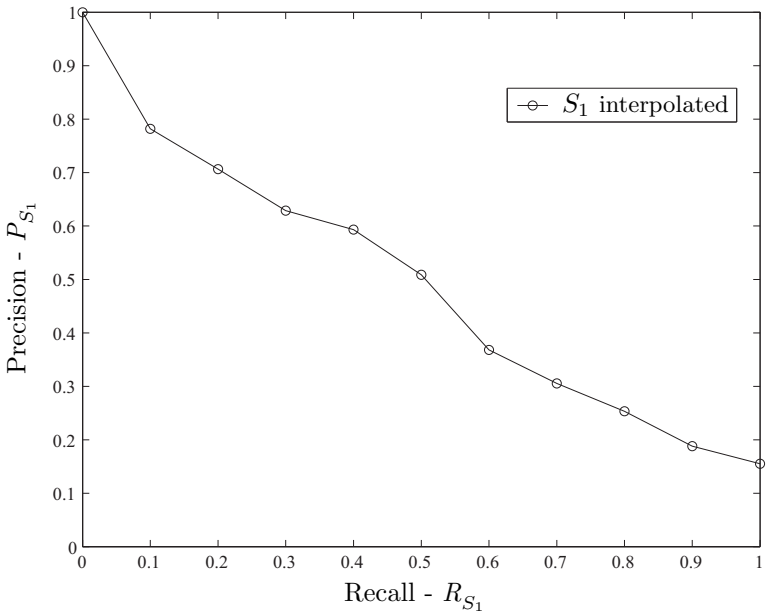


Figure 4.5 *Interpolated P/R curve.*

domain. Currently, validation of the schema matching system is performed using small, "home-made" test collections.

In large scale schema matching environments human evaluators face an insurmountable amount of work. The size of the search space SS is exponential in respect to the size of the personal schema, and polynomial in respect to the size of the repository schemas (see Sec. 2.2.2). Human inspection of the whole search space is not achievable, and the set H cannot be determined. To overcome this scalability limitation of the native precision recall computation, other techniques must be used.

The text document retrieval community has taken the lead in developing techniques to reduce the required amount of effort and in the construction of large properly evaluated test collections. For example, in TREC *pooling* was used [39]: for each keyword query, the top 100 documents produced by each participating system were merged and only these were evaluated by a human. This works under the assumption that it is highly unlikely that a significant number of answers are not found by any participating system. Zobel confirmed that the limit of 100 is adequate [97]. Another validation project similar to TREC is INEX [34]. In INEX large number of participating systems use a common collection of 12000 XML annotated IEEE documents to validate the retrieval effectiveness. Systems can validate up to 1500 answers per query. The main task of INEX is to provide

the infrastructure for the immense human effort required to create the set of correct answers H for each query. On average, for each query, the required effort of the human evaluator is one person week, and each evaluator is assigned 2 queries.

Further, Buckley and Voorhees examined techniques of measuring effectiveness that are robust to massively incomplete relevance judgments [11]. They also suggest that their techniques allow studies of operational efficiency by embedding a small test collection with known judgments in a much larger test collection of similar documents with no judgments. Zobel suggested that a shallow pool of about 30 documents could be evaluated to predict the number of relevant documents further down [97]. More recently, Sanderson and Joho review three methods of test collection construction to see whether or not query and/or system pooling can be avoided to be able to "build a new test collection quickly and with limited resources" [73].

In schema matching, Sayyadian et al., describe a system for unsupervised tuning of schema matching systems by means of synthetic scenarios [74]. The approach requires that a number of correct mappings is known beforehand. Tuning system uses transformation rules on these mappings to synthesize a larger number of different schemas, i.e., synthetic schemas, which are used in validating the effectiveness and tuning of schema matching systems.

In this thesis, we investigate efficiency improvements using large scale schema repositories built by collecting schemas from the Internet. As explained earlier, full semantic validation by a human is impractical in such scenarios. Therefore, H is unknown to us as well as all quality measures derived from it. In the next section, we introduce an approach that is independent of H but can still provide effectiveness indicators for a certain class of retrieval systems: a class to which clustered schema matching also belongs.

4.3 Introduction to effectiveness bounds approach

The clustered schema matching technique is used to improve the efficiency of an existing schema matching system. We say that S_1 is the original schema matching system and S_2 is the improved system, i.e., clustered schema matching system. As explained in Sec. 3.3, the clustered schema matching technique is a *similarity-preserving* and *non mapping-preserving* technique. It computes the same similarity indexes for mappings as the original system, but might not discover all the mappings as the original system. The beam search algorithm used in iMap [19], and the probabilistic guarantees approach of [88] are also examples of similarity-preserving and non mapping-preserving system improvements.

Fig. 4.6 shows the relation between the answer sets of systems S_1 and S_2 for some matching problem. Since S_2 is similarity-preserving and non mapping-preserving in respect to system S_1, it is guaranteed that $A^{\delta}_{S_2} \subseteq A^{\delta}_{S_1}$. The figure

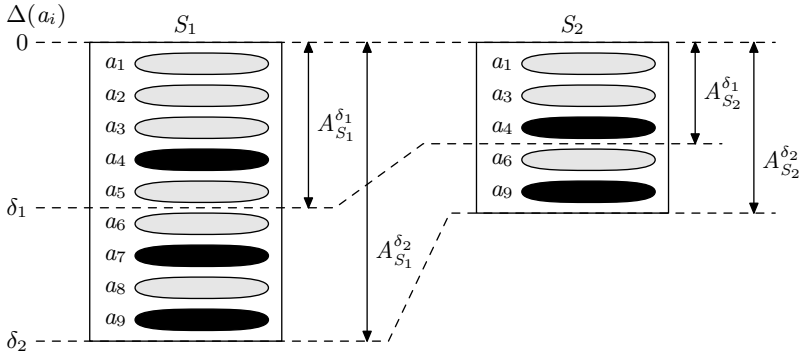


Figure 4.6 *Answer sets produced by the original system S_1 and the improved system S_2.*

shows that S_2 returned part of the original system answers.

Mappings in the answer sets are either semantically correct or incorrect. For example, Fig. 4.6 depicts correct answers (i.e., answers in H) in gray and incorrect answers in black. Improved system S_2 apparently misses answers a_2, a_5, a_7, and a_8. In this example, the improved system exhibits rather bad effectiveness: it misses three correct answers and only one incorrect answer. Another view on the sets $A^{\delta}_{S_1}$ and $A^{\delta}_{S_2}$ is given in Fig. 4.7. The figure illustrates again the relation $A^{\delta}_{S_2} \subseteq A^{\delta}_{S_1}$. Furthermore, it becomes visible that the sets of true positives are also related through subset relation $T^{\delta}_{S_2} \subseteq T^{\delta}_{S_1}$. The set H remains the same. It is obvious from the figure that the application of clustered schema matching technique changes the effectiveness, i.e., precision and recall (see Fig. 4.2), of the original matching systems. This fact calls for the validation of effectiveness of the improved system S_2 as a part of its performance analysis.

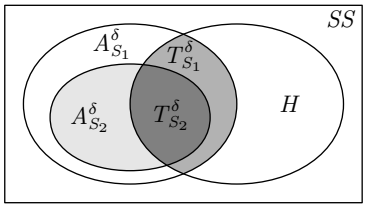


Figure 4.7 *Comparing precision and recall of systems S_1 and S_2.*

In selecting the validation approach for the effectiveness of clustered schema matching we have encountered a serious challenge: in testing the clustered schema matching technique large schema repositories must be used. The performance of clustering depends on various structural and non-structural properties of reposi-

tory schemas. To prove that the technique can handle the diversity encountered in real-world schemas, we must use large repositories with highly heterogeneous schemas. On the other hand, computing precision and recall on such large repositories requires extensive human effort, even with the use of approximate techniques such as pooling or synthesis (see Sec. 4.2.3). Faced with this problem, we have designed a different validation technique: the *effectiveness bounds technique*. The technique is based on the following intuition.

If we disregard the correctness of mappings in the answer sets $A^{\delta}_{S_1}$ and $A^{\delta}_{S_2}$, we can still make judgements of the effectiveness. For example, we can say that clustered schema matching system S_2 has the same effectiveness as the original system S_1 if it returns the same set of answers, i.e., $A^{\delta}_{S_2} = A^{\delta}_{S_1}$. Also, if S_2 looses answers otherwise produced by the original system S_1 (both the correct and the incorrect ones) we can say that S_2, in general case, is less effective than S_1. Assuming that system S_1 is properly designed, it will rank relevant mappings, i.e., correct mappings, higher. For this reason, and having in mind that S_2 is similarity-preserving system, it is better if S_2 looses answers which rank low than if it looses answers which rank high.

We convert these intuitions into a computational technique for determining the effectiveness bounds of the system S_2. The effectiveness bounds technique, illustrated in Fig. 4.8, is possible under following three assumptions:

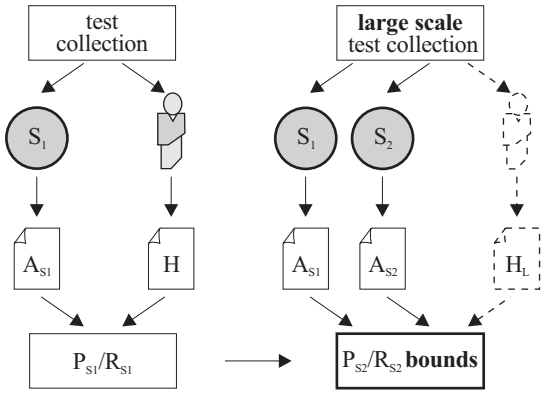


Figure 4.8 *Computing the effectiveness bounds for the improved system S_2.*

- The effectiveness of the original system S_1 must be known, for example, by means of computing the effectiveness on a small repository. Fig. 4.8 depicts the effectiveness of S_1 as a precision recall box P_{S_1}/R_{S_1}.
- This measured effectiveness of the system S_1 is assumed to be largely independent of the size of the search space. In other words, regardless of the size of the schema repository, we assume S_1 will show the same or very similar

P_{S_1}/R_{S_1} curve. In the schema matching community we found no research which investigates this assumption, but in the text document retrieval community this appears to be a reasonable assumption [6].
- For tests performed using a large scale test collection, size of the answer set $A^{\delta}_{S_1}$ (produced by the original system S_1) and size of the answer set $A^{\delta}_{S_2}$ (produced by the improved system S_2) are known for various thresholds δ (note, δ is omitted in the figure).

With these assumptions it possible to compute the bounds, i.e, a lower and upper bound between which the P_{S_2}/R_{S_2} curve of the improved system S_2 is certainly positioned. Though the technique does not compute the precise P_{S_2}/R_{S_2} curve, it can be considered as a valuable asset in the validation of effectiveness since it:

- provides effectiveness guaranties,
- requires no human involvement in the validation of the large scale tests, i.e., the set H_L in Fig. 4.8 does not need to be known. The technique can be used in an automated way to quickly get an impression on the efficiency-effectiveness trade-off and the potential effectiveness changes for many different parameter settings and different system improvements, using many different test cases,
- can be used to assess the accuracy of an effectiveness estimate which was acquired using other approximate validation techniques.

The technique can be applied in many unfavorable research situations for low-effort scalability and efficiency research. It alleviates the need for manual (human) mapping discovery on large schema matching test collections. It does not require the test collection of the original system to be known. For example, when doing research on efficiency improvements of other people's systems, the test collection associated with published effectiveness measures need not be known. Instead, abovementioned techniques [11, 97, 73, 74] can be used to construct new large test collections that are expected to produce roughly the same effectiveness measures. Furthermore, it may happen that an original system for which effectiveness results have been published, is not available. Since the objective function of a system determines the actual ranking, a reconstruction with the same objective function exactly copies its behavior, and the effectiveness measures of the original system are expected to carry over to the reconstruction.

In the next section we describe the computational technique used to determine the effectiveness bounds for the clustered schema matching technique.

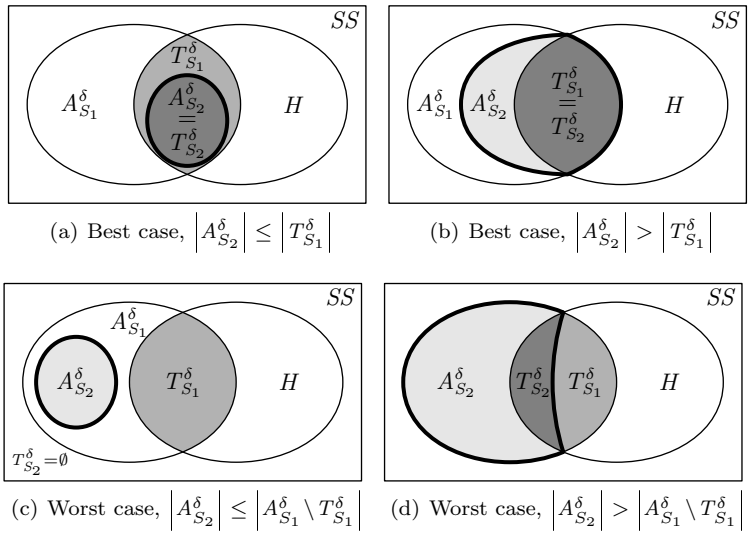
4.4 Computation of effectiveness bounds

4.4.1 Best and worst-case analysis

For the original system S_1, the improved system S_2, and any matching problem P, we know that $A^{\delta}_{S_2} \subseteq A^{\delta}_{S_1}$. We cannot know if the answers which S_2 misses are correct or incorrect ones, since the set H is not known. We can, however, perform a best and worst case analysis. In the best case, S_2 misses only incorrect mappings and preserves the maximum amount of correct, in the worst case it misses the maximum amount of correct mappings.

Best case scenario

Let us first focus on the best case scenario. In this scenario, the set of answers $A^{\delta}_{S_2}$ keeps as much as possible correct answers which exist in the set of answers $A^{\delta}_{S_1}$. Two situations can be distinguished: If $A^{\delta}_{S_2}$ is smaller than $T^{\delta}_{S_1}$ (see Fig. 4.9(a)), $A^{\delta}_{S_2}$ is fully included in $T^{\delta}_{S_1}$, i.e., $A^{\delta}_{S_2} \subseteq T^{\delta}_{S_1}$, hence $T^{\delta}_{S_2} = A^{\delta}_{S_2}$. Otherwise, $T^{\delta}_{S_1} \subseteq A^{\delta}_{S_2}$ (see Fig. 4.9(b)), hence $T^{\delta}_{S_2} = T^{\delta}_{S_1}$. Based on this observation, we can derive the equations for precision and recall of S_2 solely in terms of precision and recall of S_1 and the sizes of the answer sets $A^{\delta}_{S_1}$ and $A^{\delta}_{S_2}$. For brevity, let $\widehat{A}^{\delta}_{S_2/S_1} = \frac{|A^{\delta}_{S_2}|}{|A^{\delta}_{S_1}|}$ be the size ratio of the answer set sizes. Eq. 4.1 and Eq. 4.2 compute the best case precision and recall.



(a) Best case, $\left|A^{\delta}_{S_2}\right| \leq \left|T^{\delta}_{S_1}\right|$ (b) Best case, $\left|A^{\delta}_{S_2}\right| > \left|T^{\delta}_{S_1}\right|$

(c) Worst case, $\left|A^{\delta}_{S_2}\right| \leq \left|A^{\delta}_{S_1} \setminus T^{\delta}_{S_1}\right|$ (d) Worst case, $\left|A^{\delta}_{S_2}\right| > \left|A^{\delta}_{S_1} \setminus T^{\delta}_{S_1}\right|$

Figure 4.9 *Computing the best and worst case precision and recall.*

[**best case**]

$$
\begin{aligned}
\left|T^{\delta}_{S_2}\right| &= \min(\left|T^{\delta}_{S_1}\right|, \left|A^{\delta}_{S_2}\right|) \\
P^{\delta}_{S_2} &= \frac{\left|T^{\delta}_{S_2}\right|}{\left|A^{\delta}_{S_2}\right|} = \frac{\min(\left|T^{\delta}_{S_1}\right|, \left|A^{\delta}_{S_2}\right|)}{\left|A^{\delta}_{S_2}\right|} \\
&= \min(\frac{\left|T^{\delta}_{S_1}\right|}{\left|A^{\delta}_{S_2}\right|}, 1) \\
&= P^{\delta}_{S_1} \cdot \min(\frac{1}{\widehat{A}^{\delta}_{S_2/S_1}}, \frac{1}{P^{\delta}_{S_1}}) \qquad (4.1)
\end{aligned}
$$

$$
\begin{aligned}
R^{\delta}_{S_2} &= \frac{\left|T^{\delta}_{S_2}\right|}{|H|} = \frac{\min(\left|T^{\delta}_{S_1}\right|, \left|A^{\delta}_{S_2}\right|)}{|H|} \\
&= \min(R^{\delta}_{S_1}, \frac{\left|A^{\delta}_{S_2}\right|}{|H|}) = R^{\delta}_{S_1} \cdot \min(1, \frac{\left|A^{\delta}_{S_2}\right|}{\left|T^{\delta}_{S_1}\right|}) \\
&= R^{\delta}_{S_1} \cdot \min(1, \frac{\widehat{A}^{\delta}_{S_2/S_1}}{P^{\delta}_{S_1}}) \qquad (4.2)
\end{aligned}
$$

Worst case

In the worst case, the set of answers $A^{\delta}_{S_2}$ looses as much as possible correct answers which exist in the set of answers $A^{\delta}_{S_1}$. We can also distinguish two situations. If $A^{\delta}_{S_2}$ is smaller than the set of *false positives* $A^{\delta}_{S_1} \setminus T^{\delta}_{S_1}$ (see Figure 4.9(c)), then it may be fully 'detached' from $T^{\delta}_{S_1}$, i.e., the set $A^{\delta}_{S_2}$ will comprise no correct mappings, and precision and recall of system S_2 would be zero. Otherwise, if the set $A^{\delta}_{S_2}$ is larger, we will get a situation as depicted in Figure 4.9(d). Eq. 4.3 and Eq. 4.4 compute the worst case precision and recall.

[worst case]

$$
\begin{aligned}
\left|T^{\delta}_{S_2}\right| &= \max(0, \left|A^{\delta}_{S_2}\right| - (\left|A^{\delta}_{S_1}\right| - \left|T^{\delta}_{S_1}\right|)) \\
P^{\delta}_{S_2} &= \frac{\left|T^{\delta}_{S_2}\right|}{\left|A^{\delta}_{S_2}\right|} \\
&= \max(0, \frac{\left|A^{\delta}_{S_2}\right| - (\left|A^{\delta}_{S_1}\right| - \left|T^{\delta}_{S_1}\right|)}{\left|A^{\delta}_{S_2}\right|}) \\
&= \max(0, 1 - (\frac{\left|A^{\delta}_{S_1}\right|}{\left|A^{\delta}_{S_2}\right|} - \frac{\left|T^{\delta}_{S_1}\right|}{\left|A^{\delta}_{S_2}\right|})) \\
&= \max(0, 1 - (\frac{1}{\widehat{A}^{\delta}_{S_2/S_1}} - \frac{P^{\delta}_{S_1}}{\widehat{A}^{\delta}_{S_2/S_1}})) \\
&= \max(0, 1 - \frac{1 - P^{\delta}_{S_1}}{\widehat{A}^{\delta}_{S_2/S_1}}) \qquad (4.3)
\end{aligned}
$$

$$
\begin{aligned}
R^{\delta}_{S_2} &= \frac{\left|T^{\delta}_{S_2}\right|}{|H|} \\
&= \max(0, \frac{\left|A^{\delta}_{S_2}\right| - (\left|A^{\delta}_{S_1}\right| - \left|T^{\delta}_{S_1}\right|)}{|H|}) \\
&= \max(0, \frac{\left|A^{\delta}_{S_2}\right|}{|H|} - \frac{\left|A^{\delta}_{S_1}\right|}{|H|} + \frac{\left|T^{\delta}_{S_1}\right|}{|H|}) \\
&= \max(0, R^{\delta}_{S_1}(\frac{\left|A^{\delta}_{S_2}\right|}{\left|T^{\delta}_{S_1}\right|} - \frac{\left|A^{\delta}_{S_1}\right|}{\left|T^{\delta}_{S_1}\right|} + 1)) \\
&= \max(0, R^{\delta}_{S_1}\left(\frac{\frac{\left|A^{\delta}_{S_2}\right|}{\left|A^{\delta}_{S_1}\right|} - \frac{\left|A^{\delta}_{S_1}\right|}{\left|A^{\delta}_{S_1}\right|}}{\frac{\left|T^{\delta}_{S_1}\right|}{\left|A^{\delta}_{S_1}\right|}} + 1\right)) \\
&= \max(0, R^{\delta}_{S_1}(\frac{\widehat{A}^{\delta}_{S_2/S_1} - 1}{P^{\delta}_{S_1}} + 1)) \qquad (4.4)
\end{aligned}
$$

Note that the computation adheres to the starting assumptions given in Sec. 4.3. All formulas for best and worst case precision and recall for S_2 are defined in terms of the size ratio of the answer sets of both systems which can be acquired through experiments. Also, computation uses the precision and recall of S_1, which we assumed is known and given as a *measured P/R* curve. In Sec. 4.5.1 we will address

the problem of having an interpolated P/R curveS_1 as an input. Also note that it is not necessary to know the set H.

4.4.2 Computing the best/worst case P/R curve

Using the formulas from the previous section, we can derive the best and worst case P/R curve by varying the threshold: at each threshold value, the matching is performed by both the improved and the original system to obtain the result sizes. The obtained sizes are used to compute both best and worst case precision and recall values, in this way establishing the effectiveness bounds for that threshold value. The curves for best and worst case demarcate the area within which the actual P/R curve should lie. However, we have observed that by simply applying Eq. 4.1 to Eq. 4.4 for computing of the best and worst case recall and precision at each threshold independently, the computed *worst case* P/R curve for system S_2 was unrealistically pessimistic.

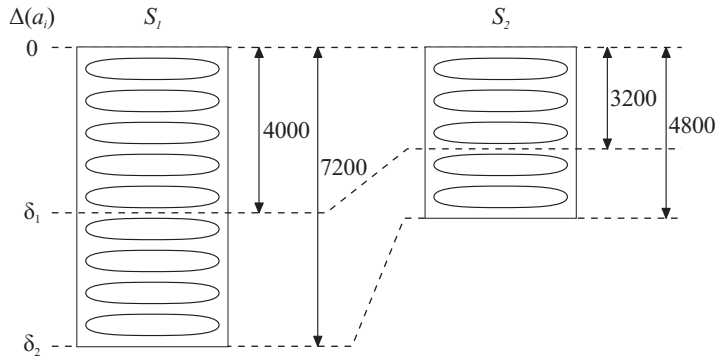


Figure 4.10 *Incremental worst case estimation example.*

We now first illustrate the described problem means of an example, and then provide a better and more accurate way for computing the bounding P/R curves by means of an incremental computation. Fig. 4.10 shows the sizes of the answer sets for a hypothetical system S_1 and its improvement S_2. Sizes are given for two thresholds: δ_1 and δ_2. We assume the experiments were performed with large scale tests and it is unknown which answers in the answers sets are correct and which incorrect. The other known input value is the precision and recall for system S_1 at both thresholds. We assume these are known from some other, small scale experiments. The following is the list of known values based on which

we can compute effectiveness bounds:

$$\begin{array}{lll} P_{S_1}^{\delta_1} = 37.5\%, & P_{S_1}^{\delta_2} = 37.5\% & \\ R_{S_1}^{\delta_1} = 50\%, & R_{S_1}^{\delta_2} = 75\% & (S_1) \\ A_{S_1}^{\delta_1} = 4000, & A_{S_1}^{\delta_2} = 7200 & \\ \hline A_{S_2}^{\delta_1} = 3200, & A_{S_2}^{\delta_2} = 4800 & (S_2) \end{array} \tag{4.5}$$

Let us look at the worst case for the precision of S_2. Using formulas Eq. 4.3 and Eq. 4.4 from the previous section, we compute worst case bounds for each threshold independently:

[**worst case**]

$$\begin{aligned} P_{S_2}^{\delta_1} = 21.875\%, \quad & P_{S_2}^{\delta_2} = 6.25\% \\ R_{S_2}^{\delta_1} = 23.33\%, \quad & R_{S_2}^{\delta_2} = 8.33\% \end{aligned}$$

We can observe the following. By increasing the threshold from δ_1 to δ_2 the answer set of S_2 changes from 3200 to 4800 mappings, i.e., 1600 mappings are added to answer set $A_{S_2}^{\delta_1}$ to create answer set $A_{S_2}^{\delta_2}$. At the same time, worst case recall for system S_2 changes from 23.33% for δ_1 to 8.33% for δ_2. This makes no sense because under no circumstances should recall decrease if the only thing S_2 does is *add* 1600 more answers. Recall $R_{S_2}^{\delta_2}$ can only grow or stay the same as $R_{S_2}^{\delta_1}$.

The reason for this error is the following one. So far, the bounds computation does not take into account the fact that $A_S^{\delta_1} \subseteq A_S^{\delta_2}$. Computation simply applies the equations Eq. 4.3 to Eq. 4.4 to each threshold independently. As such, it assumes that correct mappings which were included in $A_{S_2}^{\delta_1}$ can be excluded from $A_{S_2}^{\delta_2}$, i.e., $A_S^{\delta_1} \not\subseteq A_S^{\delta_2}$.

To overcome this problem, i.e., to make the computation of bounds aware of the relation between answer sets at different thresholds, we devise an incremental computation. An increment is defined by two consecutive threshold values $\delta_1 - \delta_2$. The answer set of the increment $A_S^{\delta_1-\delta_2}$ contains all answers with a ranking between these thresholds, i.e., answers a_i with $\delta_1 < \Delta(a_i) \leq \delta_2$. The answer set is defined by $A_S^{\delta_1-\delta_2} = A_S^{\delta_2} \setminus A_S^{\delta_1}$. Since an increment contains correct and incorrect answers, it makes sense to speak about precision and recall of an increment.

We first show that by knowing the precision and recall at thresholds δ_1 and δ_2, the precision and recall of the $\delta_1 - \delta_2$ increment can be computed. Let $T_S^{\delta_1-\delta_2} = A_S^{\delta_1-\delta_2} \cap H = T_S^{\delta_2} \setminus T_S^{\delta_1}$ be the set of correct answers of increment $\delta_1 - \delta_2$. Precision and recall of an increment are computed as follows

$$
\begin{aligned}
P_S^{\delta_1-\delta_2} &= \frac{\left|T_S^{\delta_1-\delta_2}\right|}{\left|A_S^{\delta_1-\delta_2}\right|} = \frac{\left|T_S^{\delta_2}\right| - \left|T_S^{\delta_1}\right|}{\left|A_S^{\delta_2}\right| - \left|A_S^{\delta_1}\right|} \\
&= \frac{\frac{\left|T_S^{\delta_2}\right|}{|H|} - \frac{\left|T_S^{\delta_1}\right|}{|H|}}{\frac{\left|A_S^{\delta_2}\right|}{|H|} - \frac{\left|A_S^{\delta_1}\right|}{|H|}} = \frac{R_S^{\delta_2} - R_S^{\delta_1}}{\frac{R_S^{\delta_2}}{P_S^{\delta_2}} - \frac{R_S^{\delta_1}}{P_S^{\delta_1}}} \qquad (4.6) \\
R_S^{\delta_1-\delta_2} &= \frac{\left|T_S^{\delta_1-\delta_2}\right|}{|H|} = \frac{\left|T_S^{\delta_2}\right| - \left|T_S^{\delta_1}\right|}{|H|} \\
&= R_S^{\delta_2} - R_S^{\delta_1} \qquad (4.7)
\end{aligned}
$$

By inverting the equations given above, it is also possible to compute the precision and recall at threshold δ_2 given that precision and recall at δ_1 is known as well as the precision and recall of the $\delta_1 - \delta_2$ increment.

$$
\begin{aligned}
P_S^{\delta_2} &= \frac{R_S^{\delta_1} + R_S^{\delta_1-\delta_2}}{\frac{R_S^{\delta_1}}{P_S^{\delta_1}} + \frac{R_S^{\delta_1-\delta_2}}{P_S^{\delta_1-\delta_2}}} \qquad (4.8) \\
R_S^{\delta_2} &= R_S^{\delta_1} + R_S^{\delta_1-\delta_2} \qquad (4.9)
\end{aligned}
$$

Further note that the best and worst case computation presented in the previous section can also be applied on increments. That is, having the precision and recall for the increment $A_{S_1}^{\delta_1-\delta_2}$ of the original system S_1, we can use Eq. 4.1 to Eq. 4.4 to compute the best and worst case precision and recall for the corresponding increment $A_{S_2}^{\delta_1-\delta_2}$ for the improved system S_2. Based on this, and the equations Eq. 4.6 to Eq. 4.9, we can now establish the more accurate increment-based computational approach for effectiveness bounds.

Input
The input for the computation is the measured *P/R* curve of the original system S_1. This means that the precision $P_{S_1}^{\delta}$ and recall $R_{S_1}^{\delta}$ are known for a number of threshold values $0, \delta_1, \ldots, \delta_n$. Also, by performing large scale experiments with both the original system S_1 and the improved system S_2, the sizes of the answer sets $A_{S_1}^{\delta}$ and $A_{S_2}^{\delta}$ are known for the same sequence of thresholds $0, \delta_1, \ldots, \delta_n$.

Step 1: compute the precision and recall for $A_{S_1}^{\delta_i-\delta_{i+1}}$ increments
Using the equations Eq. 4.6 and Eq. 4.7 compute the precision and recall for all the increments $\delta_i - \delta_{i+1}, i = 1, n-1$, that is, compute the precision

$P_{S_1}^{\delta_i-\delta_{i+1}}$ and $R_{S_1}^{\delta_i-\delta_{i+1}}$. Note that $P_{S_1}^{\delta_0-\delta_1} = P_{S_1}^{\delta_1}$ and $R_{S_1}^{\delta_0-\delta_1} = R_{S_1}^{\delta_1}$.

Step 2: compute best and worst case precision and recall for $A_{S_2}^{\delta_i-\delta_{i+1}}$ increments
Using formulas 4.1, 4.2, 4.3, and 4.4, compute the best and worst case precision $P_{S_2}^{\delta_i-\delta_{i+1}}$ and recall $R_{S_2}^{\delta_i-\delta_{i+1}}$ for an improved system S_2 for each increment $\delta_i - \delta_{i+1}, i = 0, n-1$. The required inputs: $A_{S_1}^{\delta_i-\delta_{i+1}}$, $A_{S_2}^{\delta_i-\delta_{i+1}}$, $P_{S_1}^{\delta_i-\delta_{i+1}}$, $R_{S_1}^{\delta_i-\delta_{i+1}}$, are all available.

Step 3: compute best and worst case effectiveness of S_2
Using formulas 4.8, and 4.9 compute the best and worst case precision and recall for S_2 at each threshold δ_{i+1} based on the precision and recall computed at threshold δ_i, and precision and recall of the increment $\delta_i - \delta_{i+1}$. The computation is preformed incrementally, starting with threshold δ_2 and proceeding to δ_n. Computation is initialized with $P_{S_2}^{\delta_1} = P_{S_2}^{\delta_0-\delta_1}$ and $R_{S_2}^{\delta_1} = R_{S_2}^{\delta_0-\delta_1}$. In the special case, it can happen that step 2 computes that the precision of $\delta_i - \delta_{i+1}$ increment is equal zero, i.e., $P_{S_2}^{\delta_i-\delta_{i+1}} = 0$. This causes division by zero error in Eq. 4.8. In such cases, precision and recall for the next threshold level are computed as follows. Having a zero precision for an increment means that no correct answers exist in the increment. Consequently, recall of the next threshold value remains the same, i.e., $R_{S_2}^{\delta_{i+1}} = R_{S_2}^{\delta_i}$. The precision is computed using the following formula, which is derived from basic precision and recall computation (see Figure 4.2):

$$P_{S_2}^{\delta_{i+1}} = \frac{P_{S_2}^{\delta_i}}{1 + \frac{\left|A_{S_2}^{\delta_i-\delta_{i+1}}\right|}{\left|A_{S_2}^{\delta_i}\right|}} \tag{4.10}$$

To illustrate incremental computation approach, let us look again at the example in Fig. 4.10 on page 81.

Input
As given in Eq. 4.6.

Step 1
Increments are $0 - \delta_1$ and $\delta_1 - \delta_2$.
The effectiveness of the first interval is initialized as follows.
$P_{S_1}^{\delta_0-\delta_1} = P_{S_1}^{\delta_1} = 37.5\%,\ R_{S_1}^{\delta_0-\delta_1} = R_{S_1}^{\delta_1} = 50\%$.
The effectiveness of the second interval is computed.
$P_{S_1}^{\delta_1-\delta_2} = 37.5\%,\ R_{S_1}^{\delta_1-\delta_2} = 25\%$.

Step 2
We here compute only the worst case effectiveness.
$P_{S_2}^{\delta_0-\delta_1} = 21.875\%,\ R_{S_2}^{\delta_0-\delta_1} = 23.33\%$ and
$P_{S_2}^{\delta_1-\delta_2} = 0\%,\ R_{S_2}^{\delta_1-\delta_2} = 0\%$.

Step 3
We initialize the worst case precision and recall of S_2 at δ_1 as follows.
$P_{S_2}^{\delta_1} = P_{S_2}^{\delta_0-\delta_1} = 21.875\%,\ R_{S_0}^{\delta_1} = R_{S_2}^{\delta_0-\delta_1} = 23.33\%$.
The worst case precision and recall of S_2 at δ_2 are computed using the described exception case.
$P_{S_2}^{\delta_2} = 14.58\%,\ R_{S_2}^{\delta_2} = 23.33\%$.

This time the computation delivers expected results. In the worst case, there are no correct answers among the 1600 answers in the increment $\delta_1 - \delta_2$, hence $P_{S_2}^{\delta_1-\delta_2} = 0\%$. The worst case precision and recall for S_2 behaves as expected, recall level for δ_2 is the same as at δ_1, but there is a drop in precision, from 21.875% to 14.58%, due to the addition of 1600 wrong answers.

4.4.3 Examples of P/R curve bounds

In this section, we show examples of *P/R* curves which give some insight in the behavior of the process.

The bounding approach given in previous sections is ultimately based on answer sizes, more concretely on $\widehat{A}_{S_2/S_1}^{\delta}$ ratio. Observe that for $\widehat{A}_{S_2/S_1}^{\delta} = 1$, the best and worst case bounds for S_2 are exactly the same and equal to the original *P/R* curve for S_1: if an improved system produces the same number of answers, then it necessarily produces the same answers (since $A_S^{\delta_1} \subseteq A_S^{\delta_2}$), and hence has the same precision and recall characteristics.

Fig. 4.11 shows the answer set ratio $\widehat{A}_{S_2/S_1}^{\delta}$ for three different system improvements. First, the hypothetical system improvement S_2–*hyp* that maintains a fixed answer size ratio $\widehat{A}_{S_2/S_1}^{\delta} = 0.9$ for each threshold δ. In other words, system S_2–*hyp* misses, in all increments, the same fraction of answers (10%) produced by the original system S_1. Fig. 4.12 shows the measured *P/R* curve of some example original system S_1 and the computed best/worst case *P/R* curve bounds of the hypothetical system improvement S_2–*hyp*. The figure should be interpreted as follows. For 30% recall the precision of the original system S_1 is 64%. Best/worst case analysis delivered that the precision of the system improvement is guaranteed to lie between the worst case precision of 56% and the best case precision of 70%. Although the figure does not show the actual *P/R* curve of the improved system, the effectiveness bounds give precise guaranties that the effectiveness remains quite high, at least for smaller recall levels ($\leq$ 40%). Also observe that, even though the answer set ratio is fixed at 0.9, with the increasing recall the bounded region

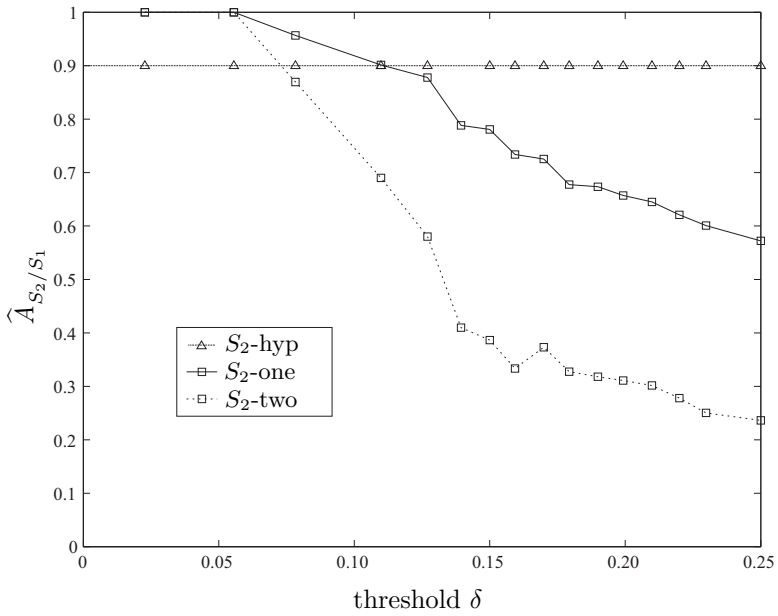


Figure 4.11 *Answer set ratio lines ($\widehat{A}_{S_2/S_1}$) lines for different system improvements.*

becomes wider. When Eq. 4.4 computes that the worst case recall in an increment $R_{S_2}^{\delta_i-\delta_{i+1}}$ to be close or equal zero the worst case bound steeply goes down, and widens the gap (see Fig 4.12). Eq. 4.4 shows that this happens when the precision of the increment is small, in particular when $P_{S_1}^{\delta_i-\delta_{i+1}} < 1 - \widehat{A}_{S_2/S_1}^{\delta_i-\delta_{i+1}}$. Having in mind that the natural behavior of S_1 is to loose precision with increasing recall, also on an increment level, this ultimately explains why the bounds rapidly widen for higher recall values.

The used $\widehat{A}_{S_2/S_1}^{\delta} = 0.9$ ratio is rather high and is rare in real-world systems. To give insight into how the effectiveness bound computation behaves for real-world systems, we chose two actual improvements the same schema matching system S_1. We call these improvements S_2-one and S_2-two. The improvements are in fact, two different setups in our clustered schema matching. These two improvements show different behavior. Fig. 4.11 shows the measured $\widehat{A}_{S_2/S_1}^{\delta}$ for both systems. S_2-one shows a smoothly declining ratio of retrieved answers, with an increasing threshold. At $\delta = 0.25$ about 60% of the answers are still retained. S_2-two is more rigorous in missing answers. Of the answers with a score higher than 0.2, only about 30% is retained. The answers with the best score still have a high chance of being retained though.

The result of determining the best and worst case *P/R* curves for both systems can be found in Figure 4.13 (the "random case" is explained in Sec. 4.4.4). Notice,

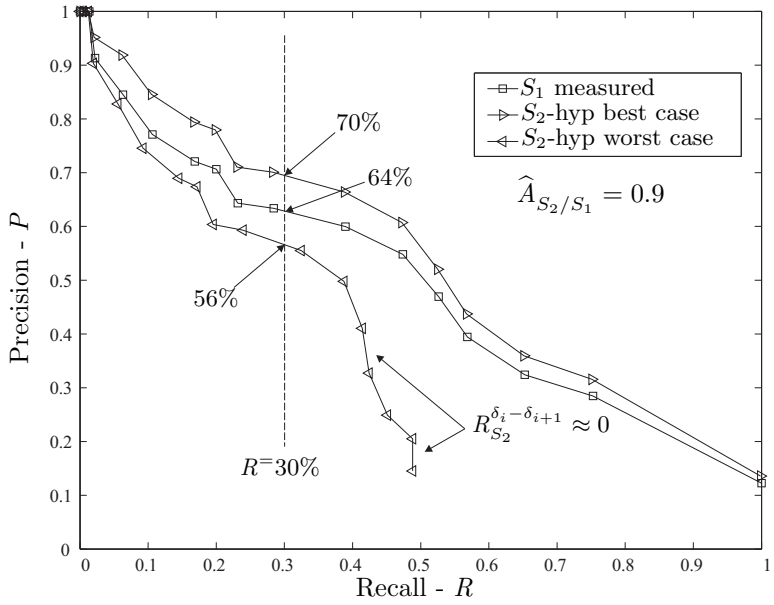


Figure 4.12 *Best/worst case P/R curve for fixed* $\widehat{A}_{S_2/S_1}$*=0.9.*

that for both systems, the best and worst case curves lie far away from each. This does not mean that the systems are bad, only that we could not establish tighter bounds for the effectiveness. However, what we can see, for example, is that for recall levels up to 0.15, S_2-one guarantees a worst case precision of 0.5, while system S_2-two gives this guaranty only for very small recall levels, e.g., at recall 0.03 it guarantees a precision of about 0.80. Based on this, we can get an impression on the potential behavior of the two systems. For applications in which only few top answers are to be considered, both system improvements perform rather well. However, for applications, in which the precision must be guaranteed for higher recall levels, the system S_2-one is likely to perform better than S_2-two. We can also observe the correlation of the computed effectiveness bounds (see Fig. 4.13) with the measured $\widehat{A}^{\delta}_{S_2/S_1}$ ratio (see Fig. 4.11) for the two systems. The faster the answer ratio drops for some system, the faster the worst case P/R curve drops for that system.

Although the computed effectiveness bounds give a rather wide range for positioning the true effectiveness of the improved system, the method has shown that distinctions can be made. Further, the technique shows that the effectiveness bounds depend on the $\widehat{A}^{\delta}_{S_2/S_1}$ curve. In our research, i.e., in chapter 5, we use $\widehat{A}^{\delta}_{S_2/S_1}$ curves of various clustered schema matching technique variants to make judgements about their effectiveness. Though we cannot say what is the exact

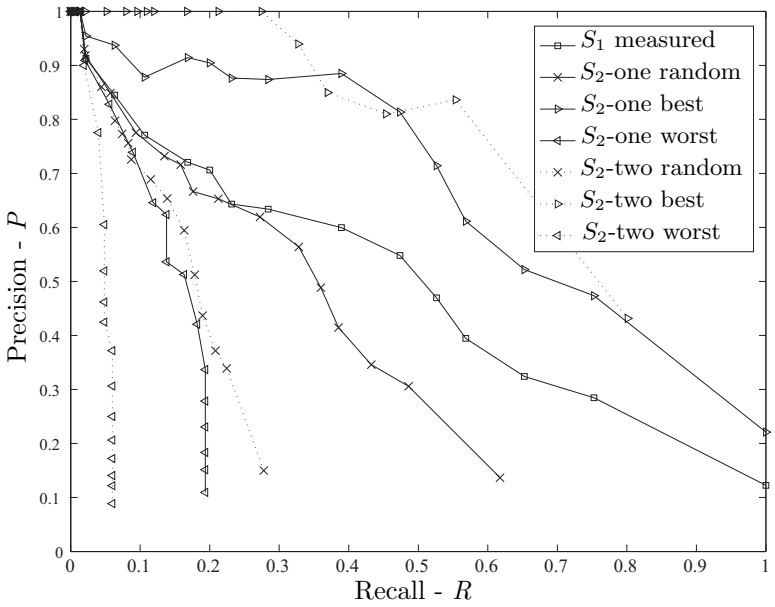


Figure 4.13 *Best/worst case P/R curve for the two experimental systems.*

effectiveness of any particular improvement, we can say that some improvements are much more likely to deliver better effectiveness than other.

4.4.4 Using the P/R curve of a random system for lower bound

The curves for the best and worst case are rather far apart. Another interpretation of what a *worst case* is, may improve this. If we assume that any non mapping-preserving improvement that we construct, produces a better set of answers than simply picking them randomly, then we may use the P/R curve of this hypothetical random system as worst case bound. In this section, we explore this idea.

Let S_1 be the original schema matching system and S_2 a similarity-preserving and non mapping-preserving improvement of S_1. Let S_{random} be a random system that simply executes S_1 and for each increment selects a certain percentage of answers randomly. Since we are using the random system to compare with S_2, we need it to produce the same number of answers as S_2. In other words, S_{random} has the same answer size ratio curve as S_2 (see Figure 4.11).

The random P/R curve is computed using the incremental computation described in Sec. 4.4.2 with the following difference. In *Step 2* the best and the worst case formulas for precision and recall of an increment are not used. Instead,

the precision and recall of the increment of the random system are computed using the following formulas.

[**random case**]

$$P^{\delta_i-\delta_{i+1}}_{S_{random}} = P^{\delta_i-\delta_{i+1}}_{S_1} \tag{4.11}$$

$$R^{\delta_i-\delta_{i+1}}_{S_{random}} = R^{\delta_i-\delta_{i+1}}_{S_1} \cdot \frac{A^{\delta_i-\delta_{i+1}}_{S_{random}}}{A^{\delta_i-\delta_{i+1}}_{S_1}} \tag{4.12}$$

These formulas are acquired as follows. The random system blindly picks mappings from every increment $A^{\delta_i-\delta_{i+1}}_{S_1}$ in order to deliver $A^{\delta_i-\delta_{i+1}}_{S_{random}}$. With such an approach, the ratio between the correct and the incorrect answers in $A^{\delta_i-\delta_{i+1}}_{S_{random}}$ remains the same as that of $A^{\delta_i-\delta_{i+1}}_{S_1}$, that is $\frac{\left|T^{\delta_i-\delta_{i+1}}_{S_{random}}\right|}{\left|A^{\delta_i-\delta_{i+1}}_{S_{random}}\right|} = \frac{\left|T^{\delta_i-\delta_{i+1}}_{S_1}\right|}{\left|A^{\delta_i-\delta_{i+1}}_{S_1}\right|}$. When combining this equation with the ones given in Fig. 4.2 the result are Eq. 4.11 and Eq. 4.12: precision of the random system increment does not change, but the recall is reduced proportional to its size.

Fig 4.13 also shows the curves of the random system corresponding to S_2-**one** and S_2-**two**. Given the expectation that an improved system performs better than the random system, this gives a more useful lower bound, since it produces a narrower interval. For example, Figure 4.13 shows that precision of 0.5 is maintained up to a recall of 0.35 for S_2-**one**.

4.5 Interpolation in effectiveness bounds computation

4.5.1 Using an interpolated *P/R* curve as input

The best and worst case analysis presented above is based on a measured *P/R* curve of system S_1 (see Fig. 4.4). An 11-point *P/R* curve taken from the literature (called interpolated *P/R* curve in Sec 4.2.2, see Fig. 4.5) seems to be equally suitable, but in fact, it lacks one kind of information: the specific threshold points. Observe that from an interpolated *P/R* curve, it is not possible to determine at which δ-value a certain precision and recall was measured. Without this information, it is not possible to correlate the published precision and recall measures of S_1 with the answer sets produced by S_1 on a different, large scale, test collection.

The following is a theoretical observation for which the practical implications are not investigated in this thesis. In the case described above, one can try to estimate, or simply *guess*, the size of H in an experiment on a large scale test collection. Knowing the experimental answer sets on a large test collection, and

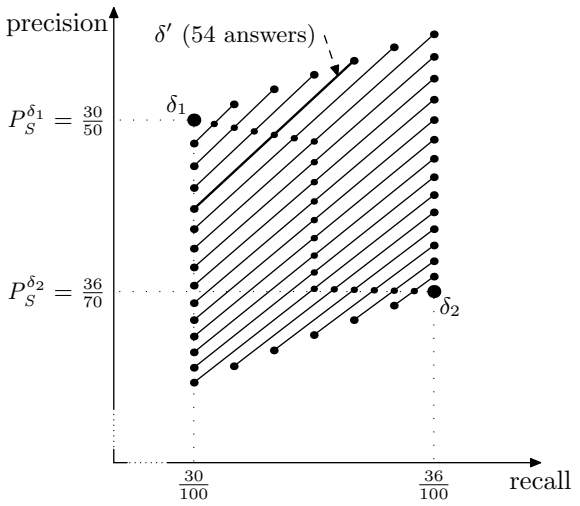


Figure 4.14 *Determining sub-increment precision and recall by interpolation.*

by guessing the size of H, it is possible to establish the correspondence between the δ-values and the given interpolated P/R curve. This correlation can however, only be done by means of interpolation, for which the basic relation is establish through equations in Fig 4.2 which show that $\left|A_S^\delta\right| = \frac{R_S^\delta}{P_S^\delta} \cdot |H|$.

4.5.2 Sub-increment level bounds

In this section we investigate the case when the experiments on large collections use more (finer) threshold points than the given measured P/R curve of S_1; the need arises to compute the P/R values for the thresholds not directly specified in the measured P/R curve. This can only be done by interpolation. Without going into detail about the effects of different interpolation approaches, we give a characterization of the boundaries between which interpolated points on the P/R curve are guaranteed to lie. We analyze this issue by means of an example.

Say, we have obtained from literature results from an experiment with a certain system S and we rebuilt this system using the same objective function. Suppose, at two thresholds δ_1 and δ_2, literature reports $|H| = 100, R_S^{\delta_1} = \frac{30}{100}, R_S^{\delta_2} = \frac{36}{100}, P_S^{\delta_1} = \frac{30}{50}, P_S^{\delta_2} = \frac{36}{70}$. This is illustrated with the two big points in Figure 4.14. Our rebuilt system produces 50 and 70 answers for these thresholds, respectively.

Let us examine some intermediary threshold $\delta_1 \leq \delta' \leq \delta_2$. We observe that our rebuilt system produces 54 answers. Since there is no quality measurement available, precision and recall for this threshold are unknown, i.e., the location of the point on the P/R curve corresponding with δ' is unknown. We do know,

however, that at δ_1, there were 30 correct answers among the 50. At δ', there are 4 more answers of which it is unknown whether or not they are correct. In the worst case, they are all incorrect ($R_S^{\delta'} = 30, P_S^{\delta'} = 30/54$), in the best case they are all correct ($R_S^{\delta'} = 34, P_S^{\delta'} = 34/54$). In other words, an interpolated point for δ' should lie on the line between $(30/100, 30/54)$ and $(34/100, 34/54)$. In Figure 4.14, this is depicted with the thick line marked "δ' (54 answers)".

By varying thresholds between δ_1 and δ_2, one obtains many lines that demarcate boundaries for interpolating points on the *P/R* curve. Note that taking the point halfway between worst and best case (small dots in the figure) is not the same as linear interpolation between δ_1 and δ_2.

The shape of the boundary can be explained as follows. Close to the measured points, there are only a few answers for which it is unknown whether or not they are correct or incorrect. This establishes restrictions on how good the best case and how bad the worst case can be. In the figure, this becomes apparent by the three sections observable in the halfway-points. The fact that precision can go up in a *P/R* curve was also observed in the appendix of [39]. Without further analysis, the figure shows, that the safest, i.e., with smallest error, interpolation choice is made by taking the mid points in the lines.

Finally, because several answers may have the same score, best and worst case points may not be as evenly distributed in practice as in the figure.

4.6 Conclusion

In this thesis (in chapter 5), the effectiveness of different clustered schema matching variants is compared using *answer set ratio* lines. This is done in an automated way, without the human evaluation.

One can argue that answer set ratio lines are not valid indicators of the system's true effectiveness, i.e., precision and recall. To prove differently, this chapter describes a computational technique which, under certain restrictions (which are valid in clustered schema matching) computes the *effectiveness bounds* using the measured answer set ratio line. The bounds, the lower bound in particular, can be used to observe the system's guaranteed performance and, as such, are an effectiveness indicator. Bounds can be used as a quick estimates of the expected effectiveness of some system. The determination of the exact effectiveness, unfortunately, still requires other, more expensive, human-based validation techniques.

Chapter 5

Clustered schema matching in tree-based repositories

5.1 Introduction

Chapter 3 introduced clustered schema matching as a technique for improving the efficiency of schema matching systems. The technique was described and the expected benefits analyzed. The aim of this chapter is to experimentally confirm the main assumptions of clustered schema matching: *improved efficiency* and *preserved effectiveness*. Clustered schema matching is a complex (i.e., multicomponent) technique, and this chapter also investigates how the sub-components of the technique cooperate and participate in determining the performance of the whole system. This chapter investigates more the holistic behavior of clustered schema matching than the individual performance of the components of the technique.

The experiments performed in this chapter are based on Bellflower: an experimental system for clustered schema matching. In previous chapters, for some algorithms which take part in the clustered schema matching technique, e.g., the k-means clustering algorithm, not all sub-algorithms were described in detail sufficient for implementation. This chapter proposes implementations for all the missing parts, and builds a complete experimental system.

This chapter has two parts. The first part, comprising Sec. 5.2 through Sec. 5.4, introduces important details of the experimental system and the validation approach. In particular, Sec. 5.2 gives an overview of the experimental approach implemented in Bellflower. It introduces and demonstrates, through one complete schema matching experiment, the way in which clustered schema matching technique is validated. Sec. 5.3 describes details of the algorithms implemented in Bellflower. Sec. 5.4 describes how the experimental schema repositories are built and discusses their main properties.

The second part of the chapter, that is, sections 5.5 through 5.9, describes and discusses, by means of experiments and analytically, various aspects of the clustered schema matching technique. Sec. 5.5 discusses the influence of element matching on other components of the clustered schema matching technique. Sec. 5.6 discusses how to configure the clustering algorithm and what is the re-

lation between the parameters of the clustering algorithm and the performance of the whole clustered schema matching system, Sec. 5.7 discusses the efficiency, effectiveness, and the balance between the two in clustered schema matching. Sec. 5.8 addresses the importance of the correlation between the schema matching algorithm and the clustering algorithm. Sec. 5.9 discusses the scalability of the clustered schema matching technique.

5.2 Overview of the experimental approach

This section uses a sample experiment to introduce a validation approach for the clustered schema matching technique. The validation experiments are executed on a prototype system Bellflower.

5.2.1 Experimental workflow

Bellflower implements both the standard, i.e., non-clustered, schema matching (see Sec. 2.2.2) and the clustered schema matching technique (see Sec. 3.3.1). Fig. 5.1 shows the workflow implemented in Bellflower. White rectangles, in the figure, represent input and output data of various processing steps, and gray ovals are the processing steps. The workflow has two branches: the upper *clustered schema matching branch* and the lower *non-clustered schema matching branch*.

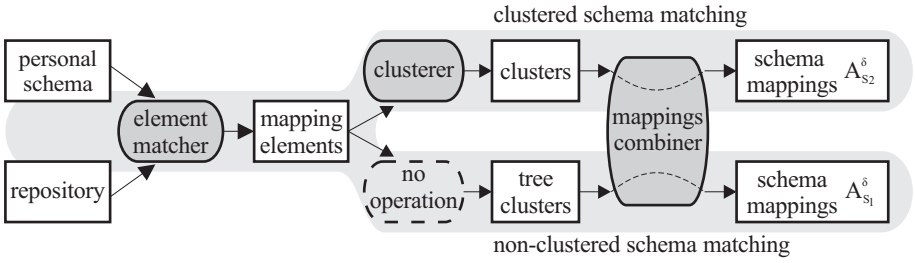


Figure 5.1 *The experimental workflow.*

Both branches start with the element matcher. For a *personal schema*, the element matcher discovers mapping elements in the repository. Note that in Bellflower the repository is a set of schemas, i.e., a forest. Mapping elements are independently discovered in each schema in the repository. Then, the workflow divides. In the lower branch mapping elements are sent to the mappings combiner without any modification. However, having in mind that the repository is partitioned into trees, we say that, even without clustering, mapping elements come partitioned into *tree-clusters*. In the upper branch a *clusterer* creates *clusters*. Compared to tree-clusters, clusters are smaller groups of mapping elements.

Finally, in both branches, (tree-)clusters are sent to a *mappings combiner* which generates a set of schema mappings $(A^{\delta}_{S_1})\ A^{\delta}_{S_2}$.

We base the validation of the clustered schema matching technique on the comparison with the non-clustered matching. The validation of efficiency measures how much faster the clustered schema matching technique solves the matching problem compared to the non-clustered matching technique. For this, it is necessary to measure the performance of all three processing steps, i.e., *element matching*, *clustering*, and *mappings combining*.

The validation of effectiveness is performed by observing the differences in the resulting sets of schema mappings $A^{\delta}_{S_2}$ and $A^{\delta}_{S_1}$, more concretely, by observing the *answer set ratio line* which shows $\widehat{A}^{\delta}_{S_2/S_1} = \frac{|A^{\delta}_{S_2}|}{|A^{\delta}_{S_1}|}$ for various values of δ threshold. The answer set ratio line is first introduced in chapter 4 (See Fig. 4.11) where it is used to compute the effectiveness bounds of the clustered schema matching system. This chapter compares the answer set ratio lines of different variants of the clustered schema matching technique to claim which of the variants delivers better effectiveness. It does not, however, determine the exact effectiveness, i.e., precision and recall.

Throughout the chapter, different kinds of tables and graphs are used to specify the input and the output of experiments. Table types are as follows.

- The *experiment table* specifies the repository, the personal schema and the parameters of the algorithms used in a specific experiment.
- The *clusters table* reports the results of clustering. Among others, the number and the size of formed clusters.
- The *iterations table* is used to monitor the properties of clusters in each iteration of the k-means algorithm.
- The *performance table* reports the measured efficiency (time and counters) for an experiment.

Graph types are as follows.

- The *answer set ratio* graph displays the ratio of answer sizes, for the clustered and the non-clustered schema matching, i.e., $\widehat{A}^{\delta}_{S_2/S_1}$, at various values of the similarity index threshold δ.
- The *cluster size distribution* graph displays the number of clusters having specific sizes.

When used for the first time, these and other tables and graphs will be explained in more detail.

Note that although the repository schema R is a collection of a large number of schema-trees, i.e., a forest, for the brevity of expressions in this chapter we shall treat the repository schema R as being a single large tree. Only when considering the scalability in Sec. 5.9 will the forest structure of the repository be explicitly addressed.

5.2.2 Definition of an experiment

Each experiment starts with a definition of the goal, algorithms, parameters, and input data used in the experiment. The *experiment table* contains most of this data.

Experiment 1: The overview experiment

This experiment is used to illustrate the validation approach. The details concerning the experiment are given in the *experiment table* Tab. 5.1.

In this experiment a user is looking for the *address* and the *e-mail* of his friend. To express his information need, the user creates a personal schema (with a rather naive XML design) shown in Fig. 5.2. The *personal schema* section in Tab. 5.1 defines the personal schema using a pseudo language: */name/*{*email, address*}. This personal schema is used throughout the chapter.

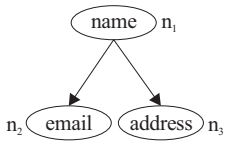


Figure 5.2 *Personal schema.*

Further, the user expects Bellflower to discover possible mappings against a repository called B, as specified in the *repository* section in Tab. 5.1. Note that the experiment table only shows the name and the size (i.e., number of elements) of the repository. Other properties of the repository are given in Tab. 5.5 in the Sec. 5.4. The B repository is a tree-based repository comprising 76 schemas with 4676 markup nodes. For illustration, Fig. 5.3 shows a small (0.5 percent) fragment of the B repository. Note that different elements can have equal names. In total 4676 elements in the repository B share 1467 different element names.

Table 5.1 *Experiment table for Exp. 1.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
B	4676	/name/{email,address}	0.4

Clusterer		
initialization	reclustering	convergence
MIN-init 0.40	join 3; remove 3	5% nodes

Mappings combiner			
α	0.5	δ	0.7

The experiment table has three more sections: *Element (El.) matcher*, *Clusterer*, and *Mappings combiner*. These provide details of algorithms and the parameters of the algorithms used in the experiment. For now, these three section of

the table are shown for illustration purposes only. The algorithms implemented in the Bellflower will be described in Sec. 5.3. The following sections describe the experimental results for the three processing steps, and the acquired performance indicators.

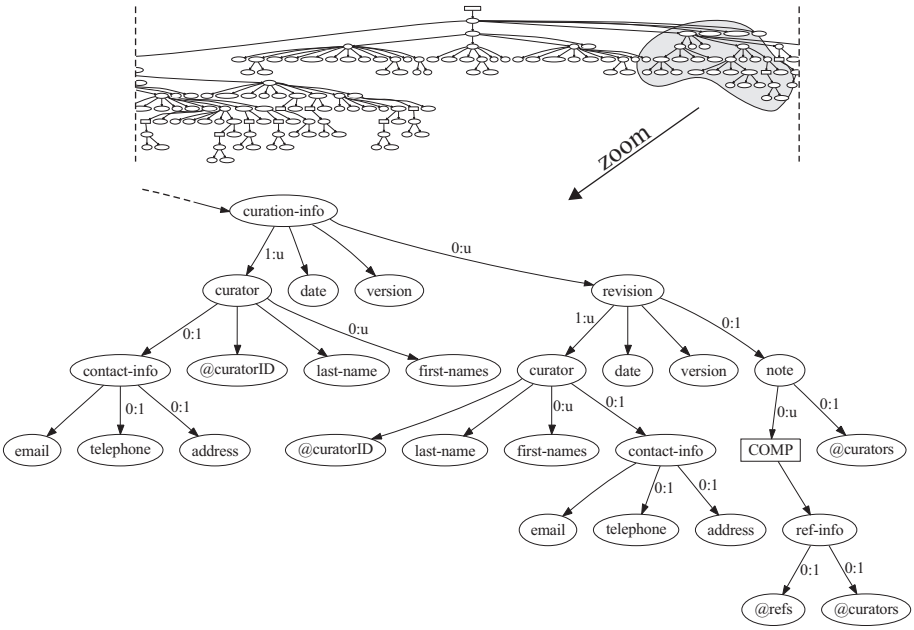


Figure 5.3 *A part of the B repository.*

5.2.3 Element matching

Element matching is the first processing step in the experimental workflow shared by both the non-clustered and the clustered matching approach (see Fig. 5.1). The outputs of the element matcher are sets of mapping elements $\mathcal{M}^j_{n_i}$, where n_i are personal schema nodes and j identifies each XML schema in the repository. In the current experiment there are three personal schema nodes n_1, n_2, and n_3 and 76 schemas in the repository B, i.e., $j = 1..76$. The union of all sets of mapping elements for the n_i personal schema node is denoted with $\mathcal{M}_{n_i}$, and the union of mapping elements for the whole personal schema is denoted with $\mathcal{M}$.

Repository schema node n' becomes a member of $\mathcal{M}_n$ if the element matcher computes the similarity between n and n' to be greater than the numerical threshold $\delta_{element} = 0.4$. This threshold is specified in the *element matcher* section in Tab. 5.1. In Bellflower, the element matcher computes element similarity index by comparing element names. Implementation details will be given in Sec. 5.3.1.

In the current experiment, the resulting sets of mapping elements have following sizes: $|\mathcal{M}_{n_1}| = 1159$, $|\mathcal{M}_{n_2}| = 788$, and $|\mathcal{M}_{n_3}| = 617$. Note that these numbers do not appear in any table. Instead, as shown later, a total number of mapping elements $|\mathcal{M}| = |\mathcal{M}_{n_1} \cup \mathcal{M}_{n_2} \cup \mathcal{M}_{n_3}| = 2564$ is reported in the clusters table (i.e., Tab. 5.2).

Bellflower used 3.18 *seconds* to generate these sets. This time, given later in the performance table Tab. 5.3, indicates the efficiency of the element matcher. Note that because element matching is not in the focus of this thesis, Bellflower's implementation of the element matcher is a simple nested loop. We expect the efficiency of an element matchers using advanced algorithms to be much better (see Sec. 3.2). Sec. 5.5 will discuss the impact that element matching has on other steps in clustered schema matching.

Fig. 5.4 illustrates mapping elements discovered in the fragment of the repository B; the fragment was previously shown in Fig. 5.3. Elements with thick solid border-line belong to $\mathcal{M}_{n_1}$. The nodes belonging to the other two sets are indicated in a similar way. Numbers in the nodes represent the element similarity index computed by the element matcher.

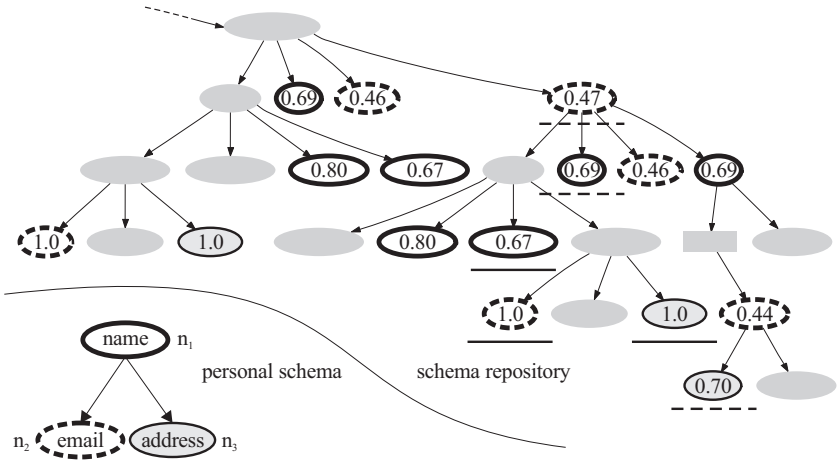


Figure 5.4 *Sample sets of mapping elements.*

5.2.4 Mappings combining

In the current experiment, sets of mapping elements are created for 76 repository schemas. For each repository schema, i.e., for each tree-cluster, sets of mapping elements are sent to the *mappings combiner*. The mappings combiner combines the mapping elements and generates an ordered list of all schema mappings a_i (where $a_i = T \mapsto \tau_i$). In Bellflower, the mappings combiner generates only schema

mappings for which the value of the objective function is larger than some given numerical threshold δ (not to be confused with $\delta_{element}$ used in element matching in Sec. 5.2.3). Tab. 5.1 specifies that in this experiment $\delta = 0.7$. In Bellflower, δ is set manually. Sec. 5.3.2 describes the implementation of the objective function and the combining algorithm used by the mappings combiner.

Fig. 5.5 illustrates two mapping schemas generated by the mappings combiner. For these mapping schemas the objective function evaluates to $\Delta(T, \tau_1) = 0.86$, $\Delta(T, \tau_2) = 0.61$. Bellflower expects mapping $T \mapsto \tau_1$ to be "more correct" than $T \mapsto \tau_2$. The two mapping schemas are also depicted in Fig. 5.4: elements which belong to mapping schemas are underlined.

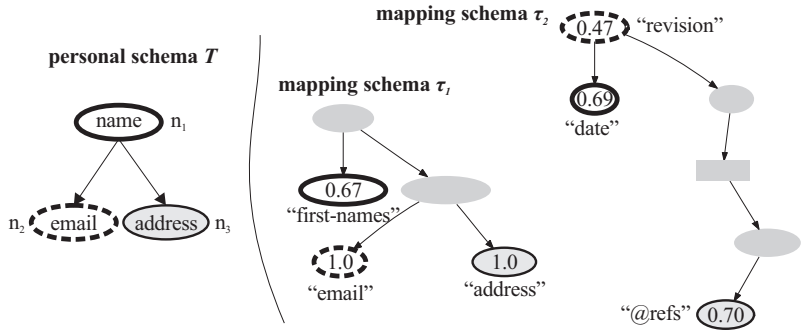


Figure 5.5 *A personal schema and two mapping schemas.*

Apart from these two, in the current experiment there exist a total of 8.26×10^6 mapping schemas. This number, which can be found in clusters table (see Tab. 5.2), is the search space size for the current matching problem. Out of these, as it will be shown later, only 9199 (i.e., 0.11%) schema mappings satisfy the condition $\Delta(a) \geq 0.7$, i.e., $|A^{\delta}_{S_1}| = 9199$, *where* $\delta = 0.7$. It took Bellflower's mappings combiner 360.4 *seconds* to discover these mappings: a time too long for a system aiming to provide a real-time response. Besides the fact that Bellflower is an experimental system, the long matching time is tributed to the complexity of schema matching problems. This complexity is the main obstacle for improving efficiency, i.e., the main problem faced in this thesis.

Fig. 5.6 shows the number of solutions for various δ threshold values. The results of the non-clustered matching (in this chapter always referred to as system S_1) are shown with the "triangle" line. Note that the y-axis is logarithmic; with a decreasing threshold, the number of possible solutions grows rapidly. For example, there exist 7 mappings (i.e., $|A^{0.9}_{S_1}| = 7$) with an objective function value larger or equal 0.9 (i.e., $\Delta(a) \geq 0.9$), and 457 mappings for $\delta = 0.8$.

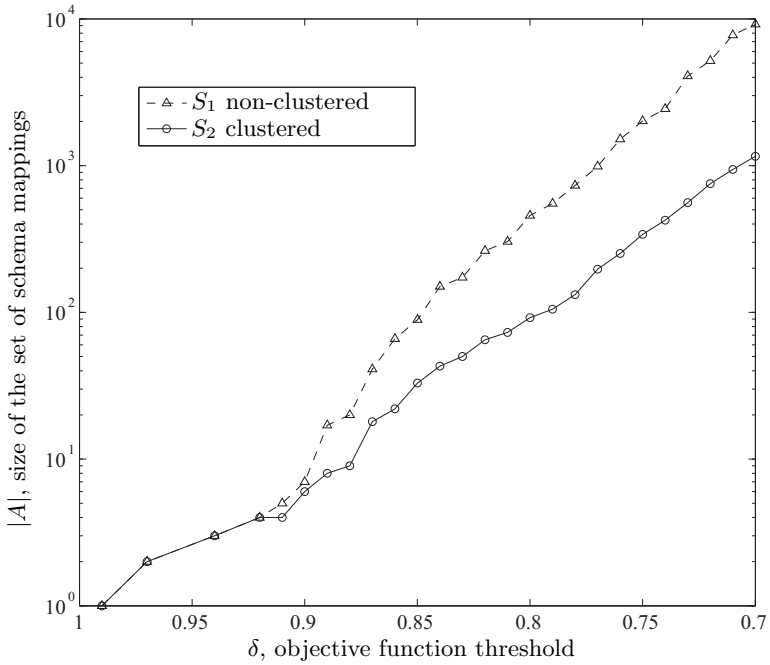


Figure 5.6 $|A^{\delta}_{S_1}|$ *and* $|A^{\delta}_{S_2}|$ *for various objective function thresholds* δ*.*

5.2.5 Clustering

The clustered schema matching branch in the Bellflower workflow starts with a clusterer. Bellflower implements the k-means clustering algorithm as proposed in Sec. 3.4. Details of the implementation are given in Sec. 5.3.3.

The results of clustering, for the current experiment, are given in the *clusters table* Tab. 5.2. Each row of the clusters table represents a different clustering

Table 5.2 *Clusters table for Exp. 1.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
k-means	121 [159]	2258 [2478]	18.7 / 25.6	0.23 (2.75%)
tree-clusters	45 [76]	2464 [2564]	54.8 / 111.3	8.26 (100%)

technique used in the experiment. The *clustering* column specifies the name of each clustering technique: the *k-means* row reports the results of the clustering performed in the clustered branch of the current experiment. The *tree-clusters* row

reports the properties of tree-clusters which are used in the non-clustered branch of the workflow.

The second column in the table, i.e., *clusters*, shows the number of clusters formed in the whole repository. In this column, a distinction is made between *all clusters* and *useful clusters*. A *useful cluster* is a cluster which comprises at least one complete mapping schema: to form one mapping schema a cluster needs at least one mapping element for each personal schema node. Numbers given in the square brackets report the total number of clusters, regardless of their "usefulness." The clusterer creates multiple clusters in each schema in the repository. Therefore, the number of clusters is expected to be larger than number of tree-clusters. In this experiment, K-means clustering creates 159 clusters, out of which 121 are useful. The tree-clusters row shows that there exist 76 tree-clusters; this is the number of schemas in the repository. Out of these, only 45 can produce schema mappings.

Next, in the table, the *mapping elements* column shows the number of mapping elements contained within the formed clusters. Again, a distinction is made between useful and all clusters. Depending on the clustering algorithm, some mapping elements can end up not being included in any cluster. For example, the tree-clusters comprise all 2564 mapping elements of the $\mathcal{M}$ set. K-means clusters, however, comprise 86 elements less, i.e., 2478. The reasons for this are explained in Sec. 5.6.1 (Exp. 3).

The next two columns consider only useful clusters. The *cluster size* column shows the average size and the standard deviation of the size of the useful clusters, in terms of the number of enclosed mapping elements. The table shows that the clusters contain on average 18.7 mapping elements, while schemas in the repository contain on average 54.8 mapping elements.

Finally, the *schema mappings (search space)* column shows the total number of mapping schemas that can be composed within useful clusters. This number represents the size of the search space within which the mappings combiner looks for highly ranked schema mappings. In the current experiment, tree-clusters can deliver a total of 8.26×10^6 mappings. The 100% in the table means that this is the maximum size of the search space, i.e., search space when clustering is not used. In the k-means row, the search space is reduced to 0.23×10^6 mappings, or 2.75% of the non-clustered case. This means that clustering reduced the search space for about 35 times.

In the current experiment, Bellflower's clusterer performed clustering in 6.0 seconds (see the performance table Tab. 5.3).

5.2.6 Validation of clustered schema matching

Clusters of mapping elements are sent to the mappings combiner (see Fig. 5.1). Both the clustered and non-clustered branches use the same mappings combiner:

the same objective function Δ and the same objective function threshold $\delta = 0.7$.

In the clustered case, the mappings combiner discovers 1157 *good*[1] schema mappings, i.e., schema mappings a with an objective function value $\Delta(a) \geq \delta$, i.e., $|A^{\delta}_{S_2}| = 1157$ (see the last column in the performance table Tab. 5.3). Fig. 5.6 shows, with a circle-line (i.e., system S_2), the distribution of discovered mappings for different threshold values. As expected the number of solutions produced by clustered matching is smaller than the number of solutions produced by the non-clustered matching. In total, for $\delta = 0.7$, non-clustered matching delivered $|A^{\delta}_{S_1}| = 9199$ mappings while clustered matching produced $|A^{\delta}_{S_2}| = 1157$ mappings, which is 12.5% of the non-clustered case. Note however, that these 12.5% of mappings were discovered in only 2.75% of the original search space. Another way to look at the effect of clustering is through the density of good mappings within the search space. Without clustering, this the density is $9199/8.26 \times 10^6 = 0.11\%$, after clustering the density is $1157/0.23 \times 10^6 = 0.5\%$, that is, 5 times larger. Though these constants do not tell much about the final practical implications of the technique, they validate the desired behavior of clustered schema matching: clustering reduces the search space and, while doing so, preserves more good mappings and cuts-off more bad mappings.

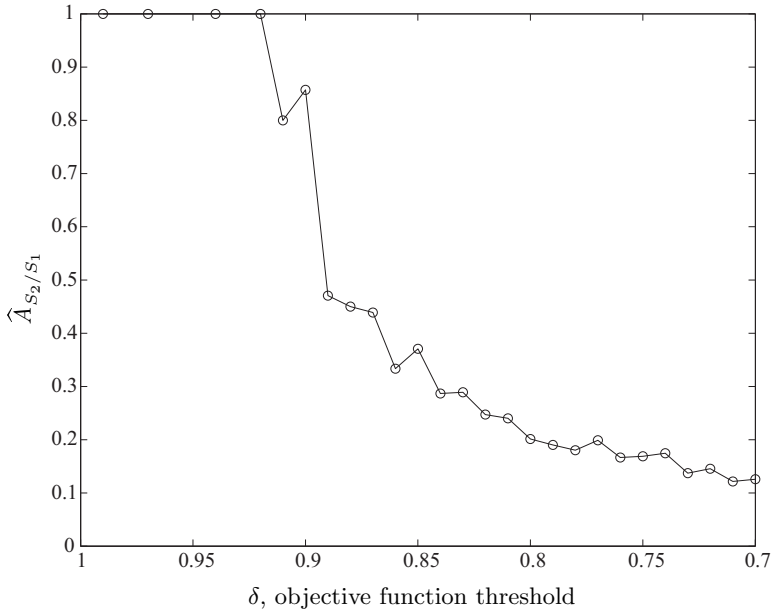


Figure 5.7 *Answer set ratio line for the Exp. 1.*

[1] *good mapping* ($\Delta(a) \geq \delta$) is not to be confused with a *correct mapping*(human judgment)

Effectiveness

To get an insight on how this reduction affects the effectiveness of the system, it is important to analyze which answers in the non-clustered set $A^{\delta}_{S_1}$ are preserved in $A^{\delta}_{S_2}$. Fig. 5.7 shows the answer set ratio line obtained in the current experiment (note, answer set ratio line can be computed by "dividing the lines" in Fig. 5.6). The figure shows that for high threshold values the clustered matching technique preserves most or all schema mappings. A drop is observed for lower threshold values: clustering looses good schema mappings. This, however, is a desired behavior: clustering preserves the mappings which rank high and looses mappings which rank low.

In this chapter we only use the answer set ratio line to judge on the effectiveness of different clustered schema matching techniques. When comparing two clustering techniques, the clustering technique which preserves more mappings, highly ranked mappings in particular, is considered more effective.

Efficiency

The efficiency of matching depends on the efficiency of the three processing components. The performance table Tab. 5.3 shows times consumed by each of these components in the current experiment.

Table 5.3 *Performance table (timers & counters) for Exp 1.*

clustering	element matching	clustering	mappings combining	Σ (sec)	partial mappings	$\lvert A^{\delta}_{S}\rvert$
k-means	3.18	6.0	22.4	31.58	8614	1157
tree-clusters	3.18	0	360.4	363.58	23439	9199

It took Bellflower's mapping combiner 22.4 *seconds* to discovers highly ranked mappings in clusters. This is 16 times faster than 360.4 *seconds* consumed in the non-clustered case. Even with the addition of clustering time, which only exists in the clustered case, and the addition of element matching time, the total execution time is $363.58/31.58 \approx 11$ times shorter. Bellflower is a proof of concept prototype, and some algorithms are implemented in a suboptimal way. A well tuned system can be expected to have much better performance. A more reliable indication of how much faster the mappings combiner completes in presence of clustering, can be acquired by counting the number of times the most consuming operations are invoked. This approach will be used later in the chapter.

Note that most time savings come from the mappings combiner. The total improvement thus also depend on the distribution of work between element matchers and the mappings combiner.

In this section, we have introduced the experimental approach used to validate the clustered schema matching technique. We have also demonstrated the main benefits of the clustered schema matching – the ability to increase schema matching efficiency by reducing the problem search space in such a way that preserve the important mappings while reducing the workload for the mappings combiner.

5.3 Clustered schema matching algorithms in Bellflower

The clustered matching workflow given in Fig. 5.1 has three processing components: *element matcher*, *clusterer*, and *mappings combiner*. This section describes algorithms used to implement these three components in Bellflower.

5.3.1 Element matcher

In this thesis, we do not do research on the effectiveness and the efficiency of the element matcher. We see the element matcher as a service which provides input for testing the clustering algorithm. Therefore, in our prototype it is sufficient for the element matcher to simulate the behavior of more advanced element matchers, such as the ones described in Sec. 2.2.2.

Bellflower uses a single element matcher called *node name matcher* (`NNM`). The `NNM` matcher is very similar to Similarity flooding's `StringMatch` [59] and `EditDistance` matcher in COMA [22]. The node name matcher is implemented using the `CompareStringFuzzy`[2] function. The `CompareStringFuzzy` function compares two strings and returns a similarity index in the range $[0, 1]$. The similarity index is computed based on character substitution, insertion, exclusion, and transposition. Examples of the computed name similarity index are given in the third column of Tab. 5.4.

Table 5.4 *Examples of using `CompareStringFuzzy` and `NNM` functions.*

XML element		`CompareStringFuzzy`		`NNM`
n	n'	n,n'	reversed n,n'	n,n'
"Name"	"Naming"	0.86	0.00	0.86
"Name"	"Company"	0.25	0.25	0.25
"Name"	"FirstName"	0.00	0.80	0.80
"Name"	"Time"	0.55	0.83	0.83

[2]part of `FuzzySearch` library available at `www.textolution.com`

Note that in the third row, contrary to expected, the similarity between elements "Name" and "FirstName" is zero. This *false negative* appears because `CompareStringFuzzy` function gives more significance to a mismatch at the beginning of a string than to a mismatch at the end. Bellflower uses a simple modification to bypass this problem: it compares not only the original strings (column 3 in the table), but also their reverse strings, i.e., "emaN" and "emaNtsriF"(column 4 in the table). The larger of two similarity indexes becomes the final similarity index (column 5 in the table). With this modification the number of false negatives is reduced, but at the same time the number of false positives, such as `NNM("Name","Time")=0.83` increases.

Bellflower creates a set of mapping elements $\mathcal{M}_n$ for every personal schema node $n \in N_T$ (T is a personal schema) by computing the value of `NNM(`n`,`n'`)` for every every repository schema node $n' \in N_R$ (R is a schema repository). Repository schema node n' becomes a member of $\mathcal{M}_n$ if it meets two criteria:

- the computed name similarity is larger than a predefined *name similarity threshold* $\delta_{element}$, i.e., $n' \in \mathcal{M}_n \Rightarrow$ `NNM`$(n, n') \geq \delta_{element}$. Sec. 5.5 discusses the selection of $\delta_{element}$ in the experiments.
- the repository schema node can belong to only one set of mapping elements; the one with the maximum name similarity, i.e., $n' \in \mathcal{M}_n \Rightarrow \nexists m \in N_T \bullet$ `NNM`$(m, n') >$`NNM`(n, n'). This restriction is introduced to simplify the implementation of Bellflower.

As a consequence of the second restriction, the sets of mapping elements $\mathcal{M}_n$ are disjunctive and smaller. This reduces the search space for the mappings combiner and eases its implementation. However, this restriction potentially leads to the loss of some correct schema mappings. As this thesis primarily studies the effects of clustering, and not the quality of schema matching, this risk is acceptable. More so knowing that the second restriction does not affect clustering. Clusterer, as implemented in Bellflower, does not distinguish between nodes in different $\mathcal{M}_n$ sets, and sees only the set $\mathcal{M}$, where $\mathcal{M} = \bigcup_{n \in T} \mathcal{M}_n$; this union is not affected by the second restriction.

In Bellflower, the efficiency of the element matcher is acquired by time measurement. Having in mind the simple implementation of the `NNM` function, its efficiency is used for illustration purposes and not as a solid indicator of the actual execution costs.

5.3.2 Mappings combiner

For a personal schema T and a repository schema R, the element matcher creates sets of mapping elements $\mathcal{M}_n$, $n \in N_T$, as explained in the previous section. Next, the mappings combiner generates a set of mapping schemas A_S^δ. A reminder: $a = T \mapsto \tau$ is a schema mapping, where the schema mapping $a \in A_S^\delta$ is

generated by selecting one mapping element $n' \in \mathcal{M}_n$ for each personal schema node $n \in N_T$. Mappings combiner also computes the value of the objective function $\Delta(a) \equiv \Delta(T, \tau)$ and adds the mapping schema a to A_S^δ if $\Delta(a) \geq \delta$.

In this section, we describe the implementation of the mappings combiner in Bellflower. In particular, we describe the *objective function* Δ and the *algorithm for generating schema mappings*.

Objective function

Bellflower's objective function uses two heuristics to compute the similarity of schemas. The first heuristic is the *average name similarity* (NS) based on the name similarity indexes computed by element matcher. The second heuristic (PL) takes into account the structural properties of the schema mapping; it compares the total path length, i.e., size of the minimum spanning tree, of the personal schema T with the total path length of the mapping schema a. Bellflower computes $\Delta(T, \tau)$ as follows.

$$\Delta_{NS}(T, \tau) = \frac{1}{|N_T|} \cdot \sum_{n \in N_T} NNM(n, n') \tag{5.1}$$

where $n \in N_T, n' \in N_\tau$ *and* $n \mapsto n'$

$$\Delta_{PL}(T, \tau) = 1 - \frac{|E_\tau| - |E_T|}{|E_T| \cdot K}, \tag{5.2}$$

where K *is a normalization constant*

Eq. 5.1 computes the average name similarity for a schema mapping. Eq. 5.2 computes the difference in path lengths of the personal schema and the mapping schema. The difference is normalized to $[0, 1]$ interval using the normalization constant K, the value of which is determined using other constraints in the system (e.g., the maximum length of a path). The two hints are combined using a weighted sum (Eq. 5.3) with a parameter α determining the relative importance of the two.

$$\Delta(T, \tau) = \alpha \cdot \Delta_{NS}(T, \tau) + (1 - \alpha) \cdot \Delta_{PL}(T, \tau) \tag{5.3}$$

This objective function, though simple, simulates the two most important types of heuristics used in schema matching systems. First, Δ_{NS} simulates heuristics based on local properties of elements, and second, Δ_{PL} simulates heuristics based on structural properties of schemas. Note that research in this thesis does not try to develop a new, or better, objective function; the objective function implemented in Bellflower probably has an inferior effectiveness when compared to that of the other schema matching system. Nevertheless, by simulating the two main kinds of heuristics, we believe that the Bellflower prototype has enough behavioral similar-

ity to other schema matching systems (see Sec. 2.2.2) to present a good platform for investigating the performance of clustered schema matching technique, and for making conclusions about its broader applicability.

Generating schema mappings

The simplest implementation of the mappings combiner enumerates all schema mappings in the search space and for each computes $\Delta(T, \tau)$. However, the total number of schema mappings is polynomial in respect to the size of schema repository, i.e., $O(|\mathcal{M}_n|^{|N_T|})$ (see Sec. 2.2.2) and even for moderate schema sizes enumeration becomes prohibitively inefficient. The mappings combiner must be implemented using an optimized algorithm. In our research there exist two more reasons for a more advanced implementation of mappings combiner.

First, our validation approach relies on the comparison of the non-clustered and the clustered schema matching approaches. Building the non-clustered schema matching system in the most naive way would not be a base for a fair comparison. Second, in chapter 3 we claim that clustered schema matching is orthogonal to other optimization techniques. By using an optimization technique in the mappings combiner we can validate this claim.

For reasons stated above, Bellflower uses the *Branch and Bound* ($B\&B$) optimization algorithm in the mappings combiner: a schema matching problem is a constraint optimization problem, as discussed in chapter 2, and the $B\&B$ algorithm is commonly used to solve such problems in an efficient way [5].

Before explaining the $B\&B$ algorithm, we first introduce the notion of partial schema mapping $\tilde{a}$. A partial schema mapping is a schema mapping in which not all personal schema elements have been assigned a corresponding mapping element. A more precise definition of partial schema mapping is given in Sec. 2.3.3: a partial mapping is the same as partial valuation used in constraint optimization problems.

The $B\&B$ ([49], pg. 141) algorithm is a recursive algorithm and in each recursive step, it gradually builds a partial mapping $\tilde{a}$ to a full mapping a. At each stage, it computes the potential quality of the partial mapping $\tilde{a}$ by means of a *bounding function* Δ_{max}. The bounding function $\Delta_{max}(\tilde{a})$ computes the best possible similarity index (i.e., best possible value of $\Delta(a)$) a partial mapping $\tilde{a}$ can have when completed to a full mapping a ($\tilde{a} \subseteq a$), i.e.

$$\forall \tilde{a},\ \forall a \bullet\ \tilde{a} \subseteq a \Rightarrow \Delta(a) \leq \Delta_{max}(\tilde{a}). \tag{5.4}$$

If the bounding function computes the value which is above a certain threshold, the partial mapping is declared as "promising" and is further extended. Otherwise, the partial mapping is discarded, and in that way time is saved which would otherwise be spent on building many schema mappings which would rank below the threshold. Savings are large because a partial schema mapping can be extended to a full mapping in an exponential number of ways. The sooner the partial mapping

is discarded, the larger the savings.

The $B\&B$ algorithm finds only one, i.e., the best, solution in the whole search space. The mappings combiner, on the other hand, has to find all the solutions within the search space which have a the similarity index above threshold δ. Therefore, Bellflower implements a modified version of the $B\&B$ algorithm shown in Alg. 2: the first recursion of the algorithm is invoked on an empty partial mapping $\tilde{a} = \{\}$. Each $B\&B$ recursion starts by checking if a partial mapping $\tilde{a}$ grew to a full mapping (line 1). If not, one of the personal schema nodes, which is not yet mapped in $\tilde{a}$ is chosen (i.e., n in line 4); it is for this node that the mapping node n' will be selected next. Mapping elements n' are taken from the corresponding set of the mapping elements $\mathcal{M}_n$ (line 5). The partial schema mapping $\tilde{a}$ is extended with a new element mapping $n \mapsto n'$, i.e., $\tilde{a} \cup (n \mapsto n')$, and the bounding function $\Delta_{Max}(\tilde{a} \cup (n \mapsto n'))$ is computed for the extended mapping (line 6). If the computed value is below the threshold δ the partial schema mapping is discarded. If, however, the value is larger than δ, the partial mapping is propagated to a new recursive cycle (line 7).

Algorithm 2 B&B($\tilde{a}$).

Require: $\forall n \in N_T \bullet \mathcal{M}_n$ is available
Initialize: $\tilde{a} = \{\},\ A = \{\}$
1: **if** $\tilde{a}$ is a full mapping **then**
2: add a to A
3: **else**
4: choose node $n \in N_T$ which is not yet mapped in $\tilde{a}$
5: **for all** $n' \in \mathcal{M}_n$ **do**
6: **if** $\Delta_{max}(\tilde{a} \cup (n \mapsto n')) \geq \delta$ **then**
7: B&B($\tilde{a} \cup (n \mapsto n')$)
8: **end if**
9: **end for**
10: **end if**

Two more issues need to be addressed to come to a full implementation of the presented $B\&B$ algorithm: implementation of the bounding function Δ_{max}(line 6), and the node selection algorithm (line 4).

The bounding function Δ_{max} computes the best possible similarity index a partial mapping can reach as follows: it uses the objective function Δ given in Eq. 5.3 with the following adaptations. The nodes and the paths which are still not known in the partial mapping are given the best case scores. For nodes, each unknown name similarity is replaced with 1.0 which is the best case similarity. For paths, the length of each unknown path is set to 1, which is the length of each edge in personal schema. When built in this way, the bounding function always satisfies Eq. 5.4.

The node selection algorithm, commonly called *variable ordering algorithm* in constraint programming, is known to have a significant impact on the performance of the $B\&B$ algorithm. However, deep investigation of this aspect is out of the line of the research in this thesis. Therefore, Bellflower implements a static variable ordering algorithm which selects personal schema nodes using breadth-first traversal.

Timers and counters in the mappings combiner

To measure the performance of the mappings combiner Bellflower uses both timers and counters. A timer monitors the time needed to run the complete $B\&B$ algorithm. The measured time is reduced with the amount of time needed to write the results (line 2). To get more reliable efficiency measurements, a counter is used to count the number of times the computation of the bounding function Δ_{max} is invoked. The Δ_{max} function is the most expensive operation in the $B\&B$ algorithm and the efficiency of the $B\&B$ algorithm is largely proportional to the number of times Δ_{max} is invoked. This counter can also be interpreted as the number of partial schema mappings which are generated and tested during the $B\&B$ execution.

The worst case complexity of the $B\&B$ algorithm is the same as the complexity of naive full enumeration of the search space. However, in practice, as our experiments also confirm, the performance of the algorithm is much better. By implementing the mappings combiner as a $B\&B$ algorithm, we brought Bellflower closer to what would be an optimized schema matcher. As such, Bellflower is a realistic and challenging environment for the clustered schema matching technique which is being validated.

5.3.3 Clusterer

In Bellflower, the clusterer implements the k-means clustering algorithm. The reasons for using the k-means clustering algorithm, and the algorithm itself, are first introduced in Sec. 3.4. Here, we describe the implementation details of the components of the k-means clustering algorithm. For easier reference, we repeat the k-means algorithm in Alg. 3.

In this section, we introduce the implementation of the following:

- centroid: definition and computation (used throughout the algorithm)
- initialization of centroids (line 1)
- distance measure (line 5)
- reclustering (line 10)
- convergence criteria (line 11)

Algorithm 3 K-means clustering algorithm **(repeated Alg. 1)**.

```
1: initialize centroids
2: repeat
3:    for each mapping element do
4:       for each centroid do
5:          compute distance(mapping element,centroid)
6:       end for
7:       assign mapping element to nearest centroid
8:    end for
9:    compute new centroids for all clusters
10:   perform reclustering
11: until convergence criterion is met
```

Definition of centroid

A cluster is set of elements. To make k-means clustering algorithm efficient, a distance between an element and a cluster is not computed by accessing all the clusters' elements. Instead, the whole cluster is represented with a single *centroid* which is used for distance computation.

In Bellflower, we have considered three different ways to define a centroid. Fig. 5.8 illustrates the three kinds of centroids on a simple cluster comprised of 5 mapping elements. Note, mapping elements in the figure are shown as painted circles, other schema elements are white circles. In all three cases, the centroid is selected among the existing elements in the repository schema, as opposed to being a synthetic mathematical entity. A special name is used if a centroid is selected from the set of mapping elements. Such centroid is called a medoid [26].

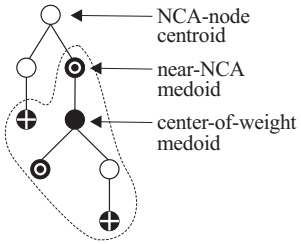


Figure 5.8 *Three kinds of centroids.*

- The *NCA-node* centroid is the simplest centroid kind. The NCA-node centroid is the nearest common ancestor of all nodes in the cluster (see Fig. 5.8). This centroid, does not perform well: with a spreading cluster the centroids quickly converges toward the root of the tree. When having more than one cluster in a tree, the NCA-node centroids of these clusters all converge to

the root node which is in contrast to the main purpose of centroid: a clear and distinctive representation of a cluster.

- The *near-NCA medoid*, is the clustered element which is closest to the NCA node. In initial experiments, near-NCA medoid showed better behavior than the NCA-node. However, the way in which near-NCA medoid is selected showed deficiencies. When there are several equally good candidate elements the leftmost element is selected to be the centroid, which is not always the best choice and leads to less effective clustering.
- The *center-of-weight medoid*, selects a centroid by computing which of the clustered elements is closest to all the other members of the cluster. Of the three proposed centroid kinds, this showed the best results in initial experiments. Bellflower implements the computation of the center-of-weight medoid in a naive way: it uses a nested loop to discover the center of weight. The efficiency of the clustering algorithm would benefit from the application of an incremental algorithm for centroid computation [54].

Initialization of centroids

In clustered schema matching, it is very hard to determine beforehand the optimal number of clusters. The number of regions in the repository, which have the potential to deliver good schema mappings, is different for each personal schema. For this reason, we let the initialization create a large number of centroids, and we then rely on the reclustering techniques (described below), to reduce the reduce the number of clusters to a desirable level.

Bellflower implements two kinds of initialization approaches. The first approach *selects initial centroids*, the second *creates initial clusters*.

- *MIN-init* is Bellflower's algorithm which selects initial centroids. The heuristics and initialization approach of this algorithm are as follows: to form a full mapping, the mappings combiner needs one mapping element for each personal schema node. The mapping elements in the smallest set of mapping elements $\mathcal{M}_{min}$ are in short-supply, but without them a schema mapping cannot be formed. The use of these elements as initial centroids increases the chance that each cluster contains at least one of these rare elements. The MIN-init algorithm also features *MIN-init-threshold*. This threshold defines that not all mapping elements in $\mathcal{M}_{min}$ are to become centroids, but only these with a *name similarity index* larger than MIN-init-threshold.
  Experiments with similar initialization approaches, such as using the $\mathcal{M}_{max}$ instead $\mathcal{M}_{min}$ to seed centroids, resulted in worse clustering effectiveness.
- *NCA-init* is Bellflower's algorithm which creates initial clusters. NCA-init is a single-pass clustering algorithm which forms clusters in a single preorder traversal of the repository schemas. Each mapping element is added to the closest cluster. If the distance between the node and the *NCA-node centroid*

(see Fig. 5.8) of the cluster is larger than some given *NCA-init-threshold*, the node becomes the first member of a new cluster. In experiments, NCA-init-threshold takes values 2,3, and 4. More details on the behavior of these two initialization approaches are presented in Sec. 5.6.1.

Distance measure

- The distance function (line 5 in Alg. 3) as implemented in Bellflower computes the actual tree distance between the mapping element and the centroid. To make the computation of distance efficient, Bellflower uses a node labeling technique similar to those described in Sec. 3.2.

Sec. 5.8 discusses the importance of designing the distance measure in accordance with the objective function used in the mappings combiner.

Reclustering

Bellflower implements two reclustering algorithms: *cluster joining* and *cluster removal*.

- The *join* algorithm discovers all cluster pairs for which centroids are closer to each other than some predefined threshold. Such clusters are joined into a single cluster.
- The *remove* algorithm disbands clusters which have too few elements. A threshold defines the minimum number of elements a cluster must have to endure. The elements which belonged to a disbanded cluster are free to join other clusters in the vicinity.

Convergence criteria

A Convergence criterion determines the end of the iterations of the k-means algorithm. Bellflower implements several:

- *Fixed* number of iterations. This criterion specifies that iteration stops after a fixed number of iterations.
- Percentage of *nodes*. This criterion specifies that iteration stops when the number of nodes that moved from one cluster to another during an iteration, drops below a given percentage of all the nodes being clustered.
- Percentage of *clusters*. This criterion specifies that iteration stops when the reduction in the number of clusters is less than a given percentage of existing clusters. Note that this criterion can only be used in combination with reclustering, as only reclustering reduces the number of clusters.

The presented criteria can be used in various combinations. The design of the clustering criteria will be reexamined in more detail in Sec. 7.

5.4 Experimental repositories

Validation of the clustered schema matching technique requires experimental XML schema repositories. The repositories have to confront the clustered schema matching technique with data structures and challenges that would normally be encountered in a real-world schema matching system. In Bellflower, such repositories were built by random selection of XML schemas available on the Internet. In this section, the approach for building the repositories is described, and the main properties of these repositories are discussed.

5.4.1 Harvesting the Internet to build schema repositories

In Bellflower, the repository building process has five steps shown in Fig. 5.9.

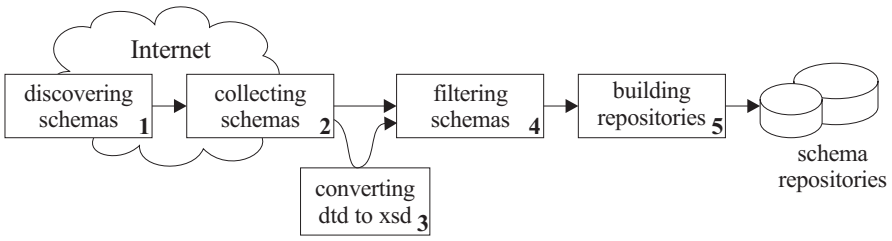


Figure 5.9 *Repository building in the Bellflower.*

Step 1: The *Google* search engine is used to discover URI-s of DTD-s and XML schemas. Google's `filetype` operator is crucial in this step. For example, a query "`+schema +name -group filetype:xsd`" asks Google to discover `*.xsd` files containing the words `schema` and `name` but not the word `group`. In this query, the used keywords are the reserved words of the XML schema language; these keywords do not determine the semantics of schemas to which they belong. Consequently, schemas discovered in this way have high semantic heterogeneity. With this approach, approximately 5000 DTD-s and XML schemas were discovered.

Step 2: Schemas are downloaded[3] from the Internet. Out of the 5000 discovered schemas, 65 percent was successfully downloaded; the other 35 percent was unavailable due to broken URI-s or servers being off-line. A total of 1815 xsd schemas originating from 835 servers, and 1887 DTD-s originating from 1461 website, were downloaded.

Step 3: Bellflower parses only schemas written in the XML schema language. For that reason, DTD schemas are converted into XML schemas[4].

[3]by means of *Internet Spider Downloader* available at `www.tensons.com`
[4]DTD to XML schema conversion tool available at `www.w3.org/2000/04/schema_hack`

Step 4: Some schemas are filtered out: a schema is discarded if Bellflower's schema parser reports a parsing error or if a schema part is not available due to broken link in an `include` statement. Also, schemas with a recursive structure cannot be represented using the tree data model and must be discarded. The filtering step reduced the corpus of schemas to 830 XML schemas and 943 DTD-s.

Step 5: This step builds the repositories. Bellflower first parses each schema and resolves all internal references. For example, each complex type definition, referenced from multiple places in the schema, is replicated and placed where referenced. This ensures the each schema becomes a tree. Schemas are then put together in a Bellflower-specific data format. All the collected schemas together contain 178252 elements distributed over 3889 trees (note, one schema can produce several trees, one tree for each root element declared in the schema). A repository of such size, proved to be too big for our experimental system and, instead of one big, four smaller repositories were built by random selection of schemas from the corpus.

5.4.2 Properties of the experimental repositories

The main properties of the four experimental repositories A, B, C, and D, are given in Tab. 5.5

Table 5.5 *Main properties of the repositories.*

name	schemas (i.e., trees)	elements	distinct element names	elements per schema avg. / dev.
A	51	2505	1040	49.1 / 89.8
B	76	4676	1467	61.5 / 159.2
C	151	10261	2728	67.9 / 219.6
D	262	9759	3331	37.2 / 152.0

There exists a large deviation in the number of elements in trees (column 5 in the table). For example, the largest tree in the C repository has 2335, and the smallest one only 3 elements. To better observe the sizes of trees in the repositories, Fig. 5.10 shows the tree size distribution. The x-axis depicts the size of a tree and the y-axis shows the number of trees of that size. For example, there are 21 trees in repository D with size $n = 6$, i.e., trees with more than $2^{6-1} = 32$ nodes, but less than $2^6 = 64$ markup nodes. Fig. 5.10 shows that there exists no distinctive pattern in tree size distribution. This supports the assumption that experimental repositories are non-biased samples of the real-world schemas.

Next, we observe the structural diversity in the repositories. Fig. 5.11 shows the

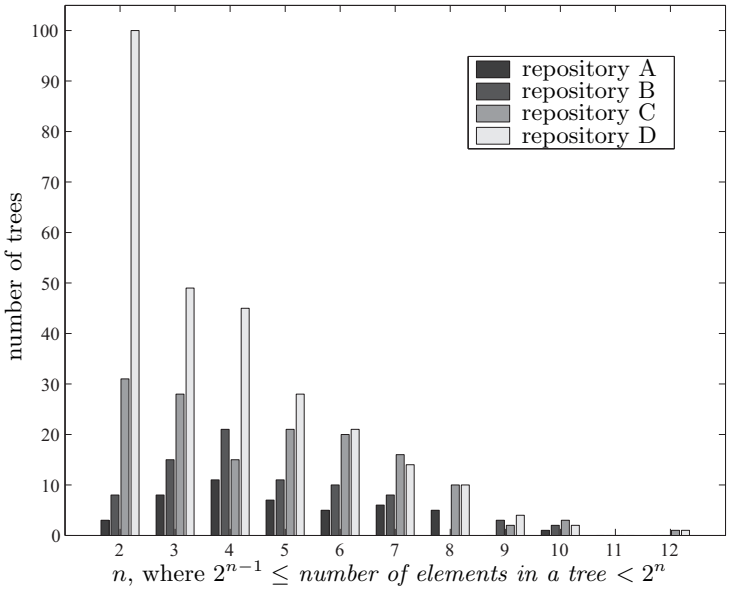


Figure 5.10 *Tree size distribution in the repositories.*

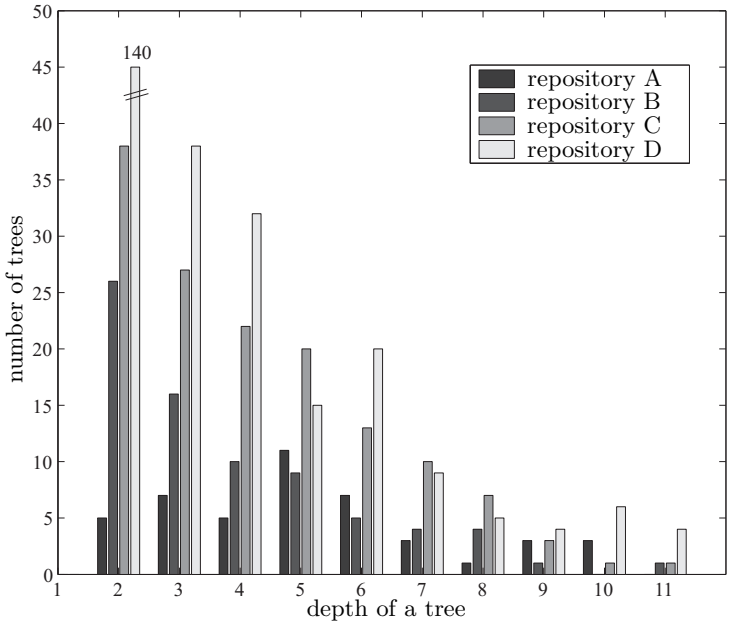


Figure 5.11 *Depths of trees in the repositories.*

number of trees (y-axis) that are of certain depth (x-axis). Trees in our repositories have depths ranging from 1 to 11. In the analysis of the structure of Web databases, Chang et al. [13] report that depths of Web database schemas in their sample range from 1 to 6. In their study, schema depths show a distribution similar to ours. Chang reports that 90 percent of schemas have depths not more than 4. In our repositories this percentage is observed at greater depths; for repositories A, B, C, and D, depths that incorporate 90 percent of all schemas are 9, 8, 8, and 7.

We conclude that the four experimental repositories comprise a syntactically and semantically heterogeneous and unbiased sample of Internet schemas. As such, these repositories challenge the clustered schema matching technique in ways sufficiently resembling the real-world scenarios.

This thesis reports only on experiments performed with the B and D repositories. However, the research also used experiments with other repositories. In all cases, clustered schema matching showed similar behavior, both in terms of acquired results and observed problems. Conclusions drawn in this thesis are thus based on both the reported and the non-reported experiments.

5.5 The influence of element matching

This section discusses the influence that element matching, i.e., `NNM` function (see Sec. 5.3.1), has on clustered schema matching technique. In particular, it addresses the following three questions.

- How does element matching affect clustering?
- How does element matching affect mappings combiner?
- How to configure the element matcher in the experiments?

Bellflower's element matcher is a name similarity function `NNM`. The function is parameterized with a name similarity threshold $\delta_{element}$ which indirectly controls the number mapping elements. The higher the threshold, the smaller the number of mapping elements, and vice versa. We here analyze how the number of mapping elements impacts other stages of clustered schema matching.

Element matching and clustering

Fig. 5.12 shows the sets of mapping elements for two different $\delta_{element}$ thresholds. Fig. 5.12a illustrates the case with a low $\delta_{element}$ which results in a large set of mapping elements densely populating the repository. In such a case, it is harder to build good clusters: mapping elements are close to each other, and the chance of cutting-off good schema mappings by clustering is high. In such a scenario, however, the clustering has an important role: if successful, clustering provides an important efficiency improvement (see Sec. 3.3).

Fig. 5.12b illustrates the case with a high $\delta_{element}$ threshold. The number of mapping elements is smaller and these elements are sparsely distributed over the

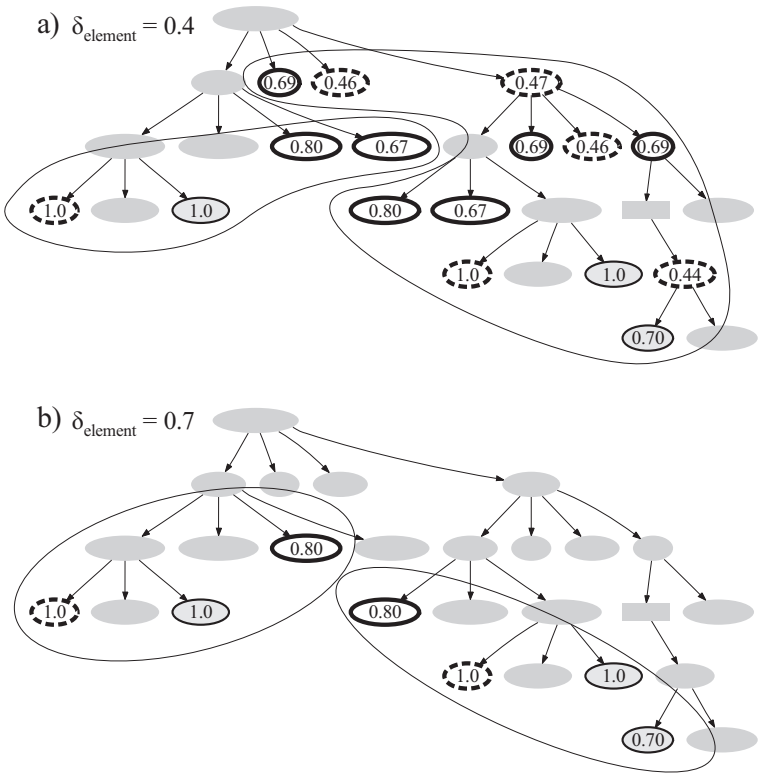


Figure 5.12 *Mapping elements and clusters for two different name similarity thresholds* $\delta_{element}$.

repository. In this case, it is easier for a clustering algorithm to correctly detect the borders of groups of mapping elements. However, the benefits of clustering are questionable: if necessary a simpler algorithm could perhaps perform the same partitioning task at a lower cost.

It is to be expected that, in practice, element matchers generate distributions of mapping elements with both dense and sparse areas in the repository. A clustered schema matching system should be able to handle both.

Since a low $\delta_{element}$ represents the hard case and also the case in which clustering is most beneficial, in experiments we choose low values for the name similarity threshold $\delta_{element}$.

Element matching and mappings combiner

There exists a polynomial dependency between the number of mapping elements and the size of the search space (see Sec. 2.2.2): the more mapping elements in

the repository, the larger the search space for the mappings combiner. Hence, the size of the search space can be indirectly controlled with the name similarity threshold $\delta_{element}$. Fig. 5.13 shows how the size of the search space changes, in Exp. 1, depending on threshold $\delta_{element}$. If $\delta_{element}$ is set to a very small value, the result could be a prohibitively large search space.

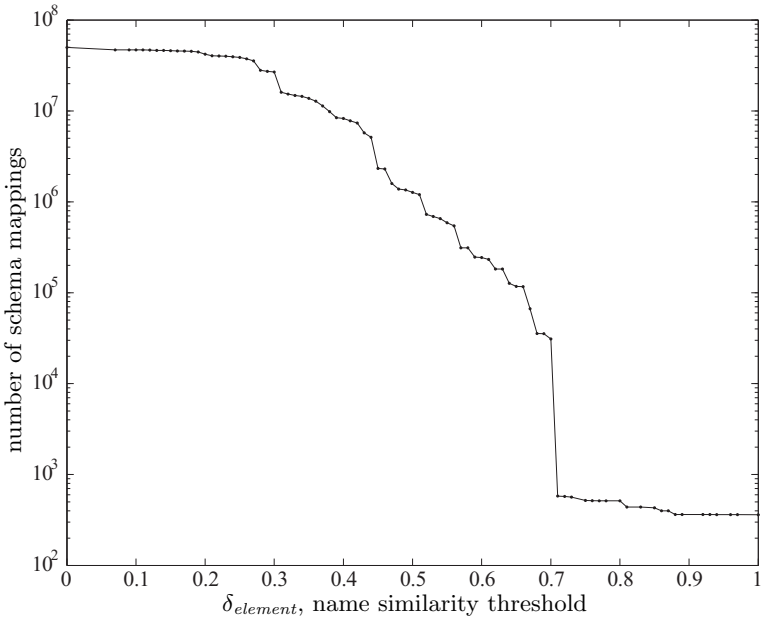


Figure 5.13 *Size of the search space depends on the name similarity threshold* $\delta_{element}$.

Setting $\delta_{element}$ in experiments

In the experiments, we have adopted the following *rule of thumb*. The value of the $\delta_{element}$ is set to generate a search space close to 10^7 mapping schemas. This ensures that the execution time of non-clustered schema matching is measured in minutes. In some cases, a trial-and-error approach was used to fine-tune the threshold and to: (a) acquire a large set of mapping elements which is needed to observe the behavior of the clustering algorithm, and (b) keep the time needed to execute the non-clustered technique reasonable. In the experiments presented in this chapter $\delta_{element} = 0.4$.

5.6 Setting up the clustering algorithm

This section addresses the problem of setting the parameters of the k-means clustering algorithm in the clustered schema matching technique. In particular the following components of the k-means clustering algorithm are discussed:

- initialization,
- centroid and distance measure,
- dynamic reclustering, and
- convergence criteria.

Each of these components makes an impact on:

Ⓒ the efficiency of clustering,

Ⓜ the efficiency of mappings combining, and

Ⓔ the effectiveness of the clustered schema matching technique.

Symbols Ⓒ, Ⓜ, and Ⓔ are used in this chapter for quick reference to the three aspects of performance: instead of writing "this improves the efficiency of clustering" we write "this improves Ⓒ."

5.6.1 Initialization

The k-means clustering algorithm starts with the initialization of centroids. Bellflower implements the *MIN-init* and the *NCA-init* initialization algorithms (see Sec. 5.3.3). In this section we experimentally observe the properties of initial clusters (note the difference between initial centroids and initial clusters). The MIN-init algorithm produces initial centroids; to form initial clusters one iteration of the k-means clustering must be executed. The NCA-init algorithm directly generates initial clusters.

The number and the size of the initial clusters

The number and the size of the initial clusters are of interest because of their potential impact on Ⓒ, Ⓜ, and Ⓔ.

The number of initial clusters affects Ⓒ. The complexity of the k-means clustering algorithm is $O(k{\cdot}i{\cdot}|\mathcal{M}|)$ where k is the number of clusters (see Sec. 3.4.2). Consequently, a large number of initial clusters degrades efficiency.

The size of initial clusters can affect Ⓜ. If the initial clusters are too big, the resulting clusters will be big as well. This is because the implemented clustering algorithm only makes clusters larger. With big clusters, the efficiency improvements, which are based on the search space partitioning, will not be significant. Therefore, smaller initial clusters are preferable.

The last two observations contradict each other. Initial clusters clusters cannot be small and few at the same time. The number and the size of the initial clusters are reciprocal; the more clusters, the smaller the clusters, and vice versa. A trade-off decision must be made.

The number and the size of initial clusters also influences Ⓔ. We have observed that large initial clusters tend to be stable, i.e., immutable, throughout the iterative part of the k-means algorithm. In such a case, the initialization algorithm takes full responsibility for Ⓔ. If, however, the initial clusters are small and in large numbers, the agglomeration performed in the iterative part of the k-means process becomes dominant in determining the composition of final clusters, and consequently the resulting Ⓔ; the initialization has less impact on Ⓔ.

The number and the size of initial clusters can be controlled. The NCA-init uses initialization NCA-init-threshold to control the size of initial clusters, and the MIN-init uses the MIN-init-threshold to controls the number of initial clusters. The following experiments analyze the behavior of the two initialization approaches.

Experiment 2: Initialization based on the NCA-init algorithm

In this experiment (Tab. 5.6), only the initialization part of the k-means algorithm is executed. The NCA-init initialization algorithm is run with NCA-init-thresholds 2,3, and 4.

Table 5.6 *Experiment table for Exp. 2.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4
Clusterer			
initialization		reclustering	convergence
NCA-init 2,3,4		not used	fixed 0

In this experiment, the element matcher discovered 2077, 1701, and 973 mapping elements for *name*, *email*, and *address* personal schema elements; a total of 4751 mapping elements is split among initial clusters. Clusters table Tab. 5.7 shows the properties of initial clusters, and the properties of tree-clusters.

Table 5.7 *Clusters table for Exp. 2.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
NCA-init 2	288 [938]	2762 [4751]	9.6 / 6.1	irrelevant
NCA-init 3	232 [557]	3412 [4751]	14.7 / 13.1	irrelevant
NCA-init 4	179 [414]	3844 [4751]	21.5 / 21.8	irrelevant
tree-clusters	95 [228]	4326 [4751]	45.5 / 86.4	irrelevant

The NCA-init 2 initialization creates a very large number of clusters, i.e., 938.

The average size of these cluster is $\frac{4751}{938} = 5.06$ (note, the average size of cluster given in the table, i.e., 9.6, refers to useful clusters only). To better observe the sizes of the created clusters we introduce the *cluster size distribution graph*.

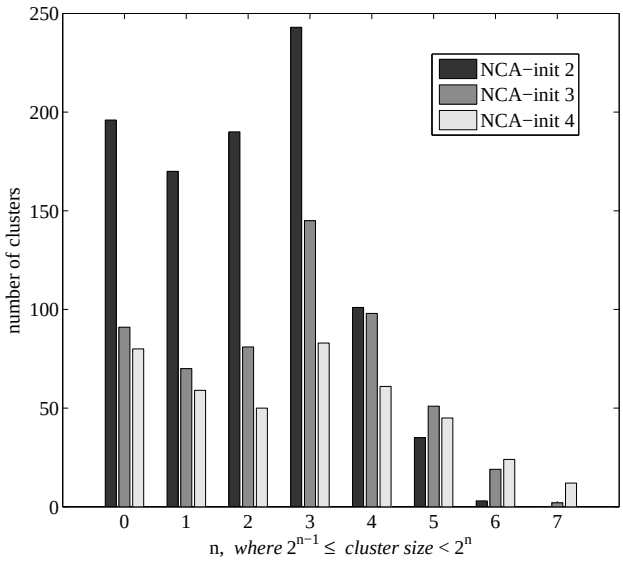


Figure 5.14 *Cluster size distribution graph for NCA-init; Exp. 2.*

Fig. 5.14 is the cluster size distribution graph for the current experiment. The x-axis depicts eight size ranges $n = 0..7$. Range $n = 0$ represents clusters that have only 1 node, $n = 1$ represents clusters with 2 nodes, $n = 3$ clusters with 3 to 4, $n = 7$ clusters with 65 to 128 nodes. The y-axis depicts the number of clusters within a given size range. For example, for the NCA-init 3 the number of initial clusters with sizes between 5 and 8 ($n = 3$) is 145.

Fig. 5.14 and Tab. 5.7 illustrate that the number and the size of the initial clusters depend on the NCA-init-threshold. The table shows that for thresholds 2,3, and 4, the number of initial clusters decreases from 938, to 557, to 414. Even though the larger thresholds lead to the formation of larger clusters, the figure shows that many very small clusters (e.g., n=0,1,2) remain even for NCA-init 4. This issue will be addressed in more details shortly.

Experiment 3: Initialization based on the MIN-init algorithm

The initialization of clusters is, in this experiment (see Tab. 5.8), performed using the MIN-init algorithm with three different MIN-init-thresholds 0.41, 0.49, and 0.50. Theses thresholds were tuned to produce a number of initial clusters comparable to the number of initial clusters created by the NCA-init 2,3, and 4 in the previous experiment (see Exp. 2).

Table 5.8 *Clusters table for Exp. 3.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4
Clusterer			
initialization		reclustering	convergence
MIN-init 0.41,0.49,0.50		not used	fixed 1

Note in Tab. 5.8, the clusterer has a convergence criterion set to "fixed 1." This is because the MIN-init initialization only selects initial centroids, and one k-means iteration is used to create initial clusters; in the iteration, every mapping element joins the nearest centroid. Tab. 5.9 shows the properties of the resulting initial clusters.

Table 5.9 *Cluster table for the Exp. 3.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
MIN-init 0.41	351 [912]	3508 [4414]	10.0 / 9.3	irrelevant
MIN-init 0.49	283 [579]	3729 [4250]	13.2 / 16.0	irrelevant
MIN-init 0.50	278 [447]	3884 [4226]	14.0 / 16.4	irrelevant
tree-clusters	95 [228]	4326 [4751]	45.5 / 86.4	irrelevant

Tab. 5.9 shows that not all 4751 mapping elements are included in clusters; with MIN-init 0.50, only 4226 elements ended up in an initial cluster. This is due to the fact that MIN-init set no initial centroids in some schemas in the repository. Consequently, mapping elements in these schemas had no centroid to join and remain unclustered. In the Bellflower experiments, this problem is ignored with the expectation that it does not significantly degrade the effectiveness. If a schema in the repository does not have any MIN-init 0.50 centroids (reminder, MIN-init 0.50 centroids are elements from the $\mathcal{M}_{min}$ set of mapping elements which have the name similarity index ≥ 0.50) this leads to one of the following situations. First, it can be the case that such repository schema cannot deliver any schema mappings, because it has no $\mathcal{M}_{min}$ elements which are necessary to form a schema mapping. Second, if the schema contains elements from $\mathcal{M}_{min}$ these elements have name similarity lower than 0.50. Consequently, schema mappings based on this elements will probably have low similarity indexes. The problem can

be solved by modifying the initialization technique to ignore the name similarity threshold in trees where all $\mathcal{M}_{min}$ elements have name similarity index below 0.50.

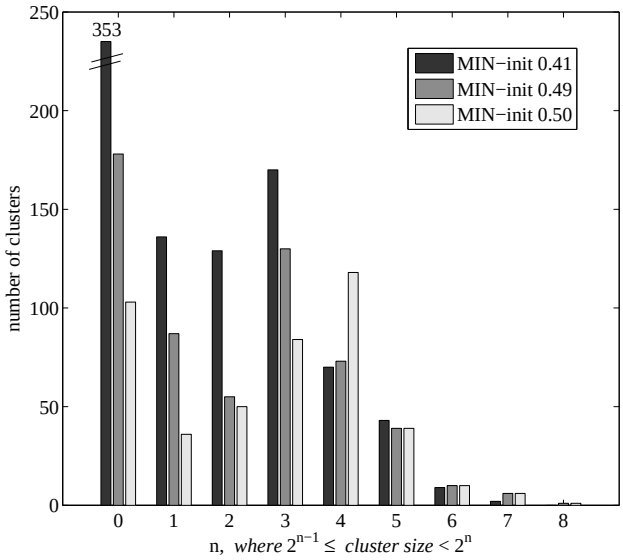


Figure 5.15 *Cluster size distribution graph for MIN-init; Exp. 3.*

Fig. 5.15 shows the cluster size distribution graph for the MIN-init initialization. The MIN-init initialization, similar to NCA-init, also creates large number of very small clusters. Also, both initialization techniques create few very large clusters (e.g., n=8 in Fig. 5.15) clusters. We now analyze the causes and the consequences of these two extreme kinds of initial clusters.

Tiny and huge initial clusters

The *tiny* initial clusters include a very small number of mapping elements. We consider clusters with up to four elements as tiny (i.e., clusters in $n = 0$, $n = 1$, and $n = 2$ size ranges in size distribution graphs). The *huge* initial clusters are clusters with hundreds or even thousands of elements. In experiments Exp. 2 and Exp. 3 largest clusters are not as large as the ones observed in few other unpublished experiments. For explanation purposes, we shall treat clusters with more than 128 elements ($n = 8+$) as huge.

As already discussed at the beginning of this section, the number and the size of initial clusters has an impact on Ⓒ and Ⓜ. Tiny clusters have a negative effect on Ⓒ, and huge clusters degrade Ⓜ.

The tiny and the huge initial clusters occur due to the following reasons:

NCA-init tiny clusters: Fig. 5.16a illustrates the clustering performed with NCA-init 2. Gray nodes are mapping elements and closed lines depict clusters. A tiny cluster occurs when a condensed group of mapping elements, such as the 4 elements on the right of the schema, spreads over the a depth slightly larger than the NCA-init-threshold. Due to the simplicity of NCA-init algorithm, the 4 mapping elements are split into two clusters: the lowest mapping element is 3 edges away from the NCA node of the other three mapping elements, which is too far according to the NCA-init-threshold. Another reason for the occurrence of tiny clusters are groups of isolated mapping elements, such as the two elements in the left of schema in Fig. 5.16.

NCA-init huge clusters: Huge clusters are created in trees where mapping elements are distributed over wide and shallow areas, as in Fig. 5.16b. With larger NCA-init-thresholds chances are higher that huge clusters will be formed.

MIN-init tiny clusters: The MIN-init algorithm creates tiny clusters in the areas with high density of initial centroids. These centroids compete to attract the same mapping elements. It happens that some centroids fail to attracting any nodes, as shown in Fig. 5.16c (black nodes are the initial centroids). These nodes form tiny, single node, clusters.

MIN-init huge clusters: Huge clusters appear in large populations of mapping elements in which only one (or few) initial centroids exists, as shown in Fig. 5.16d. In such cases, one centroid gathers all mapping elements in a single cluster.

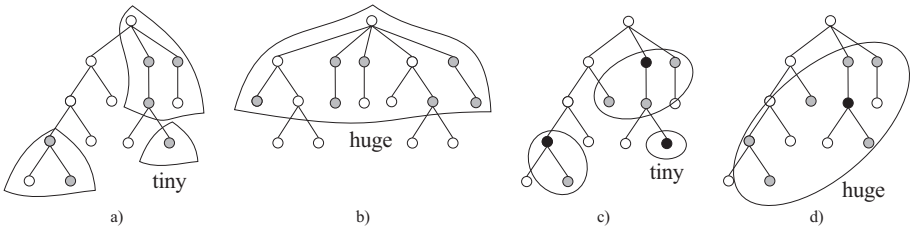


Figure 5.16 *Tiny and huge clusters. a) NCA-init tiny, b) NCA-init huge, c) MIN-init tiny, d) MIN-init huge.*

Solving the problem of tiny and huge initial clusters

The problem of tiny clusters is solved in Bellflower by means of reclustering (see Sec. 5.3.3). Reclustering joins or removes tiny clusters during the clustering iterations. Sec. 5.6.3 validates the effectiveness of this approach. There also exists a positive side to having large number of small initial clusters. The k-means clustering algorithm is known to be very sensitive to the initial choice of centroids. This however, is primarily true for applications in which the number of clusters is specified beforehand, and in which the k-means algorithm uses no reclustering.

Our approach which combines large number of initial centroids and reclustering, is less sensitive to initialization: when used with reclustering, initialization has less responsibility in determining the final clusters.

The problem of huge clusters can be solved, for example, by splitting the cluster into several smaller clusters or by introducing new centroids within the cluster. In Bellflower experiments, huge clusters rarely occur, and are dealt with manually if necessary. Bellflower does not implement procedures for automatic solving of the huge clusters.

5.6.2 Centroid

This section discusses the impact of the centroid kind on the performance of clustered schema matching; in particular how it affects Ⓒ and Ⓔ.

Centroid kind and the efficiency of clustering
Centroid kind impacts Ⓒ *directly* and *indirectly*. The *direct* impact comes through the complexity of the algorithm for computing centroids. Computations of centroids are frequent in the k-means clustering algorithm. Some versions of the algorithm compute the centroid each time an element moves from one cluster to another. Bellflower's version of the k-means algorithm computes centroids less frequent: at the end of each iterative cycle. For example, in Exp. 1, centroids are computed 1113 times (159 clusters $\times$ 7 iterations). Depending on the kind of centroid, each computation can be more or less expensive. In most experiments, we use center-of-weight centroids. The computation of center-of-weight centroid relies on distances between elements in the cluster, hence frequently executes the distance function. In Exp. 1, the distance was computed $4.21 \cdot 10^6$ times. To improve efficiency of distance computation, Bellflower uses node labeling techniques. There exist more room for further improvements, e.g., by means of incremental centroid computation or through caching.

The indirect impact is due to clustering stability. Clustering stability is used as a quality measure for clustering algorithms [69], and can be observed by varying different aspects of the clustering algorithm, such as the order in which elements are clustered, the distance measure, or, as it is the case in this section, the kind of centroid. Clustering is considered stable if it delivers similar clusters while varying one parameters.

In this section, we observe one aspect of clustering stability: the stability of clusters in respect to the kind of centroid. We say that if the addition or the removal of an element from the cluster seriously displace the centroid, the cluster becomes unstable. Unstable clusters experience larger changes in their membership between two iterations and need more iterations to meet the convergence criteria; this degrades Ⓒ. The following experiment shows the stability of clusters for two centroid kinds implemented in the Bellflower.

Experiment 4: Stability of clusters with different centroid kinds

In this experiment (see Tab. 4) two different kinds of centroids are used for the same clustering task: *near-NCA* and *center-of-weight* centroids. The goal of the experiment is to observe how the centroid kind influences the number of k-means iterations. The convergence criteria "0% nodes" means *total stability*, i.e., iterations end when no elements move from one cluster to another. The results of the experiment are given in the *iterations table* Tab. 5.11. In the table, each column

Table 5.10 *Experiment table for Exp. 4.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4

Clusterer		
initialization	reclustering	convergence
MIN-init 0.49 with a) near-NCA centroid b) center-of-weight centroid	not used	0% nodes

represents one iteration, and each row represents one kind of centroid. Numbers in cells have the following meanings. The upper number is the number of mapping elements that moved from one cluster to another during the iteration. For example, in iteration 2, when near-NCA centroid is used 775 elements switched clusters and when center-of-wight centroid is used 232 elements switched clusters. The number in square brackets denotes the number of clusters formed at the end of the iteration. In this experiment, the number is 579 and stays constant because the clustering algorithm does not use reclustering. In other experiments, this number will change.

Table 5.11 *Iterations table* ($\frac{\#\ moved\ elements}{[\#\ formed\ clusters]}$) *for Exp 4.*

iteration → centroid kind ↓	1	2	3	4
near-NCA	4250 [579]	775 [579]	221 [579]	54 [579]
center-of-weight	4250 [579]	232 [579]	16 [579]	0 [579]

The table shows that with the center-of-weight centroid, clusters reach stability faster (in 3 instead of 4 iterations); consequently, the clustering with the near-NCA centroid is less efficient. This confirms the existence of indirect impact of centroid kind on clustering efficiency. In this particular case, the reason for the instability of near-NCA clusters was already mentioned in Sec. 5.3.3: the near-NCA centroid is sometimes chosen as the leftmost node in the set of equally good candidate centroids. The lack of more accurate selection strategy causes lower stability of clusters.

Centroids, cluster shape, and the effectiveness of clustering
The k-means clustering algorithm is known to create spherical clusters, i.e., clustered elements are evenly distributed around the centroid. For illustration purposes, figures 5.17a and 5.17b visualize clusters as circles (painted schema elements are mapping elements, the black node is a centroid). The distance between clustered elements and the centroid is limited to 1 (note, white nodes in the figure do not belong to the cluster).

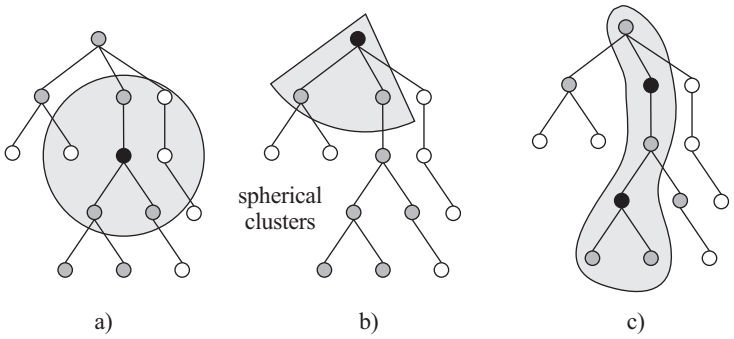


Figure 5.17 *a,b) spherical clusters, c) non-spherical cluster.*

Sphere is not necessarily the best cluster shape for clusters in clustered schema matching. It is to expect that (E) can be improved by creating non-spherical clusters. For example, by means of two centroids (and the same distance limit 1) the resulting cluster takes shape as shown in Fig. 5.17c. The validation of this hypothesis is future research.

5.6.3 Reclustering

This section validates the effects of the *join* and the *remove* reclustering techniques (see Sec. 5.3.3). A reminder: in clustered schema matching the precise number of clusters cannot be determined beforehand. Instead, we start with a large number of clusters, with as side effect, a large number of tiny clusters. Reclustering is

responsible for reducing the number of initial clusters to a desirable level. At the same time, reclustering solves the tiny clusters problem.

Join reclustering

This section observes, through an experiment, the effects of join reclustering on resulting clusters.

Experiment 5: Join reclustering

In this experiment, join reclustering is used with three different distance thresholds: 2, 3, and 4, i.e., clusters whose centroids are closer to each other than 2,3, and 4 edges are joined.

Table 5.12 *Experiment table for Exp. 5.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4
Clusterer			
initialization		reclustering	convergence
MIN-init 0.49		join 2,3,4	0% nodes

Tab. 5.13 shows the outcome of clustering. First row shows that 579 clusters, of which 268 are useful, are formed if no reclustering is used. With the application of reclustering the number of clusters is reduced (see *clusters* column), and the size of clusters increases (see *cluster size* column).

Table 5.13 *Clusters table for Exp. 5.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
no reclustering	268 [579]	3566 [4250]	13.3 / 15.1	0.11 (0.96%)
join 2	263 [445]	3735 [4250]	14.2 / 15.6	0.13 (1.07%)
join 3	251 [333]	4032 [4250]	16.0 / 16.5	0.15 (1.27%)
join 4	174 [217]	4157 [4250]	23.9 / 32.0	0.92 (7.69%)

Note that with join threshold distances 2 and 3, the number of useful clusters does not change a lot, while the total number of clusters rapidly drops from 578, to 445, to 333. This is a desired behavior – useful clusters are seldom tiny, and should not be disturbed. With a large join distance, e.g., 4, there is a sudden drop in the number of useful clusters to 174. This means that some useful clusters were

joined, potentially leading to the formation of very large clusters. This increases the search space size (note the jump in the number of solutions in the join 4 row of Tab. 5.13) and reduces the efficiency of mapping combining Ⓜ.

To see how join reclustering handles tiny clusters we use the size distribution graph shown in Fig. 5.18. With *join 3* reclustering, the number of tiny clusters decreases from 316, which is the sum of *no reclustering* $n = 0$, $n = 1$, and $n = 2$ clusters, to a total of 86 tiny clusters. At the same time, no huge clusters are formed; instead, there is an increase of $n = 4$ clusters.

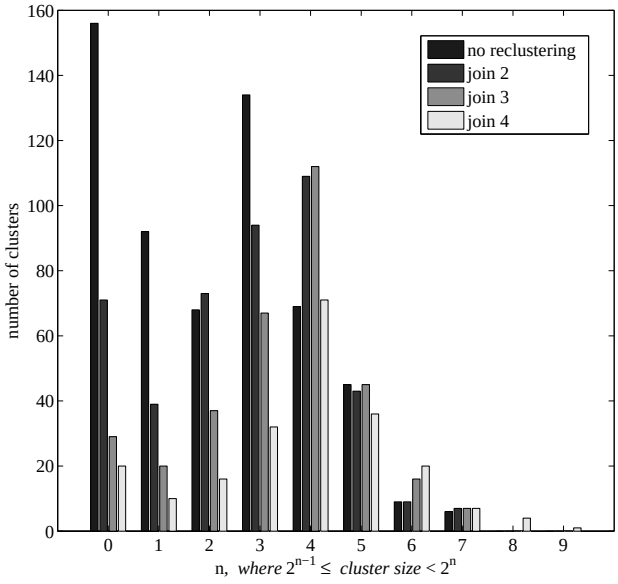


Figure 5.18 *The effects of the join reclustering.*

The join reclustering comes at a price of reduced clustering efficiency Ⓒ. Apart from the direct computational cost of the reclustering, indirect effects exist as well. The number of iterations necessary to meet the convergence criteria increases: when a whole cluster is added to another, the centroid significantly changes its position inducing cluster instability. Tab. 5.14 illustrates this effect. With no reclustering and join 2 reclustering, clusters reach total stability in iteration 3. With join 4, the stability is reached in iteration 7.

Remove reclustering

There exist at least two reasons why cluster joining cannot eliminate all tiny initial clusters. The first reason is that some tiny clusters are isolated and have no

Table 5.14 *Iterations table* ($\frac{\#\ moved\ elements}{[\#\ formed\ clusters]}$) *for Exp 5.*

iteration → centroid kind ↓	1	2	3	4	5	6	7
no reclust.	4250 [579]	232 [579]	16 [579]	0 [579]	– –	– –	– –
join 2	4250 [578]	390 [489]	140 [445]	0 [445]	– –	– –	– –
join 3	4250 [578]	1059 [422]	472 [336]	33 [333]	0 [333]	– –	– –
join 4	4250 [578]	1028 [376]	636 [265]	512 [239]	435 [225]	549 [217]	46 [217]

neighboring clusters to join with. The second reason is illustrated in Fig. 5.19a: two tiny clusters, B and C, are positioned near a larger cluster A. The *join 3* reclustering will join cluster C with cluster A. However, cluster B remains isolated; the distance between the centroids of A and B is larger than 3. To solve this problem, cluster B can simply be disbanded, and its nodes left free to join other clusters; in this case free nodes join cluster A.

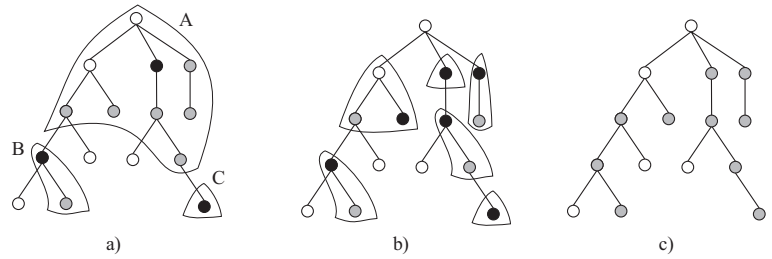


Figure 5.19 *a) A reason for removing small clusters, b) a risk of removing small clusters early.*

Care must be taken that clusters are not removed too early. For example, let the clusters in Fig. 5.19b be initial clusters. If remove reclustering with threshold 3 is applied immediately, all the clusters with less than 3 elements get removed, resulting in the situation shown in Fig. 5.19c: no clusters remain to gather nodes from disbanded clusters. To prevent this, remove reclustering is used with a delay.

Experiment 6: Remove reclustering

This experiment combines the join and the remove reclustering (see Tab. 5.15) Join distance 3 is used with three different remove thresholds: 3, 5, and 9. Re-

move reclustering is used with a delay of 1 iteration: run one iteration with join reclustering only, and then start using remove reclustering as well.

Table 5.15 *Experiment table for Exp. 6.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4
Clusterer			
initialization		reclustering	convergence
MIN-init 0.49		cluster join 3 cluster remove 3,5,9	0% nodes

Tab. 5.16 shows the results of clustering. The number of useful clusters is stable with the *remove 3* and the *remove 5* reclustering. As expected, the size of clusters slightly increases due to the nodes coming form the removed clusters. With *remove 9* reclustering, however, there is a large drop in the number of useful clusters from 124. This means that threshold 9 is too big, and causes the removal of a large number of useful clusters.

Note that remove reclustering reduces the number of mapping elements comprised within clusters, e.g., from 4250 with *no removing* to 4211 for *remove 3*. This is due to the removal of small clusters in trees where only small clusters existed, as already illustrated in Fig. 5.19b and Fig. 5.19c: clusters are disbanded but free nodes have no other clusters to join. This problem potentially degrades effectiveness and requires more attention in future research.

Table 5.16 *Clusters table for Exp. 6.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
no remove	251 [333]	4032 [4250]	16.1 / 16.5	0.15 (1.27%)
remove 3	250 [282]	4065 [4211]	16.3 / 16.6	0.15 (1.29%)
remove 5	235 [243]	4082 [4150]	17.4 / 17.1	0.17 (1.41%)
remove 9	124 [127]	3996 [4023]	32.2 / 25.6	0.37 (3.12%)

Fig. 5.20 shows the cluster size distribution graph for the current experiment. With the *remove 3* reclustering there are no more clusters with less than 3 elements ($n = 0$ and $n = 1$). The freed elements are "collected" by $n = 2$ and $n = 3$ clusters which consequently get bigger. Notice (for *remove 3*) a slight drop in $n = 2$ and $n = 3$ clusters and a slight increase in $n = 5$ clusters. *Remove 9* reclustering shows the same behavior but in a more extreme proportion, which even leads to the creation of one huge cluster.

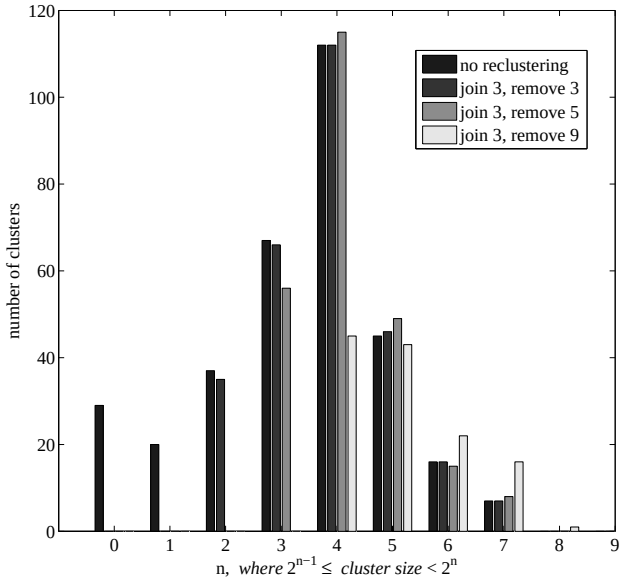


Figure 5.20 *The effects of the remove and the join reclustering.*

Tab. 5.17 is the iterations table for the current experiment. The table shows that remove reclustering has a small influence the convergence time: with *no remove* reclustering, full cluster stability is reached in cycle 4. Apart from *remove 5* reclustering, which adds one iteration, other remove variants also need 4 cycles to stabilize. We explain this as follows. With remove reclustering, the freed elements do not all join the same neighboring cluster, but instead get attracted to the nearest cluster, which is not the same for all the elements. With such "natural" distribution of freed nodes, there is less disturbance in the clusters which attract these nodes.

This shows that remove reclustering solves the problem of remaining tiny clusters when using only join reclustering (e.g., the case of cluster B in Fig. 5.19a). Remove reclustering is cheap to implement, and furthermore, improves the efficiency of clustering Ⓒ by further reducing the number of clusters. In Bellflower, the criterion for removing a cluster is the size of the cluster.

We can conclude that reclustering techniques in Bellflower determine the size and the number of resulting clusters, and also successfully solves the tiny clusters problem. Finally, together with other clustering parameters, reclustering algorithms influence the effectiveness of clustered schema matching Ⓔ; this aspect will be addressed shortly.

Table 5.17 *Iterations table* ($\frac{\#\ moved\ elements}{[\#\ formed\ clusters]}$) *for Exp 6.*

iteration → centroid kind ↓	1	2	3	4	5
no removing	4250 [579]	1059 [422]	472 [336]	33 [333]	0 [333]
remove 3	4250 [579]	1059 [422]	397 [284]	25 [282]	0 [282]
remove 5	4250 [579]	1059 [422]	391 [246]	34 [244]	3 [243]
remove 9	4250 [579]	1059 [422]	1020 [128]	59 [127]	0 [127]

5.6.4 Convergence criteria

So far in the experiments, the convergence criterion was *total stability* of clusters. A less strict criterion, i.e., the one that ends the clustering iterations before total stability is reached, leads to improved Ⓒ. This section discusses how different values, gathered during clustering, can be used to design another convergence criterion. Also, the relation between convergence criteria and Ⓔ is discussed.

Experiment 7: Convergence criteria

In this experiment (see Tab. 5.18), the convergence criterion is the total cluster stability. In the experiment mappings combining is performed at the end of each iteration step. This is used to observe Ⓔ at the end of different iterations. Note that this is not the regular way of using the mapping combiner; normally, mappings combining is performed only when the clustering ends.

Table 5.18 *Experiment table for Exp. 7.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4

Clusterer		
initialization	reclustering	convergence
MIN-init 0.49	join 4, remove 5	0% nodes

Mappings combiner			
α	0.5	δ	0.75 (after each k-means iteration)

Tab. 5.19 is an abbreviated combination of the previously used clusters and iterations tables. The table shows some properties of resulting clusters for each iteration of the k-means algorithm. The total cluster stability is reached after 7 iterations. Clustering efficiency (C) would improve by stopping iterations sooner.

Table 5.19 *Combined clusters/iterations table for the Exp. 7.*

iteration → cluster property ↓	1	2	3	4	5	6	7
elements moved [resulting clusters]	4250 [579]	1028 [376]	794 [183]	304 [174]	202 [170]	160 [169]	27 [169]
useful clusters	283	227	179	172	168	167	167
avg. cluster size	13.2	16.9	23.1	24.2	24.7	24.9	24.9
search space (%)	1.19%	1.94%	3.77%	4.07%	5.25%	8.89%	7.80%

Tab. 5.19 monitors four different parameters at each iteration stage: elements moving from one cluster to anther, the total number of clusters, the number of useful clusters, average cluster size, and the search space reduction. Observe that most parameters in the table show large changes in the first few iterations, and a more stable behavior in the remaining iterations. Also, most of the parameters show monotonic behavior. Parameters such as these can be used to design a better convergence criterion.

Using absolute values: convergence criteria can be based on absolute stability of some parameters, for example, when the number of moved elements becomes less then 10% of all the elements being clustered. In the current experiment (in which 4250 elements are being clustered) this condition is satisfied in iteration 4, because 304 is less than 10% of 4250 nodes. Similarity, the search space size can be used to determine the end of clustering, e.g., stop when search space reduction is more than 5% of the non-clustered one. In this way, the convergence criterion largely determines (M).

Using relative values: a convergence criterion can be based on comparison of values in two consecutive iterations, for example, when the average size of clusters does not change for more than 3% percent in the two consecutive iterations. In the current experiment, this condition is satisfied in iteration 5 at an average cluster size 24.7. In iteration 4 the average size is 24.2, and $24.2 \cdot 103\% = 24.926 > 24.7$.

In designing the clustering criteria it is also important to take into account the effectiveness of clustered schema matching (E). Fig. 5.21 shows the effectiveness of clustered schema matching (i.e., answer set ratio graph) for each iteration. In early iterations, e.g., iteration 1 and 2, the number of clusters is very large,

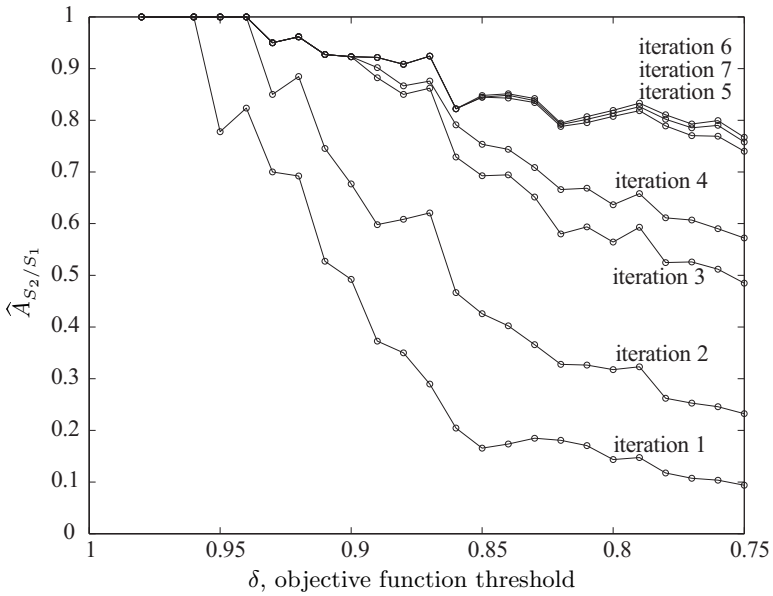


Figure 5.21 *Answer set ratio after each iteration.*

i.e., 579 and 376 and the search space is heavily reduced (see Tab. 5.19). The composition of clusters is largely dependent on initialization because there was not enough iterations yet to rearrange the clusters. All in all, large number of schema mappings is lost and the answer set ratio lines for iteration 1 and 2 in Fig. 5.21 are low. In iterations 3,4, and 5, clusters continue to grow and so does the search space and the amount of retrieved schema mappings. Next, in iterations 5,6, and 7, the effectiveness of clustering begins to converge to a stable state. The last three iterations deliver similar answer set ratio curves. It seems that regarding effectiveness, iteration 5 is a good time to stop. Iteration 5 is also good time to stop when taking Ⓜ into account. Tab. 5.19 shows that search space grows from 5.25% (iteration 5) to 8.89% (iteration 6). The growth in of the search space causes the reduction of Ⓜ, while not bringing any effectiveness improvements. Observe also that, if we are only interested in the effectiveness regarding schema mappings a with $\Delta(T, \tau) \geq 0.9$, iteration 3 is a good iteration to end with. Further iteration does not improve Ⓔ in the area above $\delta = 0.9$.

The experiments presented above illustrate the problem of designing a good convergence criteria. The few suggested convergence methods are simple to implement, but however, do not guarantee the most efficient clustering and the most effective clusters.

5.7 Effectiveness and efficiency of clustered schema matching

In optimization algorithms, efficiency improvements usually imply the reduction of effectiveness, and vice versa. In real-world applications, it is desirable for the balance between the efficiency and effectiveness to be tunable to meet some practical requirement. In this section, we observe the behavior of the efficiency/effectiveness balance in the clustered schema matching technique.

Experiment 8: Effectiveness/efficiency trade-off

This experiment performs clustered schema matching using three variants of the same clustering algorithm. The variants differ in the value of join reclustering threshold which varies over 2,3, and 4 (see Tab. 5.20).

Table 5.20 *Experiment table for Exp. 8.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4
Clusterer			
initialization		reclustering	convergence
MIN-init 0.49		join 2,3,4; remove 5	10% nodes & 5% clusters
Mappings combiner			
α	0.5	δ	0.75

Tab. 5.21 shows properties of resulting clusters. With no clustering (see the tree-clusters row) the search space has 11.96×10^6 possible mappings. With the three clustering variants, the search space is reduced to 1.29% (i.e., search space is reduced 77 times), 1.41% (i.e., 70 times) and 4.07% (i.e., 24 times).

Table 5.21 *Clusters table for Exp. 8.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
join 2	251 [262]	4046 [4131]	16.1 / 16.0	0.15 (1.29%)
join 3	235 [244]	4079 [4150]	17.4 / 17.1	0.17 (1.41%)
join 4	172 [174]	4154 [4170]	24.1 / 27.0	0.48 (4.07%)
tree-clusters	95 [228]	4326 [4751]	45.5 / 86.4	11.96 (100%)

Efficiency analysis

Clustering reduces the search space and consequently improves efficiency. It is to be expected that larger search space reductions lead to greater efficiency improvements. First question to answers is how does the search space reduction and the efficiency improvement relate?

Tab. 5.22 shows time measurements for the current experiment. *Element matcher*, a component shared by all matching techniques, needs 7.3 seconds to complete. *Clusterer* in all three variants uses approximately 12 seconds. Large differences, however exist in *mappings combiner*: 16.0, 23.8 and 56.3 seconds. With no clustering used, mappings combiner spends 106.3 in discovering schema mappings.

Table 5.22 *Performance table (timers & counters) for Exp 8.*

clustering	element matching	clustering	mappings combining	Σ (sec)	partial mappings	$\lvert A_S^\delta \rvert$
join 2	7.3	12.2	16.0	35.5	51491	620
join 3	7.3	12.1	23.8	43.2	56965	1009
join 4	7.3	11.9	56.3	75.5	109341	2444
tree-clusters	7.3	0	106.3	113.6	386817	4271

The efficiency improvements caused by clustering are as follows. The total measured non-clustered matching time is 113.6 seconds (see Tab. 5.22), and the improvements for each clustering variant are (join 2 (1/77) reads: join 2 clustering variant which reduces the search space 77 times):

join 2 (1/77) • 113.6 sec / 35.5sec = 3.20 times faster
join 3 (1/70) • 113.6 sec / 43.2sec = 2.63 times faster
join 4 (1/24) • 113.6 sec / 75.5sec = 1.51 times faster

Thought the acquired improvement factors are moderate, they undoubtedly show that efficiency improvements do exist. The amount of improvement depends on the clustering variant, or as we can also see, the efficiency depends on the amount of search space reduction. For example, the join 2 variant reduces the search space 77 times, which in turn reduces time spent in mappings combining 106.3/16.0 = 6.6 times (see the mappings combining column in Tab. 5.22), and the total improvement comes as 3.2 times faster execution.

In performing deeper analysis of the relation between the search space reduction and the gained efficiency improvements, we first faced a practical problem; time measurements in Bellflower had too much "noise." Bellflower is a large prototypical system, and many algorithms are implemented in a less than optimal way. For example, the implementation of the $B\&B$ algorithm makes use of large C++

objects in each recursion of the algorithm. With each recursive call, these objects are duplicated and handed over to next recursion level. Measurements showed that duplication and deletion of these objects, by means of `new` and `delete` C++ operators, consume significant portion of time and make intolerable "noise" in time measurements.

We overcome this problem by means of counters. In particular, as explained in Sec. 5.3.3, we count the number of times the Δ_{max} bounding function is executed during the execution of the $B\&B$ algorithm. In a well-tuned system, the efficiency of the mapping combiner would roughly be proportional to this counter. Tab. 5.22 shows the value of this counter in the *partial mappings* column (note, this number is equal to the number of partial mappings generated and tested by the $B\&B$ algorithm). We continue the efficiency analysis with this more reliable measure.

In the non-clustered case, the search space contains 11.96×10^6 schema mappings. When finding the highly ranked schema mappings by means of complete enumeration, all 11.96×10^6 mappings are generated and the objective function is computed for each. But when using the $B\&B$ algorithm to find the highly ranked mappings, the $B\&B$ algorithm generates and tests only 386817 partial mappings (see *partial mappings* column in the Tab. 5.22). The $B\&B$ algorithm cuts a significant portion of the search space, and generates and tests 31 times less (partial)mappings compared to full enumeration.

When using both the clustering, e.g., join 2 variant, and the $B\&B$ algorithms, the number of generated partial mappings is 51491, which is 232.5 times less compared to full enumeration of the non-clustered case. With the introduction of clustering the number of generated partial mappings got reduced $386817/51491 (= 232.5/31) = 7.5$ times; with clustering Ⓜ shows a 7.5 fold improvement.

It might be confusing that join 2 clustering reduced the search space 77 times, and yet, the total benefit is "only" 7.5 faster execution. This is explained as follows. Join 2 reduces the search space to a size 77 times smaller, i.e., 0.15×10^6 schema mappings (see Tab. 5.21). With such reduced input, the $B\&B$ algorithm generates 51491 partial mappings which is $0.15 \times 10^6/51491 \approx 3$ times less than with full enumeration which would generate all 0.15×10^6 mappings. Note that when no clustering is used, the $B\&B$ algorithm improves efficiency 31 times, but with clustering only 3 times. The reason for this disproportionate result is as follows. The $B\&B$ algorithm gains efficiency by discarding partial mappings for which it computes that the ending objective function returns low similarity index. In a large repository, $B\&B$ discovers a lot of such partial mappings (mind that the discovery of each consumes time), and hence delivers high pruning ratio, i.e., 31. However, when clustering is used, repository is partitioned into clusters within which most mappings rank high, in other words, there is little partial mapping supposed to be pruned away by the $B\&B$. So when first using clustering, the $B\&B$ algorithm has much less partial mappings to check due to the 77 times

reduced search space, but also, shows much lower pruning ration, i.e., 3, because much more partial mappings result in good solutions.

Effectiveness

Improved efficiency comes at the price of reduced effectiveness. The question we ask here, is how does the search space reduction, produced by clustering, reflect on the effectiveness? Fig. 5.22 shows the answer set ratio lines for the three clustering variants. The answer set ratio differs at different thresholds. At $\delta = 0.75$, clustering variants join 2, join 3, and join 4 produce sets of mappings A_S^δ with 620, 1109, 2444 schema mappings (see Tab. 5.22). It is interesting to compare the search space reduction at $\delta = 0.75$ with the reduction of the resulting set of mappings.

join 2 • search space reduced to 1.29%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 14.5\%$
join 3 • search space reduced to 1.41%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 23.6\%$
join 4 • search space reduced to 4.07%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 57.23\%$

The more the reduction of the search space, the more schema mappings lost from $A_{S_1}^\delta$. However, the percentage of preserved schema mappings is much larger than the percentage of the search space reduction; e.g., 1.41% of the search space contains 23.6% percent of schema mappings. Furthermore, Fig. 5.22, shows that the percentage of preserved mappings grows with growing threshold δ. At $\delta = 0.9$ the answer set ratio is as follows.

join 2 • search space reduced to 1.29%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 56.1\%$
join 3 • search space reduced to 1.41%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 74.0\%$
join 4 • search space reduced to 4.07%, $|A_{S_2}^\delta| = |A_{S_1}^\delta| \cdot 92.9\%$

This experiment shows the expected drop in effectiveness, but also confirms the desired property of clustered schema matching: preservation of schema mappings which rank high and loss of mappings which rank low.

In the current experiment, the three clustering variants are based on the same clustering algorithm and thus have equal clustering "intelligence." Consequently, the observed differences in (E), for the three variants, can only be attributed to different cluster sizes, and not one clustering variant being "smarter" than the other. To that end, it would be interesting to compare the effectiveness of two clustering algorithms with different "intelligence." But, in order to have fair comparison of two different clustering algorithms, the search space reduction, produced by these algorithms, has to be the same. Otherwise, the differences in effectiveness could still be attributed to different search space reduction, rather than the clustering "intelligence."

Unfortunately, in Bellflower, the requirements of such an experiment are hard to meet. First, all clustering variants are rather similar in Bellflower, thus all algorithms have similar clustering "intelligence." Second, for a pair of clustering algorithms it is very hard to set the parameters in way that leads to the same search

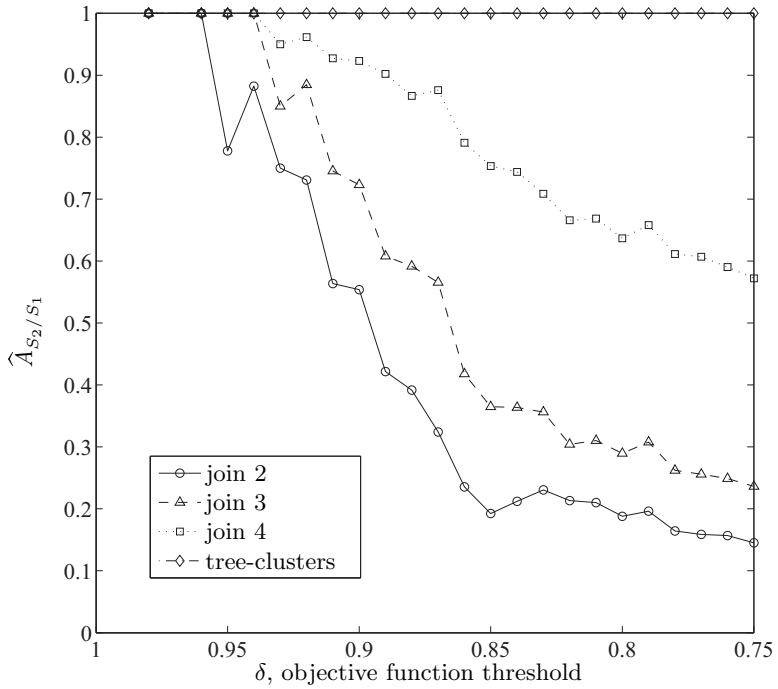


Figure 5.22 *Effectiveness.*

space reduction. Despite these problem, we present the following experiment as our best-effort in comparing the effectiveness of two different clustering algorithms.

Experiment 9: Comparing the effectiveness of different clustering algorithms

In trying to find two clustering algorithms which meet the two requirements stated above, i.e., that the algorithms differ as much as possible, and that algorithms deliver the same search space reduction, we found the following. In this experiment, of the two clustering variants, one is based on the MIN-init initialization, and the other on the NCA-init initialization. Even though the two initialization algorithms are very different, the two clustering variants as a whole can be treated as just slightly different because they share the remaining parameters (i.e., the reclustering and the convergence parameters, see Tab. 5.23). We have experimented with clustering variants showing larger differences, however, for these it was not possible to meet the second requirement - the equality of the search space reduction.

Tab. 5.24 shows the properties of the resulting clusters. The table also shows that two clustering variants produce almost the same number of clusters and with similar average sizes. Most importantly, the resulting search space reduction,

Table 5.23 *Experiment table for Exp. 9.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4

Clusterer		
initialization	reclustering	convergence
MIN-init 0.49	join 3; remove 5	0% nodes
NCA-init 3	join 3; remove 5	0% nodes

Mappings combiner			
α	0.5	δ	0.75

though slightly larger for the MIN-init clustering technique, is comparable. This satisfies the second requirement.

Table 5.24 *Cluster table for the Exp. 9.*

clustering	clusters	mapping elements	cluster size avg./dev.	schema mappings $\times 10^6$ (search space)
MIN-init	235 [243]	4082 [4150]	17.4 / 17.1	0.169 (1.41%)
NCA-init	238 [299]	4012 [4534]	16.9 / 15.7	0.160 (1.34%)

The resulting answer set ratio curves for the two clustering techniques are given in Fig. 5.23. The lines are very similar. This is because the two clustering variants are not that extremely different. However it is important to see that the two lines in the figure not identical. Most importantly, the lines show that at $\delta = 0.75$ the NCA-init approach has higher answer set ratio, even though it reduces the search space more than the other clustering variant (see Tab. 5.24). This confirms that in this experiment, (E) does not depend only on the search space reduction, but also how the clustering was done.

In order to decide which of the two clustering variants shows better (E) in this experiment, we observe that for the highly ranked solutions, the MIN-init clustering approach delivers better answer set ratio. This means that MIN-init is better in preserving the highly ranked (and loosing the lower ranked) mappings, and is in this case preferable to NCA-init variant.

This experiment has shown that it is not only the search space reduction which determines (E). Though subtle, the differences in answer set ratio seen in Fig. 5.23 show that the choice of the clustering algorithm has an impact on (E). For now, we can only expect that more different clustering algorithms show much larger differences in (E).

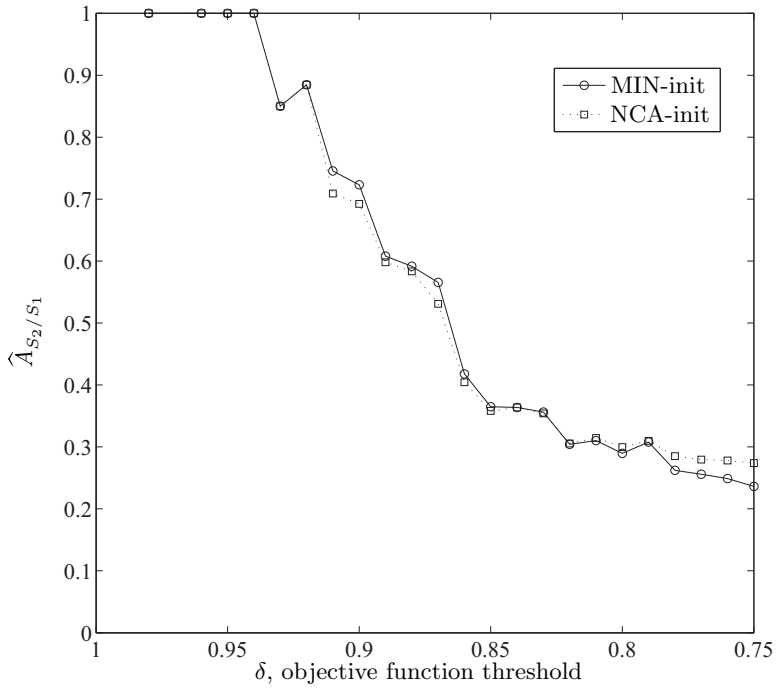


Figure 5.23 *Initialization techniques do not provide significant difference in effectiveness.*

This experiment has an interesting "side-effect." The fact that the two clustering variants, used in this experiment, differ only in the initialization algorithm, opens an opportunity to get more insight into initialization of the k-means algorithm discussed in Sec. 5.6.1. As discussed, k-means is known to be sensitive to initial centroids. Our claim in was that a larger number of initial centroids, combined with reclustering, reduces this problem. In this experiment, two very different initialization techniques have been used to produce a large number of initial clusters. Other parameters of the clustering algorithm are the same. Both approaches resulted in similar sets of clusters, and delivered similar effectiveness, i.e., answer set ratio. In accordance with our claim, the experiment shows that initialization is *not dominant* in determining the resulting clusters and the resulting (E). If the claim was incorrect, and clustering very sensitive to initialization, the lines in Fig. 5.23 would show larger differences.

5.8 Correlation between clustering and schema matching

In the experiments performed so far, various clustering variants were used. For mappings combining, however, always the same objective function was used; the one defined in Eq. 5.3 in Sec. 5.3.2:

$\Delta(a) = \alpha \cdot \Delta_{NS}(a) + (1 - \alpha) \cdot \Delta_{PL}(a)$

Sec. 3.3.2 discussed that clustering in clustered schema matching is not generic and has to be designed for each specific objective function. One cannot hope that one clustering variant delivers the same effectiveness with different objective functions. In the following experiment, we examine the extent of this claim.

Experiment 10: Single clustering for various objective functions

In this experiment (see Tab. 5.25), three different objective functions are created by varying the value of the α parameter in the objective function formula.

Table 5.25 *Experiment table for Exp. 10.*

Repository		Personal schema	El. matcher
name	size	definition	$\delta_{element}$
D	9757	/name/{email,address}	0.4

Clusterer		
initialization	reclustering	convergence
MIN-init 0.49	join 3; remove 5	0% nodes

Mappings combiner			
α	**0.25, 0.50, 0.75**	δ	0.75

The α parameter controls the relative importance distribution of the average name similarity Δ_{NS} and the normalized path length difference Δ_{PL}; the smaller the α the less important name similarity and the more important path length. In the experiment, α varies over 0.25, 0.50, and 0.75. The use of different objective functions also means that the same schema mappings are given different similarity indexes, resulting in a different number and order of generated schema mappings. Given below are numbers which illustrate this fact. In the given experiment, the three objective functions all generate different sets of schema mappings in both the non-clustered case (i.e., $A^{\delta}_{S_1}$) and in the clustered case (i.e., $A^{\delta}_{S_2}$).

- $\alpha = 0.25$ • $|A^{\delta}_{S_1}| = 32637, |A^{\delta}_{S_2}| = 13990$
- $\alpha = 0.50$ • $|A^{\delta}_{S_1}| = 4271, |A^{\delta}_{S_2}| = 1009$
- $\alpha = 0.75$ • $|A^{\delta}_{S_1}| = 5330, |A^{\delta}_{S_2}| = 536$

Fig. 5.24 shows the answer set ratio curves for the three different objective

functions. Clustering shows the best effectiveness for $\alpha = 0.25$. This is because the objective function with $\alpha = 0.25$ favors path length over name similarity and clustering is performed by taking into account path lengths between nodes. On the other extreme, clustering has lower effectiveness if it uses heuristics that are different from the one used by the schema matcher. For $\alpha = 0.75$, the objective function favors schema mappings with high name similarity indexes, and even schema mappings with long distances between elements can rank very high. Clustering cuts many of such mapping schemas. Hence the bad effectiveness of $\alpha = 0.75$ curve when compared to other two.

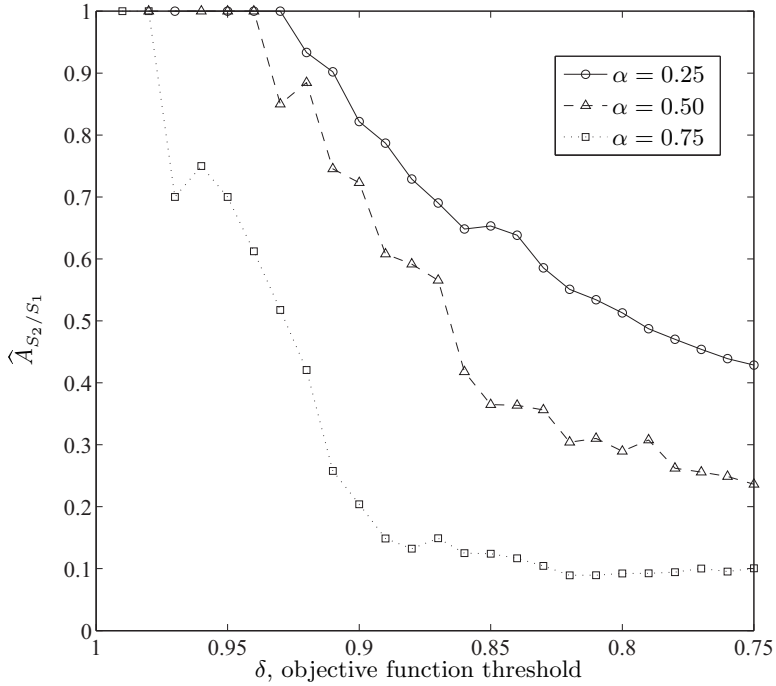


Figure 5.24 *Answer set ratio for different objective functions.*

This experiment shows that, in order to obtain good effectiveness of clustered schema matching, the clustering algorithm has to be designed and tuned to a specific objective function. In our research, so far, the design of the clustering criteria was treated as purely handcraft work. Future research needs to investigate and recommend techniques for the design of the clustering criteria. The technique should be generic enough to incorporate other heuristics, both local and structural, which are in use in the schema matching systems.

5.9 Scalability of the clustered schema matching technique

We have shown, both through the experiments in this chapter and analytically in Sec. 3.3.1, that the clustered schema matching technique improves the efficiency of non-clustered schema matching. This section analyzes the impact of the technique on the *scalability* of the matching system.

Scalability is a capability of a system to maintain total throughput under increasing load, when resources (typically hardware) are added[5]. In the clustered schema matching, the *load* primarily refers to the *size of the schema repository* and the *size of the personal schema*. The *throughput* is the number of schema matching problems that can be solved in a unit of time, and is determined by the efficiency of the system. So far, in this thesis, the efficiency of the system was analyzed by means of *"O"* expressions which show the complexity of problems and algorithms, and therefore represent the efficiency. *"O"* expressions will also be used here to discuss the *scalability* of the matching system.

In this analysis we assume that the main *load*, i.e., the schema repository, is tree-based. With this assumption, there are three specific values which impact the performance of the matching system. These are:

1. *Number of schemas in the repository*, denoted with t.
   Theoretically, schema repository R grows in two ways. The first way is the addition of new schemas in the repository, in which case, t grows.
2. *Size of schemas in the repository*, denoted with $|N_r|$.
   The second reason for the growth of R is the growth of individual repository schemas, e.g., by adding elements to these schemas (not that this is actually expected to happen often in practice).
3. *Size of the personal schema*, denoted with $|N_T|$.
   Personal schema grows when new elements are added to it.

Notation

So far in the thesis, the symbol R represented the schema repository, but for simplicity, in complexity formulas it was used to depict a single schema in the repository. The fact that repository was a forest was not explicitly represented. In this section this simplification does not apply. Instead, symbol R represents the *whole* repository, i.e., a forest of schema-trees, r represents one schema-tree in the repository, and t is the number of schema-trees in the repository. The following holds: $O(t \cdot |N_r|) = O(|N_R|)$.

[5]definition adapted form `www.answers.com/topic/scalability`

5.9.1 Revisiting the complexity of clustered schema matching

In this section, we redefine the complexity for the non-clustered and the clustered schema matching technique in terms of the three variable values t, $|N_r|$, and $|N_T|$. These complexities are later used to discuss the scalability of the two matching techniques.

Element matcher

Bellflower implements a simple element matcher: a nested loop in which every personal schema element is compared with every elements in every repository schema. The computational complexity of the element matcher is

$$O(t \cdot |N_T| \cdot |N_r|) \tag{5.5}$$

Real-world systems are expected to use more efficient string comparison techniques (see Sec. 3.2.2). For each personal schema element $n \in T$ and every repository schema r, element matcher produces a set of mapping elements $\mathcal{M}_n^r$. When a schema r grows, $|\mathcal{M}_n^r|$ grows proportionally. Therefore

$$O(|N_r|) = O(|\mathcal{M}_n^r|), \text{ and} \tag{5.6}$$

$$O(|N_T| \cdot |N_r|) = O(|\mathcal{M}^r|), \text{ where } \mathcal{M}^r = \bigcup_{n \in T} \mathcal{M}_n^r. \tag{5.7}$$

K-means clustering

Adopted from Sec. 3.4.2, the computational complexity of the k-means algorithm becomes

$$O(t \cdot k \cdot i \cdot |\mathcal{M}^r|), \text{ and after substituting } |\mathcal{M}^r| \text{ from Eq. 5.7 this becomes}$$

$$O(t \cdot k \cdot i \cdot |N_T| \cdot |N_r|) = O(t \cdot k \cdot |N_T| \cdot |N_r|) \tag{5.8}$$

where i is the number of k-means iterations and k is the number of clusters in each repository schema r. Number of iterations i is independent of all three variables t, $|N_r|$, and $|N_T|$. It is therefore treated as constant value and removed from the complexity formula. In Sec. 3.3.1 it was discussed that the size of the search space for the mappings combiner becomes linear in respect to the size of the repository schema, if the ratio $\frac{|\mathcal{M}_n^r|}{k} = \mathtt{M}$ is kept constant. When $\mathtt{M}$ is constant it is also true that

$$O(k) = O(|\mathcal{M}_n^r|) = O(|N_r|) \tag{5.9}$$

Substituting Eq. 5.9 in Eq. 5.8, the complexity of the k-means clusterer becomes

$$O(t \cdot |N_T| \cdot |N_r|^2) \tag{5.10}$$

Mappings combiner
Mappings combiner in Bellflower implements the Branch and Bound ($B\&B$) algorithm for which the worst case complexity is worse than exponential. The actual computational complexity of the $B\&B$ algorithm is problem dependent and requires additional research, which is not performed in this thesis. As a substitute, it is assumed here that the mappings combiner implements simple enumeration for which the complexity is proportional to the size of the problem search space. The size of the search space was already discussed in Sec. 3.3.1, and when adopted becomes,

$$O(t \cdot |N_r|^{|N_T|}), \quad \text{for the non-clustered case, and} \tag{5.11}$$

$$O(t \cdot |N_r| \cdot \mathtt{M}^{|N_T|}), \text{ for the clustered case} \tag{5.12}$$

Systems' scalability
When combined, Eq. 5.5 and Eq. 5.11 define the non-clustered schema matching computational complexity, and Eq. 5.5, Eq. 5.10, and Eq. 5.12 define the clustered schema matching computational complexity as follows:

[non-clustered]

$$O(t \cdot |N_T| \cdot |N_r|) \quad + \quad O(t \cdot |N_r|^{|N_T|}) \tag{5.13}$$

[clustered]

$$O(t \cdot |N_T| \cdot |N_r|) + O(t \cdot |N_T| \cdot |N_r|^2) + O(t \cdot |N_r| \cdot \mathtt{M}^{|N_T|}) \tag{5.14}$$

5.9.2 Scalability discussion

In this section we discuss the scalability of the clustered schema matching technique by varying each variable independently: e.g., when t grows, the $|N_r|$ and the $|N_T|$ are considered *constant* and are replaced with symbol $\mathtt{C}$ in equations 5.13 and 5.14.

Scalability in respect to the number of schemas in the repository
In this case, t grows and $O(|N_r|) = O(|N_T|) = \mathtt{C}$. Complexity of the systems becomes:

$$[\textbf{non-clustered}] \quad O(t \cdot \mathtt{C}^2) + O(t \cdot \mathtt{C}^{\mathtt{C}}) = O(t)$$

$$[\textbf{clustered}] \quad O(t \cdot \mathtt{C}^2) + O(t \cdot \mathtt{C}^3) + O(t \cdot \mathtt{C} \cdot \mathtt{M}^{\mathtt{C}}) = O(t)$$

Both systems scale linearly in respect to t. This is because both techniques process each schema in the repository independently of other repository schemas. Parallel processing can be used with both techniques to maintain performance

when faced with a growing t. This also shows that the clustered schema matching technique does not improve the scalability in respect to t.

Scalability in respect to the size of schemas in the repository
In this case, $|N_r|$ grows and $O(t) = O(|N_T|) = \mathtt{C}$. Complexities of the systems become (note, $|N_T|$ is not replaced with $\mathtt{C}$ where used as exponent over $|N_r|$):

$$\begin{aligned} [\textsf{non-clustered}] \quad & O(|N_r|) + O(|N_r|^{|N_T|}) \\ [\textsf{clustered}] \quad & O(|N_r|^2) + O(|N_r|) \end{aligned}$$

When individual schemas in the repository get bigger, the non-clustered schema matching has a polynomial complexity, due to the complexity of the mappings combiner. On the other hand, the clustered schema matching shows quadratic complexity due to the clusterer. We can also observe that $|N_T|$, though constant, determines which of the two approaches scales better. For $|N_T| > 2$, which is true for any reasonable personal schema, clustered schema matching technique shows better scalability. To improve the scalability further, the quadratic complexity of the clusterer must be reduced. This calls for research in different clustering techniques.

Scalability in respect to the size of the personal schema
In this case, $|N_T|$ grows and $O(t) = O(|N_r|) = \mathtt{C}$. Complexity of the systems becomes (note, for discussion purposes $|N_r|$ is not replaced with $\mathtt{C}$ where used as a base for an exponential expression):

$$\begin{aligned} [\textsf{non-clustered}] \quad & O(|N_T|) + O(|N_r|^{|N_T|}) \\ [\textsf{clustered}] \quad & O(|N_T|) + O(\mathtt{M}^{|N_T|}) \end{aligned}$$

Complexity formulas show that both approaches have an exponential complexity, i.e., clustered schema matching does not improve the scalability in respect to the size of the personal schema. However, note the difference in the base of the exponential component. With non-clustered schema matching, this is $|N_r|$, with clustered schema matching, the base is a constant $\mathtt{M}$, which is a value tunable by the clustering algorithm. The tuning of $\mathtt{M}$ balances between efficiency and effectiveness of schema matching, and may in practice enable systems to switch from not feasible to feasible.

Scalability in graph-based repositories
The scalability (and complexity) analysis as performed above is valid for tree-based repositories only. However, as introduced in Sec. 2.3.2, the natural model for XML

schemas is a graph. In graph-based repositories, schemas are interconnected by means of inter-schema links, and the repository is (theoretically) a single connected graph. Instead of containing t schemas of size $|N_r|$ as in tree-based repositories, the graph-based repository is as single graph of size $|N_R|$.

Complexity formulas for a graph-based repository can simply be derived by setting $t = 1$ and replacing r with R in Eq. 5.13 and Eq. 5.14. However, the resulting formulas are correct only if the following simplification is adopted: schema matching systems consider only one path between any two graph nodes (considering multiple paths increases the complexity). With this assumption, the complexity of the matching techniques over a graph-based repository are:

[**non-clustered**]

$$O(|N_T| \cdot |N_R|) \quad + \quad O(|N_R|^{|N_T|}) \tag{5.15}$$

[**clustered**]

$$O(|N_T| \cdot |N_R|) + O(|N_T| \cdot |N_R|^2) + O(|N_R| \cdot \mathtt{M}^{|N_T|}) \tag{5.16}$$

Results are the same as with the tree-based repositories. In respect to R, polynomial vs. quadratic, and in respect to $|N_T|$, both techniques are exponential. Note, however, that $|N_R| >> |N_r| > \mathtt{M}$, and that the complexity difference between the clustered and non-clustered techniques becomes very large with a growing $|N_T|$. This indicates that the clustered schema matching technique can bring much larger improvements to graph-based than to tree-based schema matching systems.

5.10 Summary

This chapter validates the performance of the clustered schema matching when using tree-based repositories. Validation is based on Bellflower, an experimental system which implements both the non-clustered and the clustered schema matching techniques. The validation is performed by comparing the performance of these two techniques. The chapter starts with a description of algorithms implemented in Bellflower and repositories used in the experiments.

Ten experiments are performed and discussed. These observe the clustered schema matching technique in a holistic way, showing relations between its various components and parameters. Also, experiments help discover several problems, such as *tiny* and *huge* clusters. However, the main achievement of these experiments is the confirmation that clustered schema matching technique indeed possesses the properties for which it is designed:

- it improves efficiency of schema matching, and
- it preserves the highly ranked schema mappings while loosing the ones that rank low.

In the experiments, the measured efficiency improvement factor (i.e., $t_{non\text{-}clustered}/t_{clustered}$) ranges from 1.51 to 11. Bellflower is a proof-of-concept experimental system in which some algorithms are implemented in a sub-optimal way. Consequently, the measured improvement factors are moderate and, we believe, far from the ultimate capacity of the technique.

Experiments show that the size of clusters largely determines the balance between the efficiency and effectiveness. The cluster's size, however, is not the only factor determining the effectiveness. Experiments show that the choice of the clustering algorithm, and in particular its correlation with the objective function, plays an important role regarding the effectiveness.

The analysis shows that the technique also brings scalability improvements: the search space of the mappings combiner is, in the non-clustered case, *polynomially* dependent, but in the clustered case *linearly* dependent on the size of the repository schemas. The k-means clustering algorithm, due to its quadratic complexity, lessens the total scalability improvement. In respect to the size of the personal schema, clustered schema matching remains exponential, however, the base of the exponential expression is reduced and controlled by clustering to balance between efficiency and effectiveness. In graph-based repositories, the efficiency improvement induced by the clustered schema matching technique becomes much larger.

Chapter 6

Conclusion

6.1 Introduction

Since the beginnings of the Internet one question strongly attracts the attention of the scientific community: *"How to help a user quickly find, on the Internet, the exact information he is looking for?"* Over time, various tools, such as directories, search engines, and web portals, have been proposed, improved, and successfully deployed. Nevertheless, scientific community continues to look for novel, more intelligent and faster ways for information search.

In joining this research effort, this thesis first proposes and focuses on a *personal schema based querying* technique for the XML data on the Internet. With this technique, a user is allowed to represent his information need by means of a *personal XML schema* and to ask XML queries (i.e., *personal queries*) over his personal schema. It is the task of the *personal schema query answering system* (PSQ) to discover mappings of the personal schema onto XML schemas of the actual data sources on the Internet and to evaluate the user's personal query.

The thesis further focuses on the schema matcher component of a PSQ. Note that *the question* at the beginning of this section imposes two requirements on systems for information search: systems are asked to "*quickly find*," and to find the "*exact information*," that is, systems are asked to be both *efficient* and *effective*. Current schema matching research does not pay enough attention to efficiency, and contemporary schema matchers present a serious efficiency bottleneck when used in a PSQ. Therefore, this thesis focuses on developing a technique which improves the efficiency of schema matching. Because most improvements of the efficiency come with a price of reduced effectiveness, attention is paid to provide an acceptable balance between the efficiency and effectiveness. In the development of the improvement technique the thesis addresses three related, yet distinctive, topics:

- understanding the schema matching problem (chapter 2),
- improving the efficiency of schema matching systems (chapters 3 and 5), and
- validating effectiveness (chapter 4).

The following sections separately address each of the three topics. For each

topic a *short summary* is given, followed by *contributions and conclusions*, and directions for *future research*.

6.2 Understanding the schema matching problem

Summary

Before trying to efficiently solve any difficult problem, it is first necessary to have a complete, preferably formal, problem specification. As no suitable formal specifications exist for *semantic schema matching problems*, this thesis develops its own. The proposed formalization consists of three phases (see Sec. 2.3):

1. *Modeling the semantic schema matching problem*: a *model of the semantic schema matching problem* is built by extending the *model of a generic matching problem* with the notion of *semantics*, where semantics stands for the *meaning of data*.
2. *Approximating the semantic schema matching problem to enable automatic solving*: semantics cannot (yet) be understood by machines. Therefore, to enable automated solving, all components of the semantic schema matching problem are approximated with syntactic counterparts.
3. *Representing the approximated semantic schema matching problem as a constraint optimization problem*: the approximated semantic schema matching problem is formally specified using the *constraint optimization problem formalism*.

Contributions and conclusions

The main contribution of this part of the thesis is the finding that

- *a schema matching problem is a constraint optimization problem (COP).*

Constraint optimization problems are part of a well-known generic framework for problem representation and solving – *constraint programming*. The benefit of knowing that schema matching problems are COPs, is that schema matching systems can now directly exploit existing algorithms for efficient solving of constraint optimization problems (see Sec. 2.3.4).

Additionally, this finding enables the use of COP formalism to formally specify a schema matching problem. Having an unambiguous framework for schema matching problem specification ensures, both from an engineering and a scientific point of view, a solid base for the development of matching systems.

Future research

The formalization and modeling of schema matching is, at its core, an effort to model human reasoning. This is one of the most challenging areas in computer science and is completely open to new ideas and approaches.

Concerning the formalization approach presented in this thesis, future research should analyze the capabilities of the formalization technique to accommodate the full range of schema matching problems. For example, in the thesis element mapping is restricted to $1:1$ mappings (see Def. 2 on page 38). Future research should find the useful ways to formalize the $m:n$ mappings within the same framework. Furthermore, representation of more complex XML schema features, such as recursion and referential integrity constraints, needs more attention.

6.3 Improving the efficiency of schema matching systems

Summary

The method for improving the efficiency of schema matching which is proposed in this thesis, simulates the behavior of a human matcher. When a human matcher matches a small personal schema T against a large schema repository R, he first performs a quick scan of R in order to identify *regions* in R which can potentially deliver good mappings for T. The matcher then, as a second step, takes a closer look at each discovered region and tries to build meaningful mappings for T.

The thesis proposes the *clustered schema matching technique* as an implementation of this behavior. The clustered schema matching technique extends the existing schema matching systems by adding clustering as an intermediate step. Clustering forms regions, i.e., clusters, in the schema repository R, such that these regions have a high potential for delivering good schema mappings. After clustering, the remainder of the matching process is performed per cluster. Effectively, this reduces the search space, i.e., the workload, for the *mappings combiner* used in schema matching, and improves the efficiency (see Sec. 3.3.1). The improved efficiency does not come without a cost. Due to partitioning of R, some schema mappings can be split across clusters and never discovered. This reduces the effectiveness of the system.

The thesis performs a thorough analysis of how various parameters of the clustered schema matching technique correlate and how they influence the resulting efficiency and the effectiveness. An experimental system Bellflower is used for validation.

Contributions and conclusions

The main contribution of this thesis is the

- *clustered schema matching technique.*

Both analytically, and through validation, the thesis shows that clustered schema matching, when compared to the non-clustered matching systems,

- improves the efficiency of schema matching, and

- preserves the highly ranked schema mappings, while loosing the ones that rank low.

In the experiments, the measured efficiency improvement factor (i.e., $t_{non\text{-}clustered}/t_{clustered}$) ranges from 1.51 to 11. Bellflower is a proof-of-concept system, consequently, the measured improvement factors are moderate and, we believe, far from the ultimate capacity of the technique.

Experiments show that the size of clusters largely determines the balance between the efficiency and effectiveness (see Sec. 5.7). The cluster's size, however, is not the only factor determining the effectiveness. Experiments show that the choice of the clustering algorithm, and in particular its correlation with the objective function (see Sec. 5.8), plays an important role regarding the effectiveness.

In clustered schema matching several highly parameterized algorithms cooperate on a common task. Experimental and analytical investigation of the technique uncovered various problems and properties of the technique, some listed below.

- In clustered schema matching the number of clusters is not known beforehand. This problem is solved in the k-means algorithm by creating large number of initial clusters and by using the reclustering step to reduce the number of clusters to a desirable count. Additionally, this approach reduces the known dependency of the k-means algorithm on the initial clusters (see Sec. 5.6 and Sec. 5.7 Exp. 9).
- *Tiny* and *huge* clusters can harm the performance of the technique. Thesis shows that the *join* and the *remove* reclustering solves the problem of tiny clusters (see Sec. 5.6.1 and Sec. 5.6.3).
- Clustering can be orthogonally combined with other optimization techniques, even if these also aim at reducing the search space of the mappings combiner. In this thesis clustering is combined with the Branch and Bound ($B\&B$) algorithm. Thesis shows how each technique contributes to the total improvement (see Sec. 5.7). When combined, the total performance improvement $i_{combined}$ is larger than when the techniques are used individually, i.e., $i_{combined} > i_{clustering}$, $i_{combined} > i_{B\&B}$, but it is less than the multiplication of improvements acquired by the individual use of the techniques $i_{combined} < i_{clustering} \times i_{B\&B}$.

The thesis shows that the clustered schema matching technique serves the purpose for which it is designed, however, the complexity and high parameterization of the technique, makes it a challenging task to tune the technique for maximum performance gains.

Future research

Future research on clustered schema matching goes in two directions. First direction tries to improve the performance of the technique, while the second direction tries to expand the capabilities and the applicability of the technique.

The performance of the technique could be improved by (1) using different clustering algorithms for faster clustering and better clusters, (2) using different or adaptable distance measures (e.g., multi-feature distance measures) for better correlation with the objective function, (3) computing the *cluster quality index* to predict the quality of mappings enclosed within clusters. Using this index, clusters could be ranked to improve the time-to-first good schema mapping.

The applicability of the technique can be improved by (1) supporting partial mappings and inter-cluster mappings, (2) supporting large personal schemas, (3) using the technique on schema-graphs (instead of schema-trees), (4) using the technique with "non-standard" schema matching techniques, such as, holistic matching.

6.4 Validating effectiveness

Summary

This thesis shows how to improve the efficiency of some *original*, i.e., *non-clustered* schema matching system by extending it to a clustered schema matching system. The technique improves efficiency but reduces the effectiveness of the original system. While efficiency is simple to measure, effectiveness measurement is a challenge.

Effectiveness of a search system is traditionally expressed by means of precision and recall. This requires human efforts which become very expensive (i.e., time consuming) in large scale test environments.

To sufficiently overcome this problem, this thesis relies on an automated method for effectiveness assessment. This method exploits the fact that clustered schema matching is built on top of an existing schema matching system. The method uses only the relation between the answer sets produced by the clustered and the non-clustered systems, i.e., the *answer set ratio* curve, to assess the effectiveness of the clustered schema matching technique. In order to show that the answer set ratio curve is a reflection of the actual precision and recall of the improved system, a computational method is presented which converts, under certain conditions, the answer set ratio curve, into lower and upper bounds for the precision recall curve (see Sec. 4.4).

Contributions and conclusions

This part of the thesis contributes, under certain restrictions,

- *the computational method for establishing effectiveness bounds.*

Effectiveness bounds can be computed without human involvement under the assumption that the effectiveness of the original system, e.g., acquired on small scale test sets, is known. Also, both the original and the improved system must be available for testing on large scale test collections.

The technique is limited in the sense that it only determines the bounds, hence does not provide the actual effectiveness, the determination of which still requires human effort.

Future work

The effectiveness bounds computational technique is exact, and does not require validation of correctness. Nevertheless, apart from its clear theoretical value, the technique still has to show its practical utility. For example, it would be interesting to see where the actual precision and recall curve of improved systems lie within the bounds computed with this technique. In schema matching systems, the experimental data needed to make such an observation is hard to come by. This can be solved by testing the technique in a more traditional IR community, e.g., text retrieval, in which large test collections exist.

Appendix A

The experimental framework

A.1 Introduction

Bellflower is the experimental system used for the validation of the clustered schema matching technique. The implementation of Bellflower proved to be a challenging engineering task. This appendix provides insight in the implementation of Bellflower and the experimental framework around it. Sec. A.2, introduces the experimental framework of which Bellflower is part. Sec. A.3 lifts the hood of Bellflower, and shows its main components, their responsibilities, and discusses some implementation details.

A.2 Experimental framework

Fig. A.1 illustrates the experimental framework. The components in the framework are divided in those responsible for executing the experiments (left of the dividing line), and those responsible for analyzing the results of the experiments (right of the dividing line).

Experiment execution
Bellflower is a program responsible for executing the whole clustered schema matching workflow (see Sec. 5.2.1). Its work is externally controlled by means of *Bellflower script files* `(*.bfscr)` (a). Scripts are written in Bellflower's custom scripting language and are used to specify the sequence of actions, the parameters of various algorithms, the locations of input data, and the locations for storing the results.

Fig. A.1 also illustrates the two inputs without which experiments cannot be performed: the personal schema (b) and the repository (c). The personal schema is a regular XML schema file `(*.xsd)`, while the repository comes as a `(*.rep)` file, which is the Bellflower's file format for saving schema-graphs (i.e., schema-trees). Sec. 5.4 describes how repositories are formed.

For each experiment, Bellflower creates several *Microsoft® Access* database files (`*.mdb`) (d) in which it stores the data generated during the experiment. This data includes sets of mapping elements, the composition of clusters, sets of

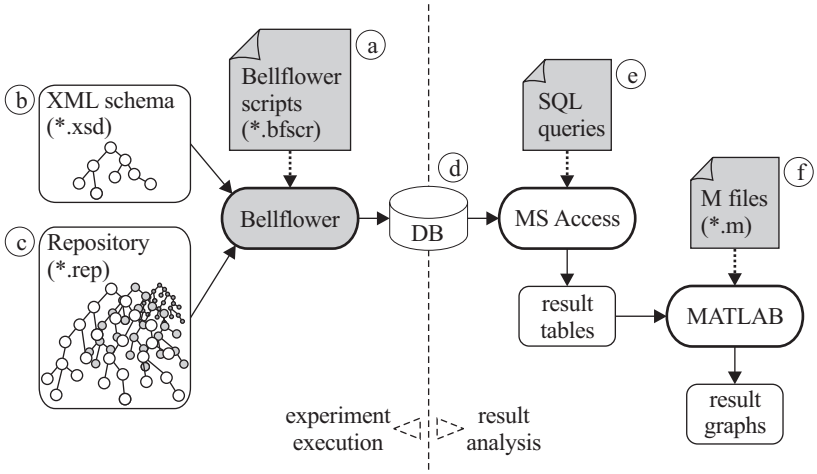


Figure A.1 *The experimental framework.*

discovered schema mappings, and performance indicators, that is, values of various timers and counters.

Result analysis

To extract useful data, the raw data stored in the result databases (d) needs further processing. For this the Microsoft Access database engine and four sets of SQL queries (e) are used.

- *Repository statistics queries.* Among others, these queries compute (1) the number of distinct element names in the repository, (2) the schema size distribution, and (3) the average number of elements per schema and per schema depth level.
- *Mapping element statistics queries.* These compute (1) the sizes of sets of mapping elements, (2) the name similarity index distribution of mapping elements, and (3) the size of the non-clustered search space.
- *Cluster statistics queries.* These compute (1) the average cluster size, (2) the cluster size distribution, (3) the number useful clusters, and (4) the size of the clustered search space.
- *Schema mapping statistics queries.* These compute (1) the similarity index distribution of the discovered schema mappings, and (2) the answer set ratio at various thresholds (e.g., see Fig. 5.7).

These queries produce all the tabular experimental results presented in this thesis. For graphical representation, tabular data was exported to MATLAB where it was further processed and displayed by means of MATLAB programs: the so

called M-files (`*.m`). An M-file also implements the computation of the effectiveness bounds discussed in chapter 4.

A.3 Architecture of Bellflower

Bellflower is developed in C++. Bellflower's source contains 25000 lines of code and includes several third-party libraries. Main tools used for Bellflower's development are *Microsoft Visual C++* for coding, compiling, and debugging, and *Rational Rose* for class diagram modeling, automatic code generation, and documenting.

Fig. A.2 shows the architecture of Bellflower. The gray ovals represent Bellflower's components which are developed from the scratch, while the white rectangles stand for third party libraries or tools integrated with Bellflower. This section discusses the responsibilities of these components and presents some implementation details.

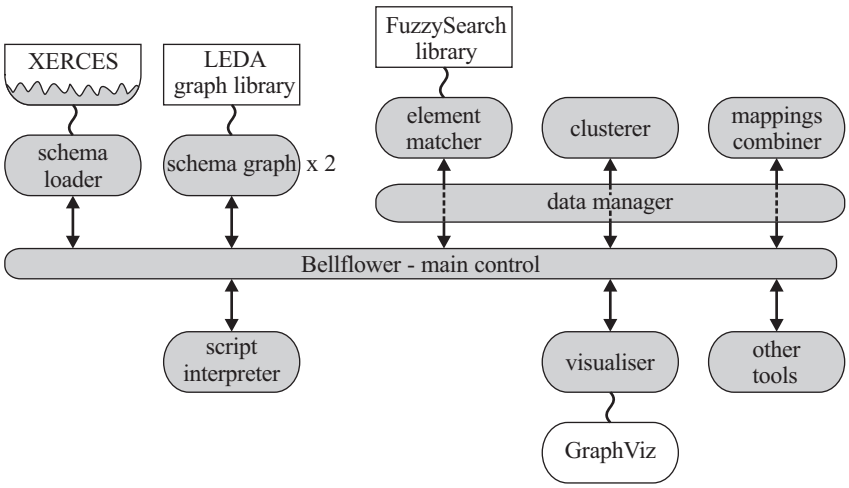


Figure A.2 *The architecture of Bellflower.*

Bellflower main control

Main control is the component responsible for holding the Bellflower system together. It bootstraps the whole system, that is, it creates and links together all other objects. It is also responsible for dispatching script commands to appropriate components for execution.

Script interpreter
Bellflower is controlled through scripts. It is the task of the *script interpreter* component to parse the script files, to resolve variable assignments, and to correctly sequence the commands, sending them one by one to the *main control* for execution.

The following is a self-explanatory excerpt from a Bellflower script. Note, variable names begin with `$d_` and are placed between quotes.

```
initMatching();                    //reset the matching system
setVar("$d_NNSThr", 40);           //set a variable $d_NNSThr=40
openRepository("repD.rep");        //open a repository
labelTree();                       //compute node labels
parseXsdFile("pSchema.xsd");       //load a personal schema
setIntParam(21, "$d_NNSThr");      //set param. ID=21 to 40(/100)
doNodeNameMatching();              //perform element matching
```

The two "magic numbers" in the code mean the following: number 40 becomes the value of the name similarity threshold $\delta_{element} = \mathbf{40}/100$ (see Sec. 5.3.1). Number 21 is the hard-coded ID of the name similarity threshold parameter, i.e., the command `setIntParam(21, "$d_NNSThr");` assigns the value of the variable `$d_NNSThr` to parameter with the ID 21.

In the experimental framework, the development of manageable experimental scripts is a challenge of its own. To enable modular script development, the Bellflower script language supports the use of global *variables* (i.e., `setVar(`*varName, varValue*`)`), the invocation of other scripts (i.e., `runScript(`*scriptFile*`)`), and simple program flow control (i.e., `skipCommands(`n`)` which skips n commands).

Schema loader
Schema loader is the component responsible for parsing the XML schema files (`*.xsd`) and creating the schema-graph (see Def. 1). At the time of development, several libraries offering *validated XML document parsing* were available; validated parsing checks if the XML document conforms to a specified XML schema. However, these libraries offered programming interfaces for accessing the XML document only, and not the associated XML schema.

To avoid developing an XML schema parser from scratch, Bellflower's XML schema parser is built by extending the open source *Xerces C++ validating XML parser*[1]. First development attempt tried to directly use Xerces' data structure which store the parsed XML schema. Due to unavailable documentation, this approach did not bring results. Instead, a different approach is used: Xerces is modified to become a SAX parser for XML schemas. To achieve this functionality, Xerces code for XML schema parsing, is extended to be able to sends messages to

[1] see *Xerces C++ XML parser* at `http://xml.apache.org/xerces-c`

Bellflower's *schema loader*. The modified Xerces sends three kinds of messages to the loader.

- XSD_CONSTRUCT_START
  This message informs the schema loader that a new XML schema construct has just been discovered by Xerces' schema parser, for example, an `element`, a `complexType`, or a `group`.
- XSD_CONSTRUCT_DATA_AVAILABLE(data)
  This message informs the schema loader that all the data related to the construct is known to Xerces. This data is sent to Bellflower's schema loader together with the message. Depending on the type of the construct, this data includes some of the following: type of the construct (e.g., <`xs : element`>, <`xs : complexType`>), name of the construct (e.g., `name="airport"`, `name="publicationType"`), cardinality (e.g., `maxOccurs="5"`), simple data type (e.g., `type="xs:string"`), name of the referred global element (e.g. `ref="Department"`).
- XSD_CONSTRUCT_END
  This message informs the schema loader that the current XML schema construct is closed.

In this way, the hard work such as namespace resolving, or XML schema inclusion, is still performed by Xerces. The schema loader only collects the useful information. However, the schema loader has an additional task of converting this message stream into a schema-graph.

Schema-graph

Two schema-graph objects are used in Bellflower. One for the repository, the other for the personal schema. Schema-graphs are built on top of LEDA 4.2 graph library[2]. A schema-graph object provides services such as graph navigation, node and edge iteration and access to stored graph data, graph merging, graph editing, save and load functionality. Some LEDA graph algorithms are also used: for example, to check if the parsed schema has cyclic structure.

Element matcher, clusterer, mappings combiner

The three steps of the clustered schema matching workflow are implemented as separate components. In addition, a *data manager* component stores and serves intermediate results.

The *element matcher* implements the node name matcher described in Sec. 5.3.1. The *clusterer* implements the k-means clustering algorithm described in Sec. 3.4.2 and Sec. 5.3.3, and the mappings combiner implements the branch and bound algorithm given in Sec. 5.3.2. Among the three, the *clusterer* is the most complex

[2]see LEDA 4.2 at `http://www.mpi-sb.mpg.de/LEDA`

component. To give an impression of how the clusterer is designed in a customizable and extensible way, Fig. A.3 shows the clusterer's UML class diagram (note, names in the diagram are changed for readability purposes).

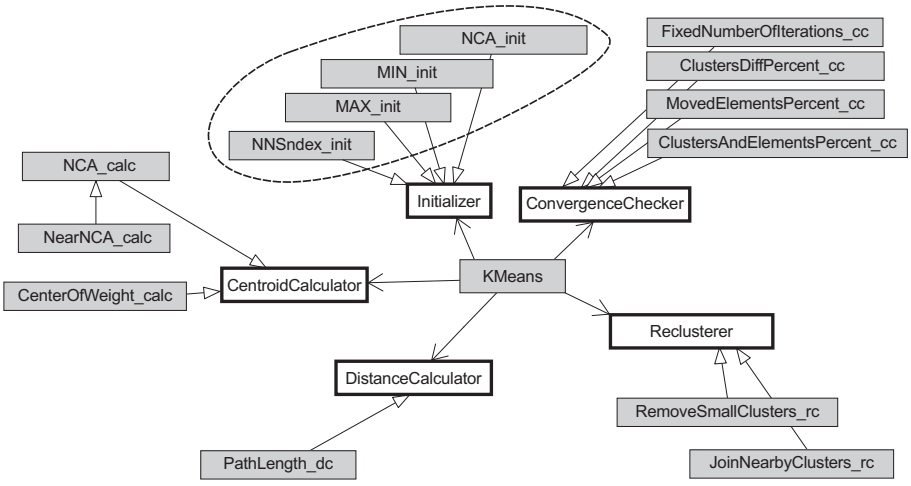


Figure A.3 *UML class diagram of the clusterer.*

White rectangles in the diagram represent interfaces (i.e., in C++ interfaces are implemented as virtual classes) through which the skeleton of the k-means algorithm is allowed to communicate with other five components of the algorithm: `Initializer`, `DistanceCalculator`, `CentroidCalculator`, `Reclusterer`, and `ConvergenceChecker`. The gray rectangles are classes which implement the five interfaces. For example, the four classes circled with a dashed line are the four different initialization techniques we have experimented with. The experiments in this thesis use only two: *MIN-init* and the *NCA-init* initialization techniques.

Visualizer and other tools

The *visualizer component* is responsible for generating graphical representations of XML schemas. For this it uses the graph visualization software GraphViz[3]. The visualizer can display the personal schema and the repository, together with the mapping elements within the repository, and the clusters. Fig. 5.3 on page 97 shows a sample schema drawn using the *visualizer*.

Bellflower contains a number of smaller supporting components, such as a node labeler, performance measurement components, and database access components. These are all represented with the *other tools* block in Fig. A.2.

[3]see Graphviz at `http://www.graphviz.org`

Bibliography

[1] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: A survey and a new distributed algorithm. In *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002.

[2] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, pages 496–513, London, UK, 2002. Springer-Verlag.

[3] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. Flexpath: flexible structure and full-text querying for xml. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 83–94, New York, NY, USA, 2004. ACM Press.

[4] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD Conference*, pages 906–908, 2005.

[5] R. Bartak. Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS)*, pages 555–564, June 1999.

[6] M. Beigbeder and A. Imafouo. An experimental methodology to study collections size impact on retrieval effectiveness. In *DIR '05: Proc. of 5th Dutch-Belgian Information Retrieval Workshop*, Jan. 2005.

[7] G. Bellinger, D. Castro, and A. Mills. Data, information, knowledge, and wisdom, 1997. http://www.outsights.com/systems/dikw/dikw.htm.

[8] A. Bergholz and J. C. Freytag. Querying Semistructured Data Based on Schema Matching. *Lecture Notes in Computer Science*, 1949, 2000.

[9] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-Strength Schema Matching. *SIGMOD Rec.*, 33(4):38–43, 2004.

[10] T. A. Brooks. Web Search: how the Web has changed information retrieval. *Information Research*, 8(3):paper no. 154, 2003.

[11] C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *Proc. of the 27th Int. Conf. on Research and Development in Information Retrieval SIGIR)*, pages 25–32. ACM Press, 2004.

[12] S. Castano, V. D. Antonellis, and S. D. C. di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.

[13] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: observations and implications. *SIGMOD Rec.*, 33(3):61–70, 2004.

[14] C. Chekuri, M. Goldwasser, P. Raghavan, and E. Upfal. Web search using automated classification. In *Proc. of 6th International World Wide Web Conference*, 1997.

[15] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. Xsearch: A semantic search engine for xml. In *VLDB2003*, pages 45–56, 2003.

[16] Computer Industry Almanac. Worldwide Internet Users will Top 1 Billion in 2005, September 2004. http://www.c-i-a.com/pr0904.htm.

[17] B. Convent. Unsolvable problems related to the view integration approach. In *Proceedings on International conference on database theory*, pages 141–156, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[18] B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *The VLDB Conference*, pages 341–350, 2001.

[19] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: discovering complex semantic matches between database schemas. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394. ACM Press, 2004.

[20] Y. Ding and S. Foo. Ontology research and development. Part 1 - a review of ontology generation. *Journal of Information Science*, 28(2):123–136, 2002.

[21] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases*, 2002.

[22] H. H. Do and E. Rahm. COMA — A System for Flexible Combination of Schema Matching Approaches. In P. A. Bernstein et al., editors, *VLDB 2002: proceedings of the Twenty-Eighth International Conference on Very Large Data Bases, Hong Kong SAR, China, 20–23 August 2002*, pages 610–621, Los Altos, CA 94022, USA, 2002. Morgan Kaufmann Publishers.

[23] A. Doan. *Learning to Map between Structured Representations of Data.* PhD thesis, University of Washington, 2002.

[24] A. Doan, P. Domingos, and A. Halevy. Learning to Match the Schemas of Data Sources: A Multistrategy Approach. *Machine Learning*, 50(3):279–301, 2003.

[25] A. Doan and A. Y. Halevy. Semantic integration research in the database community: A brief survey. *AAAI AI Magazine, Special Issue on Semantic Integration*, 26(1):83–94, 2005.

[26] M. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2003.

[27] R. M. Duwairi. Clustering semantically related classes in a heterogeneous multidatabase system. *Inf. Sci.*, 162(3-4):193–210, 2004.

[28] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 3rd ed.* Addison Wesley, 2000.

[29] Erhard Rahm and Hong-Hai Do and Sabine Mamann. Matching Large XML Schemas. *SIGMOD Rec.*, 33(4):26–31, 2004.

[30] L. Floridi. From data to semantic information. *Entropy*, 5(5):124–145, 2003.

[31] W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[32] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 252–257, New York, NY, USA, 1983. ACM Press.

[33] N. Fuhr and K. Großjohann. XIRQL – an extension of XQL for information retrieval. ACM, 2000.

[34] N. Fuhr, M. Lalmas, S. Malik, and Z. Szlavik, editors. *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl Castle, Germany, December 6-8, 2004, Revised Selected Papers*, volume 3493. Springer-Verlag GmbH, may 2005. http://www.springeronline.com/3-540-26166-4.

[35] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *The VLDB Journal*, 14(1):50–67, 2005.

[36] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. *Lecture Notes in Computer Science*, 2161, 2001.

[37] K. Gibran. *The Prophet*. Knopf, September 1923.

[38] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB 2001*, pages 491–500, 2001.

[39] D. Harman. Overview of the first text retrieval conference. In *Proc. of the 16th Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 36–47. ACM Press, June 1993.

[40] B. He and K. C.-C. Chang. A holistic paradigm for large scale schema matching. *SIGMOD Rec.*, 33(4):20–25, 2004.

[41] B. He, T. Tao, and K. C.-C. Chang. Organizing structured web sources by query schemas: a clustering approach. In *CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pages 22–31. ACM Press, 2004.

[42] M. R. Henzinger and V. King. Maintaining Minimum Spanning Trees in Dynamic Graphs. In *Automata, Languages and Programming*, pages 594–604, 1997.

[43] R. Hull. Managing semantic heterogeneity in databases: a theoretical prospective. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 51–61. ACM Press, 1997.

[44] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[45] E. M. C. Jr. and O. Grumberg. *Model Checking*. The MIT Press, 1999.

[46] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. *Lecture Notes in Computer Science*, 2125, 2000.

[47] H. Kaplan, T. Milo, and R. Shabo. A comparison of labeling schemes for ancestor queries. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 954–963. Society for Industrial and Applied Mathematics, 2002.

[48] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Comput. Surv.*, 32(4):422–469, Dec. 2000.

[49] D. L. Kreher and D. R. Stinson. *Combinatorial algorithms : generation, enumeration, and search*. CRC Press LLC, Boca Raton, Florida, 1999.

[50] D. Lee and W. W. Chu. Comparative analysis of six xml schema languages. *SIGMOD Rec.*, 29(3):76–87, 2000.

[51] J.-O. Lee and D.-K. Baik. Semql: a semantic query language for multidatabase systems. In *CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*, pages 259–266, New York, NY, USA, 1999. ACM Press.

[52] M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 292–299. ACM Press, 2002.

[53] Y. Li, C. Yu, and H. V. Jagadish. Schema-free xquery. In *VLDB*, 2004.

[54] Y. Lu, S. Lu, F. Fotouhi, Y. Deng, and S. J. Brown. Incremental genetic k-means algorithm and its application in gene expression data analysis. *BMC Bioinformatics*, 5, 2004.

[55] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68. IEEE Computer Society, 2005.

[56] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and reasoning about mappings between domain models. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02)*, pages 80–86, Menlo Parc, CA, USA, July 28– Aug. 1 2002. AAAI Press.

[57] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01)*, pages 49–58, Orlando, Sept. 2001. Morgan Kaufmann.

[58] K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. MIT Press, 1998.

[59] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, February 2002.

[60] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, December 1999.

[61] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 77–88, 2000.

[62] T. Milo and D. Suciu. Index structures for path expressions. *Lecture Notes in Computer Science*, 1540:277–295, 1999.

[63] Netcraft. January 2006 Web Server Survey. http://news.netcraft.com.

[64] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, 2004.

[65] OCLC Online Computer Library Center. Introduction to Dewey Decimal Classification. http://www.oclc.org/dewey.

[66] T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[67] C. Pluempitiwiriyawej and J. Hammer. Element matching across data-oriented XML sources using a multi-strategy clustering model. *Data Knowl. Eng.*, 48(3):297–333, 2004.

[68] N. Polyzotis and M. Garofalakis. XCluster Synopses for Structured XML Content. In *Proc. 22nd Int. Conf. on Data Engineering (ICDE)*. IEEE Comp. Soc. Dig. Library, 2006.

[69] V. V. Raghavan. Approaches for measuring the stability of clustering methods. *SIGIR Forum*, 17(1):6–20, 1982.

[70] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, Dec. 2001.

[71] RFC Editor. Official Internet Protocol Standards. http://www.rfc-editor.org/rfcxx00.html.

[72] C. J. v. Rijsbergen. *Information Retrieval*. Butterworths, 1979. See also: http://www.dcs.glasgow.ac.uk/Keith/.

[73] M. Sanderson and H. Joho. Forming test collections with no system pooling. In *Proc. of the 27th Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 33–40. ACM Press, 2004.

[74] M. Sayyadian, Y. Lee, A. Doan, and A. S. Rosenthal. Tuning schema matching software using synthetic scenarios. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 994–1005. VLDB Endowment, 2005.

[75] T. Schlieder and F. Naumann. Approximate tree embedding for querying xml data. In *Proceedings of the ACM SIGIR Workshop On XML and Information Retrieval*, July 2000.

[76] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[77] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. 22(3):183–236, Sept. 1990.

[78] P. Shvaiko and J. Euzenat. A Survey of Schema-based Matching Approaches. Technical Report DIT-04-087, Informatica e Telecomunicazioni, University of Trento, Italy, 2004.

[79] M. Smiljanić, H. Blanken, M. van Keulen, and W. Jonker. Distributed XML Database Systems. Technical Report TR-CTIT-02-46, CTIT, Oct. 2002.

[80] M. Smiljanić, L. Feng, and W. Jonker. *Web-Based Distributed XML Query Processing*. Number 2818 in LNCS. Springer-Verlag, 2003.

[81] M. Smiljanić, M. van Keulen, and W. Jonker. Defining the XML Schema Matching Problem for a Personal Schema Based Query Answering System. Technical Report TR-CTIT-04-17, Centre for Telematics and Information Technology, Apr. 2004.

[82] M. Smiljanić, M. van Keulen, and W. Jonker. Formalizing the XML Schema Matching Problem as a Constraint Optimization Problem. In *Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA2005), Copenhagen, Denmark*, Aug. 2005.

[83] M. Smiljanić, M. van Keulen, and W. Jonker. Effectiveness Bounds for Non-Exhaustive Schema Matching Systems. In *Proceedings of ICDE Workshop on XML Schema and Data Management (XSDM'06), Atlanta, GA, USA*, 2006.

[84] M. Smiljanić, M. van Keulen, and W. Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *Proceedings of ICDE Workshop on Challenges in Web Information Retrieval and Integration (WIRI'06), Atlanta, GA, USA*, 2006.

[85] U. Srinivasan, A. H. H. Ngu, and T. Gedeon. Managing heterogeneous information systems through discovery and retrieval of generic concepts. *J. Am. Soc. Inf. Sci.*, 51(8):707–723, 2000.

[86] D. Stenmark. The relationship between information and knowledge. In *IRIS 24*, 2001.

[87] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *Extending Database Technology*, pages 477–495, 2002.

[88] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 648–659. Morgan Kaufmann, 2004.

[89] A. Tombros, R. Villa, and C. J. V. Rijsbergen. The effectiveness of query-specific hierarchic clustering in information retrieval. *Inf. Process. Manage.*, 38(4):559–582, 2002.

[90] A. Trotman and R. A. O'Keefe. The Simplest Query Language That Could Possibly Work. In N. Fuhr, M. Lalmas, and S. Malik, editors, *Proceedings of the Second Workshop of the INitiative for the Evaluation of XML retrieval (INEX)*, ERCIM Publications, 2004.

[91] A. Vakali, J. Pokorn, and T. Dalamagas. An overview of web data clustering practices. In A. V. W. Lindner, M. Mesiti, C. Türker, Y. Tzitzikas, editor, *In EDBT 2004 Workshops, Greece, March 14-18*, volume 3268 of *Lecture Notes in Computer Science (LNCS)*, Berlin, 2004. Springer-Verlag.

[92] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.

[93] World Wide Web Consortium. Various documents. http://www.w3.org.

[94] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 95–106, New York, NY, USA, 2004. ACM Press.

[95] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 443–454. ACM Press, 2004.

[96] H. Zhao and S. Ram. Clustering schema elements for semantic integration of heterogeneous data sources. *Journal of Database Management*, 15(4):88–106, 2004.

[97] J. Zobel. How reliable are the results of large-scale information retrieval experiments? In *Proc. of the 21st Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 307–314. ACM Press, Aug. 1998.

SIKS Dissertation Series

1998-01 Johan van den Akker (CWI), *DEGAS - An Active, Temporal Database of Autonomous Objects*

1998–02 Floris Wiesman (UM), *Information Retrieval by Graphically Browsing Meta-Information*

1998-03 Ans Steuten (TUD), *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

1998-04 Dennis Breuker (UM), *Memory versus Search in Games*

1998-05 E.W.Oskamp (RUL), *Computerondersteuning bij Straftoemeting*

1999-01 Mark Sloof (VU), *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*

1999-02 Rob Potharst (EUR), *Classification using decision trees and neural nets*

1999-03 Don Beal (UM), *The Nature of Minimax Search*

1999-04 Jacques Penders (UM), *The practical Art of Moving Physical Objects*

1999-05 Aldo de Moor (KUB), *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

1999-06 Niek J.E. Wijngaards (VU), *Re-design of compositional systems*

1999-07 David Spelt (UT), *Verification support for object database design*

1999-08 Jacques H.J. Lenting (UM), *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*

2000-01 Frank Niessink (VU), *Perspectives on Improving Software Maintenance*

2000-02 Koen Holtman (TUE), *Prototyping of CMS Storage Management*

2000-03 Carolien M.T. Metselaar (UVA), *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.*

2000-04 Geert de Haan (VU), *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

2000-05 Ruud van der Pol (UM), *Knowledge-based Query Formulation in Information Retrieval.*

2000-06 Rogier van Eijk (UU), *Programming Languages for Agent Communication*

2000-07 Niels Peek (UU), *Decision-theoretic Planning of Clinical Patient Management*

2000-08 Veerle Coup (EUR), *Sensitivity Analyis of Decision-Theoretic Networks*

2000-09 Florian Waas (CWI), *Principles of Probabilistic Query Optimization*

2000-10 Niels Nes (CWI), *Image Database Management System Design Considerations, Algorithms and Architecture*

2000-11 Jonas Karlsson (CWI), *Scalable Distributed Data Structures for Database Management*

2001-01 Silja Renooij (UU), *Qualitative Approaches to Quantifying Probabilistic Networks*

2001-02 Koen Hindriks (UU), *Agent Programming Languages: Programming with Mental Models*

2001-03 Maarten van Someren (UvA), *Learning as problem solving*

2001-04 Evgueni Smirnov (UM), *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

2001-05 Jacco van Ossenbruggen (VU), *Processing Structured Hypermedia: A Matter of Style*

2001-06 Martijn van Welie (VU), *Task-based User Interface Design*

2001-07 Bastiaan Schonhage (VU), *Diva: Architectural Perspectives on Information Visualization*

2001-08 Pascal van Eck (VU), *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*

2001-09 Pieter Jan 't Hoen (RUL), *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

2001-10 Maarten Sierhuis (UvA), *Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*

2001-11 Tom M. van Engers (VUA), *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002-01 Nico Lassing (VU), *Architecture-Level Modifiability Analysis*

2002-02 Roelof van Zwol (UT), *Modelling and searching web-based document collections*

2002-03 Henk Ernst Blok (UT), *Database Optimization Aspects for Information Retrieval*

2002-04 Juan Roberto Castelo Valdueza (UU), *The Discrete Acyclic Digraph Markov Model in Data Mining*

2002-05 Radu Serban (VU), *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*

2002-06 Laurens Mommers (UL), *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*

2002-07 Peter Boncz (CWI), *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

2002-08 Jaap Gordijn (VU), *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

2002-09 Willem-Jan van den Heuvel(KUB), *Integrating Modern Business Applications with Objectified Legacy Systems*

2002-10 Brian Sheppard (UM), *Towards Perfect Play of Scrabble*

2002-11 Wouter C.A. Wijngaards (VU), *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

2002-12 Albrecht Schmidt (Uva), *Processing XML in Database Systems*

2002-13 Hongjing Wu (TUE), *A Reference Architecture for Adaptive Hypermedia Applications*

2002-14 Wieke de Vries (UU), *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

2002-15 Rik Eshuis (UT), *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

2002-16 Pieter van Langen (VU), *The Anatomy of Design: Foundations, Models and Applications*

2002-17 Stefan Manegold (UVA), *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003-01 Heiner Stuckenschmidt (VU), *Ontology-Based Information Sharing in Weakly Structured Environments*

2003-02 Jan Broersen (VU), *Modal Action Logics for Reasoning About Reactive Systems*

2003-03 Martijn Schuemie (TUD), *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

2003-04 Milan Petkovic (UT), *Content-Based Video Retrieval Supported by Database Technology*

2003-05 Jos Lehmann (UVA), *Causation in Artificial Intelligence and Law - A modelling approach*

2003-06 Boris van Schooten (UT), *Development and specification of virtual environments*

2003-07 Machiel Jansen (UvA), *Formal Explorations of Knowledge Intensive Tasks*

2003-08 Yongping Ran (UM), *Repair Based Scheduling*

2003-09 Rens Kortmann (UM), *The resolution of visually guided behaviour*

2003-10 Andreas Lincke (UvT), *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*

2003-11 Simon Keizer (UT), *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

2003-12 Roeland Ordelman (UT), *Dutch speech recognition in multimedia information retrieval*

2003-13 Jeroen Donkers (UM), *Nosce Hostem - Searching with Opponent Models*

2003-14 Stijn Hoppenbrouwers (KUN), *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

2003-15 Mathijs de Weerdt (TUD), *Plan Merging in Multi-Agent Systems*

2003-16 Menzo Windhouwer (CWI), *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses*

2003-17 David Jansen (UT), *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

2003-18 Levente Kocsis (UM), *Learning Search Decisions*

2004-01 Virginia Dignum (UU), *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2004-02 Lai Xu (UvT), *Monitoring Multi-party Contracts for E-business*

2004-03 Perry Groot (VU), *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*

2004-04 Chris van Aart (UVA), *Organizational Principles for Multi-Agent Architectures*

2004-05 Viara Popova (EUR), *Knowledge discovery and monotonicity*

2004-06 Bart-Jan Hommes (TUD), *The Evaluation of Business Process Modeling Techniques*

2004-07 Elise Boltjes (UM), *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*

2004-08 Joop Verbeek(UM), *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise*

2004-09 Martin Caminada (VU), *For the Sake of the Argument; explorations into argument-based reasoning*

2004-10 Suzanne Kabel (UVA), *Knowledge-rich indexing of learning-objects*

2004-11 Michel Klein (VU), *Change Management for Distributed Ontologies*

2004-12 The Duy Bui (UT), *Creating emotions and facial expressions for embodied agents*

2004-13 Wojciech Jamroga (UT), *Using Multiple Models of Reality: On Agents who Know how to Play*

2004-14 Paul Harrenstein (UU), *Logic in Conflict. Logical Explorations in Strategic Equilibrium*

2004-15 Arno Knobbe (UU), *Multi-Relational Data Mining*

2004-16 Federico Divina (VU), *Hybrid Genetic Relational Search for Inductive Learning*

2004-17 Mark Winands (UM), *Informed Search in Complex Games*

2004-18 Vania Bessa Machado (UvA), *Supporting the Construction of Qualitative Knowledge Models*

2004-19 Thijs Westerveld (UT), *Using generative probabilistic models for multimedia retrieval*

2004-20 Madelon Evers (Nyenrode), *Learning from Design: facilitating multidisciplinary design teams*

2005-01 Floor Verdenius (UVA), *Methodological Aspects of Designing Induction-Based Applications*

2005-02 Erik van der Werf (UM), *AI techniques for the game of Go*

2005-03 Franc Grootjen (RUN), *A Pragmatic Approach to the Conceptualisation of Language*

2005-04 Nirvana Meratnia (UT), *Towards Database Support for Moving Object data*

2005-05 Gabriel Infante-Lopez (UVA), *Two-Level Probabilistic Grammars for Natural Language Parsing*

2005-06 Pieter Spronck (UM), *Adaptive Game AI*

2005-07 Flavius Frasincar (TUE), *Hypermedia Presentation Generation for Semantic Web Information Systems*

2005-08 Richard Vdovjak (TUE), *A Model-driven Approach for Building Distributed Ontology-based Web Applications*

2005-09 Jeen Broekstra (VU), *Storage, Querying and Inferencing for Semantic Web Languages*

2005-10 Anders Bouwer (UVA), *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*

2005-11 Elth Ogston (VU), *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*

2005-12 Csaba Boer (EUR), *Distributed Simulation in Industry*

2005-13 Fred Hamburg (UL), *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*

2005-14 Borys Omelayenko (VU), *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*

2005-15 Tibor Bosse (VU), *Analysis of the Dynamics of Cognitive Processes*

2005-16 Joris Graaumans (UU), *Usability of XML Query Languages*

2005-17 Boris Shishkov (TUD), *Software Specification Based on Re-usable Business Components*

2005-18 Danielle Sent (UU), *Test-selection strategies for probabilistic networks*

2005-19 Michel van Dartel (UM), *Situated Representation*

2005-20 Cristina Coteanu (UL), *Cyber Consumer Law, State of the Art and Perspectives*

2005-21 Wijnand Derks (UT), *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006-01 Samuil Angelov (TUE), *Foundations of B2B Electronic Contracting*

2006-02 Cristina Chisalita (VU), *Contextual issues in the design and use of information technology in organizations*

2006-03 Noor Christoph (UVA), *The role of metacognitive skills in learning to solve problems*

2006-04 Marta Sabou (VU), *Building Web Service Ontologies*

2006-05 Cees Pierik (UU), *Validation Techniques for Object-Oriented Proof Outlines*

2006-06 Ziv Baida (VU), *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*

2006-07 Marko Smiljanić (UT), *XML schema matching – balancing efficiency and effectiveness by means of clustering*

Summary

XML schema matching – balancing efficiency and effectiveness by means of clustering

To provide information to humans has been both the greatest achievement and the most challenging problem of the Internet. While machines continue to handle data with increasing speeds, their inability to understand what data means, i.e., the *semantic gap*, nowadays consumes large amount of research. In the effort to bridge the gap, research has proposed novel techniques for information representation and processing. On the representation side, the addition of metadata and structure is proposed to evolve simple text and HTML to semantically richer XML and RDF. On the processing side, new ways for using rich data representations are investigated. Various heuristics, advanced learning algorithms, and inference are explored to bring machines closer to how humans understand information.

In this thesis we place our research in the scope of a tool which looks for information within XML data on the Internet. We envision a *personal schema querying system* which enables a user to express his information need by specifying a personal XML schema. The user can also ask queries over his personal schema. The first task the personal schema querying system must perform is to match the personal schema against the real schemas of the XML data on the Internet. In this way, the system establishes a meaningful link between the user's information need and the data available on the Internet. The second task is the evaluation of the personal query, which is performed by means of distributed query processing.

Of these two tasks, the second one can be fulfilled by contemporary distributed query processing techniques which are capable of operating in large scale on-demand environments. The first task needs to be performed by a schema matcher. However, current schema matching techniques can only handle smaller off-line matching problems. Consequently, schema matching presents a serious performance bottleneck in personal schema querying. In this thesis we investigate ways to eliminate this bottleneck by improving the efficiency of schema matching. In doing so, we confront three challenges.

The first challenge is to understand the sources of schema matching complexity. Schema matching research largely focuses on inventing new approaches for improving the effectiveness of matching systems with almost no work addressing formal specification of the matching problem which is actually being solved by

these systems. To acquire such formal understanding, we observe how existing schema matching systems work and identify the building blocks of schema matching problems. Further, we find that these can be formally specified using a known generic framework for problem solving – *constraint programming*. In particular we show how to represent a schema matching problem as a *constraint optimization problem*. The result is a clear understanding of the roots of schema matching complexity: among other, we observe the polynomial dependency of the problem search space on the size of the Internet schemas; this complexity we need to reduce to improve performance in presence of large Internet schemas. More benefits of this formalization approach come from a variety of available optimization techniques for problem solving offered by the constraint programming framework.

The second challenge is to propose a technique which improves the efficiency of schema matching. We observe that it is possible to identify regions in Internet schemas in which good mappings, for a particular personal schema, can be found. Hence, to solve the schema matching problem, the matching algorithm needs to look for mappings only in these regions. The efficiency is improved by not having to look for mappings in Internet schemas as a whole. This reduces the problem complexity from polynomial to linear, enabling faster matching in presence of large Internet schemas. In this thesis, we propose and investigate the use of clustering for identifying such interesting regions in the Internet schemas, hence the name of the technique – *clustered schema matching*. By means of a clustered schema matching prototype we validate that the technique indeed delivers the expected efficiency improvements. Along, we identify and analyze a number of sensitive issues which influence the technique. Beside the efficiency, validation of the technique includes the effectiveness measurements. This brings us to the third challenge.

The third challenge is the inapplicability of classical effectiveness measurements, that is, precision and recall. In large scale schema matching problems the number of possible mappings can be immense. For such problems, the use of human inspection which is required for measuring the precision and recall, is very expensive. To sufficiently overcome this problem, we propose an automated validation technique based on comparison of results delivered by an original schema matching system and the system which is improved by means of clustered schema matching. The technique uses no human inspection, and relies on best/worst case analysis to deliver precise effectiveness bounds. These bounds determine the area in the precision/recall graph within which the actual effectiveness of the clustered schema matching system is guaranteed to lie. The bounds provide enough information to observe the efficiency/effectiveness trade-off introduced by clustered schema matching technique, and also to observe the changes in effectiveness when applying different variants of the technique.

Samenvatting

XML schema matching - het balanceren van efficiëntie en effectiviteit m.b.v. clustering

Mensen voorzien van informatie is zowel het grootste succes als ook het meest uitdagende probleem van het Internet. Hoewel computers steeds sneller gegevens kunnen verwerken, is het niet kunnen begrijpen wat die gegevens betekenen, de zogenaamde "semantic gap", een probleem waaraan tegenwoordig veel onderzoek wordt gedaan. Het toevoegen van meta-gegevens en structuur aan tekst en HTML d.m.v. de semantisch rijkere talen XML en RDF is een bijdrage aan de kant van (meer betekenisvolle) informatierepresentatie. Daarnaast worden nieuwe manieren onderzocht voor het verwerken van deze semantisch rijkere gegevens, zoals heuristieken en geavanceerde algoritmen voor leren en redeneren die computers dichter moeten brengen bij hoe wij mensen informatie begrijpen.

Het onderzoek in dit proefschrift betreft het gericht zoeken van informatie in de beschikbare hoeveelheid XML-gegevens op het Internet. We voorzien een "personal schema querying system" dat een gebruiker in staat stelt zijn informatiebehoefte te specificeren m.b.v een persoonlijk schema. De gebruiker kan vragen stellen in termen van dit eigen schema. De eerste taak van het personal schema querying system is het zoeken naar een goede match op het Internet voor het persoonlijk schema. Op deze manier legt het systeem de benodigde relatie tussen de informatiebehoefte van de gebruiker en de beschikbare gegevens op het Internet. De tweede taak is dan het beantwoorden van de vraag van de gebruiker.

Voor de tweede taak voldoen bestaande gedistribueerde query-evaluatietechnieken. Deze zijn toepasbaar in grootschalige ad-hoc omgevingen zoals het Internet. Voor de eerste taak is een zogenaamde schema matcher nodig. Bestaande schema matching-technieken kunnen echter alleen maar kleinere matching-problemen aan. Schema matching is daarom een serieuze bottleneck in personal schema querying. Dit proefschrift onderzoekt technieken om deze bottleneck weg te nemen en daarmee de efficiëntie van schema matching te verbeteren. Drie uitdagingen spelen hierbij een belangrijke rol.

De eerste uitdaging is het grondig inzicht krijgen in de oorzaken van de complexiteit van schema matching. Bestaand onderzoek naar schema matching richt zich hoofdzakelijk op manieren om de effectiviteit te vergroten zonder veel aandacht voor formele analyse van het probleem dat men probeert op te lossen.

Het welbekende en generieke raamwerk van "constraint programming" blijkt zeer geschikt om inzicht op formeel niveau te krijgen in deelproblemen en de werking van schema-matching systemen. We laten zien hoe het schema matching-probleem gespecificeerd kan worden als een constraint-optimalisatieprobleem. De oorzaken van complexiteit worden hiermee duidelijk: er is o.a. een polynomiale afhankelijkheid tussen de grootte van de zoekruimte en de grootte van de XML schema's op het Internet. Deze complexiteit moet teruggebracht worden om de efficiëntie te kunnen vergroten voor grote Internet-schema's.

De tweede uitdaging omvat het daadwerkelijk ontwikkelen van een techniek die de efficiëntie van schema matching verbetert. Het blijkt mogelijk om snel gebieden in Internet-schema's te herkennen met hoge kans op goede matches voor een gegeven persoonlijk schema. De efficiëntieverbetering zit in het niet meer hoeven doorzoeken van de volledige Internet-schema's. De complexiteit gaat hiermee van polynomiaal naar lineair, wat sneller doorzoeken van enorme hoeveelheden grote Internet-schema's mogelijk maakt. Specifiek onderzoeken we het gebruik van clustering-technieken voor het herkennen van deze gebieden, vandaar de naam van de techniek: "clustered schema matching." M.b.v. een prototype laten we zien dat de techniek inderdaad de verwachte efficiëntieverbetering oplevert. Tegelijkertijd analyseren we hiermee een aantal kwesties die de efficiëntie en effectiviteit van de techniek beïnvloeden. Om ook dit te kunnen valideren, doen we ook de effectiviteitsmetingen wat ons bij de derde uitdaging brengt.

De derde uitdaging is de lastige toepasbaarheid van de klassieke technieken voor effectiviteitsmetingen op basis van precision en recall. In grootschalige schema matching-problemen zijn de gevonden hoeveelheden matches enorm. Het is echter onoverkomelijk arbeidsintensief om in deze context voldoende matches te kunnen laten evalueren door een mensen, iets wat wel nodig is voor de bepaling van precision en recall. Als voldoende oplossing voor dit probleem presenteren we een geautomatiseerde validatietechniek gebaseerd op het vergelijken van resultaten van een origineel schema-matching systeem met een verbeterde versie daarvan die werkt met clustered schema matching. De validatietechniek doet een best/worst case-analyse waarmee precieze effectiviteitsgrenzen berekend worden. De daadwerkelijke effectiviteitscurve van het clustered schema matching-systeem ligt gegarandeerd binnen het hiermee begrensde gebied in de precision/recall-grafiek. Zonder handmatige evaluaties geven de effectiviteitsgrenzen voldoende informatie om een beeld te krijgen van de balans tussen efficiëntie en effectiviteit en de invloed daarop door variaties op de techniek.