

# OBJECT MATCHING ON REAL-WORLD PROBLEMS

Von der Fakultät für Mathematik und Informatik  
der Universität Leipzig  
angenommene

## D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM  
(Dr. rer. nat.)

im Fachgebiet Informatik

vorgelegt von

Diplom-Informatikerin Hanna Köpcke  
geboren am 31. Dezember 1978 in München

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Erhard Rahm (Universität Leipzig)
2. Prof. Dr. Peter Christen (Canberra, Australien)

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am  
09. Mai 2014 mit dem Gesamtprädikat *magna cum laude*.



# Acknowledgements

My special thanks go to Prof. Dr. Erhard Rahm. He has supported and guided my scientific development. I owe him the interesting topic of my thesis. He always found the time to discuss with me my ideas. His valuable comments and suggestions added a great deal to the success of this work. With great engagement he supported my scientific publications.

I would not have been able to finish my thesis without financial support from different scholarships and grants, especially from the DFG (Deutsche Forschungsgemeinschaft - German Research Foundation). Hence, I thank Prof. Dr. Gerhard Brewka. In his function as the speaker of the Research Training Group "knowledge representation" he enabled me to pursue an important part of my work as a scholarship holder of the DFG.

I thank my colleagues of the database group and of the WDI lab for the great working atmosphere, fruitful discussions and the productive scientific cooperation. At this point I also would like to particularly point out our annual "workshop-vacations" in Zingst under the direction of Prof. Dr. Rahm.

Thanks to Kerstin Wurdinger and Salvatore Raunich for sparing a significant amount of time to proofread this thesis - it draws much from their comments.

Last but not least, I would like to thank my family. They supported me throughout my whole life in every respect.



# Abstract

Object matching (also referred to as duplicate identification, record linkage, entity resolution or reference reconciliation) is a crucial task for data integration and data cleaning. The task is to detect multiple representations of the same real-world object. This is a challenging task particularly for objects that are highly heterogeneous and of limited data quality, e.g., regarding completeness and consistency of their descriptions.

To gain a better overview about the current state of the art in object matching, we survey the existing frameworks and their evaluations. According to the defined criteria, we review various frameworks published in the literature. We characterize them in some detail and compare them with each other and with our own framework, FEVER.

With FEVER we introduce a new generic and comprehensive framework for object matching and comparative object matching evaluation. FEVER offers numerous operators for constructing non-learning as well as learning-based match workflows. Moreover, FEVER allows match approaches to be automatically executed and evaluated under different parameter configurations. Therefore FEVER sets the platform for conducting a comparative evaluation on the relative effectiveness and efficiency of alternate match approaches.

Despite the huge amount of recent research efforts on object matching there has not yet been such an evaluation. With FEVER we fill this gap and present an evaluation of existing implementations on challenging real-world match tasks. We use the FEVER framework to automatically execute the approaches and to find favourable parameter settings in a comparable way. We consider approaches both with and without using machine learning to find suitable parameterization and combination of similarity functions. In addition to approaches from the research community we also consider a state-of-the-art commercial object matching implementation. Our results indicate significant quality and efficiency differences between different approaches. We also find that some challenging matching tasks such as matching product offers from online shops are not sufficiently solved with conventional approaches based on the similarity of attribute values.

Furthermore, this thesis addresses the product offer matching problem. Product of-

---

fer matching is a special case of object matching that is needed to identify equivalent offers referring to the same real-world product. The thesis proposes a tailored overall approach for the product offer matching problem. The approach supports category-specific match strategies based on two pillars: a comprehensive preprocessing and machine learning. The preprocessing extracts and cleans new attributes usable for matching. In particular, the approach extracts and uses so-called product codes to identify products and distinguish them from similar product variations. After the preprocessing machine learning is employed to semi-automatically determine a match strategy utilizing several attributes and similarity functions.

# Contents

<b>I</b>	<b>Introduction</b>	<b>11</b>
<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Motivation . . . . .	13
1.2	Scientific contributions . . . . .	15
1.3	Outline . . . . .	17
<b>II</b>	<b>Object matching approaches</b>	<b>19</b>
<b>2</b>	<b>Preliminaries</b>	<b>21</b>
2.1	Object matching . . . . .	21
2.2	Evaluation measures . . . . .	25
2.3	Requirements for object matching frameworks . . . . .	27
<b>3</b>	<b>Blocking</b>	<b>29</b>
3.1	Blocking evaluation measures . . . . .	30
3.2	General idea . . . . .	30
3.3	Disjoint blocking . . . . .	31
3.4	Overlapping blocking . . . . .	32
3.5	Further approaches . . . . .	36
3.6	Concluding remarks . . . . .	37
<b>4</b>	<b>Matchers</b>	<b>39</b>
4.1	Similarity measures . . . . .	39

## CONTENTS

---

4.2	Context matchers . . . . .	45
<b>5</b>	<b>Combination of matchers</b>	<b>49</b>
5.1	Numerical approaches . . . . .	49
5.2	Rule-based approaches . . . . .	50
5.3	Learning-based approaches . . . . .	50
5.4	Workflow-based approaches . . . . .	53
<b>III</b>	<b>Comparison of object matching approaches</b>	<b>55</b>
<b>6</b>	<b>Comparison of existing object matching frameworks</b>	<b>57</b>
6.1	Frameworks without training . . . . .	58
6.2	Learning-based frameworks . . . . .	59
6.3	Hybrid frameworks . . . . .	61
6.4	Functional comparison . . . . .	62
6.5	Evaluation comparison . . . . .	65
6.6	Summary and discussion . . . . .	73
<b>7</b>	<b>FEVER - A framework for object matching</b>	<b>75</b>
7.1	System architecture . . . . .	76
7.2	Data structures . . . . .	77
7.3	Operators . . . . .	78
7.4	Training selection operators . . . . .	83
7.5	Match workflows . . . . .	88
7.6	Configuration strategies for comparative object matching evaluation .	90
7.7	Implementation and use . . . . .	91
7.8	Functional comparison with competitive frameworks . . . . .	95
<b>8</b>	<b>Comparative evaluation of object matching approaches with FEVER</b>	<b>97</b>
8.1	Evaluation match tasks . . . . .	98
8.2	Evaluation of non-learning workflows . . . . .	100



---

8.3	Evaluation of learning-based workflows . . . . .	107
8.4	Comparative evaluation of match approaches and frameworks . . . . .	114
<b>IV</b>	<b>Product offer matching</b>	<b>129</b>
<b>9</b>	<b>The product offer matching problem</b>	<b>131</b>
9.1	Problem outline . . . . .	131
9.2	Special challenges . . . . .	133
<b>10</b>	<b>Tailored product offer matching approaches</b>	<b>135</b>
10.1	Overall product offer matching workflow . . . . .	135
10.2	Product offer classification . . . . .	137
10.3	Manufacturer cleaning . . . . .	140
10.4	Product code extraction . . . . .	142
<b>11</b>	<b>Evaluation of product offer matching</b>	<b>145</b>
11.1	Evaluation dataset . . . . .	145
11.2	Evaluation of product offer classification . . . . .	147
11.3	Evaluation of product code extraction . . . . .	150
11.4	Evaluation of match quality . . . . .	151
<b>V</b>	<b>Closing</b>	<b>155</b>
<b>12</b>	<b>Summary</b>	<b>157</b>
<b>13</b>	<b>Future directions</b>	<b>159</b>
	<b>References</b>	<b>161</b>



# **Part I**

## **Introduction**



# 1

## Introduction

### 1.1 Motivation

The recent explosion of data produced and circulated by organizations on and beyond the internet coincides with an increase of data quality problems. As clean data are a key factor for all business critical analysis and business intelligence tasks, poor data quality can seriously hinder or damage the efficiency and effectiveness of organizations and businesses.

Multiple representations of the same real-world object in the data are particularly problematic. Examples of such so-called duplicates include multiple managed customers, different representations of the same product within an online catalogue, and single type proteins stored in many different scientific databases. Duplicates decrease the usability of data, causing unnecessary expenses, customer dissatisfaction and, incorrect performance indicators. In short they decrease data quality.

Data cleaning [117, 6, 104], also termed data cleansing or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve its quality. Object matching (also referred to as duplicate identification, record linkage, entity resolution or reference reconciliation) is a crucial task for data integration and cleaning. It is a particularly challenging task for objects that are highly heterogeneous and of limited data quality, e.g., regarding the completeness and consistency of their descriptions.

The object matching problem was originally defined by Newcombe et al. [107] in 1959 and was formalized by Fellegi and Sunter [54] ten years later. It has since then

been considered under various facets and within different communities, including the artificial intelligence research community, the database research community, and industry. Numerous approaches and frameworks have been proposed for object matching especially for structured data.

The high number and diversity of different object matching approaches make it difficult for a user to decide when to choose which approach. Comparative evaluations of different schemes could provide guidance and support the user in his decision. Unfortunately, to date most object matching approaches have been evaluated individually using diverse methodologies, configurations, and test problems making it difficult to assess the overall quality of each approach, let alone their comparative effectiveness and efficiency. Only few attempts for comparative evaluations of some sub-approaches have been made, e.g., evaluation of different string similarity metrics [38] and of blocking approaches [7]. Some benchmark proposals for object matching have been made [106, 140] but they have not yet been implemented or applied.

A difficulty when comparing object matching algorithms is that they require different parameters to be set such as the similarity functions for comparing attribute values or similarity thresholds to be exceeded by matching entities. Many proposed approaches also make use of machine learning algorithms requiring specific parameters such as the size and characteristics of training data. Obviously, the chosen algorithm configuration is one of the predominant factors for the resulting match quality and in many published evaluation results significant details of it (e.g., on the used training data) remain unspecified.

This thesis deals with the topic of comparing, evaluating, and improving object matching approaches. It introduces a generic, customizable and scalable framework for object matching approaches and comparative object matching evaluation. The framework provides support for non-learning as well as learning-based approaches. It allows match approaches to be automatically executed and evaluated under different parameter configuration. It sets a platform for conducting a comparative evaluation on the relative effectiveness and efficiency of alternate match approaches and conduct such an evaluation on challenging real-word match tasks.

Furthermore, this thesis focuses on the product offer matching problem. Product offer matching is a special case of object matching that is needed to identify equivalent offers referring to the same real-world product. Product offer matching for e-commerce websites introduces several specific challenges that make this problem much harder than other forms of object matching, e.g., to match objects representing references of scientific publications. In particular, there is a huge degree of heterogeneity since offers come from thousands of merchants using different names and descriptions of the products. Furthermore, product offers frequently have missing or wrong values and are mostly not well structured but mix different product characteristics in text fields such as product name or description [72].

## 1.2 Scientific contributions

Focusing on the open problems discussed above, the thesis makes a number of contributions, which can be grouped into the following four areas:

**Survey of match approaches and frameworks:** To obtain a better overview about the current state of the art in object matching and object matching evaluation, we survey the existing frameworks and their evaluations. According to the defined criteria, we review various frameworks published in the literature. We characterize them in some detail and compare them with each other.

**Framework for object matching:** We introduce FEVER, a generic, customizable and comprehensive framework for object matching and for evaluating object matching approaches. Based on an open multi-component architecture it provides high flexibility for extension and adaptation. The framework offers the following key features:

- FEVER supports the flexible construction and comparative evaluation of many different object matching workflows based on so-called operator trees. Operator trees support the combined application of different blocking and match algorithms in order to achieve a high effectiveness (precision, recall). Individual operator implementations can be based on virtually any previously proposed algorithm, e.g., for blocking or object matching, so that these can be evaluated with FEVER.
- An important aspect of the FEVER framework is the support of learning-based match approaches. FEVER includes several learning-based approaches and methods to (semi-)automatically generate training data for object matching. Hence, FEVER can be used to compare non-learning and learning-based approaches for object matching. We specifically analyze the effectiveness for small training sizes which incur only a modest effort for labeling.
- FEVER allows each match approach to be automatically executed and evaluated under different parameter configurations. We first use this feature to determine the necessary effort for training and parameter tuning (e.g., finding suitable similarity thresholds) to obtain a reasonable match quality. This way we can conduct a comparative comparison of different object matching algorithms under comparable tuning effort. Hence, an approach **A** with better effectiveness than approach **B** is only superior if it does not incur a substantially higher configuration effort.
- FEVER can also be used to fine-tune match approaches by letting the system automatically evaluate a large number of parameter settings for

test data. The best performing configuration can then be used for subsequent match tasks on similar and larger input data.

**Comparative evaluation of match approaches and frameworks:** In this thesis we present a comparative evaluation on the relative effectiveness and efficiency of alternate object matching approaches. Main characteristics of our evaluation are:

- The approaches are uniformly evaluated on four real-world e-commerce and the bibliographic match tasks. In particular we consider matching of product objects from different web shops.
- We consider individual algorithms as well as frameworks offering different approaches. Furthermore we study approaches that do and do not require training data. In addition we consider a state-of-the art commercial object matching approach. More than 20 different approaches are evaluated under different parameter settings.
- Our evaluation considers both match quality in terms of precision, recall, and F-measure, as well as efficiency in terms of runtime.
- We use the FEVER framework to automatically execute the approaches and to find favourable parameter settings in a comparable way. In particular, we always apply the same blocking method to reduce the search space and use a uniform approach for providing training to the machine-learning approaches. For the approaches not based on machine learning we spend the same effort for optimizing parameters such as similarity thresholds.

**Product offer matching:** To address the product offer matching challenge we present an overall approach for matching product offers. It supports category-specific match strategies and is based on machine learning to semi-automatically determine a match strategy utilizing several attributes and similarity functions. We propose the use of tailored approaches for product offer matching based on a preprocessing of product offers to extract and clean new attributes usable for matching. In particular, we propose a new approach to extract and use so-called product codes to identify products and distinguish them from similar product variations.

Parts of the thesis have been published in refereed conferences, workshops and journals. In particular, the survey of object matching frameworks and evaluations is presented in [84]. Compared to the journal article we include two additional frameworks (DuDe [50] and FRIL [70]) in our evaluation. The architecture of the FEVER framework has been introduced as a demo for the VLDB conference [85]. The learning-based approaches are described in [83, 87]. The comparative evaluation on the relative effectiveness and efficiency of existing approaches and frameworks



is presented in [86]. The approach for matching product offers is published in [82]. Compared to the conference article we provide more details on the categorization and preprocessing steps and further evaluate the categorization approach.

## 1.3 Outline

The thesis is structured in five parts. In this first part of the thesis, Part I, we introduced the problem of object matching and motivated the need for semi-automatic support to solve the task. We then discussed the open issues in the current state of the art and gave an overview about the main contributions of the dissertation.

The rest of the thesis is organized in the four following parts:

In Part II – Object matching approaches – we give a short overview over the state of the art in object matching. We survey and discuss relevant literature. The overview is structured into the following chapters:

**Chapter 2** defines basic terms of object matching. It provides a definition of object matching and introduces standard evaluation measures to capture the effectiveness and efficiency of object matching. Furthermore requirements for object matching frameworks are defined.

**Chapter 3** introduces the blocking concept. Blocking is needed for large inputs to reduce the search space for object matching from the Cartesian product to a small subset of the most likely matching object pairs. The chapter defines measures to capture blocking effectiveness and outlines the most popular blocking approaches.

**Chapter 4** gives an overview over different matchers. Object matching requires a way to determine whether two objects are alike enough to represent the same real-world entity. A matcher is an algorithm specifying how the similarity between two objects is computed.

**Chapter 5** reviews approaches for combining several matchers. Due to the large variety of data sources and objects to match there is no single “best” solution. Instead it is often beneficial and necessary to combine several matchers to improve the effectiveness of object matching.

Part III - Comparison of object matching approaches - deals with the evaluation and comparison of object matching approaches. The description is organized into the following chapters:

**Chapter 6** analyzes and compares the functionality and evaluation of several object matching frameworks. The study considers both frameworks which do

or do not utilize training data to semi-automatically find an object matching strategy to solve a given match task. It considers the support for blocking, matching and the combination of different match algorithms as introduced in the previous three chapters. We further study how the different frameworks have been evaluated. The study aims at exploring the current state of the art in research prototypes of object matching frameworks and their evaluations.

**Chapter 7** introduces the ideas and concepts of our framework FEVER for object matching and object matching evaluation. It focuses on the concept of object matching workflows modeled as operator trees and offers methods for the comparative evaluation of object matching approaches.

**Chapter 8** first presents several real-world scenarios for object matching. We describe datasets within each scenario that we use for the evaluation of our own approaches as well as for the comparative evaluation of competitive approaches. We conduct a comparative evaluation of our own non-learning and learning-based workflows implemented within FEVER as well as of existing match approaches and frameworks.

Part IV - Product offer matching - deals with product offer matching as a special case of object matching. The part comprises the following chapters:

**Chapter 9** outlines the product offer matching problem as a challenging variation of object matching to identify representations and offers referring to the same product. We point out the particular challenges that have to be mastered.

**Chapter 10** proposes the use of tailored approaches for product offer matching based on a preprocessing of product offers to extract and clean new attributes usable for matching. In particular, we propose a new approach to extract and use so-called product codes to identify products and distinguish them from similar product variations.

**Chapter 11** evaluates the effectiveness of the proposed approaches with challenging real-life datasets with product offers from online shops. We also show that the UPC information in product offers is often error-prone and can lead to insufficient match decisions.

Part V concludes the thesis by summarizing the main contributions made and discussing relevant directions for future research.

## **Part II**

# **Object matching approaches**



# 2

## Preliminaries

In order to provide readers with an understanding of the problem, this chapter defines the basic terms. A definition of object matching is provided in Section 2.1. Section 2.2 introduces standard evaluation measures to capture the effectiveness and efficiency of object matching. Having formally defined the problem we address, we then discuss requirements for object matching frameworks in Section 2.3.

### 2.1 Object matching

Object matching is the process of determining whether two objects are referring to the same entity or two different entities. Objects to be resolved may reside in distributed, typically heterogeneous data sources or may be stored in a single data source, e.g., in a database or search engine store. They may be physically materialized or dynamically be requested from sources, e.g., by database queries or keyword searches.

The high importance and difficulty of the object matching problem has triggered a huge amount of research on different variations of the problem. Numerous approaches have been proposed especially for structured data. There exist several surveys [59, 142, 53] as well as books [103, 34].

In the research literature the problem was first called record linkage and since then has been studied under various different names:

deduplication [119], duplicate detection [18], duplicate record elimination [22], entity

object	title	manufacturer	price
$o_1$	HP Photosmart 6510	HP	113.37
$o_2$	HP Photosmart 6510 e-All-in-One	Hewlett-Packard	116.50
$o_3$	Nikon Coolpix S3100 14 MP Digital Camera	Nikon	54.72
$o_4$	Nikon Coolpix S4100 14 MP Digital Camera	Nikon	52.72
$o_5$	Coolpix S4100		64.85
$o_6$	Kyocera FS C5150DN	Kyocera	273.89
$o_7$	Kyocera FS C5150DN Color Laser printer - 21 ppm - 300 sheets		264.90
$o_8$	Kyocera FS C5150DN	Kyocera	274.00

Table 2.1: Single source object matching problem for objects representing product offers

identification [92], entity matching [84], entity reconciliation [43], entity resolution [12], fuzzy duplicate identification [26], identity resolution [69], object identification [124], object matching [46], object consolidation [28], record linkage [54], reference matching [94], or reference reconciliation [48].

In this thesis, we refer to the problem as object matching.

### Definition 2.1 Object

An object  $o$  is an instance of a real-world entity. It is defined and described through a set of attributes. An object  $o$  is represented as

$$o = \{\{A_1, v_1\}, \dots, \{A_n, v_n\}\}.$$

The attribute-value pair  $\{A_i, v_i\}$  denotes the value  $v_i$  for the attribute  $A_i$ . The value  $v_i$  can be elemental or set-valued.  $\square$

Table 2.1 gives an example for objects representing product offers. The objects are characterized by three attributes: title, manufacturer and price.

Obviously the example set contains several offers referring to the same real-world product. For example, the objects  $o_1$  and  $o_2$  are offers for the same printer. To identify all objects referring to the same real-world entity, the objects have to be matched against each other. The example set also illustrates that object matching is challenging. Objects are often highly heterogeneous and of limited data quality, e.g., regarding completeness and consistency of their descriptions. The objects in the above example contain numerous quality problems such as missing and erroneous values as well heterogeneous manufacturer denominations.

The next step is to provide a formal definition of object matching. For this definition the concept of similarity measure has to be introduced.

### Definition 2.2 Similarity Measure

Let  $O = \{o_1, \dots, o_n\}$  be a set of  $n$  objects,  $n < \infty$ .

A real-valued function  $s : O \times O \rightarrow [0, \infty[$  is called a **similarity measure**, if

1.  $s(o_i, o_j) = s(o_j, o_i)$ , for all  $o_i, o_j \in O$
2.  $s(o_i, o_j) \leq s(o_i, o_i)$ , for all  $o_i, o_j \in O$ .

Additionally it is often required that

- $s(o_i, o_j) \geq 0$  and  $s(o_i, o_i) = 1$ , for all  $o_i, o_j \in O$

□

The object matching problem can now be formally defined as follows:

**Definition 2.3 Object Matching**

Object matching is the process of identifying all objects  $O$  of a particular semantic entity type referring to the same real-word entity.

The objects in  $O$  are taken from data sources. Let  $S_A, S_B$  be data sources.

The *input* to the process is either a single set of objects  $O_A \subseteq S_A$  (**single source case**) or two sets of objects  $O_A \subseteq S_A$  and  $O_B \subseteq S_B$  (**double source case**). In the double source case we consider  $S_A$  to be duplicate free.

$O = O_A$  in case of a single object set  $O_A$ , while for two sets  $O_A$  and  $O_B$  it applies  $O = O_A \cup O_B$ .

The *output* of the match process is either:

1. a mapping  $M$   
A **mapping** is a set of correspondences

$$M = \{(o_i, o_j, s_{i,j}) | o_i \in S_A, o_j \in S_B, s(o_i, o_j) \in [0, 1]\}.$$

A correspondence  $c = (o_i, o_j, s_{i,j}) \in M$  interrelates two objects  $o_i$  and  $o_j$ . The objects originate either from the same source  $S_A = S_B$  or from two different sources  $S_A \neq S_B$ . An optional similarity value  $s_{i,j} = s(o_i, o_j) \in [0, 1]$  indicates the similarity or strength of the correspondence between the two objects.

or

2. a set of clusters  $\mathcal{C} = \{C_1, \dots, C_n\}$ .  
 $C_i$  is a set of objects  $C_i = \{o_{i,1}, \dots, o_{i,k}\}$  where the objects are deemed to represent the same real-word entity.

The *process* is a sequence of several steps called a **match strategy**. A match strategy consists of the application of an optional blocking step and one or several matchers. A matcher  $m$  uses a similarity measure  $s(\cdot, \cdot)$  to determine the similarity between two objects  $o_i$  and  $o_j$ . □

## CHAPTER 2. PRELIMINARIES

object	title	author	venue	year
$o_{A,1}$	A survey of approaches to automatic schema matching	Erhard Rahm, Philip A. Bernstein	VLDB Journal 10 (4)	2001
$o_{A,2}$	A survey of schema-based matching approaches	Pavel Shvaiko, Jérôme Euzenat	J. Data Semantics IV	2005

(a) Clean bibliographic reference source  $S_A$

object	title	author	venue	year	citation count
$o_{B,1}$	A survey of approaches to automatic schema matching	Rahm, E.; Bernstein, P.A.	VLDB J.	2001	3183
$o_{B,2}$	A survey of approaches to automatic schema matching	AB Philip			17
$o_{B,3}$	A survey of approaches to automatic schema matching	PA Bernstein	VLDB Journal	2001	17
$o_{B,4}$	A survey of schema-based matching approaches	P Shvaiko	Journal on Data Semantics IV	2005	1113

(b) Bibliographic source with citation counts  $S_B$

Table 2.2: Double source object matching problem for two bibliographic sources

The example in Table 2.1 illustrates the single source case. An example for the double source case is given in Table 2.2. The objects in this case are references for scientific papers from two sources. The task is to collect all citations of publications for a citation analysis. We have a clean reference source with publications for which we want to determine the citations counts. On the other hand, we have a source providing references with citations counts. The references in the second source contain numerous quality problems such as misspelled author names, different ordering of authors, heterogeneous venue denominations, etc. Since the reference source has no duplicates the object matching result between the two sources can also be used for determining the duplicates in the second source. This is because all entries from the second source matching the same publication from the reference source can be considered duplicates.

A mapping is a symmetric relation. If two objects are duplicates, then the order of expressing identicalness does not matter. To say that object  $o_i$  is identical to object  $o_j$  is the same as saying that  $o_j$  is identical to  $o_i$ . It is therefore sufficient to store only one of the two correspondences  $(o_i, o_j)$  and  $(o_j, o_i)$  in the result mapping. Without loss of generality we store the correspondence  $(o_i, o_j)$  with  $i < j$ .

The maximum number of correspondences is  $|O_A| \frac{|O_A|-1}{2}$  for matching objects within a single source. That is the case when all objects refer to the same real-world entity. When we match two sources the resulting mapping can at most contain  $|O_B|$  correspondences assuming that the reference source is duplicate free.



A plausible object matching result for our single source example problem illustrated in Table 2.1 would return the following mapping:

$$M = \{(o_1, o_2), (o_4, o_5), (o_6, o_7), (o_6, o_8), (o_7, o_8)\}.$$

For our double source example problem illustrated in Table 2.2 we would obtain the following mapping:

$$M = \{(o_{A,1}, o_{B,1}), (o_{A,1}, o_{B,2}), (o_{A,1}, o_{B,3}), (o_{A,2}, o_{B,4})\}.$$

If we are interested in all representations of a real-world entity the cluster representation of the matching result is more suitable.

The cluster representation can be obtained from the mapping result by applying the transitive closure over the set of correspondences.

For our single source example problem illustrated in Table 2.1 the cluster representation of the match result consists of four clusters:

$$\mathcal{C} = \{\{o_1, o_2\}, \{o_3\}, \{o_4, o_5\}, \{o_6, o_7, o_8\}\}.$$

For our double source example problem illustrated in Table 2.2 the cluster representation of the match result consists of two clusters:

$$\mathcal{C} = \{\{o_{A,1}, o_{B,1}, o_{B,2}, o_{B,3}\}, \{o_{A,2}, o_{B,4}\}\}.$$

The cluster representation provides a compact and space saving representation of the match result. The cluster assignment can be added as an additional attribute to each object representation. However, the similarities between the objects are not preserved and a later refinement of the match result is not possible. The advantage of the mapping representation is that similarities are stored. This allows to further process the mapping result, e.g., to combine it with other mappings computed by different approaches or to refine it. Clusters can be derived from a mapping but not vice versa. In this thesis we therefore prefer the mapping representation to the cluster representation.

The optional blocking step within a match strategy reduces the search space for object matching from the Cartesian product to a subset of the most likely matching object pairs. We introduce diverse strategies in Chapter 3. In Chapter 4 we describe various similarity measures and approaches that can be used by a matcher to compute the similarity between objects based on the attribute values as well as on context information. There are various possibilities for combining multiple matchers within a match strategy. We discuss different approaches in Chapter 5.

## 2.2 Evaluation measures

The goal is to develop approaches that effectively and efficiently perform object matching in a scalable way. To measure effectiveness and efficiency several measures

can be employed. In the following subsections we introduce common effectiveness and efficiency measures.

### 2.2.1 Effectiveness measures

The effectiveness describes the quality of the match result. The match quality of an object matching algorithm is typically evaluated by running it on a dataset and comparing the results to a gold standard. The gold standard is an object matching result that is assumed to be correct. In many cases, the gold standard is manually determined by a group of human experts.

Precision, recall and F-measure are common measures to assess the effectiveness. To better illustrate the different measures, we employ again the single source object matching problem depicted in Table 2.1.

Let us assume that the mapping result  $M_R$  of some approach is

$$M_R = \{(o_1, o_2), (o_4, o_5), (o_6, o_8)\}.$$

The gold standard mapping  $M_G$  is

$$M_G = \{(o_1, o_2), (o_4, o_5), (o_6, o_7), (o_6, o_8), (o_7, o_8)\},$$

Based on the comparison of the correspondences in the mappings  $M_R$  and  $M_G$  *precision*  $Pr$  and *recall*  $Re$  can be calculated.

*Precision* calculates the proportion of the identified actual matches as share of all identified matches. It is thus a measure for accuracy, it measures how precise an approach is in identifying actual matches. It is defined as:

$$Pr = \frac{|M_R \cap M_G|}{|M_R|} \quad (2.1)$$

For the example mapping  $M_R$  all of the three identified correspondences are contained in the gold standard mapping  $M_G$ . Thus the precision is:

$$Pr = \frac{3}{3} = 1.0.$$

*Recall* measures the proportion of the identified actual matches as share of all actual matches. It is thus a measure for completeness. It is defined as:

$$Re = \frac{|M_R \cap M_G|}{|M_G|} \quad (2.2)$$

For the example mapping  $M_R$  three of the five correspondences from the gold standard mapping  $M_G$  have been identified. Thus the recall is:

$$Re = \frac{3}{5} = 0.6.$$

However, neither precision nor recall alone can accurately assess the match effectiveness. In particular, recall can easily be maximized at the expense of a poor precision by returning as many correspondences as possible. On the other side, a high precision can be achieved at the expense of a poor recall by returning only few but correct correspondences.

To weigh the tradeoff between precision and recall, the F-measure is often used. It is defined as the harmonic mean of precision and recall:

$$\text{F-measure} = \frac{2Pr \cdot Re}{Pr + Re} \quad (2.3)$$

For our example the F-measure can be calculated as:

$$\text{F-measure} = \frac{2 \cdot 1 \cdot (3/5)}{1 + (3/5)} = 0.75.$$

## 2.2.2 Efficiency measures

The efficiency of an object matching approach is defined by an analysis or measure of its computational/time complexity. The time complexity correlates with the efficiency: the lower the complexity, the more efficient the approach. Analysis consists of a theoretical time complexity analysis examining the best, worst, or average case. The time complexity of an object matching approach is typically measured in terms of the number of required object comparisons and the corresponding runtime.

## 2.3 Requirements for object matching frameworks

In the following, several requirements and desiderata for a comprehensive support of object matching are discussed.

**Effectiveness:** The main goal of object matching is to achieve a high-quality match result with respect to recall and precision, i.e. all real corresponding objects but no others should be included in the result. Achieving this goal for different match tasks typically requires the flexible combination and customization of different match methods. A key concern will thus be which match approaches are supported and how they can be combined.

**Efficiency:** Object matching should be fast even for voluminous datasets. For very large datasets this typically prescribes the use of blocking methods to reduce the search space for object matching (see next section).

**Scalability:** With the ever increasing amount of data, the ability to handle very large input sets is an important aspect for object matching frameworks.

**Genericity:** A general framework should support methods applicable to different match tasks from various domains (e.g., enterprise data, product data, life science data) and for different data models (e.g., relational, XML).

**Offline/online matching:** Furthermore, an object matching framework should be applicable to offline and online match tasks. Online match tasks arise for interactive data integration steps such as mediated queries or data mashups based on specific user input. Offline object matching is less time-critical than online matching which can thus better deal with large datasets and may allow for more match algorithms to be applied. Object matching during the ETL (extract, transform, load) process of data warehouses is a sample case for offline matching.

**Low manual effort / self-tuning:** The manual effort to employ an object matching framework should be as low as possible, in particular for selecting the methods for blocking and matching, their parameters and their combination. Ideally, the framework is able to solve these tasks automatically in a self-tuning manner, e.g., with the help of machine learning methods utilizing training data. On the other hand, selecting and labeling training data may also incur manual effort which should therefore be low.

A key challenge in developing a successful object matching framework is that some of the posed requirements are in conflict with each other, e.g., effectiveness and efficiency or genericity and ease-of-use. For example the use of blocking methods improves efficiency by reducing the search space. However, this may eliminate some relevant object pairs from consideration and thus reduce effectiveness (recall) of object matching. On the other hand, the combined use of several match algorithms may improve effectiveness but will typically lead to increased computational overhead and thus reduce efficiency. Successfully resolving entities in diverse domains with the help of a generic object matching framework is more difficult than for only one domain. Non-generic frameworks may thus incur a reduced manual effort to provide training or to find a suitable combination and customization of algorithms.

# 3

## Blocking

The standard (naive) approach to find matches in  $n$  input objects is to compare each object with every other object. This requires  $|O_A| \frac{|O_A|-1}{2}$  comparisons for a single input set  $O_A$ . Similarly, for two input sets  $O_A$  and  $O_B$  containing  $|O_A|$  and  $|O_B|$  objects,  $|O_A| \cdot |O_B|$  comparisons are needed. This means that the complete Cartesian product ( $O_A \times O_B$ ) is examined.

The resulting quadratic complexity of  $\mathcal{O}(n^2)$  results in infeasible execution times in particular for large input sets. Therefore, an initial step in the matching process called blocking is commonly applied to reduce the search space to a small subset of the most likely matching object pairs. Numerous blocking algorithms have been proposed in the past years (see [35, 7, 49] for overviews and comparisons). They typically use a key to partition the objects to be matched into groups (blocks). Matching of an object can then be restricted to the objects in the same block.

In this chapter we introduce some of the most popular blocking approaches discriminating between disjoint and overlapping methods. We focus on the approaches that have been implemented within the frameworks we are going to evaluate in Chapter 6 and are also provided within our own framework FEVER. Before detailing disjoint and overlapping methods in Section 3.3 and Section 3.4 we introduce in Section 3.1 measures to evaluate the effectiveness of blocking techniques and describe the general ideas of blocking in Section 3.2. Section 3.5 gives an overview over some further approaches. We conclude the chapter by giving some concluding remarks in Section 3.6.

## 3.1 Blocking evaluation measures

For the evaluation of blocking techniques three measures have been proposed: pairs completeness, reduction ratio and F-score.

Pairs completeness ( $PC$ ) indicates which share of the truly matching object pairs are preserved after blocking.  $PC$  thus corresponds to a recall measure and a high value is important for effectiveness.

The reduction ratio ( $RR$ ) measure indicates the fraction of all possible object pairs which is eliminated by blocking; it indicates how far the search space is reduced and thus efficiency is improved.

$$RR = 1 - \frac{c}{pn} \quad (3.1)$$

where  $c$  is resulting number of candidate object pairs after blocking and  $pn$  is the number of all possible combinations ( $pn = |O_A| \frac{|O_A|-1}{2}$  for  $A = B$  or  $pn = |O_A \times O_B| = |O_A| \cdot |O_B|$  for  $A \neq B$ ).

F-score combines pairs completeness ( $PC$ ) and reduction ratio ( $RR$ ) via a harmonic mean,

$$F\text{Score} = \frac{2 \cdot PC \cdot RR}{PC + RR} \quad (3.2)$$

## 3.2 General idea

Blocking approaches semantically partition the input data into blocks of similar objects and restrict object matching to objects of the same block.

The partitioning into blocks is usually done with the help of blocking keys based on the objects' attribute values. Blocking keys utilize the values of one or several attributes, e.g., product manufacturer (to group together all products sharing the same manufacturer) or the combination of manufacturer and product type. Often, the concatenated prefixes of a few attributes form the blocking key. The blocking key is typically determined manually. Few approaches exist to derive the key (semi-)automatically based on training data [17, 98].

The definition of the key is a critical issue with all blocking methods. A suboptimal choice may lead to over-selection of many dissimilar object pairs that impedes efficiency, or, worse, sorting out true matching object pairs thus decreasing match quality. A strategy to diminish the influence of poor blocking keys (e.g., due to dirty data) is to repeatedly execute a blocking method using different blocking keys. This is called a multi-pass strategy.

For most blocking techniques, an inverted index [143] can be used. The blocking key values will become the keys of the inverted index, and the record identifiers of

all records that have the same blocking key value will be inserted into the same inverted index list.

When matching objects from two sources, either a separate index data structure is built for each source, or a single data structure with common key values is generated. For the second case, each object identifier needs to include a flag that indicates from which source the object originates.

## 3.3 Disjoint blocking

Disjoint blocking builds mutually exclusive blocks by assigning each object into one block only (assuming a single blocking key definition). The default implementation is called traditional or standard blocking and is detailed in the following subsection.

### 3.3.1 Traditional or standard blocking

This technique has been used in record linkage since the 1960s [54]. The implementation uses an inverted index to build the blocks. All objects with the same blocking key value (BKV) are inserted into the same block, and only objects within the same block are then compared with each other. Figure 3.1 illustrates an example execution for blocking keys based on manufacturer values.

While this approach does not have any explicit parameters, the way blocking keys are defined will influence the quality and number of candidate pairs that are generated. A major drawback is that it is sensitive regarding errors and variations in the object values used to generate the blocking keys. True matching object pairs are missed when they are assigned to the different blocks due to heterogeneous blocking key values. This leads to a decrease of pairs completeness. A second drawback is that the sizes of the blocks generated depend upon the frequency distribution of the BKVs, and thus it is difficult in practice to predict the total number of candidate pairs that will be generated.

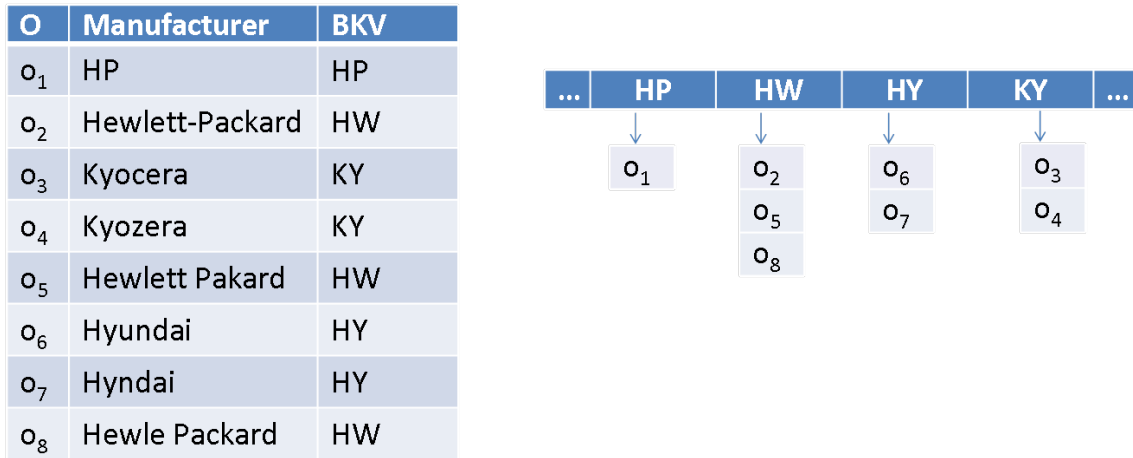


Figure 3.1: Example objects with manufacturer values and the first two consonants as BKVs, and the corresponding inverted index data structure as used for traditional blocking

## 3.4 Overlapping blocking

Overlapping methods may result in overlapping blocks of objects; implementations include the (multi-pass) sorted neighborhood approach [63], bi-gram indexing [7], canopy clustering [94] and iterative blocking [141]. These methods can require an object to be matched against multiple blocks (increased overhead) but may lead to a higher pairs completeness value than disjoint methods.

### 3.4.1 Sorted Neighborhood

The Sorted neighborhood (SN) [63] first sorts the objects according to their blocking key  $K$ . A window of a fixed size  $w$  is then moved over the sorted objects and in each step objects within the window, i.e., objects within a distance of  $w - 1$ , are compared.

Figure 3.2 illustrates by an example the execution of SN for a window size of  $w = 3$ . The objects ( $o_1 - o_9$ ) are first sorted by their blocking keys (1, 2, or 3). The sliding window then processes the first block ( $o_1, o_4, o_2$ ) resulting in the three pairs ( $o_1, o_4$ ), ( $o_1, o_2$ ), and ( $o_4, o_2$ ) for later comparisons. The window is then moved by one step to cover the block ( $o_4, o_2, o_5$ ). This leads to two additional pairs ( $o_4, o_5$ ) and ( $o_2, o_5$ ). This procedure is repeated until the window has reached the final block ( $o_3, o_7, o_9$ ). Figure 3.2 lists all pairs generated by the sliding window.

SN reduces the complexity from  $\mathcal{O}(n^2)$  (matching  $n$  input objects without blocking) to  $\mathcal{O}(n) + \mathcal{O}(n \cdot \log_2 n)$  for blocking key determination and sorting and  $\mathcal{O}(n \cdot w)$  for



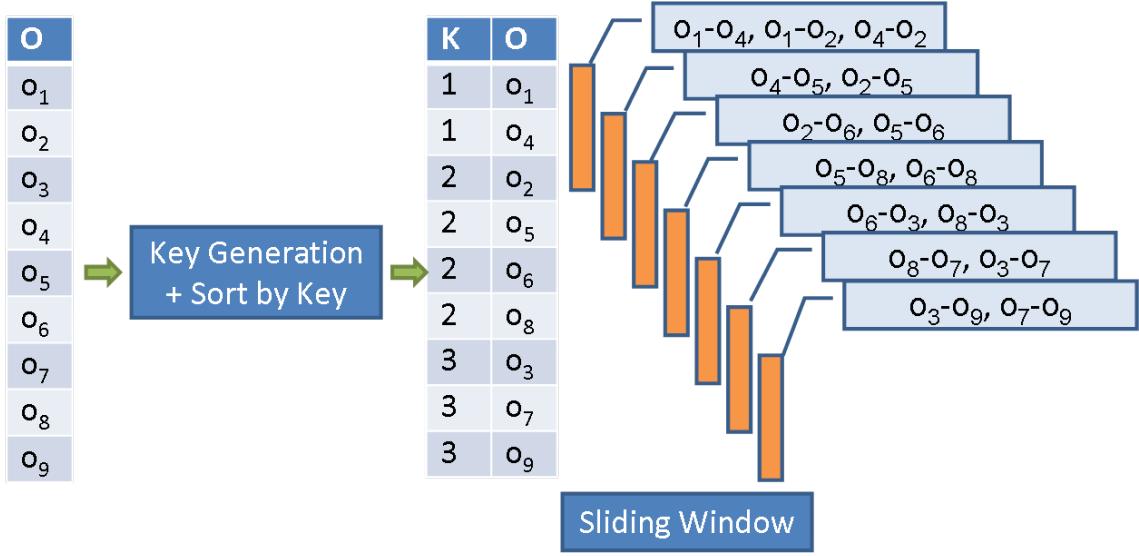


Figure 3.2: Example execution of sorted neighborhood with window size  $w = 3$  adapted from [81]

matching. Thereby matching large datasets becomes feasible and the window size  $w$  allows for a dedicated control of the runtime. The SN approach is able to compare objects with a different (but similar) blocking key and can therefore compensate a suboptimal choice of the blocking key.

As an extension the Multi-Pass approach [64] has been proposed. It supports a repeated execution with multiple independent blocking keys.

In [51] the Sorted Blocks method is proposed as a generalization of the standard blocking and the sorted neighborhood approach. The basic idea is to first sort all objects so that duplicates are close in the sort sequence, then partition the objects into disjoint sorted subsets, and finally to overlap the partitions. The size of the overlap can be defined using  $u$ , e.g.,  $u = 3$  means that three objects of each neighbouring partition are part of the overlap, which hence has a total size of  $2u$ . Within the overlap, a fixed size window with size  $u + 1$  is slid across the sorted data and all objects within the window are compared.

[147] introduces two adaptive approaches to dynamically set the window size. Incrementally Adaptive-SNM (IA-SNM) is an algorithm that incrementally increases the window size as long as the distance of the first and the last object in the current window is smaller than a specified threshold. The increase of the window size depends on the current window size. Accumulative Adaptive-SNM (AA-SNM) creates windows with a single overlapping object. By considering transitivity, multiple adjacent windows can then be grouped to one block, if the last object of a window potentially matches with the last object in the next adjacent window. Both algo-

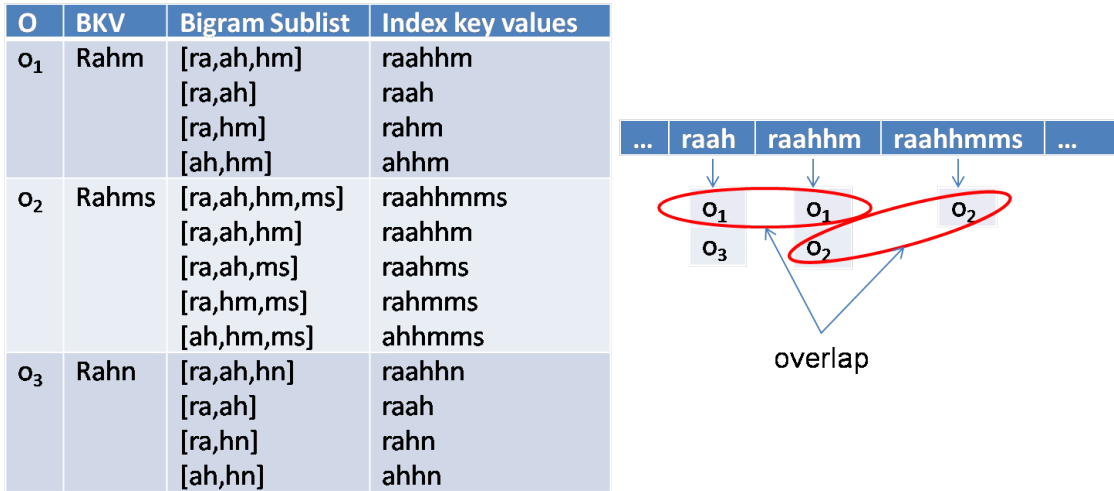


Figure 3.3: Example execution of  $q$ -gram based indexing with  $q = 2$  and  $t = 0.8$

rithms have after the enlargement of the windows a retrenchment phase, in which the window is decreased until all records within the block are potential duplicates.

### 3.4.2 Q-gram Based Indexing

The basic idea of this approach is to generate variations of each blocking key by converting it into a list of  $q$ -grams (sub-strings of length  $q$ ). Sub-lists of the  $q$ -gram list are then computed. This is controlled by a threshold parameter  $t$ , which designates the fraction of the shortest sub-lists to be generated relative to the length of the  $q$ -gram list. The concatenated  $q$ -grams in the resulting  $q$ -gram sub-lists are then used as keys in an inverted index. Each object is inserted into several blocks according to how many index keys have been generated from its blocking key. With  $t = 1.0$ , only one  $q$ -gram sub-list is generated per blocking key value. Thus each object is only inserted into a single list resulting in a disjoint blocking with mutually exclusive blocks.

Figure 3.3 illustrates  $q$ -gram based indexing for three example objects,  $q = 2$  (bi-grams), and a threshold  $t = 0.8$ . The BKV Rahm of the first object ( $o_1$ ), for example, contains three ( $k = 3$ ) bigrams: ra, ah, hm (assuming all letters have been converted into lower case beforehand). The length  $l$  of the shortest sub-lists for this value can be calculated as  $l = \lceil 3 \cdot 0.8 \rceil = 2$ . Therefore, one sublist with three bigrams and three sub-lists each containing two bigrams will be generated for this BKV: [ra,ah,hm], [ra,ah] [ra,hm], and [ah,hm]. Each of the sub-lists containing two bigrams is generated by removing one of the three original bigrams. The bigrams in the generated sub-lists are concatenated to form the actual key values used in the inverted index, as is shown in Figure 3.3. Object  $o_1$  will be inserted into the four

inverted index lists with key values 'raahhm', 'raah', 'rahm', and 'ahhm'. With an even lower threshold ( $t < 0.75$ ), sublists of length one would be generated recursively from the sub-lists of length three.

The advantage of this approach is that it can overcome errors and variations in the blocking key values. Objects that are true matches are more likely inserted into the same block resulting in a higher pairs completeness and thus an improved matching quality. The drawback is that a large number of sub-list are generated. The recursive generation of sub-lists is computationally expensive, especially for long blocking key values and threshold values.

### 3.4.3 Canopy Clustering

This blocking technique inserts objects into one or more overlapping clusters, called canopies [94, 39], based on the similarities between the BKVs [143, 39, 30]. Commonly used similarity measures to compute the similarities between the BKVs are Jaccard and TFIDF. Further details on these two measures can be found in Section 4.1.

With Canopy Clustering overlapping clusters are iteratively generated repeating the following steps:

1. Initially all objects are inserted into a candidate list.
2. One object is randomly picked and removed from the candidate set. This object becomes the centroid of a new canopy cluster.
3. All objects that have a similarity value above a loose threshold  $t_l$  are inserted in the canopy cluster.
4. All objects that have a similarity value above a tight similarity threshold  $t_t$  (with  $t_t \geq t_l$ ) are removed from the candidate list.

Steps 2 to 4 are repeated until the candidate list is empty.

We illustrate the approach by means of an example with six objects  $O = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ . Table 3.1 indicates the similarity values between the BKVs of the objects. We choose  $t_l = 0.8$  and  $t_t = 0.9$ . We pick  $o_1$  as the centroid of the first cluster and remove it from the candidate set. As  $s_{1,2} = 0.8 > t_l$  and  $s_{1,3} = 0.95 > t_l$  the objects  $o_2$  and  $o_3$  are inserted into the cluster. As  $s_{1,3} = 0.95 > t_t$  we can remove  $o_3$  from the candidate set. We pick  $o_2$  as the centroid of the second cluster and remove it from the candidate set. As  $s_{2,4} = 0.91 > t_l$ ,  $s_{2,5} = 0.97 > t_l$ , and  $s_{2,6} = 0.75 > t_l$  the objects  $o_4$ ,  $o_5$  and  $o_6$  are inserted into the cluster. We can remove object  $o_4$ . We pick  $o_5$  as the centroid of the third cluster, remove it from the candidate set and insert  $o_6$  into the cluster as  $s_{5,6} = 0.98 > t_l$ . Object  $o_6$  is removed from the

candidate set as  $s_{5,6} = 0.98 > t_t$ . The candidate set is now empty and the algorithm terminates. The final set of cluster is  $\mathcal{C} = \{\{o_1, o_2, o_3\}, \{o_2, o_4, o_5, o_6\}, \{o_5, o_6\}\}$

	$o_1$	$o_2$	$o_3$	$o_4$	$o_5$	$o_6$
$o_1$	1	0.8	0.95	0.5	0.6	0.7
$o_2$	0.8	1	0.9	0.91	0.97	0.75
$o_3$	0.95	0.9	1	0.5	0.6	0.7
$o_4$	0.5	0.91	0.5	1	0.5	0.7
$o_5$	0.6	0.97	0.6	0.5	1	0.98
$o_6$	0.7	0.75	0.7	0.7	0.98	1

Table 3.1: Similarity matrix for Canopy Clustering example

In [31] a nearest neighbour based approach is proposed as a variant. Instead of two global threshold parameters two nearest neighbour parameters are introduced:  $n_l$  and  $n_t$ .  $n_l$  is the number of closest objects to the randomly chosen centroid object (according to the similarities of their blocking key values) that will be inserted into the canopy cluster, and of these the  $n_t$  closest objects will then be removed from the pool of candidate objects (with  $n_t \leq n_l$ ). This approach results in blocks of similar sizes, with the maximum size known beforehand as  $n_l$ . This not only prevents very large blocks, but also allows an estimate of the number of object pairs generated, as the number of blocks (canopy clusters) corresponds to  $n/n_t$ , with  $n$  being the total number of different blocking key values.

## 3.5 Further approaches

In the previous sections we introduced the most popular blocking approaches. However, many more approaches have been proposed. In this section we briefly mention some recent developments.

The basic idea of suffix based blocking [2] is to insert the blocking key values and their suffixes into a suffix array-based inverted index. A suffix array contains strings or sequences and their suffixes in an alphabetically sorted order.

In [141] an iterative blocking approach is proposed where the correspondences detected within one block are reflected to subsequently processed blocks. Blocks are iteratively processed until no block contains any more matching objects.

There is a smooth transition from blocking approaches to search space reduction techniques for similarity computation. A large amount of work has also been conducted on reducing the search space for similarity measures such as edit distance

or Jaccard. For instance, LIMES [108] utilizes the mathematical characteristics of metric spaces to filter out a large number of those instance pairs that do not suffice a given minimal similarity threshold.

### 3.6 Concluding remarks

There exist a few comparative studies of different blocking approaches: [7, 31, 35]. The experimental results in these studies showed that there are large differences in terms of pairs completeness regarding the different techniques. Moreover several approaches are rather sensitive towards changes in the parameter setting. The variety of parameters that have to be set by an user, and the sensitivity of some of them (especially global thresholds) with regard to the candidate record pairs generated, makes it somewhat difficult to give a general recommendation, as parameter settings depend both upon the quality and characteristics of the input objects.



# 4

## Matchers

Object matching requires a way to determine whether two objects are alike enough to represent the same real-world entity. A matcher applies a similarity measure to specify how the similarity between two objects is computed.

In Section 4.1 we discuss various similarity measures that have been used for object matching throughout the literature. In Section 4.2 we describe context matchers that consider the context or semantic relationships of different objects for similarity computation.

### 4.1 Similarity measures

Matchers use a similarity function and apply it on the values of a pair of corresponding attributes or attribute concatenations of the input datasets. They typically return a value between 0 and 1 indicating the degree of similarity between two entities.

Numerous similarity functions may be employed, in particular generic string similarity measures (see [38] and [53] for a comprehensive comparison). Similarity computation may also utilize different kinds of auxiliary information, such as dictionaries, thesauri or domain-specific lookup tables [27], e.g., to deal with synonyms, homonyms, abbreviations, acronyms, or geographic name variations. In offline data integration, e.g., for data warehousing, such auxiliary sources are often used by separate data cleaning steps to resolve representational differences before object matching begins.

In the following, we discuss various similarity measures that have been used for object matching throughout the literature. In our discussion, we distinguish different classes of measures. We focus on the measures that have been implemented within FEVER.

### 4.1.1 String similarity measures

We distinguish character-based, token-based, and hybrid measures. Character-based measures compute string similarity by viewing strings as contiguous sequences of either characters or tokens. Token-based measures, on the other hand, do not view strings as contiguous sequences but sets of tokens. Hybrid measures combine both approaches.

#### Character-based similarity measures

**Edit Distance** : The edit distance between two strings  $\sigma_1$  and  $\sigma_2$  is the minimum number of character insertions, deletions, and replacements needed to transform the string  $\sigma_1$  into  $\sigma_2$ . In the simplest form, each edit operation has cost 1. This version of edit distance is also referred to as Levenshtein distance [91]. Given two strings  $\sigma_1$  and  $\sigma_2$ , their Edit Distance similarity can be calculated as:

$$\text{EditDistance}(\sigma_1, \sigma_2) = 1 - \left( \frac{\text{dist}(\sigma_1, \sigma_2)}{\max(|\sigma_1|, |\sigma_2|)} \right)$$

The basic dynamic programming algorithm [105] for computing the edit distance between two strings takes  $\mathcal{O}(|\sigma_1| \cdot |\sigma_2|)$  time for two strings of length  $|\sigma_1|$  and  $|\sigma_2|$ , respectively.

Several variants of the edit distance have been proposed, including the constrained edit distance [112] and the normalized edit distance [93].

#### Example 4.1.1.1 Edit Distance

As an example, consider the two strings  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlet Peckard"}$ . The edit distance of these two strings  $\text{dist}(\sigma_1, \sigma_2)$  is 2, as we need to (i) delete the  $t$  in  $\sigma_1$  and (ii) replace the first  $a$  in  $\sigma_1$  by an  $e$ .

Thus the Edit Distance similarity is

$$\text{EditDistance}(\sigma_1, \sigma_2) = 1 - \left( \frac{2}{15} \right) \approx 0.866$$

**Jaro-Winkler** : The Jaro comparison value is:

$$\text{Jaro}(\sigma_1, \sigma_2) = \frac{1}{3} \left( \frac{c}{|\sigma_1|} + \frac{c}{|\sigma_2|} + \frac{c - t/2}{c} \right)$$



where  $c$  is the number of common characters and  $t$  is the number of transpositions of common characters. Common are all the characters  $\sigma_1[i]$  and  $\sigma_2[j]$  for which  $\sigma_1[i] = \sigma_2[j]$  and  $|i - j| \leq \frac{1}{2} \min \{|\sigma_1|, |\sigma_2|\}$ . A transposition occurs when the common character at position  $i$  of string  $\sigma_1$  is not equal to the common character at position  $i$  of string  $\sigma_2$ .

Jaro-Winkler distance uses a prefix scale  $p$  which gives more favourable ratings to strings that match from the beginning for a set prefix length  $\ell$ . Given two strings  $\sigma_1$  and  $\sigma_2$ , their Jaro-Winkler similarity is:

$$\text{JaroWinkler}(\sigma_1, \sigma_2) = \text{Jaro}(\sigma_1, \sigma_2) + (\ell \cdot p(1 - \text{Jaro}(\sigma_1, \sigma_2)))$$

#### Example 4.1.1.2 Jaro-Winkler

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlet Peckard"}$ . These two strings have 13 common characters. The common prefix "Hewlet" is of length  $\ell = 6$ . There are no permutations of characters, thus it follows that  $\text{Jaro}(\sigma_1, \sigma_2) \approx 0.932$ . Assuming a scaling factor of  $p = 0.1$  the Jaro-Winkler similarity is equal to:

$$\text{JaroWinkler}(\sigma_1, \sigma_2) = 0.932 + 6 \cdot 0.1 \cdot (1 - 0.932) = 0.9728$$

### Token-based similarity measures

Token-based similarity measures split strings into pieces called tokens. Intuitively, tokens correspond to substrings of the original string. A simple tokenization splits a string into tokens based on whitespace characters. The advantage of token-based similarity measures is their robustness towards natural word moves and swaps as they do not regard the word order (e.g., "Erhard Rahm" is equivalent to "Rahm Erhard"). The disadvantage is that typographical errors within tokens are not captured, especially if they are pervasive and affect many tokens of the strings. For example, the strings "Hewlett Packard" and "Hewlet Peckard" will have zero similarity. An exception is the Q-Gram similarity measure that splits strings into q-grams.

**Q-Gram** : Q-grams are short character substrings of length  $q$  [132, 131]. Given a string  $\sigma$ , its q-grams are obtained by "sliding" a window of length  $q$  over the characters of  $\sigma$ . Since q-grams at the beginning and the end of the string can have fewer than  $q$  characters, the string is conceptually extended by "padding" the beginning and the end of the string with  $q - 1$  occurrences of a special padding character, not in the original alphabet.

Positional q-grams [127] also consider the position of the q-gram in the string. Given two sets of q-grams similarity is computed using the Dice's coefficient [44]:

$$\text{QGram}(\sigma_1, \sigma_2) = \frac{2|T_{\sigma_1} \cap T_{\sigma_2}|}{|T_{\sigma_1}| + |T_{\sigma_2}|}$$

**Example 4.1.1.3 Q-Gram**

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlet Peckard"}$ . We compute 3-grams (also called trigrams). We use underscore ( $\_$ ) to represent a whitespace. The padding characters at the beginning and the end of a string are denoted as  $\#$ . The respective sets of trigrams are

$$T_{\sigma_1} = \{\#\#\text{H}, \#\text{He}, \text{Hew}, \text{ewl}, \text{wle}, \text{let}, \text{ett}, \text{tt}\_, \text{t\_P}, \_ \text{Pa}, \text{Pac}, \text{ack}, \text{cka}, \text{kar}, \text{ard}, \text{rd}\#, \text{d}\#\#\}$$

and

$$T_{\sigma_2} = \{\#\#\text{H}, \#\text{He}, \text{Hew}, \text{ewl}, \text{wle}, \text{let}, \text{et}\_, \text{t\_P}, \_ \text{Pe}, \text{Pec}, \text{eck}, \text{cka}, \text{kar}, \text{ard}, \text{rd}\#, \text{d}\#\#\}.$$

Thus the Trigram similarity is

$$\text{Trigram}(\sigma_1, \sigma_2) = \frac{2 \cdot 12}{17 + 16} = 0.727$$

[58, 57] showed how to efficiently compute positional q-gram similarity within a relational database.

**Jaccard** : The Jaccard coefficient [66, 67] measures similarity between token sets, and is defined as the size of the intersection divided by the size of the union of the token sets:

$$\text{Jaccard}(\sigma_1, \sigma_2) = \frac{|T_{\sigma_1} \cap T_{\sigma_2}|}{|T_{\sigma_1} \cup T_{\sigma_2}|}$$

**Example 4.1.1.4 Jaccard**

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlett Peckard"}$ . The respective token sets are  $T_{\sigma_1} = \{\text{"Hewlett"}, \text{"Packard"}\}$  and  $T_{\sigma_2} = \{\text{"Hewlett"}, \text{"Peckard"}\}$ . There is one token in the intersection of the two sets  $T_{\sigma_1} \cap T_{\sigma_2} = \{\text{"Hewlett"}\}$  and a total of three tokens that appear in  $T_{\sigma_1}$  or  $T_{\sigma_2}$  or both,  $T_{\sigma_1} \cup T_{\sigma_2} = \{\text{"Hewlett"}, \text{"Packard"}, \text{"Peckard"}\}$ . Thus the Jaccard similarity is

$$\text{Jaccard}(\sigma_1, \sigma_2) = \frac{1}{3} = 0.333$$

**Cosine** : The cosine similarity is a well known measure from information retrieval. It computes the cosine of the angle  $\alpha$  between the two  $d$ -dimensional vectors  $\vec{\sigma}_1$  and  $\vec{\sigma}_2$  of the two strings  $\sigma_1$  and  $\sigma_2$ . The  $d$  dimensions of these vectors correspond to all  $d$  distinct tokens that appear in any string in a given finite domain. For example, we assume that  $\sigma_1$  and  $\sigma_2$  originate from the same attribute  $A$ . The vector for a string  $\sigma$  is  $\vec{\sigma}_1 = [a_{1\sigma}, a_{2\sigma}, \dots, a_{d\sigma}]^T$ , where  $a_{i\sigma} = 1$  if string  $\sigma$  contains the token  $a_i$ .

$$\text{Cosine}(\sigma_1, \sigma_2) = \frac{\vec{\sigma}_1 \cdot \vec{\sigma}_2}{\|\vec{\sigma}_1\| \cdot \|\vec{\sigma}_2\|} = \frac{\sum_{i=1}^d a_{i\sigma_1} a_{i\sigma_2}}{\sqrt{\sum_{i=1}^d a_{i\sigma_1}^2} \sqrt{\sum_{i=1}^d a_{i\sigma_2}^2}}$$

**Example 4.1.1.5 Cosine**

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlett Peckard"}$ . We obtain the vectors  $\vec{\sigma}_1 = [1, 1, 0]^T$  and  $\vec{\sigma}_2 = [1, 0, 1]^T$ . Thus the Cosine similarity is

$$\text{Cosine}(\sigma_1, \sigma_2) = \frac{1.0 \cdot 1.0}{\sqrt{1.0^2 + 1.0^2} \cdot \sqrt{1.0^2 + 1.0^2}} = 0.5$$

**TFIDF** The TFIDF similarity measure is an extension of the Cosine similarity measure. It considers the frequency of a token within the attribute value of an object as well as across all values of all objects to be matched. The weight vector for string  $\sigma$  is  $\vec{\sigma}_1 = [w_{1\sigma}, w_{2\sigma}, \dots, w_{d\sigma}]^T$ , where

$$w_{t,\sigma} = \text{tf}_{t,\sigma} \cdot \log_2 \frac{|\Sigma|}{|\{\sigma' \in \Sigma \mid t \in \sigma'\}|}$$

and

- $\text{tf}_{t,\sigma}$  is the token frequency of token  $t$  in string  $\sigma$
- $\log_2 \frac{|\Sigma|}{|\{\sigma' \in \Sigma \mid t \in \sigma'\}|}$  is the inverse token frequency.  $|\Sigma|$  is the total number of strings across all objects;  $|\{\sigma' \in \Sigma \mid t \in \sigma'\}|$  is the number of strings containing the token  $t$ .

The TFIDF similarity of two strings  $\sigma_1$  and  $\sigma_2$  is calculated as:

$$\text{TFIDF}(\sigma_1, \sigma_2) = \frac{\vec{\sigma}_1 \cdot \vec{\sigma}_2}{\|\vec{\sigma}_1\| \cdot \|\vec{\sigma}_2\|} = \frac{\sum_{i=1}^d w_{i\sigma_1} w_{i\sigma_2}}{\sqrt{\sum_{i=1}^d w_{i\sigma_1}^2} \sqrt{\sum_{i=1}^d w_{i\sigma_2}^2}}$$

**Example 4.1.1.6 TFIDF**

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlett Peckard"}$ . Table 4.1a shows ten objects representing manufacturers. We consider these objects for calculating the inverse string frequency values. We observe that any token occurs at most once in a value of attribute *Name*, so the token frequency  $\text{tf}_{t,\sigma}$  is either 0 or 1. We further observe that among the ten attribute values, four contain the token "Hewlett" so  $\log_2 \frac{|\Sigma|}{|\{\sigma' \in \Sigma \mid t \in \sigma'\}|} = \log_2 \frac{10}{4}$ . Based on these values, we obtain  $w_{\text{Hewlett},\sigma} = 1 \cdot \log_2 \frac{10}{4} \approx 0.4$ ,  $w_{\text{Packard},\sigma} = 1 \cdot \log_2 \frac{10}{1} = 1.0$ , and  $w_{\text{Peckard},\sigma} = 1 \cdot \log_2 \frac{10}{1} = 1.0$ . We obtain the following vectors  $\vec{\sigma}_1$  and  $\vec{\sigma}_2$  as depicted in Figure 4.1b.

$$\text{TFIDF}(\sigma_1, \sigma_2) = \frac{0.4 \cdot 0.4}{\sqrt{0.4^2 + 1.0^2} \cdot \sqrt{0.4^2 + 1.0^2}} \approx 0.14$$

Object	Name	Token	$\vec{\sigma}_1$	$\vec{\sigma}_2$
$o_1$	Apple Inc.	Apple	0	0
$o_2$	Microsoft Corp.	Corp	0	0
$o_3$	Lenovo Inc.	Hewlett	0.4	0.4
$o_4$	Lenovo Corp.	Inc	0	0
$o_5$	Hewlett Packard	Lenovo	0	0
$o_6$	Hewlett Peckard	Leonovo	0	0
$o_7$	Hewlett Packart	Microsoft	0	0
$o_8$	Hewlett Peckart	Packard	1	0
$o_9$	Leonovo Inc.	Peckard	0	1
$o_{10}$	Microsoft	Packart	0	0
		Peckart	0	0

(a) Example objects

(b) Weight vectors

Table 4.1: Example computation of TFIDF

### Hybrid measures

Hybrid techniques combine multiple string similarity techniques. For example TFIDF can be extended to additionally consider the similarity between individual tokens. This approach is called Soft-TFIDF [38]. Based on the same principal is the Monge-Elkan similarity measure.

**Monge-Elkan** : The measure proposed by Monge and Elkan [102] uses an internal similarity function  $sim'(t_1, t_2)$  to measure the similarity between two individual tokens  $t_1$  and  $t_2$ . Given two strings  $\sigma_1$  and  $\sigma_2$ , with  $|T_{\sigma_1}|$  and  $|T_{\sigma_2}|$  being their respective number of tokens, and an external inter-token similarity measure  $s'$ , the Monge-Elkan measure is computed as follows:

$$\text{MongeElkan}(\sigma_1, \sigma_2) = \frac{1}{|T_{\sigma_1}|} \sum_{i=1}^{|T_{\sigma_1}|} \max_{j=1, \dots, |T_{\sigma_2}|} s'(t_{i\sigma_1}, t_{j\sigma_2})$$

#### Example 4.1.1.7 Monge-Elkan

Let  $\sigma_1 = \text{"Hewlett Packard"}$  and  $\sigma_2 = \text{"Hewlet Peckard"}$ . We use as internal measure  $s'$  Edit Distance similarity .

$$\begin{aligned}\sigma_1 &= \text{"Hewlett Packard"}; & t_{1\sigma_1} &= \text{"Hewlett"}; & t_{2\sigma_1} &= \text{"Packard"} \\ \sigma_2 &= \text{"Hewlet Peckard"}; & t_{1\sigma_2} &= \text{"Hewlet"}; & t_{2\sigma_2} &= \text{"Peckard"}\end{aligned}$$

$$\begin{aligned}s'(t_{1\sigma_1}, t_{1\sigma_2}) &\approx 0.8571; & s'(t_{1\sigma_1}, t_{2\sigma_2}) &= 0 \\ s'(t_{2\sigma_1}, t_{1\sigma_2}) &= 0; & s'(t_{2\sigma_1}, t_{2\sigma_2}) &\approx 0.8571\end{aligned}$$

$$\begin{aligned}\text{MongeElkan}(\sigma_1, \sigma_2) &= \frac{1}{2}(\max(s'(t_{1\sigma_1}, t_{1\sigma_2}), s'(t_{1\sigma_1}, t_{2\sigma_2})) + \\ &\quad \max(s'(t_{2\sigma_1}, t_{1\sigma_2}), s'(t_{2\sigma_1}, t_{2\sigma_2}))) \\ &= \frac{1}{2}(0.8571 + 0.8571) = 0.8571\end{aligned}$$

### 4.1.2 Other measures

The measures discussed above are the most widely used for object matching. In this section, we summarize similarity measures for two further cases.

- **Phonetic similarity:** Phonetic similarity measures consider the sounds of spoken words, which may be very similar despite large spelling differences. For example, Kageonne is phonetically similar to Cajun despite the different spelling. Soundex [23] is the most common phonetic coding scheme. Soundex assigns identical code digits to phonetically similar groups of consonants.
- **Numeric similarity:** The existing approaches for comparing numerical data are rather primitive. Often, numbers are treated as strings. However, none of the previously discussed string similarity measures yield satisfactory results when applied to numerical data. A more reasonable approach is for to measure the absolute or relative difference of numbers.

## 4.2 Context matchers

Context matchers consider the context or semantic relationships of different objects for similarity computation. Context matchers commonly represent contextual information (e.g., semantic relationships, hierarchies) in a graph structure, see for example [29, 71, 14, 13, 28, 41, 48, 123]. The graph structure allows the propagation of similarity information (e.g., represented as weighted edges or auxiliary nodes) to related entities. For example to disambiguate several persons with the same name one may additionally consider contextual information such as their affiliations or co-authors.

A special case of contexts are hierarchies such as they appear in data warehouse dimensions or XML data. [3] introduces an approach for data warehouses utilizing hierarchical dimensions. A geographical dimension with levels city, state and country exemplifies the approach. Even though we might conclude that the countries USA and Great Britain are duplicates due to their similar spelling (USA, UK), an equality can be excluded when considering the other levels of the dimensions.

In [139] an approach for XML data is presented. It compares XML elements not only based on their content but also by taking into account the similarity of the parents and children nodes. This is a common approach in schema and ontology matching. A similar approach can be found in [24] and in [89].

In [121] constraints are used to express relationships between objects. For example when resolving authors the list of co-authors can be utilized to reconcile persons.

Another approach is to use machine learning [114, 123, 41, 125]. The context information is modelled by conditional random field, markov logic or relation probability models. Based on these models probabilities for the equality of objects are deduced and are then considered as similarity values.

A further possibility to consider context information is to enrich objects with associated information. In [95, 96, 97] attributes of associated objects from other data sources (secondary sources) are added to the original object information from the primary source. An example is the look up and complementation of area codes with the help of the address. This improved the comparison of telephone numbers.

The MOMA framework [130] provides a special context matcher called neighborhood matcher. We provide more details on this approach in the following subsection.

### 4.2.1 Neighborhood matcher

In [130] the authors propose a so-called neighborhood matcher. The neighborhood matcher utilizes association mappings to generate same-mappings. An association mapping interrelates objects of two different semantic types, e.g. venues and publications whereas a same-mapping is a mapping between objects of the same semantic type. The neighborhood matcher requires as input two association mappings of inverse semantic mapping type (e.g., Venue-Publication and Publication-Venue) and a same-mapping. The matcher is a sequence of two compose operations. The first step is the composition of the first association mapping and the same-mapping. In a second step the resulting mapping is composed with the second association mapping.

Figure 4.1 illustrates a concrete example for the execution of the neighborhood matcher. We assume a same-mapping for publications between source  $S_1$  and source  $S_2$  as well as association mappings of type publications of venue. Starting from a venue of source  $S_1$  we can then traverse to the associated publications of source  $S_1$ , utilize the same-mapping to find the corresponding publications of source  $S_2$ ,

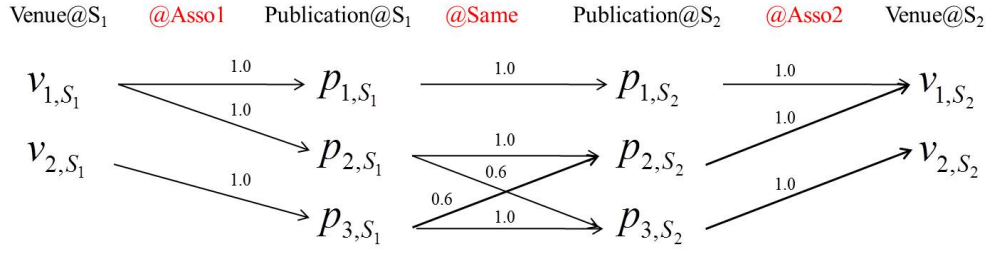


Figure 4.1: Example for the application of the neighborhood matcher

map1			@Asso2			compose		
$v_{1,S_1}$	$p_{1,S_1}$	1	$p_{1,S_1}$	$v_{1,S_2}$	1	$v_{1,S_1}$	$v_{1,S_2}$	$0.8 = 2 \cdot (1 + 1)/(3 + 2)$
$v_{1,S_1}$	$p_{2,S_1}$	1	$p_{2,S_1}$	$v_{1,S_2}$	1	$v_{1,S_1}$	$v_{2,S_2}$	$0.3 = 2 \cdot 0.6/(3 + 1)$
$v_{1,S_1}$	$p_{3,S_1}$	0.6	$p_{3,S_1}$	$v_{2,S_2}$	1	$v_{2,S_1}$	$v_{1,S_2}$	$0.3 = 2 \cdot 0.6/(2 + 2)$
$v_{2,S_1}$	$p_{2,S_1}$	0.6				$v_{2,S_1}$	$v_{2,S_2}$	$0.67 = 2 \cdot 1/(2 + 1)$
$v_{2,S_1}$	$p_{3,S_1}$	1						

Table 4.2: Example execution of compose operator

and traverse the association mapping to reach the venue of source  $S_2$ . Thus, the neighborhood matcher allows to generate a venue same-mapping by composing the association mappings with the publication same-mapping. For the second composition the similarity function *Relative* is applied to prefer correspondences reached via multiple compose paths.

Table 4.2 illustrates the execution of the second composition. Consider the result correspondence between venues  $v_{1,S_1}$  and  $v_{1,S_2}$  which can be reached via two publications  $p_{1,S_1}$  and  $p_{2,S_1}$ . Both compose paths contribute with their path similarity of  $\min(1, 1) = 1$ . The *Relative* function divides twice the sum of all path similarity values, i.e.  $2 \cdot 2 = 4$ , by the number of correspondences for  $v_{1,S_1}$  in map1 (the resulting mapping after composing the first association mapping and the same mapping) ( $=3$ ) and the number of correspondences of  $v_{1,S_2}$  in @Asso2 (the second association mapping) ( $=2$ ). Hence, we obtain a final similarity value of  $s = 4/(3 + 2) = 0.8$ . The example illustrates that the *Relative* similarity function takes into account the number of compose paths as well as the sum of the compose path similarity values. In the example, the output correspondence  $(v_{1,S_1}, v_{1,S_2})$  receives a higher similarity value than  $(v_{1,S_1}, v_{2,S_2})$  since it is supported by more compose paths (2 matching publications vs. 1).

Although both venues from source  $S_1$  have correspondences to both venues from

source  $S_2$ , the similarity values indicate the confidence of such correspondences. The *Relative* combination function thus prefers correspondences reached via multiple compose paths and reduces the influence of wrong correspondences in the underlying publication same-mapping. Therefore a threshold-based selection could be applied afterwards to determine the desired venue mapping.



# 5

## Combination of matchers

There are many possibilities for combining multiple matchers within a match strategy. In general, the combination may be expressed by a decision function which applies matchers from a given set of matchers to determine for each pair of objects whether or not the objects match.

In this chapter we outline different combination approaches. In Section 5.1 we review numerical approaches. Rule-based approaches are discussed in Section 5.2 and learning-based approaches in Section 5.3. In Section 5.4 we introduce workflow-based approaches.

### 5.1 Numerical approaches

*Numerical approaches* combine the similarity values of object pairs  $(o_i, o_j)$  determined by different matchers  $m_1, m_2$ , etc. by a numerical combination function  $f$ . The combination function determines how the similarity values  $s_{m_k}(o_i, o_j)$  should be combined for the merged correspondence  $(o_i, o_j, s)$ . Typical combination functions are average, minimum and maximum [130]. The resulting numerical value computed by the combination functions is mapped to a match or non-match decision usually with the help of a selection step. Selections are typically specified by constraints on the similarity values, e.g., a *Threshold* constraint returns all correspondences above a given similarity value. Additionally the selection step can consider domain-specific constraints, e.g., to require that the publication year of matching publications should not differ by more than one year.

## 5.2 Rule-based approaches

*Rule-based approaches* derive the match decision by a logical combination (or predicate) of match conditions. A match condition can be a threshold condition defined on the similarity value computed by a single matcher  $m$ :  $C = m(o_i, o_j) > t$ , or by several matchers (e.g., using a numerical combination function  $f$ ) or express a hard constraint (e.g., every paper has a single publisher) [45, 121]. A *simple match rule*  $R$  consists of the logical conjunction of  $n$  match conditions:  $R = \bigwedge_{i=1}^n C_i$ . For example, two books may be considered to match if the string similarities of their title and author attributes both exceed certain thresholds. Such simple match rules consisting only of threshold conditions for single attribute value matchers have also been called *similarity joins*; their restricted structure permits an efficient execution in many cases [4]. *Complex match rules* allow the combination of multiple simple match rules, e.g., by disjunction  $\bigvee_{i=1}^n R_i$ . Many approaches, e.g. [63, 88, 5], require a human expert to specify such match rules declaratively.

## 5.3 Learning-based approaches

Key decisions to be made for the specification of a combination strategy expressed as a numerical combination function, match rules, or a match workflow include selecting and configuring the matchers to be used. The chosen configuration can have a large impact on the overall quality but even experts will find it difficult and time-consuming to determine a good selection. The use of supervised (learning-based) approaches or learners aims at automating the process of object matching to reduce the required manual effort. Learning-based approaches, e.g., Naive Bayes [119], logistic regression [115], Support Vector Machine (SVM) [20, 18, 100, 119] or decision trees [149, 60, 119, 128, 129, 135] have so far been used for some subtasks, e.g., determining suitable parameterizations for matchers or adjusting combination functions parameters (weights for matchers, offsets). However, learning-based approaches require suitable training data and providing such data typically involves manual effort. Furthermore, some decisions (e.g., selection of the similarity functions and attributes to be evaluated) may still have to be determined manually. Hence it is important to analyze for learning-based approaches which tasks still require manual decisions. In the following subsections we describe in more detail the learning-based approaches that have been implemented within FEVER.

### 5.3.1 Decision Tree

A decision tree specifies which matchers are to be applied and in what order (see Figure 5.1). Each node of the tree contains a test whether or not a certain similarity threshold is exceeded for a selected matcher. The match decision is reached by

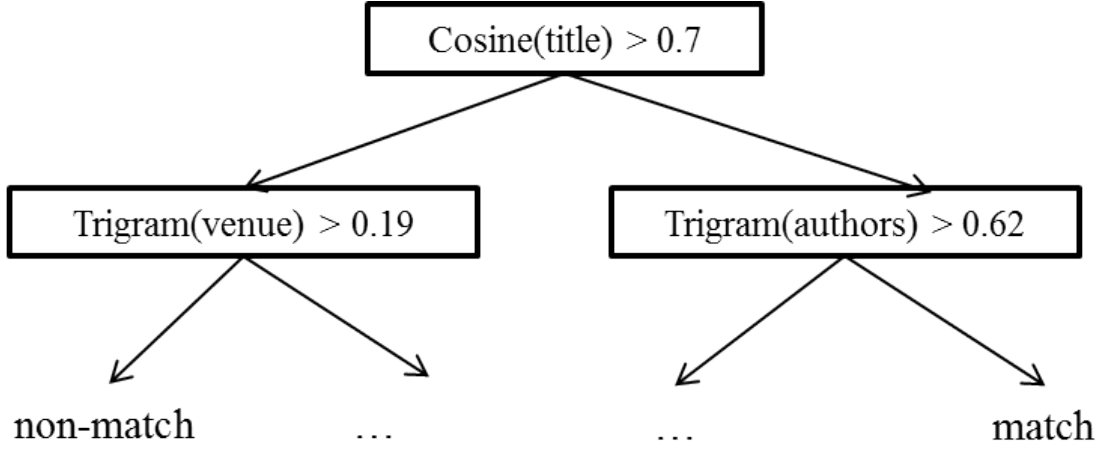


Figure 5.1: Example of learning-based matcher combination with decision tree

starting with the test of the root node and continuing with the further tests until a leaf node is reached. A decision tree is created by determining the most useful matchers to classify examples. A measure called information gain measures how well a matcher divides the given set of training examples by their classification (match/non-match). Creating a decision tree is an iterative process, where the matcher with the greatest information gain is chosen at each level [101].

### 5.3.2 Logistic Regression

Logistic Regression uses a weighted sum of the similarity values of the individual matchers to determine a probability whether two entities match. The combination corresponds to a function of the form:

$$Pr(y = 1|x_1, \dots, x_k) = \frac{\exp(\beta_0 + \sum_{j=1}^k \beta_j x_j)}{1 + \exp(\beta_0 + \sum_{j=1}^k \beta_j x_j)}$$

where  $x_1, \dots, x_k$  are the similarity values given by the selected matchers,  $\beta_j, j = 1, \dots, k$  are weights for the similarity values and  $\beta_0$  is a constant. The following equation is an example for a combination determined by Logistic Regression for a bibliographic match task (with attributes for publication title and authors):

$$f_{\text{LogisticRegression}}(a,b) = \begin{cases} \text{match} & \text{if } \frac{\exp(10.3 - 10.9 \cdot \text{Trigram}(\text{title}_a, \text{title}_b) - 3.9 \cdot \text{Cosine}(\text{authors}_a, \text{authors}_b))}{1 + \exp(10.3 - 10.9 \cdot \text{Trigram}(\text{title}_a, \text{title}_b) - 3.9 \cdot \text{Cosine}(\text{authors}_a, \text{authors}_b))} > 0.5 \\ \text{non-match} & \text{otherwise} \end{cases}$$

The strategy combines  $k = 2$  matchers: Trigram is applied to the title values and Cosine to the authors values. The resulting similarity values for Trigram and Cosine

are weighted with coefficients determined by the learner (10.9 for Trigram, 3.9 for Cosine). The constant  $\beta_0$  is 10.3. For two objects  $a \in A$  and  $b \in B$  a probability is calculated according to the formula and the entities are assumed to match if the probability exceeds a threshold of 0.5.

### 5.3.3 Support Vector Machine (SVM)

For the SVM the  $k$  selected matchers span a  $k$ -dimensional space. The matching and non-matching object pairs from the training set are represented as coordinates in this space. The basic idea of the Support Vector Machine (SVM) is to seek an optimal hyperplane that separates the coordinates representing matching object pairs from the coordinates representing non-matching object pairs with maximum distance. SVM solves the problem that in some cases there is no linear separating hyperplane. It therefore maps the coordinates to a higher dimension space using a kernel function, and seeks a hyperplane in that space. The hyperplane is defined as

$$H = \{x \mid \langle w, x \rangle + b = 0\}$$

where  $w$  is normal to the hyperplane,  $|b|/\|w\|$  is the perpendicular distance of the hyperplane to the origin (offset or bias), and  $\|w\|$  is the Euclidean norm of  $w$ . The vector  $w$  and the offset  $b$  define the position and orientation of the hyperplane in the input space. After the optimal parameters  $w$  and  $b$  are found, the learned object matching strategy is a function of the form:

$$f(x) = \text{sgn} \left( \sum_{i=1}^k w_i x_i + b \right)$$

That is we determine a weighted sum of similarity values and assume a match (non-match) if the sum is positive (negative).

A combination strategy generated with the SVM might look like the following:

$$f_{\text{SVM}}(a,b) = \begin{cases} \text{match} & \text{if } 0.8 \cdot \text{Trigram}(\text{title}_a, \text{title}_b) + 0.5 \cdot \text{TFIDF}(\text{authors}_a, \text{authors}_b) - 1.1 > 0 \\ \text{non-match} & \text{otherwise} \end{cases}$$

The strategy combines  $k = 2$  matchers: Trigram is applied to the title attribute values and TFIDF to the authors values. The resulting similarity values for Trigram and for TFIDF are weighted with weight  $w_1 = 0.8$  and weight  $w_2 = 0.5$ . The offset  $b$  is  $-1.1$ . For two entities  $a \in A$  and  $b \in B$  the weighted sum is calculated and the entities are assumed to match if the sum is positive.

### 5.3.4 Multiple Learning

In machine learning there exists a large body of work on how to combine several learners for improving classification accuracy. In [76] the authors proposed various combination strategies.

*Majority voting* is one of the most popular strategies. With this strategy a combined decision is derived from the majority consensus of several basic learners. The motivation for the combined learning is to compensate weaknesses of individual learners and to thus improve the overall match quality and robustness.

Let  $M = \{M_1, \dots, M_k\}$  be the set of mappings computed by  $n$  basic learners. Then majority voting determines the result mapping from the  $n$  input mapping as

$$M_{\text{majority voting}} = \left\{ (o_i, o_j) \mid (o_i, o_j) \in M_j \ \forall j \in J \subset \{1, \dots, k\}, |J| \geq \frac{k}{2} \right\}.$$

That means the resulting mapping contains all correspondences that appear in at least half of the input mapping determined by the  $k$  base learners.

Other popular approaches to combine multiple learners include Bagging, Boosting and Stacking [148, 29].

In Bagging (short for bootstrap aggregating) several learners are parallel trained on different training examples and the learned models are combined by voting. On the other hand, Boosting learns the base learners sequentially. The examples that are classified incorrectly by a previous learner will be given a higher weight when training the new learner in order to improve the new learner on these difficult examples.

Bagging and boosting are designed to combine multiple learners of the same type, while Stacking can combine different types of learners. The basic idea of Stacking [144] is to first train several base-level learners and then train a meta-level learner on the predictions of the base-level learners.

## 5.4 Workflow-based approaches

A *Workflow-based combination* of matchers allows almost arbitrary complex combinations of matchers, e.g., to apply a sequence of matchers to iteratively refine a match result or to combine the results of independently executed matchers. For schema matching, Rahm distinguishes in [116] between a sequential, parallel or mixed combination of matchers. This classification can be transferred to object matching. Figure 5.2 illustrates the three combination possibilities. In the sequential approach the matchers are executed consecutively and the results of initial matchers are used as input by subsequent matchers. In the parallel matcher strategy, individual matchers are autonomous and can be executed independently from each

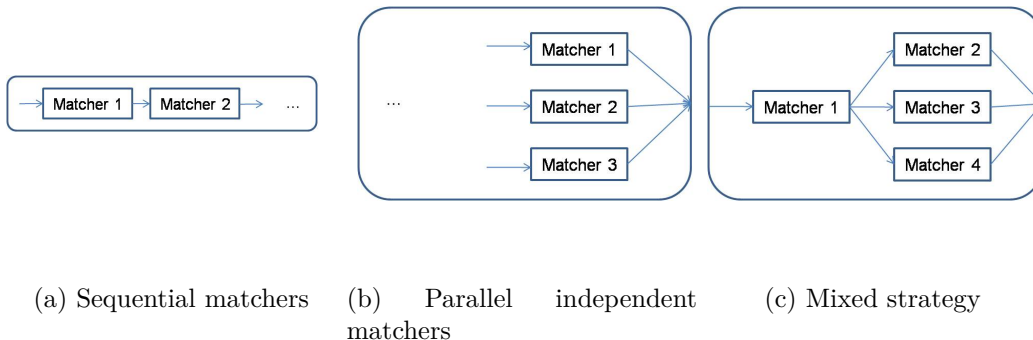


Figure 5.2: Workflow-based combination following [116]

other. This offers a high flexibility to select matchers for execution and combination. Furthermore, these matchers may also physically be executed in parallel, e.g. on multi-core or multi-server hardware. On the other hand, the autonomy of individual matchers may introduce redundant computations, e.g. of attribute similarities to be used for matching. The mixed strategy supports the combination of sequential and parallel matcher execution and is thus most complex.

The MOMA framework [130] supports such a flexible workflow-based combination. The match process is a workflow consisting of a sequence of steps. Each workflow step consists of two parts: matcher execution and mapping combination. The combination of mappings in a step is processed by a specific mapping combiner. A combiner is specified by a mapping operator followed by an optional selection. The mapping operator specifies how the resulting correspondences are determined from the input mappings, e.g. by a merge or compose operation. The selection step filters the correspondences to restrict the mapping to the most similar instances.

In [62] the authors study the composition of mappings in general, i.e., for arbitrary mapping topologies and paths of arbitrary length. The study is conducted in the related field of link discovery. The authors propose different methods to select and combine composed mappings along different paths. They further propose a link-based composition approach for selecting and composing individual links instead of entire mappings. The evaluation on real-world link discovery problems shows that focusing on the most effective mapping paths / links is a good strategy to produce mappings of high quality in very short execution times. For scenarios with only few mapping paths one can apply a selection strategy or the all strategy to create new mappings. For more complex networks with a large number of possible paths the link-based strategy is most promising.

## **Part III**

# **Comparison of object matching approaches**





# 6

## Comparison of existing object matching frameworks

In this chapter we compare twelve proposed frameworks for object matching (Table 6.1 and Table 6.2). The comparison has been published in [84]. Compared to the journal paper the comparison has been extended by two additional frameworks (DuDe [50] and FRIL [70]).

Our selection tries to cover a broad spectrum of different approaches.

We focus on research prototypes but do not consider the more general system approaches on data cleaning and data integration, such as AJAX [55], HumMer [21] and Potters’s Wheel [118]. We also exclude commercial systems such as Choice-Maker, DataCleanser (EDD), Merge/Purge Library (Sagent/QM Software) or MasterMerge (Pitney Bowes) from our discussion since they are not widely available and their algorithms are not described in the public literature.

We have selected five frameworks that need no training and seven training-based frameworks. We focused on those frameworks which allow the combined use of several match algorithms and for which evaluation results have been published. We believe that our methodology to compare different frameworks can be used to evaluate further systems or to update the characteristics when frameworks improve or new evaluations become known.

In the following sections we focus on the functionality of the different frameworks and discuss specific features of the twelve frameworks. In Section 6.1 we start with the five approaches that do not require training. The learning-based frameworks

are highlighted in Section 6.2 and hybrid frameworks supporting both combination strategies are considered in Section 6.3. In Section 6.4 we first compare the frameworks by comparing their, we then analyze in Section 6.5 their published evaluation results. In Section 6.6 we summarize and discuss the results.

## 6.1 Frameworks without training

### 6.1.1 BN

Leitão et. al. [89] propose a framework for matching XML entities based on a Bayesian network (BN) model. Bayesian networks provide a graph-based formalism to explicitly represent the dependencies among the entities of a domain. This model is derived from the structure of the XML entities to be matched. The approach numerically combines the similarity for direct attributes value of two XML entities as well as the similarity for descendant XML entities. The user has to specify a match probability threshold above which entities are considered matches.

### 6.1.2 MOMA (mapping-based object matching)

MOMA [130] is a domain-independent framework for object matching providing an extensible library of matchers, both attribute value and context matchers. To solve a particular match problem MOMA allows the specification of a workflow of several matchers and combination operators. Each matcher and workflow step determines a so-called same-mapping (set of corresponding object pairs) that can be refined by additional matchers and steps. The final mapping determined by a match workflow is stored in a mapping repository and can be re-used in other workflows. A so-called neighborhood matcher implements a context-based match approach and utilizes semantic relationships between different entities, such as publications of authors or publications of a conference. MOMA supports compose and merge operators to combine different mappings for the same match problem. Different combination functions (avg, min, max, weighted, prefer) can be used to derive a combined similarity value for input correspondences to be combined into a merged correspondence. MOMA does not explicitly offer blocking methods. However, a blocking-like reduction of the search space could be implemented by a liberal attribute matching within a first workflow step whose result is then refined by further match steps.

### 6.1.3 SERF (Stanford Entity Resolution Framework)

The SERF project [10] develops a generic object matching infrastructure with the focus on improving the efficiency of object matching. The authors do not study the

internal details of matchers (similarity functions) but view them as "black boxes" to be invoked by the object matching engine. Different algorithms are provided to minimize the number of invocations to these potentially expensive black boxes by keeping track of previously compared values thus avoiding redundant comparisons. Multiple matchers can be combined by a disjunction of manually defined simple match rules.

### **6.1.4 DuDe (duplicate detection toolkit)**

The duplicate detection toolkit DuDe [50] provides multiple methods and datasets for duplicate detection and consists of several components with interfaces that can be served with individual code. The toolkit offers two overlapping blocking methods: The Sorted-Neighborhood Method (SNM) and a variant of this method that dynamically changes the window size depending on whether the last returned pair was interpreted as a duplicate or not. Multiple matchers can be combined by so-called multi-comparators. At present, multi-comparators for the calculation of the minimum or maximum value, the weighted average, and the harmonic mean are supported. Multi-comparators can also be nested.

### **6.1.5 FRIL (Finegrained Record Integration and Linkage Tool)**

FRIL [70] is a Java-based tool that incorporates a collection of record distance metrics, search methods, and analysis tools. Along its workflow, FRIL provides a set of user-tunable parameters augmented with graphic visualization tools to assist users in understanding the effects of parameter choices. Sorted Neighborhood is the only blocking method provided. Four string matchers are available: Edit Distance, Soundex, Q-gram, and equality. Normalized weighted sum is provided to combine multiple matchers.

## **6.2 Learning-based frameworks**

### **6.2.1 Active Atlas**

The Active Atlas system proposed by Tejada et. al. [128, 129] allows the learning-based determination of match rules by utilizing a combination of several decision tree learners. Training selection is semi-automatic to minimize the number of required training examples. This is achieved by letting the decision tree learners vote on the most informative example to be classified next by a user ("active learning"). Attribute value matchers use a variant of TF-IDF that allows considering a variety of additional information (e.g., stemming, abbreviations) to determine the common

tokens of two attribute values. A disjoint blocking strategy based on hashing is supported.

### 6.2.2 MARLIN (Multiply Adaptive Record Linkage with INduction)

MARLIN [18, 19] employs a learning-based approach for combining multiple matchers using the SVM at two levels. At the first level attribute value matchers are tuned. At the second level the SVM is applied to determine a combination of the tuned matchers from the previous step. MARLIN utilizes the canopies clustering method using Jaccard similarity for overlapping blocking. Two methods for semi-automatic training selection are supported: Static-active and weakly-labeled negative selection. Static-active selection compares the entities to be resolved with some string similarity measure and selects only pairs that are fairly similar according to this measure to find near-duplicate object pairs for training. For "legacy" datasets with few duplicate entries the weakly-labeled negative training selection is proposed. It randomly selects object pairs with few shared tokens for training; these pairs are thus likely non-duplicates.

### 6.2.3 Multiple Classifier System

In [148] a multiple classifier system approach is proposed that employs a variety of supervised learners for combining matchers, including decision trees, 1-rule, Naïve Bayes, linear and logistic regression, back propagation neural network, and k-nearest neighbors. Furthermore, meta-combination approaches for combining several supervised learners are supported, namely cascading, bagging, boosting, and stacking. While bagging and boosting combine multiple supervised learners of the same type, cascading and stacking are used to combine supervised matchers of different types (e.g., logistic regression and decision tree learning). Blocking support is not explicitly mentioned, but the evaluation suggests that some blocking method has been applied.

### 6.2.4 Operator Trees

Chaudhuri et al. [25] specify object matching strategies by operator trees which correspond to the union (disjunction) of multiple similarity joins. Manually labeled training samples are used to construct the operator trees by a recursive divide and conquer strategy. The maximum number of similarity joins in an operator tree and the maximum number of similarity functions per similarity join can be restricted by the user. Blocking is not explicitly supported. However, the authors state that

the canopies clustering method using Jaccard similarity is applied in the evaluation. In their evaluation they compare the operator tree approach to a numeric matcher combination utilizing the SVM is considered.

## 6.3 Hybrid frameworks

### 6.3.1 TAILOR

TAILOR [52] is a toolbox for record linkage supporting numerical and rule-based combination approaches for multiple matchers without training as well as learning-based. Five similarity functions are provided for attribute value matching, namely hamming distance, edit distance, Jaro's algorithm, q-grams and soundex. For learning-based combination the user can choose among three probabilistic approaches for numerical combination and two rule-based approaches utilizing decision tree learning. Training data must be manually provided by the user. Disjoint as well as overlapping blocking methods are supported.

### 6.3.2 FEBRL (Freely Extensible Biomedical Record Linkage)

FEBRL [33, 32] is a hybrid framework supporting a learning-based numerical combination approach utilizing the SVM as well as numerical approaches without training. FEBRL is the only one of the considered frameworks that is freely available on the web under an open source software license. It was originally developed for object matching in the biomedical domain (hence the name). A large selection of 26 different similarity measures is available for attribute value matching. FEBRL supports one disjoint as well as three overlapping blocking methods. Besides manual training selection two strategies for automatic training are supported [32]: threshold- and nearest-based. Both methods select object pairs automatically and do not require manual labeling by a user. To determine matching / non-matching training examples the threshold method selects object pairs whose similarity values are within a certain distance to exact similarity or total dissimilarity for all considered matchers. The nearest method sorts the similarity vectors of the object pairs according to their distances from the vectors containing only exact similarities and only total dissimilarities, respectively, and then selects the nearest object pairs for training.

### 6.3.3 Context Based Framework

The Context Based Framework [29] is a graph-based hybrid framework supporting a two stage learning-based numerical combination approach. The first stage tunes

attribute value and context matchers using mainly SMOreg, a support vector regression approach. At the second level a second learner, e.g., Logistic Regression is applied to determine a combination of the tuned matchers from the previous step. The Context Based Framework utilizes a canopy-like technique for blocking. Methods for training selection are not supported. Manually labeled training samples have to be provided for both stages of the combination approach.

## 6.4 Functional comparison

In this section we focus on the functionality of the different frameworks. In the next section, we analyze published evaluation results for the frameworks. We compare the frameworks regarding to the following comparison criteria:

- Object type
- Blocking methods
- Matchers
- Combination of matchers
- Training selection

Table 6.1 and Table 6.2 compare the twelve selected frameworks for object matching within three groups. The first group includes (five) frameworks that do not utilize training data, while the (four) frameworks of the second group depend on training data. The third group includes hybrid approaches which support supervised matcher combinations as well as the manual specification of object matching strategies without training. For each group, the frameworks are listed in chronological order of their year of publication. Except for DuDe all other inspected frameworks support only one type of entities either relational (eleven frameworks) or XML (one frameworks). All frameworks focus on offline matching, i.e. they do not yet cover online matching. Online matching has been addressed to some extent in [15] and [16]. Five of the twelve frameworks operate without training; three of the seven learning-based frameworks, the Context based framework [29], FEBRL [33, 32], and TAILOR [52], also support the manual specification of object matching strategies without training (hybrid frameworks). Four of the learning-based frameworks expect the users to provide suitable training data manually, while Active Atlas [128, 129], MARLIN [18, 19], and FEBRL [33, 32] have also investigated (semi-)automatic training methods.

Most learning-based frameworks provide explicit support for blocking based on disjoint or/and overlapping object partitioning. This seems to be influenced by the fact

	BN [89]	MOMA [130]	SERF [10]	DuDe [50]	FRIL [70]
<b>Object type</b>	XML	relational	relational	relational	relational
<b>Blocking</b>	-	-	-	manual	manual
key definition					
partitioning					
disjoint					
overlapping				sorted neighborhood	sorted neighborhood
<b>Matchers</b>	attribute value, context	attribute value, context	attribute value	attribute value	attribute value
<b>Matcher combination</b>	numerical	workflow	rules	workflow	workflow

Table 6.1: Overview of frameworks without training (- means not present, ? means not clear from publication)

CHAPTER 6. COMPARISON OF EXISTING OBJECT MATCHING FRAMEWORKS

	<b>Active Atlas</b> [128, 129]	<b>MARLIN</b> [18, 19]	<b>Multiple Classifier System</b> [148]	<b>Operator Trees</b> [25]	<b>TAILOR</b> [52]	<b>FEBRL</b> [33, 32]	<b>Context Based Framework</b> [29]
<b>Object type</b>	relational	relational	relational	relational	relational	XML, relational	relational
<b>Blocking</b>							
key definition	manual	manual	manual	manual	manual	manual	manual
partitioning							
disjoint			?		Sorting, hashing	Sorting	
overlapping	hashing	canopy clustering	?	canopy clustering	sorted neighborhood	Sorted neighborhood, q-gram, canopy clustering	canopy like
<b>Matchers</b>	attribute value	attribute value		attribute value	attribute value	attribute value	attribute value, context
<b>Matcher combination</b>	rules	numerical, rules	numerical, rules	rules	numerical, rules	numerical	numerical, rules
Learners	decision tree	SVM, decision tree	7 (SVM, decision tree, etc.)	operator tree algorithm, SVM	probabilistic, decision tree	SVM	diverse (SMOreg, Logistic, etc.)
<b>Training selection</b>	manual, semi-automatic	manual, semi-automatic	manual	manual	manual	manual, automatic	manual

Table 6.2: Overview of learning-based and hybrid frameworks (- means not present, ? means not clear from publication)



that the trained methods to determine matches, such as SVM or decision trees, are computationally expensive so that blocking is mandatory to reduce the search space even for small-sized match tasks. Canopy-like clustering and sorted neighborhood are the most common blocking techniques. The definition of the blocking key is not yet derived from training data but has to be specified manually in all learning-based frameworks. This is a serious limitation for the optimization potential of learning-based methods.

All frameworks support attribute value matchers utilizing a large variety of string similarity functions. Table 6.3 and Table 6.4 indicate which similarity functions have been used in the evaluations). Three frameworks additionally provide context matching but only one of the learning-based frameworks. All frameworks support the combination of multiple matchers. Most frameworks support (complex) match rules, numerical combination functions, or both approaches. Three frameworks MOMA [130], DuDe [50], and FRIL [70] allow the definition of match workflows.

Six of the seven learning-based frameworks (Active Atlas [128, 129], Context Based Framework [29], MARLIN [18, 19], Multiple Classifier System [148], Operator Trees [25], TAILOR [52]) utilize training for learning match rules, mostly by employing decision tree algorithms. Training data is utilized to automatically determine the order in which different matchers are applied as well as threshold conditions on the similarity values computed by the matchers. Another approach pursued by six learning-based frameworks (Context Based Framework [29], MARLIN [18, 19], Multiple Classifier System [148], Operator Trees [25], FEBRL [33, 32], TAILOR [52]) is to utilize training for automatically determining a numerical combination function  $f$ . The combination function is determined through the choice of the employed supervised learning algorithm. The most often employed learner for this task is the SVM. The Context Based Framework [29], the Multiple Classifier System [148] and FEBRL [33, 32] frameworks consider other supervised learners besides the SVM (e.g., Logistic Regression) for determining a combination function. All learning-based frameworks optimize the combination of a manually predetermined set of matchers, i.e. they do not explicitly select which attributes or similarity functions should be used.

## 6.5 Evaluation comparison

In this section we compare the published evaluations of the considered frameworks. Table 6.3 and Table 6.4 summarize the evaluations.

To compare the evaluations of object matching frameworks we consider the following criteria:

- **Type of test problems:** Test problems may involve real-world data sources or may be artificially generated.

- **# Domains/# Sources/# Tasks:** How many domains, sources and match tasks are considered?
- **Semantic object types:** What kinds of match problems have been solved?
- **Min/Max # entities:** What is the minimum/maximum number of entities involved in a match task?
- **Min/Max # attributes:** What is the minimum/maximum number of attributes used for solving a match task?
- **Used Matchers:** Which matchers (similarity functions) have been used?
- **# Training examples:** How many examples were used for training?
- **Blocking performance measures:** Which measures as described in Section 3.1 were used for the evaluation of blocking techniques?
- **Effectiveness measures (achieved max. values):** Which measures as described in Section 2.2 have been used to determine effectiveness?

The values in parentheses in Table 6.3 and Table 6.4 give the maximum reported values for the considered effectiveness measures.

- **Efficiency measures:** The efficiency is commonly determined in terms of the execution time (*ET*). For learning-based approaches the training time (*TT*) has to be considered additionally.

The values in parentheses in Table 6.3 and Table 6.4 give the reported value ranges for the considered efficiency measures.

Table 6.3 and Table 6.4 give a summary about the evaluations of the considered frameworks as reported in the corresponding research papers. All frameworks offer a selection of matchers and combination approaches resulting in a huge number of possible configurations. Since it is impossible to exhaustively explore the configuration space, the reported evaluations cover only a restricted choice of configurations.

The summarized results indicate a substantial diversity between the different studies. The evaluations employ up to seven test problems from one to five domains. Popular domains for evaluation are the bibliographic, E-commerce and personal data domains. Some test problems from the RIDDLE repository (e.g., Cora, Restaurant) are used in the evaluations of five frameworks (Dude, Active Atlas, MARLIN, Operator Trees, FEBRL). Most test problems are small and deal only with a few hundred entities; the largest test problem used in the evaluations matches 1,26 million entities. All frameworks are evaluated on real-world test problems; the evaluations of BN, FEBRL and TAILOR additionally utilized artificially created test problems. While the use of real datasets is important to test for real-life conditions it is difficult to determine a perfect match result for them, especially for very large datasets.

	BN [89]	MOMA [130]	SERF [10]	DuDe [50]	FRIL [70]
Type of test problems	real, artificial	real	real	real	real
# Domains/# Sources/# tasks	2/3/3	1/3/6	2/1/2	2/2/2	1/2/1
Semantic object types	CDs, movies	publications, authors, venues	products, hotels	CDs, restaurants	persons
Min/Max # entities	1,000/10,000	258/66,879	5,000/14,574	864/9,763	1,26 mio
Min/Max# attributes		1/3	3/8	1/2	5/5
Used Matchers	edit distance, match probability of descendants	Trigram, neighborhood	(exact) equality, Jaro-Winkler, numeric difference	edit distance, soundex	equality, edit distance
Blocking performance measures					
Effectiveness measures (achieved max values)	P/R (1.0/0.99)	P/R/F (?/?/0.988)	P (1.0)	-	P/R (0.99/0.95)
Efficiency measures (achieved values)		ET (<1-15h)	ET (<1-15h), # feature value comparisons	# comparisons	ET (20min)

Table 6.3: Overview of evaluations for frameworks without learning

Moreover, real datasets are highly different in match difficulty making it problematic to generalize findings to other datasets. Artificially created match problems allow for more controlled test conditions for arbitrary many entities. However, artificially introduced errors are not domain-specific, do not necessarily cover all error types, and their distribution might be unrealistic.

Up to 18 attributes are considered for solving a given object matching task. Most commonly the used attribute value matchers are based on the string similarity measures edit distance, (exact) equality, Jaro-Winkler and TF-IDF. From the large selection of 26 different similarity measures available for attribute value matching with FEBRL only two are used in the evaluation. For the evaluation of TAILOR it is unclear which of the five supported similarity measures are applied in the evaluation.

The effectiveness of the frameworks is commonly evaluated in terms of precision, recall, and F-measure. The weaker accuracy measure has been used in the evaluations of Active Atlas, the Multiple Classifier System and TAILOR. As expected the reported effectiveness values are typically high. However these values do not allow a representative comparison of the relative effectiveness of the frameworks due to the high diversity of the test problems and employed evaluation methodology (e.g., different measures, training sizes, etc.).

Efficiency is evaluated for seven of the twelve frameworks, mostly by measuring the execution times for matching (SERF, Dude, FRIL, Multiple Classifier System, Operator Trees, FEBRL, Context Based System). For three learning-based frameworks (Context Based System, Multiple Classifier System, Operator Trees) the time needed for training is reported.

The evaluations concerning blocking methods are rather unsatisfying so far. Only for FEBRL and TAILOR different blocking methods are compared in terms of reduction ratio, pairs completeness and F-score on artificial test problems. Experimental comparisons of blocking algorithms on real world test problems are missing.

The influence of training selection is an important issue in evaluating learning-based frameworks. Unfortunately, some evaluations provide few details on the selection (Context Based System, Multiple Classifier System, Operator Trees, TAILOR) and size of training data (FEBRL). Three of the seven learning-based frameworks (Active Atlas, FEBRL, MARLIN) have investigated the issue of training selection and evaluated (semi-)automatic approaches. The influence of different training set sizes is investigated for Active Atlas, MARLIN, Operator Trees, and TAILOR. In the evaluation of the Multiple Classifier System a relatively large amount of training data is used thus favoring good match quality at the expense of a high manual effort for labeling.

In the following, we briefly discuss selected evaluation details for each framework. A summary of the effectiveness performance is given for those frameworks where the evaluation reported F-measure values.

### Frameworks without training

**BN:** The evaluation considered artificial as well as real-world datasets on movies and CDs (IMDB, IMDB+FilmDienst, FreeDB). The evaluation investigated the impact of the choice of the threshold on effectiveness, the impact of data quality on effectiveness, i.e., how errors (e.g., typos or missing data) and error frequencies affect the result, and a comparison to DogmatiX [139], a previously proposed XML object matching framework. In the evaluation all attribute values are considered as textual strings and edit distance is used for similarity computation. The framework reported high precision and recall values in all cases.

**MOMA (mapping-based object matching):** The authors evaluated the framework on match tasks from the bibliographic domain. The evaluation considered three data sources (Google Scholar, DBLP, ACM Digital Library) and matching tasks for publications, authors, and venues. The combination of several matchers and mappings was shown to compensate weaknesses of individual strategies; the neighborhood matcher (see 4.2.1) proved to be very valuable.

**SERF (Stanford Entity Resolution Framework):** The evaluation regarded comparison shopping and hotel datasets from Yahoo. Two manually defined match strategies were used for the evaluation. The evaluation considered the efficiency in terms of the runtimes of the match strategies. The runtime results mainly depend on the number of attribute value comparisons. The most dominant factor of the runtime for any algorithm compared in the evaluation turns out to be the total time for comparing string values.

**DuDe (duplicate detection toolkit):** The evaluation only considered the efficiency in terms of the number of comparisons compared to Febrl on two match tasks (restaurants and CDs from the Riddle repository). DuDe states to take less time than Febrl for both match tasks (6.5 sec vs. 95 sec, 21 sec. vs. 65 sec.).

**FRIL (Finegrained Record Integration and Linkage Tool):** The evaluation regarded linking personal data in a database of birth defects to a birth certificate database. Four manually defined match strategies using equality and edit distance matchers on five attributes were used for the evaluation. The evaluation considered the effectiveness in terms of precision and recall and the efficiency in terms of runtime.

### Learning-based frameworks

**Active Atlas:** The evaluation considered three datasets involving restaurants, companies and airports. The evaluation compared decision tree learning for

rule-based matcher combination with two baseline combination approaches without training. For training selection a random approach and the semi-automatic active learning were compared. The experimental results show that the learning-based approaches achieve higher accuracy values than the manually specified baseline object matching strategies utilizing no training. Active learning required fewer labeled examples than random training selection.

**MARLIN:** The evaluation compared the performance of support vector machines (SVM) to decision trees revealing that SVM significantly outperform decision trees when training data is limited. Further experiments demonstrate the comparative utility of static-active, weakly labeled negative and random training selection using TF-IDF and edit distance for attribute value matching and SVM as the learner. The highest performance was achieved when training data is a mix of examples selected using the static-active strategy and randomly chosen object pairs. In situations where human labeling of negatives is expensive or infeasible (e.g., due to privacy issues), using weakly-labeled non-duplicates is found to be valuable for automatic acquisition of negative examples.

**Multiple Classifier System:** The evaluation of the system is rather limited. It is restricted to a single dataset (airline passengers) from a single domain. The dataset consisted of 25,000 passenger pairs of which 5,000 were matching pairs. Various experiments compare accuracy, training time and matching time of different learning-based combination approaches. For training a relatively large amount of training data is used (66% of the 25,000 examples) thus favoring good match quality however at the expense of a high manual overhead for labeling.

**Operator Trees:** The evaluation considered several datasets from different domains: organization names and addresses, personal data of hurricane evacuees, and three datasets from the RIDDLE repository (Cora, Restaurant, Bird). The approach is compared with a domain specific address cleansing solution as well as with the SVM. On the evacuees dataset the SVM offers better recall at higher precision. However, at lower precision the recall of operator trees is close to that of SVMs. Therefore the authors propose to use operator trees as an efficient filter (with low target precision) before invoking SVMs. On the smaller RIDDLE datasets, the recall values achieved by operator trees are comparable to that of the SVM at the same precision. For the Bird dataset, significantly better recall values are obtained. The efficiency evaluations illustrate that a DBMS-based operator tree implementation executes significantly faster than SVM models, even if they employ blocking.

### Hybrid frameworks

**TAILOR:** The evaluation compares various string similarity measures (Bigrams, Trigrams, EditDistance, Jaro) for attribute value matching, disjoint (sorting) and overlapping (sorted neighborhood) blocking, and matcher combination strategies utilizing training (probabilistic, decision tree, hybrid) and without training. The blocking evaluation showed no clear advantage for overlapping approaches with larger window size over disjoint methods with short key sizes. Maximum pairs completeness and reduction ratios of about 0.95 are achieved. The other evaluation results show that (i) attribute value matchers based on Jaro’s algorithm perform better than the other attribute value matchers. (ii) learning-based approaches outperform approaches without training.

**FEBRL:** In [7] two disjoint (sorting, sorted neighborhood) and two overlapping (bigram indexing, canopy clustering) blocking methods are compared in terms of reduction ratio, pairs completeness and F-score on artificial test problems with varying object sizes. Both bigram indexing and canopy clustering outperform the two disjoint blocking methods with the right parameter settings. Pairs completeness with the two disjoint methods had a maximum of about 0.96, whereas the maximum for canopy clustering with TFIDF is 0.98. Canopy clustering also achieved the best reduction ratio of 0.9 and best F-score of over 0.98. However, the method seems highly dependent on the choice of the threshold. With a suboptimal choice pairs completeness drops to 0.8. The evaluation in [32] evaluates the threshold and the nearest-based training selection methods. The evaluations showed that a SVM-based matcher combination with automatic training selection is often better than a numeric matcher combination without training.

**Context Based Framework:** The evaluation considered match tasks from two domains, personal webpages and bibliographic references (publications, authors, departments and organizations). The authors compare various attribute value and context matchers (eTFIDF, connection strength), matcher combination strategies utilizing training (mainly SMOreg, Logistic) and manually configured strategies. The evaluation shows that matcher combination strategies considering context and utilizing training can outperform manually configured combined strategies. Effectiveness performance measured in terms of F-measure range between 0.599 and 0.89 on the personal webpages task. Training times differ depending on the utilized learner from 0.11 s to 53.66 s . The application times range from less then 1 s to less than 5 s depending on the size of the dataset.

CHAPTER 6. COMPARISON OF EXISTING OBJECT MATCHING FRAMEWORKS

	Active Atlas [128, 129]	MARLIN [18, 19]	Multiple Classifier System [148]	Operator Trees [25]	TAILOR [52]	FEBRL [33, 32]	Context Based Framework [29]
Type of test problems	real	real	real	real	real, artificial	real	real, artificial
# Domains/# Sources/# tasks	3/6/3	2/4/6	1/1/1	5/5/5	2/2/2	4/4/7	2/2/2
Semantic object types	restaurants, companies, weather data	restaurants, publications	persons	companies, persons, publications, restaurants, birds	products, persons	persons, restaurants, publications	publications, authors, departments, organizations, persons
Min/Max # entities	862/12,428	295/1,295	?	?	100,000	841/10,000	1085/14,590
Min/Max # attributes	2/3	1/6	18	1/8	7	?/6	?/?
Used Matchers	TF-IDF variant	edit distance, TF-IDF	(exact) equality, soundex, sub-string, edit distance	(exact) equality, jaccard, (generalized) edit distance	?	Jaro-Winkler, numeric difference	eTFIDF, connection strength
# Training examples	Varying (up to 700)	10/20/40/60/-1000	66%	1000/5000/-100000	Varying (1%-100%)		?
Blocking performance measures					RR, PC, $F_s$	RR, PC, $F_s$	
Effectiveness measures (achieved max values)	P A (1.0)	P/R/F (?/?/0.966)	A (0.998)	P/R (0.98/1.0)	A/R (1.0/0.9)	F (0.95)	F (0.89)
Efficiency measures (achieved values)			TT (0.86 - 6050.18 s), ET (0.21 - 3197.25 s)	TT (<1 - 1000s), ET (61 s - 10 d)		ET (<1 - 100.000 s)	TT (0.11 - 53.66 s), ET (1 - 5 s)

Table 6.4: Overview of evaluations for learning-based and hybrid frameworks



## 6.6 Summary and discussion

Object matching frameworks allow the combined use of blocking and multiple matcher algorithms to effectively solve diverse match tasks. They have to meet challenging and partly contradicting requirements, in particular high effectiveness, efficiency, genericity and low manual effort. To assess the current state of art we comparatively analyzed the functionality and published evaluation results of eleven proposed research prototypes for object matching. The comparisons are based on a common set of criteria. Criteria for the functional comparison include the supported choices for blocking methods, matchers, combination of matchers, and training selection. Criteria for the evaluation comparison consider the used test problems, applied match strategies and the achieved effectiveness and efficiency performance. The proposed criteria have value beyond the systems considered here but should enable to categorize and comparatively assess further object matching frameworks and their evaluations.

Our study indicates a research trend towards using supervised (learning-based) and hybrid approaches to semi- automatically determine an object matching strategy for a given match task. Most of the considered frameworks including Operator Trees and the Context Based Framework employ supervised learners to derive numerical combination functions or match rules specifying how multiple matchers should be combined for deriving a match decision. Hybrid frameworks provide the largest scope of methods for solving object matching tasks, in particular for blocking and the combined use of several matchers. Among the considered hybrid frameworks, FEBRL offers the largest selection of different blocking strategies and attribute value matchers, while the Context Based Framework supports the largest number of learners as well as context matching. Unfortunately, the flexibility of the hybrid frameworks comes at the price of an increased complexity for users to choose an appropriate method (selection of matchers to be considered, size and selection of training data) - despite the use of supervised machine learning approaches.

All frameworks focus on offline matching, i.e., they do not yet cover online matching. The definition of the blocking key is not yet derived (semi-)automatically from training data but has to be specified manually in all considered frameworks. While attribute value matchers are well supported the combination of context and attribute matchers is not and should be further studied. Learning-based object matching frameworks should provide more support for (semi-)automatic selection of suitable training data with low labeling effort. So far learning-based approaches only helped to optimize some decisions, e.g., determining parameters for matchers (e.g., similarity thresholds) and combination functions (e.g. weights for matchers) while other decisions (e.g., selection of the similarity functions and attributes to be evaluated) still have to be determined manually.

The published framework evaluations used diverse methodologies, measures, and

test problems making it difficult to assess the effectiveness and efficiency of each single system. While the reported evaluation results are usually very positive, the tests so far mostly dealt with small match problems so that the scalability of most approaches is unclear. Hence, scalability to large test cases needs to be better addressed in future frameworks. Some recent work regarding scalability has focused on computational aspects of string similarity computation [120, 4, 61, 145] and time-completeness trade-offs [90]. Furthermore, we see a strong need for comparative performance evaluations of different frameworks and object matching strategies. Standardized benchmarks for object matching are needed for comparative investigations; first proposals exist [106, 140] but have not yet been implemented or applied. Published evaluation results should also be reproducible by other researchers, ideally by providing the prototype implementations and test data.

# 7

## FEVER - A framework for object matching

The analysis of existing frameworks in the previous part revealed several shortcomings regarding the support of training-based approaches, product matching and the evaluation of different approaches. This motivated us to develop a novel comprehensive framework for object matching - FEVER. The framework supports a wide-range of the previously introduced methods for blocking, matchers and the combination of matchers. Furthermore FEVER offers methods for the comparative evaluation of match approaches and frameworks.

In this chapter we give an overview of our generic object matching framework FEVER (Framework for Evaluating Entity Resolution).

After introducing the architecture and illustrating the overall match process in Section 7.1, we introduce in Section 7.2 the data structures within FEVER. In Section 7.3 we define and describe the various operators that are provided to design object matching workflows. An important aspect of the FEVER framework is the support of training selection methods. Operators for training selection are introduced in Section 7.4. In Section 7.5 we introduce example workflows for non-learning as well as learning-based match approaches. In Section 7.6 we discuss configuration strategies for comparative object matching evaluation. In Section 7.7 we illustrate the implementation and use of the graphical user interface. Finally, we compare in Section 7.8 the functionality of FEVER with the competitive frameworks analyzed in Chapter 6.

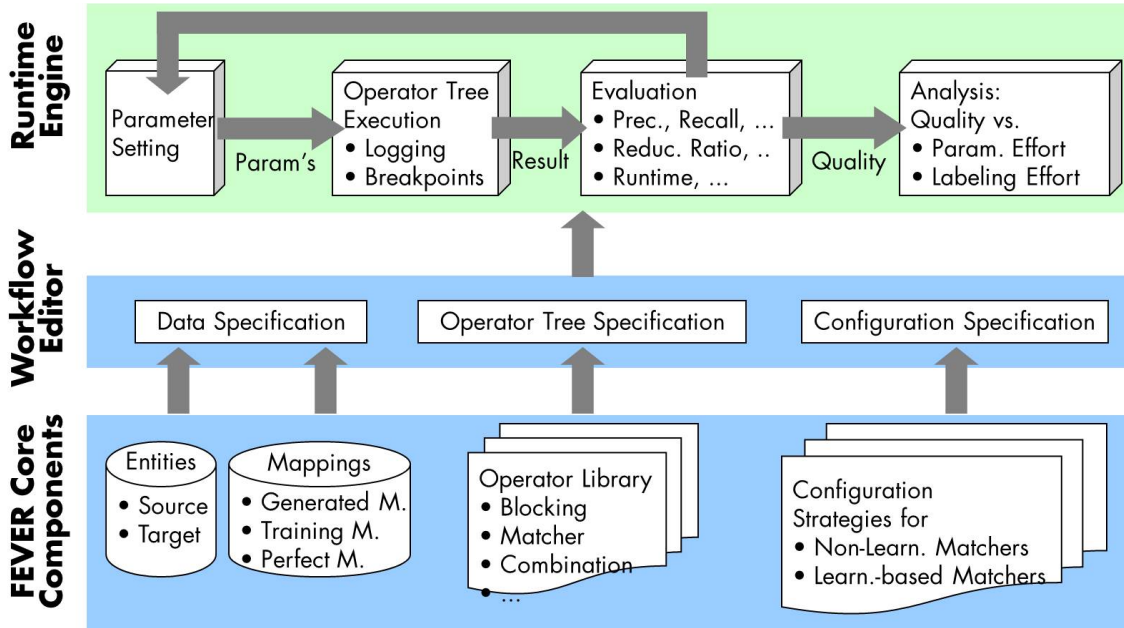


Figure 7.1: Architecture of FEVER

## 7.1 System architecture

Figure 7.1 illustrates the architecture of FEVER consisting of three layers, the *Core Components* to handle match-related data and providing a library of operators for specifying match workflows, the *Workflow Editor* to configure matchers and match workflows, and the *Runtime Engine* to control the execution of match workflows.

FEVER supports match processing as a workflow of several match steps.

A FEVER workflow specification has three parts: input data definition, operator tree, and configuration specification. All parts can be edited in the GUI-based workflow editor.

The input data includes the objects of one or two data sources to be resolved, and the perfect match result for evaluating the effectiveness of the object matching workflow. For training-based match approaches, the training data also needs to be provided.

The actual match workflow is declaratively described in terms of an operator tree. Operator trees are a common modeling concept for numerous database problems (e.g., query optimization) and have previously also been used to model object matching approaches [25]. We build on those previous approaches and extend the idea to a flexible method for defining match workflows. The input data sources form the leaves of the tree and non-leaf nodes are operators. The output of the root operator is the final match result specifying the identified correspondences within a so-called mapping (see next section). FEVER provides an extensible operator library with a variety of operators for training selection, blocking, matching, and mapping combi-

nation. Operators define their expected input and delivered output as well as their mandatory and optional parameters. The tree is executed in a post-order traversal sequence and the results of the child operators are input to the father operator. An operator tree is well formed if all operators are provided with their necessary input and mandatory parameters are set.

The operator tree concept provides a high flexibility to specify a tailored workflow for a given match task and supports its comparative evaluation with other approaches. In particular, it allows the selection and combination of several match approaches. Developed operator trees can be saved as building blocks and can then be reused as a sub-tree in other workflow definitions. The operator tree concept and the supported operators are described in the following section.

The third workflow component specifies the execution of the operator tree according to a configuration strategy defining the parameter settings of all operators of the tree. For evaluation purposes the operator tree is usually executed multiple times with different parameter settings. This way the effectiveness of the match workflow for different settings can be evaluated and the effort to find an effective configuration can be determined. Configuration strategies are explained in more detail in Section 7.6.

Specified workflows are stored in a repository and can be executed by the FEVER Runtime execution (upper part of Figure 7.1). The match result of each operator tree execution can be automatically evaluated with the help of the given perfect result and, thus, evaluation measures such as precision, recall, and F-measure can be determined. In addition, performance indicators are recorded, e.g., execution runtime and used main memory. The calculated quality measures are stored and can be compared to the effort of configuring the parameter setting. Hence, FEVER not only allows an evaluation of the match quality, but also facilitates an analysis of the effort expended to reach the respective match quality. As a consequence, FEVER aims at supporting a comparative evaluation of match algorithms by comparing the match quality reached under the same effort. For smaller datasets the execution of a match workflow can be started from the GUI and the results interactively inspected via the build-in plotter. For larger datasets the match workflow can be modelled within the GUI and then executed in batch mode. The FEVER implementation is written in Java and uses the Rapid Miner [99] library of machine learners.

## 7.2 Data structures

FEVER distinguishes between two principal data structures: object set and mapping.

An object set  $O_A$  represents as set of objects from a defined data source  $A$  (e.g., DBLP or Amazon). Objects are of a particular semantic type (e.g., publication or product). They are represented by a set of attribute values.

Objects are interconnected by so-called mappings.

Like [73] we represent mappings by a mapping table with three columns. Each row represents a correspondence consisting of the ids of the domain and range entities and the corresponding similarity value.

We distinguish three kinds of mappings: generated, perfect and training mappings.

A *Generated Mapping*  $M_G$  is the result of an object matching process. Here the similarity value of a correspondence represents the algorithm's confidence that the two objects represent actual the same real-world entity.

A *Perfect Mapping*  $M_P$  is needed to evaluate the effectiveness of an object matching approach, i.e., to calculate evaluation measures such as precision, recall, and F-measure. A Perfect Mapping only contains true correspondences and, thus, the similarity value is equal to 1 for all correspondences.

A Training Mapping  $M_T$  represents training data and is needed as input by training-based object matching approaches. A Training Mapping contains correspondences that represent true duplicates  $(o_i, o_j, 1)$  as well as true non-duplicates  $(o_i, o_j, 0)$ .

## 7.3 Operators

FEVER's operator library offers a variety of operators. The main operator types for blocking, matching and training selection generate mappings as operator output. The uniform mapping data structure is the foundation for the flexible combination of operators within trees. Some previous match approaches represent their results as clusters of entities that are considered to be the same. FEVER can also support such methods by interpreting clusters as a mapping containing pairwise correspondences between all objects of the cluster.

The supported operators can be divided into five classes. The five classes with all corresponding operators are listed in Table 7.1 and are described in the following. Operators for training selection are in more detail discussed in Section 7.4:

Name	Input	Parameters
<b>Attribute operators</b>		
setAttribute	$(O_A)$	Attr <sub>A</sub> : attribute name v: attribute value
<b>Blocking operators</b>		
standardBlocking	$(O_A), (O_A, O_B)$	Attr <sub>A</sub> : attribute name Attr <sub>B</sub> : attribute name
sortedNeighborhood	$(O_A)$	Attr <sub>A</sub> : attribute name Attr <sub>B</sub> : attribute name w: window size
qGramIndexing	$(O_A), (O_A, O_B)$	Attr <sub>A</sub> : attribute name Attr <sub>B</sub> : attribute name q: q gram length $t \in [0, 1]$ : threshold
canopyClustering	$(O_A), (O_A, O_B)$	Attr <sub>A</sub> : attribute name Attr <sub>B</sub> : attribute name $t_l \in [0, 1]$ : loose threshold $t_t \in [0, 1]$ : tight threshold $s \in \{\text{Jaccard, TFIDF}\}$ : similarity measure
<b>Match operators</b>		
simJoin	$(O_A), (O_A, O_B), (M_G)$	Attr <sub>A</sub> : attribute name Attr <sub>B</sub> : attribute name $s \in \{\text{EditDistance, JaroWinkler, MongeElkan, Cosine, Jaccard, Q-Gram, TFIDF, Numeric}\}$ : similarity measure $t \in [0, 1]$ : threshold
<b>Operators for combining matchers</b>		
<i>Workflow-based combination</i>		
union	$(M_G)$	f: combination function
intersect	$(M_G)$	f: combination function
vote	$(M_G)$	f: combination function

Name	Input	Parameters
<i>Filter operators</i>		
threshold	$(M_G)$	$t$ : threshold
maxN	$(M_G)$	$n$ : maxN
maxDelta	$(M_G)$	$d$ : tolerance value
<i>Learning-based combination</i>		
svm	$(M_T)$	$\mathcal{M} = \{m\}$ : set of matchers
logisticRegression	$(M_T)$	$\mathcal{M} = \{m\}$ : set of matchers
decisionTree	$(M_T)$	$\mathcal{M} = \{m\}$ : set of matchers
modelApplication	$(O_A), (O_A, O_B), (M_G)$	
<i>Training selection operators</i>		
random	$(O_A), (O_A, O_B), (M_G)$	$n$ : number of training pairs $t \in [0, 1]$ : minimal threshold $s$ : similarity measure
ratio	$(O_A), (O_A, O_B), (M_G)$	$n$ : number of training pairs $t \in [0, 1]$ : minimal threshold $r \in [0, 0.5]$ : ratio $s$ : similarity measure

Table 7.1: Overview over FEVER operators

### 7.3.1 Attribute operators

Attribute operators manipulate attribute values of objects. With the help of the attribute operator `setAttribute` attributes can be generated and changed. Attribute operators expect as input a single set of objects  $O_A$ .

The operator `setAttribute` has a parameter  $Attr_A$  specifying the name of the attribute that should be added or changed if it already exists. The parameter  $attributeValue$  defines the value of the attribute. The value can be a static value or an expression using the existing attribute values of the object. For example, for a product object we can calculate a new attribute *lump sum price* as the sum of the retail price and the shipping costs. the attribute that should be deleted within all objects.



### 7.3.2 Blocking operators

Blocking operators realize blocking approaches as introduced in chapter 3. They reduce the search space to a subset of the most likely matching object pairs. Blocking operators expect as input either a set of objects  $O_A$  from a single source or two sets of objects  $O_A$  and  $O_B$  from two sources. As result they return a mapping  $M$ . FEVER supports disjoint as well as overlapping blocking methods. All blocking operators have to specify a blocking key. The blocking key is given as an attribute and can be generated beforehand with the available attribute operators. Additional parameters such as the window size for the sorted neighborhood operator might be required.

### 7.3.3 Match operators

FEVER supports a generic attribute value matcher that takes as input either a set of objects  $O_A$  from a single source or two sets of objects  $O_A$  and  $O_B$  from two sources or a mapping  $M$ . Parameters include the (string) similarity measure to be applied and the threshold above which objects are considered to match. Currently the seven string similarity functions discussed in Section 4.1 are implemented as well as a numeric function for comparing numerical values. For two numbers  $a$  and  $b$  this function calculates the similarity as  $s(a, b) = \frac{\min(a,b)}{\max(a,b)}$ .

### 7.3.4 Operators for combining matchers

Operators for combining matchers combine the result mappings from multiple matchers. FEVER supports two kind of combination operators: Workflow-based and Training-based. Workflow-based combination operators combine mappings generated by independently executed matchers while training-based combination operators utilize machine learning algorithms for the combination.

#### Workflow-based combination

FEVER supports the following set operators for combining mappings generated by independently executed matchers:

- **union:** This operator unions the correspondences from all input mappings. Thus, a high recall is achieved. This may lead to a reduction of precision as a correspondence only has to appear in one of the input mappings.
- **intersect:** This operator is very restrictive. It determines only those correspondences that appear in all input mappings. This way the precision is increased at the expense of recall.

- **vote**: This operator realizes a majority vote. It determines all correspondences that appear in more than half of the input mappings.

All three operators obtain a set of  $n, (n \geq 2)$  mappings  $M = \{M_1, \dots, M_n\}$  of the same semantic object type as input.

A required parameter of all three operators is a combination function  $f$  determining the resulting similarity value  $s(o_i, o_j)$  for a correspondence between object  $o_i$  and object  $o_j$  from the similarity values  $s_i$  of the input correspondences  $(o_i, o_j, s_m) \in M_m$ . Consecutively we define all available functions.

- **Max**: This strategy returns the maximal similarity value of any matcher. It is optimistic, in particular in case of contradicting similarity values. Furthermore, matchers can maximally complement each other.

$$s_{\mathbf{Max}}(o_i, o_j) = \max_{m \in M} s_m(o_i, o_j)$$

- **Weighted**: This strategy determines a weighted sum of similarity values of the individual matchers and needs relative weights which should correspond to the expected importance of the matchers.

$$s_{\mathbf{Weighted}}(o_i, o_j) = \sum_{m \in M} w_m \cdot s_m(o_i, o_j)$$

- **Average**: This strategy represents a special case of Weighted and returns the average similarity over all matchers, i.e., considers them equally important.

$$s_{\mathbf{Average}}(o_i, o_j) = \frac{1}{|M|} \sum_{m \in M} s_m(o_i, o_j)$$

- **Min**: This strategy chooses the lowest similarity value of any matcher. As opposed to Max, it is pessimistic.

$$s_{\mathbf{Min}}(o_i, o_j) = \min_{m \in M} s_m(o_i, o_j)$$

Filter operators select all correspondences of a mapping that fulfill a given filter condition. The following operators are supported:

- **threshold**: This strategy selects all correspondences with a similarity value above a given threshold  $t$ .
- **maxN**: This strategy selects for each source object the  $n$  correspondences with maximal similarity.
- **maxDelta**: This strategy selects for each source object the correspondence with the maximal similarity plus all correspondences with a similarity differing at most by a tolerance value  $d$ , which can be specified either as an absolute or relative value.

### Learning-based combination

In FEVER a learning-based combination is realized by two kind of operators. So-called *model generator* operators utilize a supervised learning algorithm to determine an effective combination of several input matchers. Currently FEVER offers the choice between three different operators named after the underlying learning algorithms: Support Vector Machine (`svm`), Logistic Regression (`logisticRegression`), and Decision Tree (`decisionTree`).

All model generator operators require a training mapping that contains manually labeled correspondences representing examples for matching (similarity value equals 1) and non-matching (0) objects. Furthermore, they expect as a parameter a list of attribute matcher specifications  $\mathcal{M} = \{m\}$  indicating which similarity measure should be evaluated on which entity attributes. The learned combination is returned as a model. In case of SVM and Logistic Regression the model is a numerical combination function, whereas for Decision Tree it is a decision tree. The learned model can then be applied by the *model application* operator to one or two object sets or an input mapping to determine the match correspondences.

FEVER also supports a multiple learning approach by combining the mapping result of several base learners (SVM, Logistic Regression or Decision Tree) using the `vote` operator. This approach thus derives its match decisions from a set of base learning's majority consensus (two entities match if at least half of the base learners vote for the match). The motivation for combined learning is to compensate for individual learning's weaknesses and thus improve overall match quality and robustness. This comes with the highest execution cost because Fever must execute the match strategies the three basic learners have determined before it can combine their results.

## 7.4 Training selection operators

The effectiveness of a learning-based combination critically depends on the size and quality of the available training data. For object matching it is important that the training data is representative for the objects to be matched and exhibit the variety and distribution of errors observed in practice. Furthermore, the training data should allow the observation of differences between the available matchers so that an effective combination of different matchers can be learned. This requires that the training data contain a sufficient number of both matching as well as non-matching object pairs. On the other hand, it is important to limit the manual overhead for labeling. Hence, we wish to keep the number of training object pairs to be labeled manually as low as possible.

Most previous studies on training-based object matching have provided little details

on the selection and size of training data and have not studied the influence of different training sets [18, 37]. In other cases, the authors used a relatively large amount of training data thus favoring good match quality, however at the expense of a high manual overhead for labeling [25, 100, 115].

A naive method for training selection is to randomly choose objects from the sets of objects to be resolved for labeling. However, this way it cannot be guaranteed that we obtain representative training data. This is because for a given object from the first dataset, most objects from the second dataset are most likely non-matches while only few will match. Hence, a random selection of object pairs will result in an imbalanced training set where the non-matching object pairs will heavily outnumber the matching pairs. This phenomenon is known as the class imbalance problem and often reported as an obstacle to the generation of good classifiers by supervised machine learning algorithms [68].

A better training strategy is the so-called static-active selection of object pairs proposed in [19]. This method compares the entities to be resolved with some state-of-the-art string similarity measure and selects only pairs that are fairly similar according to this measure, i.e. their similarity value meets a certain threshold. By asking the user to label those object pairs a training sample with a high proportion of matching pairs can be obtained. At the same time, non-matching object pairs selected using this method are likely to be "near-miss" negative examples that are more informative for training than randomly selected pairs most of which tend to be "easy" non-matches. The idea is easy to implement and seems a reasonable way to obtain representative training data.

In this thesis we propose different generic methods for automatically selecting training data to be labeled. These methods are provided in FEVER by TrainSelect operators. They select correspondences from one or two object sets or an input mapping and prompt users to label them interactively as match or non-match. By doing so the correspondences are annotated with similarity 0 (non-match) or 1 (match) and, thus, a training mapping is compiled. Labeled correspondences are additionally stored in a repository to avoid repeated labeling of the same correspondence. The interactive labeling can be renounced if there is a perfect mapping available. In this case the selected correspondences are aligned against the perfect mapping to automatically annotate it with similarity 0 or 1. The chosen TrainSelect operator determines the resulting training mapping of a specified size. They support a fair comparison of learning-based matchers by ensuring that learners are provided with training data of the same size and quality.

Currently, FEVER offers two operators for training selection: *Random* and *Ratio*. Both approaches are discussed in the following.

### 7.4.1 Random training selection

This strategy selects  $n$  object pairs randomly among the ones satisfying a given minimal threshold  $t$  applying a similarity measure  $m$ . This approach is very simple but may be prone to the class imbalance problem, e.g. for lower (higher) thresholds the number of non-matching (matching) pairs may dominate the training.

Algorithm 1 shows the pseudo-code for the proposed strategy for a single set of objects  $O_A$ . Similarly, the invocation for two sets of objects  $O_A$  and  $O_B$  or a mapping  $M$  can be carried out. The pseudo-code refers to the following functions:

- **random\_select**: calculates a mapping based on the specified similarity measure  $m$  and the threshold  $t$  and then randomly chooses  $n$  object pairs from this mapping.
- **label\_pairs**: labels the selected object pairs. If a perfect mapping is available the labeling is obtained by alignment against this mapping. Otherwise the chosen object pairs are presented the user for labeling.

### 7.4.2 Ratio training selection

The ratio method is an extension of random that aims at a certain ratio of matching and non-matching object pairs in the training data. It uses a ratio parameter from the range 0 to 0.5, indicating the minimal percentage of both matching and non-matching pairs. The ratio 0 corresponds to the random strategy enforcing no restrictions on the share of matching or non-matching pairs. For ratio values greater than 0, the number of randomly selected object pairs is reduced so that either the number of matching or non-matching object pairs satisfies the ratio restriction. For example, a ratio of 0.4 guarantees that at least 40% of all training pairs are either matching or non-matching — in other words, at most 60% are non-matching or matching. By ensuring a minimum number of pairs, the ratio approach aims to enhance the training data’s discriminative value for learning effective match strategies.

Algorithm 2 shows the pseudo-code for the proposed strategy. Like for the random approach the algorithm demonstrates the implementation for a single set of objects  $O_A$ . Similarly, the invocation for two sets of objects  $O_A$  and  $O_B$  or a mapping  $M$  can be realized. The functions **random\_select** for the initial random selection of object pairs and **random\_select** for determining the labeling of the selected object pairs correspond to the same functions for the random selection approach as shown in Algorithm 1. The function **reduce\_pairs** reduces the selected and labeled object pairs until the specified ratio of matching and non-matching pairs is achieved.

**Algorithm 1:** Random training selection

---

```
1  $O_A$ : input object set;
2  $n$ : number of training pairs;
3  $t$ : minimal threshold;
4  $s$ : similarity measure;
5  $M_T$ : training mapping;
6  $M_P$ : perfect mapping (optional);
7  $\text{RAND}(i)$  : returns a uniformly distributed integer number between 0 and  $i - 1$ ;

8  $\text{random\_select}(O_A, n, t, s, M_P)$ 
9    $M_T \leftarrow \text{select\_pairs}(O_A, n, t, s)$ ;
10   $M_T \leftarrow \text{label\_pairs}(M_T, M_P)$ ;
11  return  $M_T$ 

12  $\text{select\_pairs}(O_A, n, t, s)$ 
13   $M \leftarrow \{(o_i, o_j, s_{i,j}) \mid s_{i,j} = s(o_i, o_j) \geq t; o_i, o_j \in O_A\}$ ;
14   $M_T \leftarrow \emptyset$ ;
15  while  $|M_T| < n \wedge |M| > 0$  do
16     $a \leftarrow \text{RAND}(|M|)$ ;
17     $(o_i, o_j, s_{i,j}) \leftarrow M[a]$ ;
18     $M \leftarrow M \setminus \{(o_i, o_j, s_{i,j})\}$ ;
19     $M_T \leftarrow M_T \cup \{(o_i, o_j, s_{i,j})\}$ ;
20  end
21  return  $M_T$ 

22  $\text{label\_pairs}(M_T, M_P)$ 
23  if  $M_P = \emptyset$  then
24    present  $M_T$  to user for labeling;
25  else
26     $M_T^* \leftarrow \emptyset$ ;
27    foreach  $(o_i, o_j, s_{i,j}) \in M_T$  do
28      if  $(o_i, o_j, 1) \in M_P$  then
29         $M_T^* \leftarrow M_T^* \cup \{(o_i, o_j, 1)\}$ ;
30      else
31         $M_T^* \leftarrow M_T^* \cup \{(o_i, o_j, 0)\}$ ;
32      end
33    end
34  end
35   $M_T \leftarrow M_T^*$ ;
36  return  $M_T$ 
```

---

---

**Algorithm 2:** Ratio training selection
 

---

```

1   $O_A$ : input object set;
2   $n$ : number of training pairs;
3   $t$ : minimal threshold;
4   $r$ : ratio;
5   $s$ : similarity measure;
6   $M_T$ : training mapping;
7   $M_P$ : perfect mapping (optional);

8  ratio_select( $O_A, n, t, r, s, M_P$ )
9     $M_T \leftarrow$  select_pairs( $O_A, n, t, s$ );
10    $M_T \leftarrow$  label_pairs( $M_T, M_P$ );
11    $M_T \leftarrow$  reduce_pairs( $M_T, n, r$ );
12   return  $M_T$ 

13 reduce_pairs( $M_T, n, r$ )
14    $TM \leftarrow \emptyset$ ;
15    $TN \leftarrow \emptyset$ ;
16   foreach  $(o_i, o_j, s_{i,j}) \in M_T$  do
17     if  $s_{i,j} = 1$  then
18        $TM \leftarrow TM \cup \{(o_i, o_j, 1)\}$ ;
19     else
20        $TN \leftarrow TN \cup \{(o_i, o_j, 0)\}$ ;
21     end
22   end
23    $sM \leftarrow \lfloor n \cdot r \rfloor$ ; /* required number of matching pairs */
24    $sN \leftarrow n - sM$ ; /* required number of non-matching pairs */
25   if  $|TM| < sM$  then
26     /* actual number of matching pairs is less than required */
27      $rn \leftarrow |TN| - \lfloor \frac{|TM|}{r} - |TM| \rfloor$ 
28     ; /* number of non-matching pairs to be removed */
29     for  $i \leftarrow 1$  to  $rn$  ; /* remove non-matching pairs */
30     do
31        $a \leftarrow$  RAND( $|TN|$ ); /* choose random index */
32        $(o_i, o_j, 0) \leftarrow TN[a]$ ; /* get non-matching pair at random index */
33        $TN \leftarrow TN \setminus \{(o_i, o_j, 0)\}$ ; /* remove selected non-matching pair */
34     end
35   else
36     if  $|TN| < sN$  then
37       /* actual number of non-matching pairs is less than required */
38        $rm \leftarrow \lfloor \frac{|TN|}{1-r} - |TN| \rfloor$ ; /* number of matching pairs to be removed */
39       for  $i \leftarrow 1$  to  $rm$  ; /* remove matching pairs */
40       do
41          $a \leftarrow$  RAND( $|TM|$ ); /* choose random index */
42          $(o_i, o_j, 1) \leftarrow TM[a]$ ; /* get matching pair at random index */
43          $TM \leftarrow TM \setminus \{(o_i, o_j, 1)\}$ ; /* remove selected matching pair */
44       end
45     end
46   end
47    $M_T \leftarrow TM \cup TN$ ;
48   return  $M_T$ 

```

---

## 7.5 Match workflows

In the following we show how non-learning as well as learning-based workflows can be realized with the operator tree concept of FEVER. Figure 7.2 illustrates example workflows for realizing a non-learning as well as a learning-based combination. For both subfigures the operator tree is shown on the left, while relevant operator parameters are shown on the right. The tree is executed in a post-order traversal sequence and the results of the child operators are input to the father operator.

### 7.5.1 Non-learning workflow

Figure 7.2a shows how a non-learning match workflow is realized as an operator tree within FEVER. The workflow consists of five operators: a blocking operator, two match operators, a combination operator, and a filter operator. Input are two sets of objects from two sources. The blocking operator produces as output a mapping that is the input to both match operators. The resulting two mappings are combined by the `union` operator. The filter operator `threshold` filters the correspondences to restrict the mapping result from the `union` operator to the most similar instances.

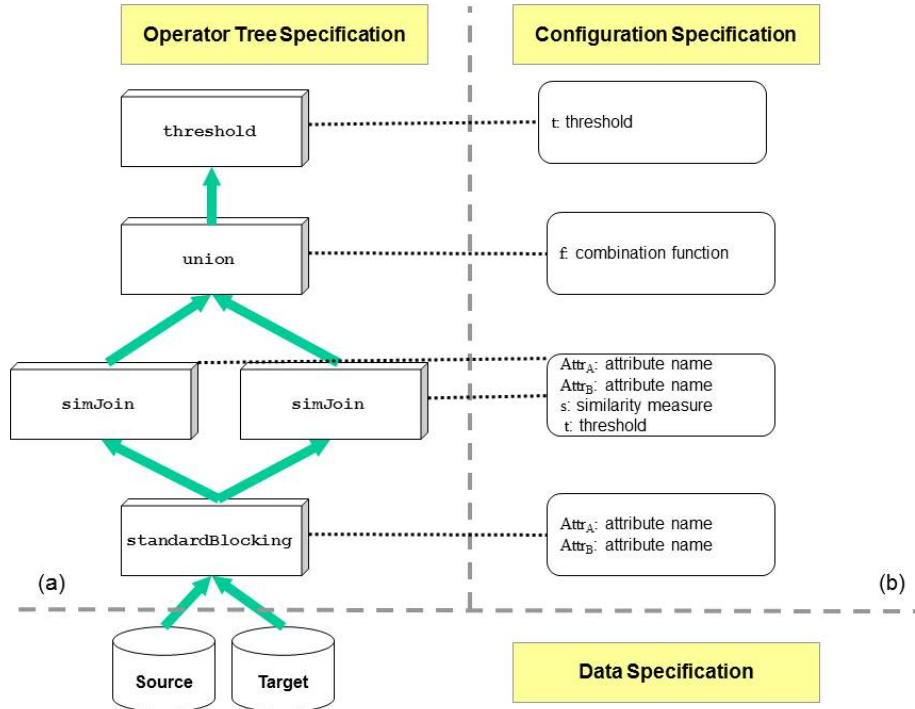
### 7.5.2 Learning-based workflow

Figure 7.2b shows how a learning-based match workflow is realized as an operator tree within FEVER.

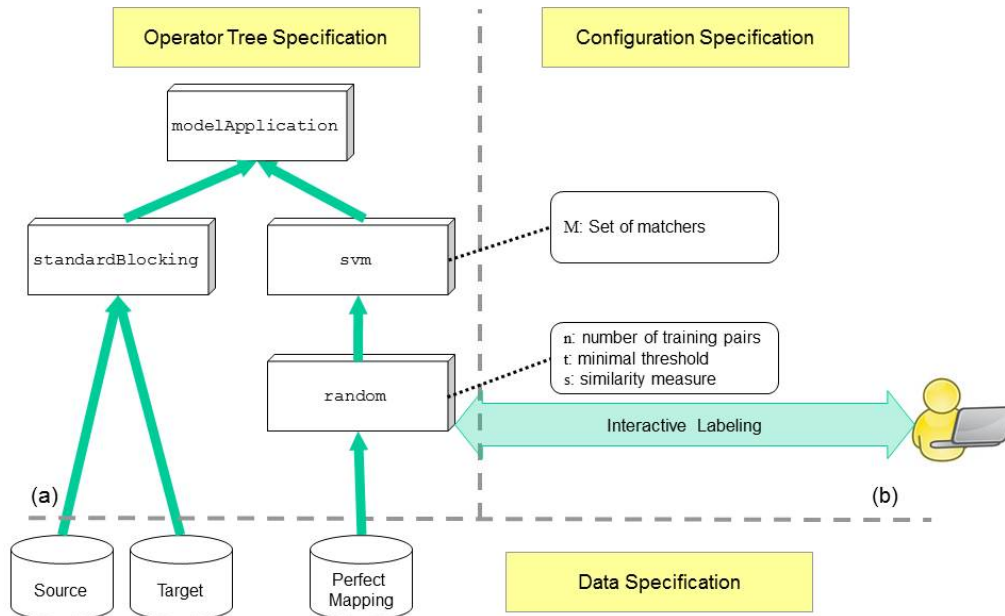
The execution falls into two phases: model generation and model application. The model generation (right part of the operator tree) requires a training mapping that contains manually labeled correspondences representing matching (similarity value equals 1) and non-matching (0) object pairs. The training mapping is obtained from the training selection operator (in this case `random`). The training selection operator (in this case `random`) semi-automatically chooses object pairs as training examples. The corresponding label (match or non-match) is either obtained by automatic alignment against a perfect mapping if such a mapping is available or through interactive labeling by a user. The learning algorithm (in this case `svm`) applies the specified matchers to the object pairs in the training mapping. The learner then uses the resulting similarity values to automatically determine a match strategy model, i.e. combination of the specified matchers to derive a match decision for any object pair.

The second phase (root of the operator tree) applies the determined model for the real match task (model application) to match a source and target dataset (or to find duplicates within one dataset). A blocking operator is executed first to reduce the search space to the most likely matching object pairs.





(a) Fever match workflow realizing a non-learning combination



(b) Fever match workflow realizing a learning-based combination

Figure 7.2: Fever match workflows. We can see (a) the operator tree and (b) its relevant configuration specification. 89

## 7.6 Configuration strategies for comparative object matching evaluation

An operator tree typically comprises several operators each having several parameters that need to be specified in order to apply the operator tree to a match problem. Typical operator parameters have been discussed above. Permissible parameter values can mostly be defined by a set of possible values, or by a range of real or integer numbers. For example, the match operator `simJoin` (see Figure 7.3) has a set parameter to specify the similarity measure  $s$  to be applied and a range parameter for the similarity threshold  $t$ . We further distinguish between bounded and free parameters. A bounded parameter is already assigned a value through the user in the operator tree definition. Parameters that are not bounded are called free parameters. They are treated as parameters of the operator tree and, thus, have to be assigned a value dynamically according to a configuration strategy. For the example in Figure 7.3, the names of the attributes to be compared are bound parameters (manually provided) while the similarity function and threshold are free parameters.

An assignment of all free parameters of an operator tree with a valid value is called a parameter setting. For operator trees without learning-based match operators the parameter setting is sufficient for the configuration. Training-based operator trees additionally require a training mapping for the configuration. To allow for comparative evaluations, we take an effort-based approach. We evaluate the quality of an operator tree against the effort spent to determine the match configuration. We consider both, the parameterization and the labeling effort. The parameterization effort can be represented as the number of parameter settings that have been evaluated to identify the best setting, i.e., the setting for which the operator tree result has the best quality (e.g., F-measure). For training-based workflows the labeling effort regards the number of correspondences that have to be labeled by the user. We can thus ensure a similar labeling effort for a comparative comparison of different training-based approaches. Parameterization and labeling effort are considered independently and are not set off into a single effort measure. This facilitates an

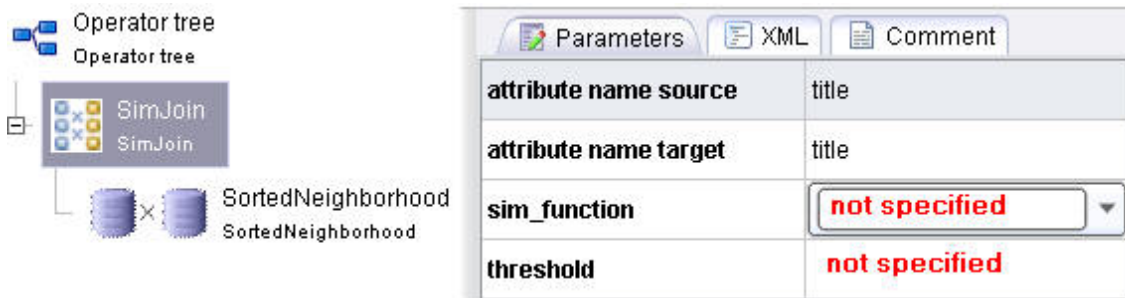


Figure 7.3: Graphical representation of an operator tree (left) incl. parameters for selected similarity join operator (right)

analysis of the reached match quality measured in terms of a quality measure (e.g. precision, recall, F-measure) against the effort spent for parameterization as well as for labeling.

The evaluation of an operator tree is subject to a specified maximal labeling effort  $M$  and maximal parameterization effort  $N$ . Hence, for training-based operator trees we restrict the user to label  $M$  training pairs. Additionally, we generate  $N$  different parameter settings according to a specified configuration strategy and determine from all obtained evaluation results the best ones. A configuration strategy takes as input the free parameters of an operator tree and the maximal parameterization effort  $N$ . To generate  $N$  different parameter settings FEVER currently supports the following configuration strategies:

- **UserDefined:** In a user defined strategy the parameter settings are completely specified by the user. This manual strategy can be applied if a defined set of parameter values should be evaluated, e.g., a threshold parameter value varies from 0 to 1 in 0.05 steps.
- **Random:** The random strategy is a straightforward way that assigns parameter values out of their possible values randomly, i.e.,  $N$  parameter settings are selected by random assignment of parameter values.
- **Grid:** The grid strategy realizes a simple grid search, dividing the multidimensional parameter space into a uniform grid. The coarseness of the grid is controlled by the number of parameter settings  $N$  and, thus determines the search efficiency and the quality of the solution.
- **GradientDescent:** The gradient descent strategy is more goal-oriented and iteratively refines a parameter setting by considering the quality of previously generated settings.

Note that all (except user-defined) strategies are independent from the match approach modeled by the operator tree. Thus the configuration strategies can be applied to different match workflows.

## 7.7 Implementation and use

FEVER has been implemented entirely in Java. A user-friendly interface is essential for the practicability and effectiveness of a match framework. The graphical user interface of FEVER, implemented in Java Swing, provides the user with many ways to interactively influence the match process. The user can configure the matchers to be employed. He can then iteratively refine the proposed correspondences by adding further operators. Furthermore, the user can manipulate the obtained match result.

Figures 7.4 illustrate how to set up and configure a workflow within FEVER.

Figure 7.4a shows how to define required input data. Input data comprises the data sources of the objects to be matched. FEVER allows to match objects from a single data source as well as two sources. Additionally to the input object sources, input mappings can be configured. When available a perfect mapping can be specified to automatically evaluate the match quality. For training-based match workflows a precomputed training mapping can be loaded if already available. FEVER can import input data from various database systems and different file formats. The figure illustrates importing objects from a MySQL database. This requires to specify the connection parameters (e.g., connection url)

Figure 7.4b illustrates the specification of the actual match workflow as an operator tree. The various operators supported by FEVER can be selected from the context menu and added to the tree. The user can configure the operator specific parameters.

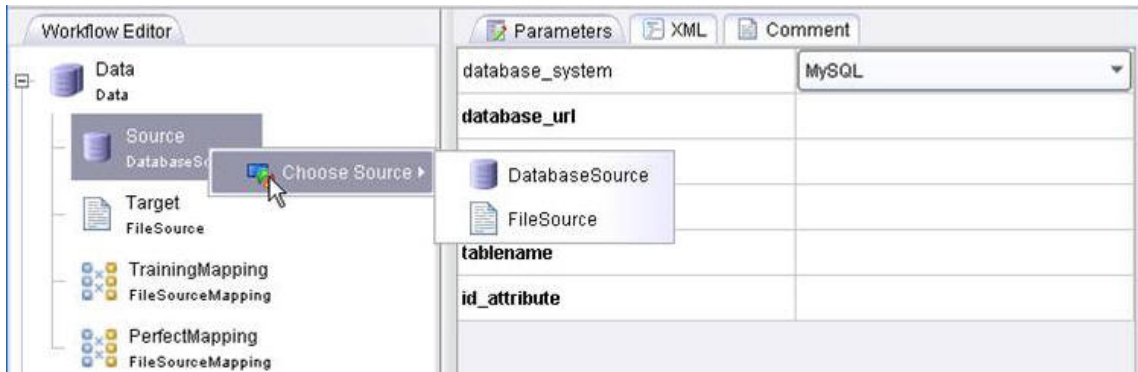
Figure 7.4c demonstrates how to configure the execution of an operator tree. FEVER offers several configuration strategies to choose from.

Figures 7.5 illustrates the features for result inspection and manipulation offered by FEVER.

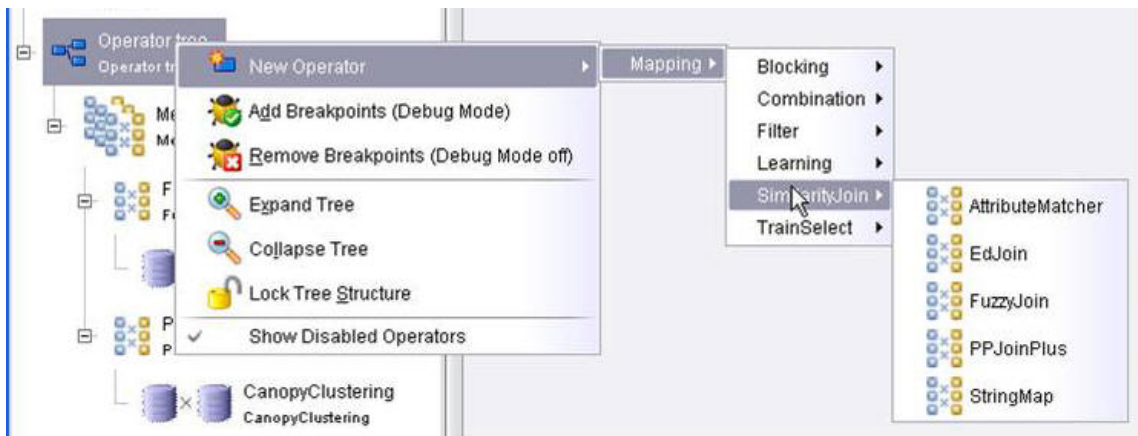
Figure 7.5a shows the result of a match workflow with the identified correspondences. The similarity values of the correspondences are highlighted in colour to indicate the match probability. Green is used for a high similarity, while red is used for a low similarity. Thus correspondences highlighted in green are very likely true matches, while red correspondences might be false matches. The user can rework the result by removing false correspondences or adding missing ones.

Figure 7.5b shows the evaluation details for several executions of a match workflow with different parameter configurations. For each execution the table shows the match quality in terms of precision, recall and F-measure. Furthermore, the column duration indicates the required execution time. The corresponding operator trees can be inspected by clicking on the magnifiers in the column configuration.

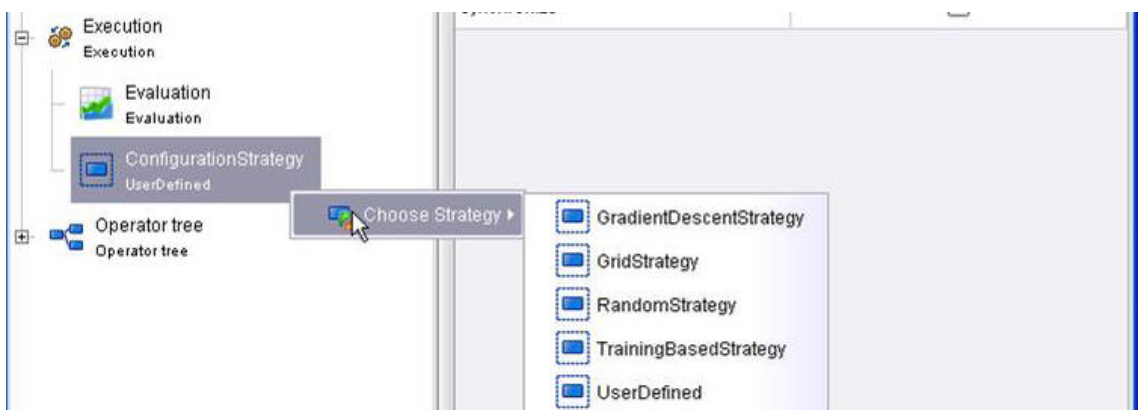
Figure 7.5c shows the graphical analysis feature of FEVER. It shows a graph comparing several training-based match strategies using different learning algorithms. The x-Axis indicates the required effort in terms of the number of training examples, while the y-Axis plots the F-measure values.



(a) Input data specification



(b) Operator tree specification



(c) Configuration specification

Figure 7.4: Workflow specification in FEVER

## CHAPTER 7. FEVER - A FRAMEWORK FOR OBJECT MATCHING

599 correspondences

Select source attributes:      Select target attributes:

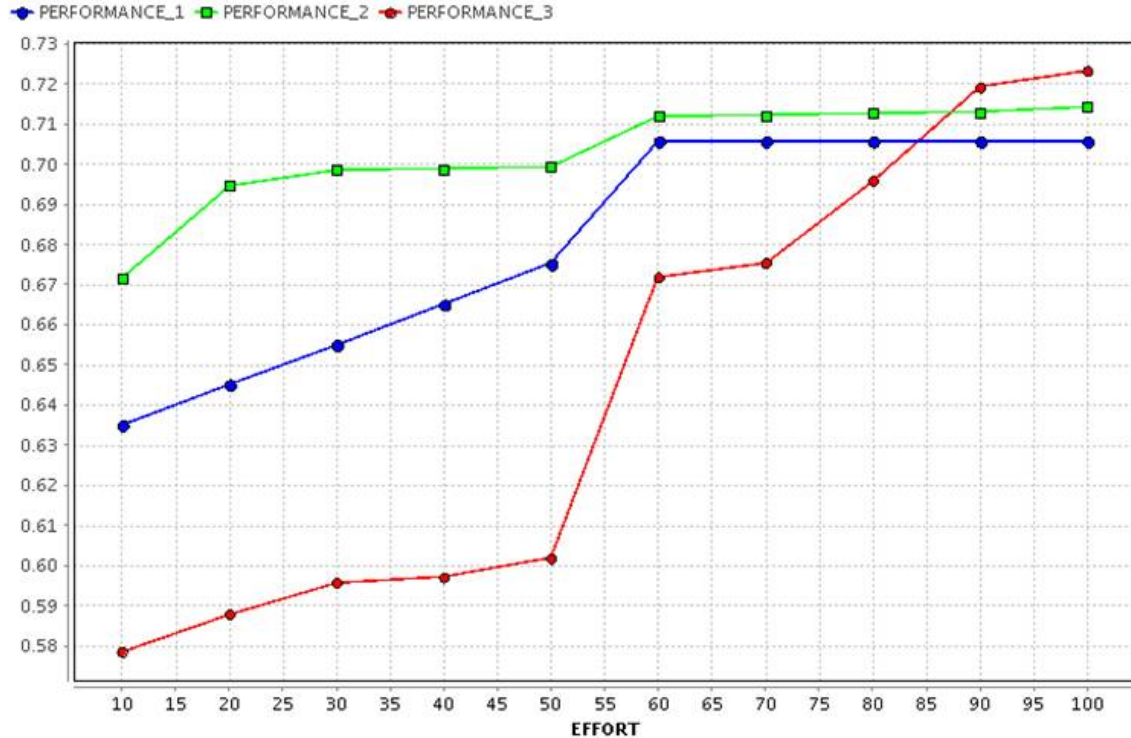
id                       id  
 name                     name  
 price                     listprice  
 availability             ourprice  
 description            shipping  
 upc                       instock  
 Approximate Weight    Manufacturer

name	$\epsilon(M)$	O(M)	
Canon Cyan Photo Ink Cartridge - Cyan - CLI8PC	0.857	1	Canon CLI-8PC Photo Cyan Ink Cartridge - 0624B002
Canon Magenta Photo Ink Cartridge - Magenta - CLI8PM	0.857	1	Canon CLI-8PM Photo Magenta Ink Cartridge - 0625B002
Sony 2GB Memory Stck Micro (M2) - MSA2GU2	0.857	1	Sony 2GB Memory Stick Micro (M2) - MSA2GD
Sirius STILETTO 2 Portable Satellite Radio - SL2PK1	0.857	1	SIRIUS SL2PK1 SIRIUS Stiletto 2 Portable Radio
Sony 1GB Memory Stck PRO Duo Mark 2 Media Card - MSMT1G	0.853	1	Sony 1GB Memory Stick PRO Duo Card - MSMT1G
Sony 2GB Memory Stck PRO Duo Mark 2 Media Card - MSMT2G	0.853	1	Sony 2GB Memory Stick PRO Duo Card - MSMT2G
Sony 4GB Memory Stck PRO Duo Mark 2 Media Card - MSMT4G	0.853	1	Sony 4GB Memory Stick PRO Duo Card - MSMT4G
Sony 8GB Memory Stck PRO Duo Mark 2 Media Card - MSMT8G	0.853	1	Sony 8GB Memory Stick PRO Duo Card - MSMT8G
Sony Digital SLR Camera With Lens Kit - DSLRA200W	0.853	1	Sony alpha DSLR-A200 Digital SLR Camera with Dual Lens Kit -
Sony Xplod 10-Disc Add-On CD/MP3 Changer - CDX565MXRF	0.845	1	Sony CDX-565MXRF 10-Disc CD/MP3 Changer

(a) Match result inspection

ID	EVALUATIONID	CYCLE	STARTTIME	CONFIGURATION	PRECISION	RECALL	FMEASURE	DURATION	EFFORT
12905	199	1	2014-01-13 02:09:20.008820000		0.957	0.387	0.551	00:00:01	100
12906	199	2	2014-01-13 02:09:22.008120000		0.837	0.630	0.719	00:00:01	100
12907	199	3	2014-01-13 02:09:24.002490000		0.887	0.550	0.679	00:00:01	100
12908	199	4	2014-01-13 02:09:25.005340000		0.541	0.794	0.643	00:00:01	100
12909	199	5	2014-01-13 02:09:26.008400000		0.541	0.794	0.643	00:00:01	100
12910	199	6	2014-01-13 02:09:28.000800000		0.929	0.476	0.629	00:00:01	100
12911	199	7	2014-01-13 02:09:29.002700000		0.874	0.577	0.695	00:00:01	100
12912	199	8	2014-01-13 02:09:30.003980000		0.841	0.556	0.670	00:00:01	100
12913	199	9	2014-01-13 02:09:31.005230000		0.541	0.794	0.643	00:00:01	100
12914	199	10	2014-01-13 02:09:32.006460000		0.884	0.568	0.691	00:00:01	100
12915	199	1	2014-01-13 02:09:33.007360000		0.837	0.605	0.703	00:00:01	250

(b) Inspection of evaluation details



(c) Configuration specification

Figure 7.5: Result inspection in FEVER

## 7.8 Functional comparison with competitive frameworks

In the following we compare the functionality of FEVER to the competitive frameworks analyzed in Section 6.4 regarding the five criteria: Entity type, blocking methods, matchers, combination of matchers, and training selection. As FEVER supports non-learning as well as learning-based approaches, FEVER ranks among the hybrid frameworks.

- Entity type: FEVER supports only relational objects.
- Blocking methods: FEVER provides explicit support for blocking based on disjoint as well as on overlapping object partitioning. It supports all four most popular blocking approaches, namely Standard Blocking, Sorted Neighborhood, Q-gram Indexing, and Canopy Clustering. Besides FEBRL, FEVER offers the most comprehensive selection of blocking approaches.
- Matchers: FEVER supports seven string similarity measures and a numeric measure for comparing numerical attribute values. This is more than most of the competitive frameworks offer. However, FEBRL provides a even greater selection of similarity measures. FEVER currently focuses on attribute value matchers.
- Combination of matchers: FEVER provides the most comprehensive support for combining multiple matchers. It supports numerical, rule-based, workflow-based and training-based approaches.
- Training selection: FEVER provides explicit support for (semi-)automatic training selection.





# 8

## Comparative evaluation of object matching approaches with FEVER

In this chapter we use FEVER to evaluate several non-learning and learning-based workflows for the real-world match problems summarized in Table 8.1.

We introduce the used evaluation match tasks in Section 8.1. To illustrate the difficulty of finding an effective object matching strategy and thus the need for learning-based approaches we extensively evaluated non-learning match workflows and show some selected results for illustration in Section 8.2. The study of non-matching workflows also includes the evaluation of a commercial object matching system representing the current state of the art. We use FEVER to specify and tune match strategies for this system. The results will also be used to determine a baseline configuration for comparison with the learning-based workflows.

The evaluation of learning-based workflows in Section 8.3 investigates the effectiveness and training effort of learning-based methods to semi-automatically determine suitable match strategies. In particular, we study different approaches to select training data and study how much training is needed to find effective combined match strategies and their configuration.

In Section 8.4 we present a comparative evaluation of existing match approaches and frameworks.

## CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

---

Match task			Source size (#entities)		Mapping size (#correspondences)		
Domain	Attributes	Sources	Source 1	Source 2	Full mapping (Cartesian product)	Reduced mapping (blocking result)	perfect result
Bibliographic	-title	DBLP- ACM	2,616	2,294	6 million	494,000	2,224
	-authors						
	-venue	DBLP- Scholar	2,616	64,263	168.1 million	607,000	5,347
E-commerce	-year						
	-name	Amazon-	1,363	3,226	4.4 million	342,761	1,300
	-description	GoogleProducts					
	-manufacturer	Abt-	1,081	1,092	1.2 million	164,072	1,097
	-price	Buy					

Table 8.1: Overview of real-world evaluation match tasks

### 8.1 Evaluation match tasks

The evaluation match tasks comprises four match tasks of two semantic types (bibliographic and ecommerce data objects). Table 8.1 provides some statistics on these tasks which are named after the involved web sources. For each of the seven data sources we consider up to four attributes for matching. The number of objects per source ranges from about 1,100 to more than 64,000; the size of the Cartesian product for the four tasks ranges from about 1.2 million (Abt-Buy task) to 168 million (DBLP-Scholar) object pairs. We use a fixed blocking strategy for all experiments and evaluated systems to guarantee equal effectiveness and efficiency of the blocking step. Thus blocking is not subject to our evaluation. The blocking strategy employs Trigram on a low string similarity threshold to reduce the search space to the numbers shown in Table 8.1 (up to 607,000 pairs). To investigate the scalability of the match approaches we evaluate the match runtimes not only on the blocking output but also on the full Cartesian product.

To determine the match quality we further created the perfect match results with the cardinalities as shown in Table 8.1. Selected attributes of the seven data sources are also listed.

The match tasks were chosen to represent a spectrum of different data characteristics and difficulty levels. The first task is expected to be of low difficulty as it deals with publication references from two well-structured bibliographic data sources (DBLP, ACM digital library) that are at least partially under manual curation.

#### 8.1.1 Bibliographic match tasks

The bibliographic tasks match publication sets of three computer science bibliographies: the DBLP bibliography, ACM Digital Library (ACM) and Google Scholar

## CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

---

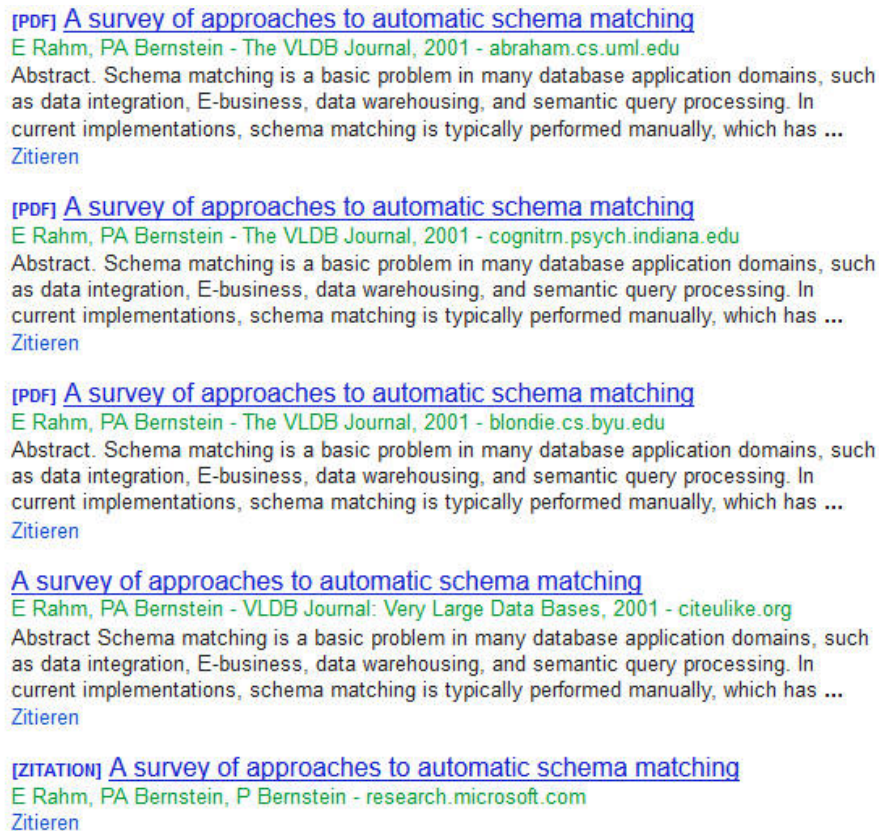


Figure 8.1: Duplicate paper entities in Google Scholar

(Scholar). Each of them provides information about publications, authors and venues but they are very different in terms of data quality, coverage and accessibility.

Scholar maintains a huge collection of publication entries automatically extracted from documents.

Figure 8.1 illustrates some of the problems on five duplicate entries for the same paper.

The bibliographic objects have automatically been extracted from web pages or PDF documents and contain numerous quality problems such as misspelled author names, different ordering of authors, heterogeneous venue denominations etc.. Google Scholar performs already an object matching by clustering references to the same publication to aggregate their citations and fulltext sources. However as the duplicates in the example show, the obtained results are far from perfect influenced by the mentioned quality and heterogeneity problems. This illustrates that there is a big potential for improving data quality by better object matching techniques. This would be critical for tasks requiring the examination of all duplicates, e.g., to collect all citations of publications for a citation analysis.

DBLP and ACM focus on computer science publications and are manually maintained. Compared to Scholar they are of higher quality, especially DBLP. Since DBLP has almost no duplicates the mapping result between Scholar and DBLP can also be used for determining the duplicates in Scholar. This is because all Scholar entries matching the same DBLP publication can be considered duplicates.

As shown in Table 8.1, our evaluation datasets cover 2,616 publications from DBLP, 2,294 publications from ACM and 64,263 publications from Google Scholar. We thus have up to 169 million object pairs (DBLP-Scholar) in the Cartesian product between these datasets.

### 8.1.2 E-commerce match tasks

The e-commerce tasks deal with sets of related product offers from the online retailers Abt.com, Buy.com (Abt-Buy task), Amazon.com and the product search service of Google accessible through the Google Base Data API (Amazon-GoogleProducts task). In order to obtain the perfect match result we included only product entities with a valid UPC (Universal Product Code) in our datasets which allows a unique identification of a product. Of course, the match strategies to be evaluated could not make use of these UPCs but only of the attributes listed in Table 8.1 (especially product name and description). This is because in reality many websites do not provide the UPC information so that object matching cannot rely on these in general.

The Abt, Buy, and Amazon datasets were created by selecting products from predefined categories. Based on the Amazon products, the GoogleProducts dataset were generated by sending queries on the product name.

## 8.2 Evaluation of non-learning workflows

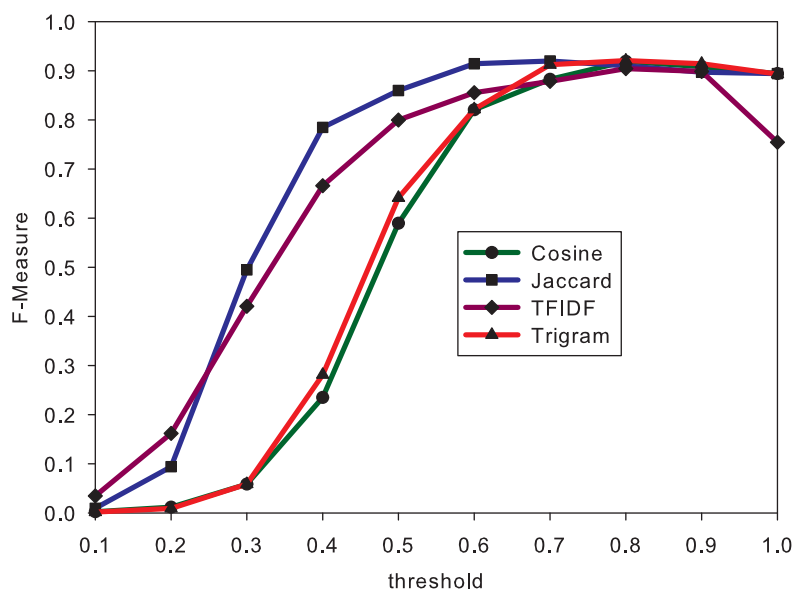
For non-learning workflows the number of possible configurations grows with the number of attributes and the number of matchers. For a given match problem with  $n$  attributes and  $m$  provided matchers we can choose from  $n * m$  single-attribute matchers which need to be provided with a similarity threshold to determine whether or not two entities match or not. Of course, the number of possible configurations including threshold choices explodes when considering the combination of two or more matchers.

We perform two experiments for the bibliographic match tasks. In a first experiment we analyze the effectiveness of different similarity measures for matching on one selected attribute (title). Figure 8.2 shows the F-measure results for different similarity measure and threshold configurations we systematically evaluated with the user defined parameterization strategy. As expected there are huge differences

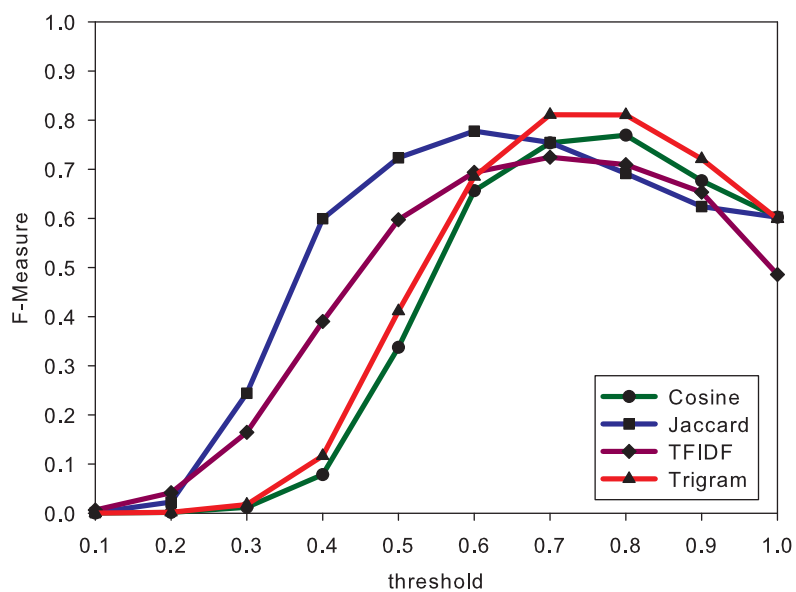
between the configurations, indicating the difficulty of choosing manually the right similarity function and threshold. For example, Trigram achieved for both match tasks the best F-measure overall (92% for DBLP-ACM and 81% for DBLP-Scholar) for threshold 0.8 but appears highly dependent on the choice of the similarity threshold. For a lower similarity threshold (0.5) Trigram was clearly outperformed by the Jaccard and TF/IDF measures which appear less sensitive w.r.t. the choice of similarity threshold.

In the second experiment we compare different match configurations for the two problems DBLP-Scholar and DBLP-ACM. For each problem, we consider 18 configurations using the trigram similarity measure for two threshold values (0.5 and 0.8) either on one attribute (title or authors), two attributes (3 combinations), three attributes (3 combinations) or all four attributes. In case of multiple attributes we require for a matching object pair that the similarity threshold is exceeded for each attribute.

## CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER



(a) DBLP-ACM



(b) DBLP-Scholar

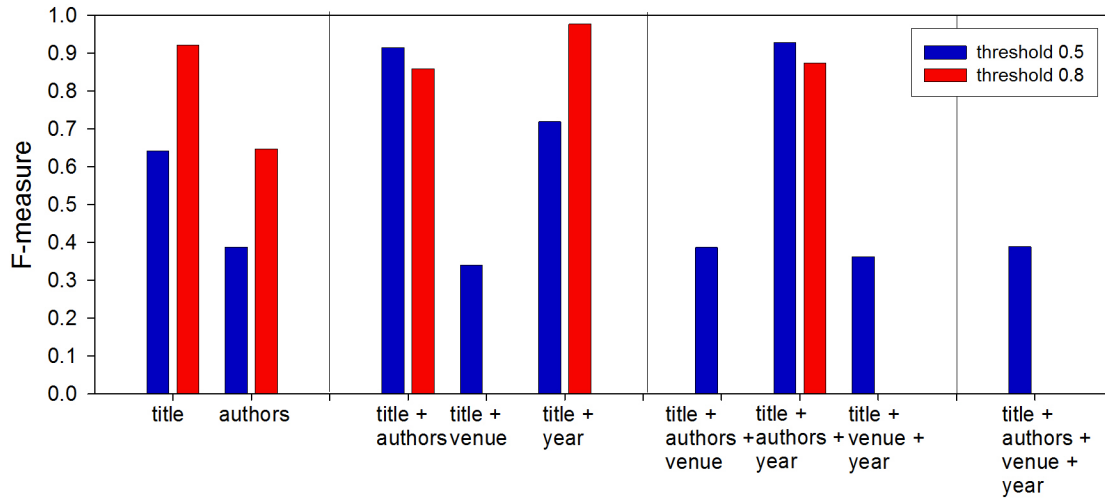
Figure 8.2: Match accuracy of manually configured single-attribute matcher for DBLP-ACM and DBLP-Scholar

Figure 8.3 shows the resulting F-measure results for the two match tasks. We observe that there are not only big differences between the configurations for each of the problems but also between the problems. While some configurations using multiple attributes outperform the single-attribute results it turns out to be quite challenging to find the right attributes to consider jointly and to determine useful similarity thresholds. For the DBLP-Scholar problem the quality limitations of Scholar on years and venues render only two attributes useful: title and authors. But even for the DBLP-ACM task we observe a poor performance for all configurations using the venue attribute. A closer inspection of the data revealed large differences in the venue names between ACM and DBLP (use of different abbreviations, etc.) indicating that this attribute would need a large normalization overhead to become useful for matching these sources. While for both object matching tasks the best single attribute configuration is title similarity with threshold 0.8 the two-attribute configurations differ already. For DBLP-Scholar the combined consideration of title and authors with a lower threshold 0.5 performed best. For DBLP-ACM this configuration also performed well but was outperformed by the consideration of title and year. The match workflow using trigram similarity on both title and authors with a threshold of 0.5 performed reasonably well on both matching tasks (F-measure 91.4% for the DBLP-ACM task and 82.3% for DBLP-Scholar).

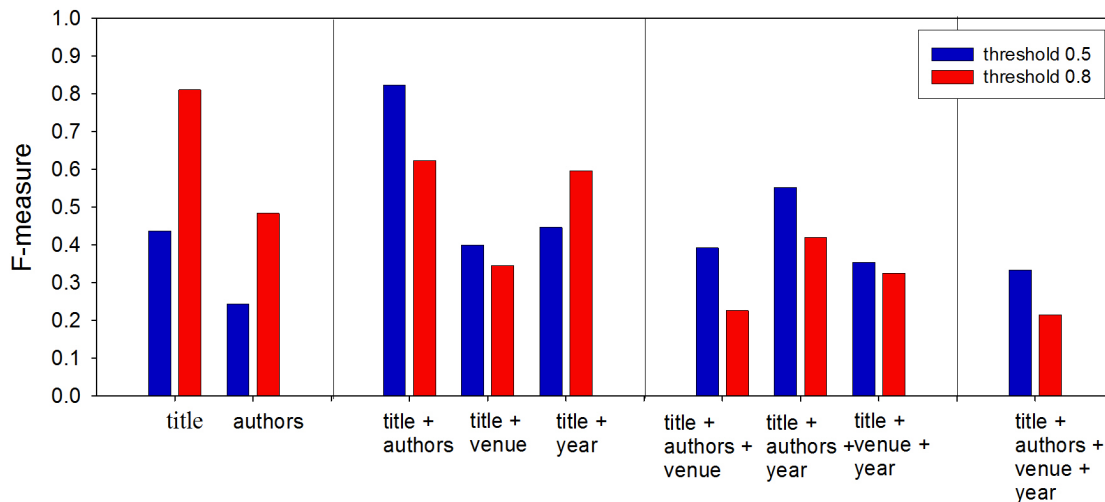
To better assess the quality of the learning-based match strategies we applied a state-of-the-art object match system (referred to as COSY) to our match tasks. Due to license restrictions we cannot provide the name of the evaluated system. The approach has several parameters that need to be configured. The most important parameter is the overall *MinimumSimilarity* threshold. An object pair will be considered a match only if it has a similarity that is greater than or equal to this threshold. Additional *attribute-level similarity thresholds* can optionally be specified for each attribute pair that should be considered in the computation of the object similarity. Hence, the number of parameters grows with the number of attributes.

Figure 8.4 shows the precision, recall and F-measure results for the four match tasks using either one or two attributes using the standard configurations (0.5 for overall *MinimumSimilarity*, 0.0 for attribute *MinimumSimilarity*). For these tests we used the first or first two attributes listed in Table 8.1 (publication title and authors for the bibliographic tasks, product name and description for the ecommerce tasks). Figure 8.4 reveals significant differences for the four match tasks. While the first bibliographic match task could effectively be solved (F-measure > 92%) the results for the three other tasks are much worse especially for the ecommerce tasks. Furthermore, the default parameters result in a reduced match quality for two attributes compared to only one attribute for all four tasks indicating a strong need for manually finding better parameter settings.

However, finding suitable parameter settings is very challenging even for domain experts due the large number of possible parameter combinations. To find a better baseline results than using the default parameters we used FEVER on smaller sub-



(a) DBLP-ACM



(b) DBLP-Scholar

Figure 8.3: Comparison of non-learning match workflows for DBLP-ACM and DBLP-Scholar using Trigram

sets of the match tasks (500 randomly selected object pairs with a minimal string similarity, analogous to the Random training selection approach (as described in 7.4.1) to find the best settings for the three similarity thresholds when using two attributes for matching. For each of the three MinimumSimilarity thresholds we considered 11 values (0 to 1 in 0.1 steps) resulting in a total of 1,331 configurations that we evaluated for each of the four match tasks. For each task, we choose the configuration with the highest F-measure as the baseline strategy. The corresponding results for the whole datasets are indicated in Figure 8.4 in the third bar (“tuned” for 2 attributes) for each match task. We observe that the tuned



threshold	Cosine	Jaccard	TFIDF
0.1	1504.2	835.2	171.1
0.2	365.3	39	199.1
0.3	85.8	7.2	154.6
0.4	18.2	3.6	149.3
0.5	20.0	3.7	153.3
0.6	34.0	4.8	208.3
0.7	26.0	3.0	164.9
0.8	15.2	2.6	152.8
0.9	19.7	3.7	156.0
1.0	23.1	3.3	148.8

Table 8.2: Execution times for non-learning matchers for DBLP-Scholar

strategies always outperform the default configuration for two attributes and for the three more challenging tasks also the default strategy on one attribute. The high tuning effort spent indicates that the reported results are rather optimistic for manually determined match strategies with state-of-the art implementations. The fact that the absolute match effectiveness remains comparatively low especially for the ecommerce tasks underlines that these are really challenging problems to deal with.

### 8.2.1 Efficiency evaluation

The efficiency evaluation focuses on the DBLP-Scholar match task for the Cartesian product. Table 8.2 shows the execution times for matchers based on four different similarity measures, namely Cosine, Jaccard, and TFIDF, subject to the chosen similarity threshold.

The implementation of the Cosine and Jaccard similarity measures are based on the filtering techniques described in [146]. Thus matchers based on these two similarity measures require less time for higher thresholds. The figure shows that matchers based on Cosine or Jaccard are most efficient.

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

---

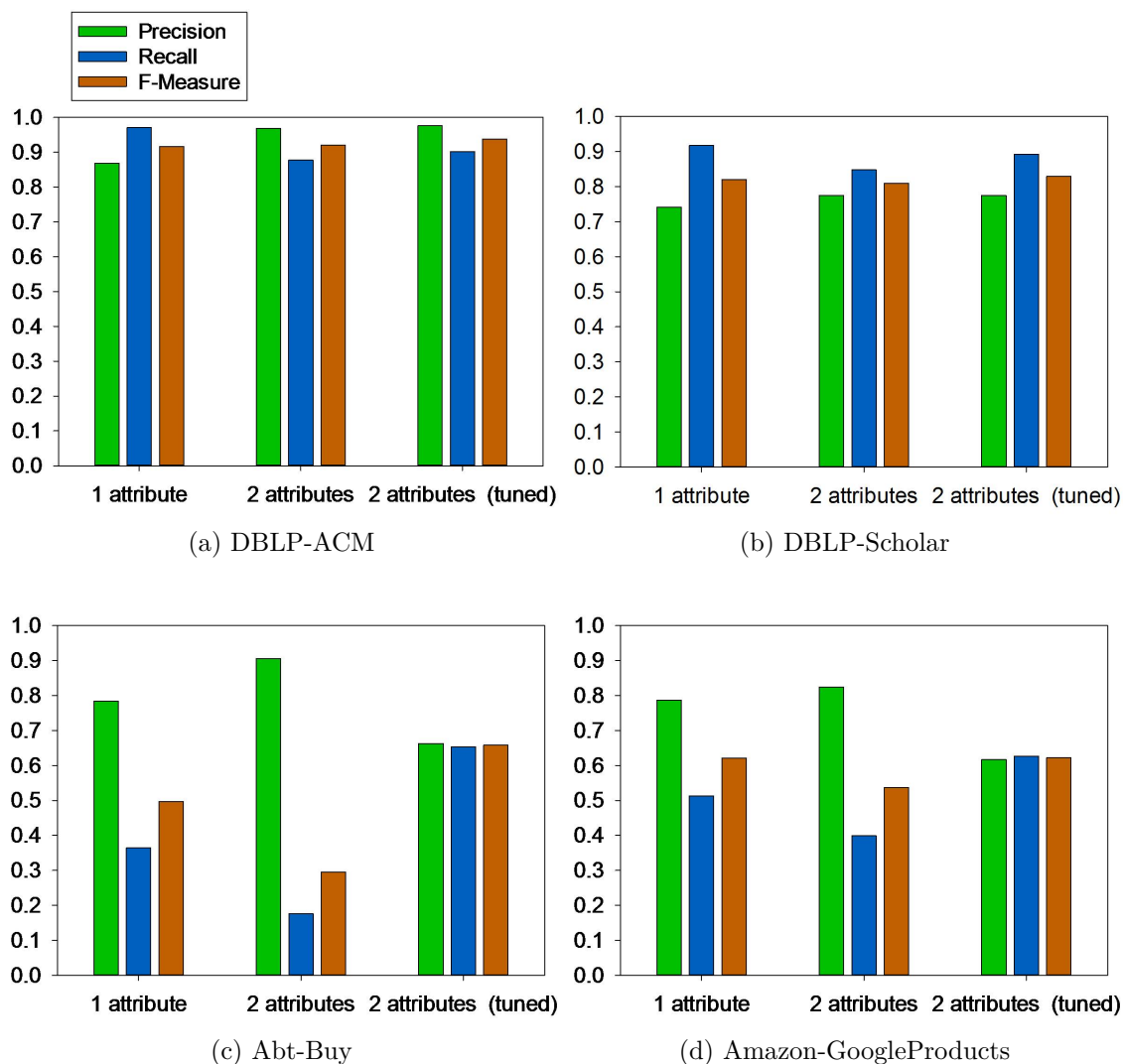


Figure 8.4: Match accuracy for a state-of-the-art approach (COSY) with default and tuned configurations

## 8.3 Evaluation of learning-based workflows

### 8.3.1 Random versus Ratio training selection

We compared random and ratio training selection considering the results for two training sizes of 50 and 500 selected object pairs, representing a rather small-to-moderate labeling effort. We varied the minimal similarity threshold for the TFIDF similarity (on the first attribute listed in Table 8.1) from 0.3 to 0.8.

Figures 8.5 and 8.6 display the F-measure results for the four match tasks DBLP-ACM, DBLP-Scholar, Abt-Buy, and Amazon- GoogleProducts and compares the random and ratio training selections. Figure 8.5 shows the results for labeling effort 50 and Figure 8.6 the results for labeling effort 500. We obtained the F-measure results with eight matchers and with SVM as the learner. For comparison, we also show the F-measure results for the manually determined baseline match configurations. The matchers used for learning operate on the same two attributes as the baseline strategy but apply one of the four similarity measures (Cosine, Jaccard, TFIDF, or Trigram) on them, resulting in eight matchers.

We first observe that even for the small training size of 50, the learned match strategies mostly outperform the baseline strategies, especially for the more difficult e-commerce tasks (about 14% improved F-measure values). Although the random and ratio approaches perform largely similarly, random is consistently somewhat less effective and more dependent on the chosen similarity threshold and training size. For higher similarity thresholds ( $\geq 0.6$ ), random mainly selects matching object pairs and thus provides few nonmatching pairs, making it difficult to learn how to identify nontrivial, nonmatches. Furthermore, the nonmatching object pairs selected with a high threshold might be rare outliers, and we risk the learned model being overfitted to those special cases, preventing it from classifying other object pairs correctly.

The ratio approach is generally better than random because it maintains a better balance between matching and nonmatching object pairs by eliminating entities from a randomly selected set of pairs. Although this reduces the remaining number of training data, our results show that this is more than offset by the better quality for learning. We experimented with different values for the ratio parameter and found rather stable results in the range from 0.2 to 0.5, with 0.4 as a good compromise value. The results show that ratio is also relatively stable for similarity thresholds between 0.3 and 0.6, even for smaller training sizes. For Abt-Buy, a similarity threshold of  $\geq 0.7$  left almost no nonmatches due to a heterogeneous representation; ratio thus became unable to retain a sufficient number of training pairs.

Based on this experiment, we conclude that the ratio approach is effective for selecting training data for learning-based object matching.

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

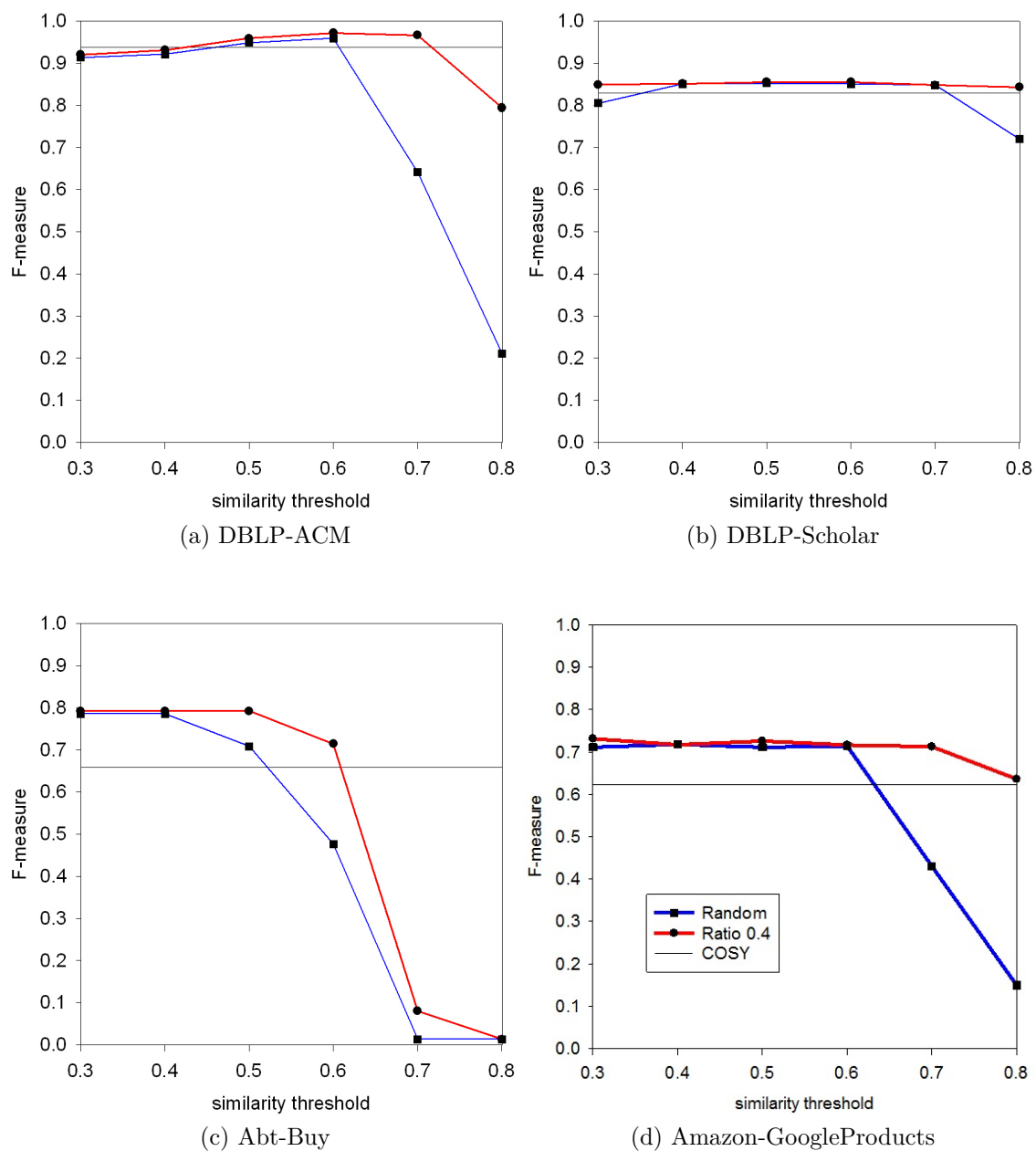


Figure 8.5: Comparison of Random and Ratio training selection using SVM with 8 matchers and 50 training pairs

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

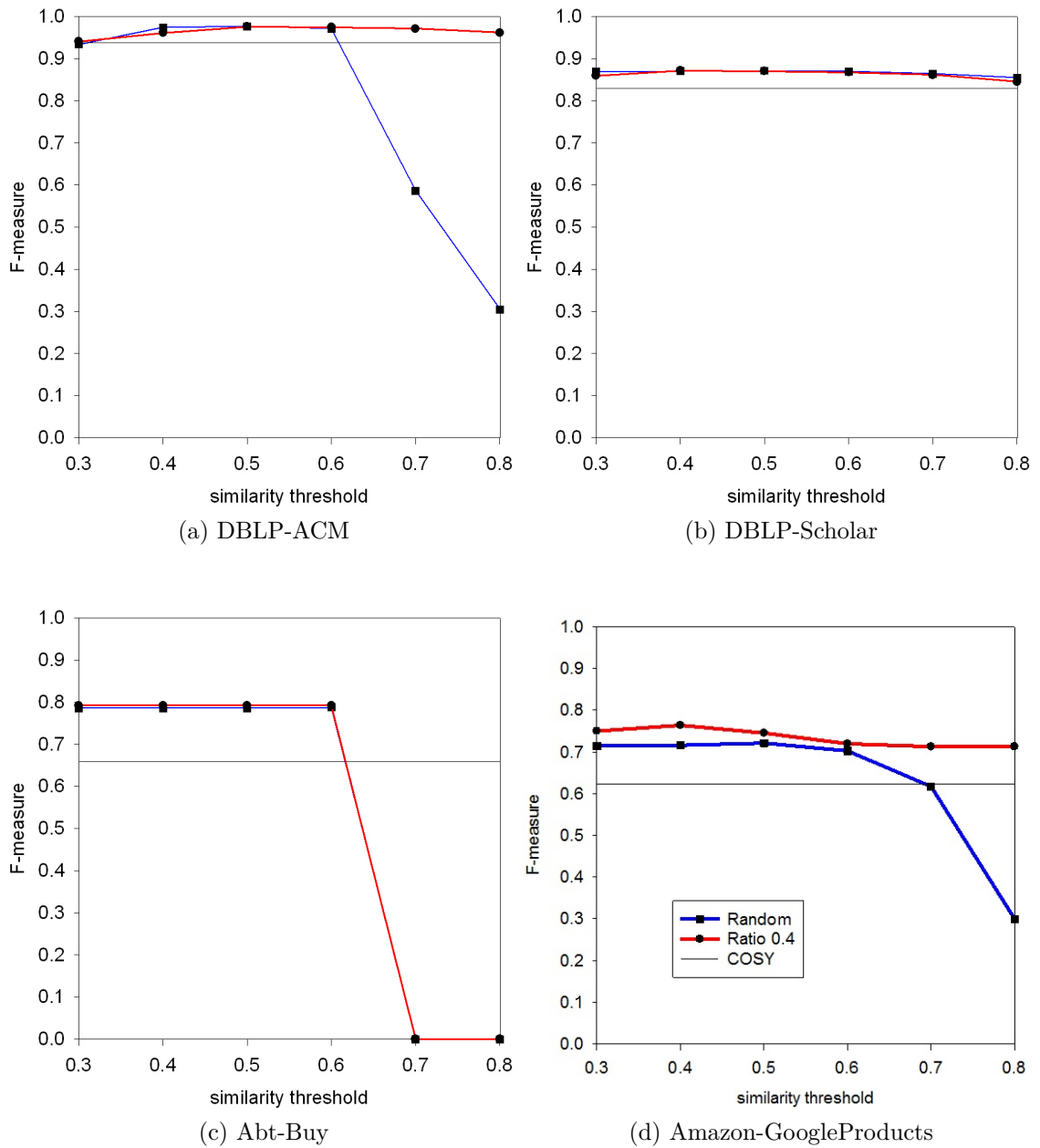


Figure 8.6: Comparison of Random and Ratio training selection using SVM with 8 matchers and 500 training pairs

### 8.3.2 Learner evaluation

In this experiment, we want to evaluate the relative quality of the four learner strategies for determining a combined object matching strategy: decision tree, logistic regression, SVM and the multiple learning approach. We used the same eight matchers than in the previous experiment.

Figure 8.7 shows the F-measure results for the four match tasks achieved with the four learners and different labeling efforts (x-axis). The labeling effort varies between 20 and 500 object pairs. We observe that all learners benefit from increasing the training size especially for the DBLP-Scholar and Abt-Buy problems. For all match tasks the baseline strategy could be clearly outperformed in most cases even for very small training sizes of 20 or 50 labeled object pairs. For the maximal training size of 500 the F-measure results could be improved to about 97% (vs. 94% for the baseline strategy) for DBLP-ACM, 91% (vs. 83%) for DBLP-Scholar, 86% (vs. 66%) for Abt-Buy and 77% (vs. 62%) for Amazon- GoogleProducts.

We observe that the three basic learners perform differently for the four match tasks so that no single basic learner consistently outperforms the others. For example, decision tree performs worst for DBLP-ACM but best for Abt-Buy. The decision tree and logistic regression approaches benefit most from more training data while SVM performs relatively well even for small training sizes.

An important observation is that the rather simple multiple learning approach consistently performs best for all match tasks and training sizes. This shows that it is able to effectively combine the strengths of the individual basic learners and compensate their weaknesses. The effectiveness of multiple learning approaches has also been demonstrated in other areas than object matching [133].

### 8.3.3 Influence of match configuration

To examine the usefulness of using a greater selection of matchers we now compare the performance of the four learners for two match configurations. For this comparison we focus on the more challenging match task DBLP-Scholar. In addition to the previously considered configurations with the eight matchers we now consider the four similarity measures on three attributes (title, authors, and venue) resulting in a total of 12 matchers.

Figure 8.8 contains a separate diagram for each of the four learners comparing the F-measure results for the two match configurations and different training sizes. The curves indicate that for small amounts of training examples the increased choice of matchers does not significantly improve performance but mostly decreases performance. This indicates that the learners need more training to effectively deal with the much increased solution space for selecting, ordering and weighting the applicable matchers. We therefore studied additional training sizes of up to 10,000 training

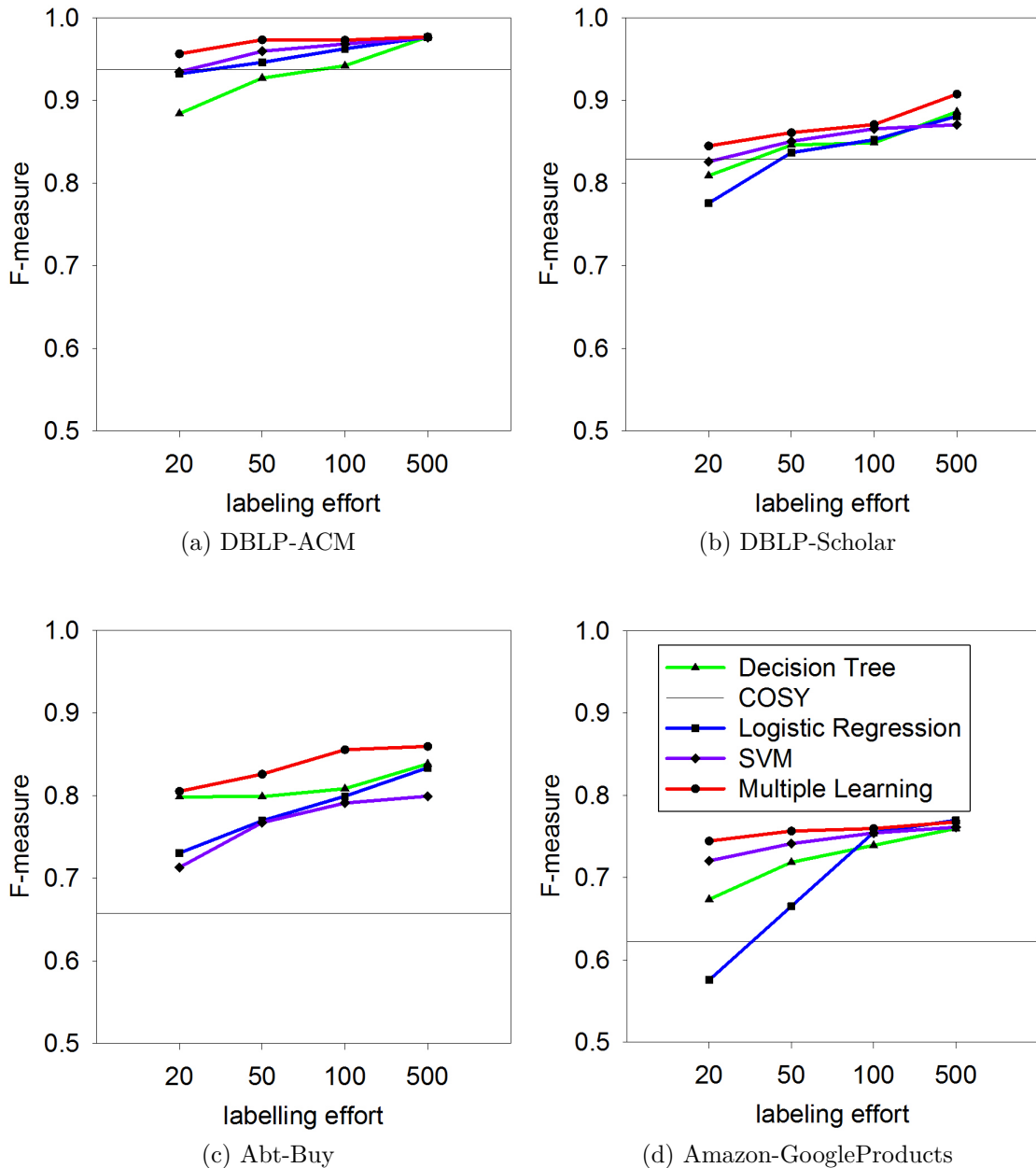


Figure 8.7: Comparison of different learners

pairs for this experiment. With more training, all learners eventually benefit from the increased number of matchers and outperform the object matching strategies based on only eight matchers. The improvements are significant for all learners especially for 500 and more training samples. The decision tree learner achieves this already for 50 training examples. It also benefits the most from increased training sizes and achieves the absolute best F-measure (93.4% for  $n = 10,000$ ).

To summarize we observe that our tuning framework can utilize a large choice of matchers to improve performance albeit at the expense of more training. The decision tree learner needs the least training data to utilize the increased optimization potential.

### 8.3.4 Concluding remarks

We investigated the use of supervised learners to semi-automatically determine effective object matching strategies for web data. We showed that the automatically found combinations of different matchers can clearly outperform manually tuned matcher combinations using a state-of-the art commercial match implementation. This is especially true for difficult match tasks such as matching heterogeneous product entities of different web sources. The improvements are achieved even for very small training sizes incurring a low manual effort compared to the high tuning effort needed for non-learning-based match strategies.

Using our evaluation platform FEVER we evaluated two methods for selecting training data and found the so-called Ratio method a simple and effective approach providing a balanced number of matching and non-matching training examples for learning. For learning we devised a simple yet effective multiple learning approach that is able to compensate weaknesses of basic learners even for small training sizes.



CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

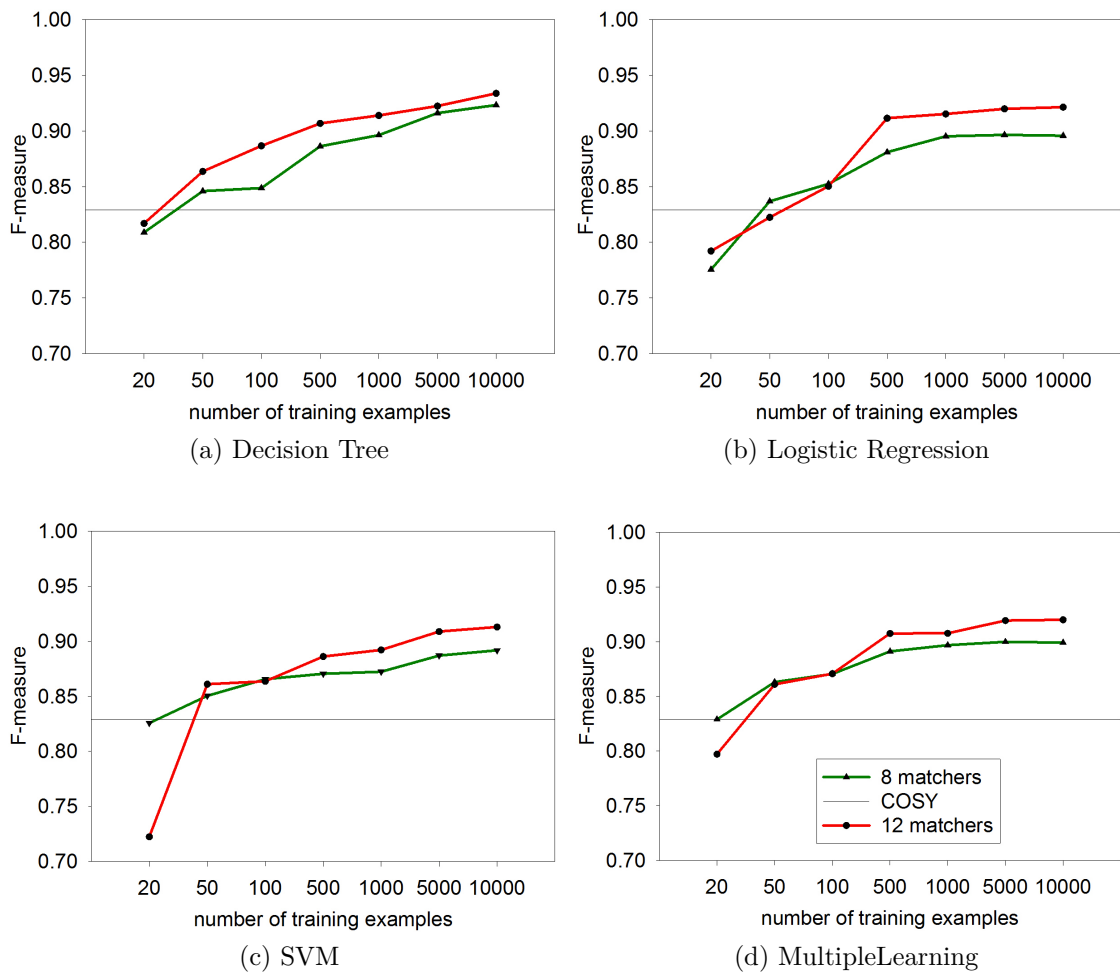


Figure 8.8: Evaluation of different matcher selections

## 8.4 Comparative evaluation of match approaches and frameworks

Despite the huge amount of recent research efforts on object matching there has not yet been a comparative evaluation on the relative effectiveness and efficiency of alternate approaches. We therefore present in this chapter such an evaluation of existing implementations on challenging real-world match tasks. We consider approaches both with and without using machine learning to find suitable parameterization and combination of similarity functions. In addition to approaches from the research community we also consider a state-of-the-art commercial object matching implementation. Our results indicate significant quality and efficiency differences between different approaches. We also find that some challenging resolution tasks such as matching product entities from online shops are not sufficiently solved with conventional approaches based on the similarity of attribute values.

The chapter is organized as follows: Section 8.4.1 describes the use of the FEVER framework to perform the comparative evaluation. It introduces the considered non-learning and learning-based match approaches and illustrates the FEVER operator trees applied for the evaluation. The evaluation results are presented and discussed in Section 8.4.2.

### 8.4.1 Evaluation setup

#### Non-learning match approaches

Figure 8.9a illustrates the FEVER operator tree that was applied in our evaluation of non-learning match approaches. We first apply a blocking operator to reduce the search space to the most likely matching object pairs. For comparability, we use a fixed blocking strategy for all non-learning and learning-based match approaches, i.e., blocking is not subject of the evaluation. The blocking result is input to the non-learning match approaches to be evaluated. In this study all considered match approaches are based on so-called attribute matchers that evaluate the similarity of attribute values based on some similarity function (e.g., an approximate string similarity). The approaches may evaluate only a single matcher (for a specific attribute pair and similarity function) or multiple matchers using different attribute pairs or similarity functions. In the latter case the approaches also need to support a combination of the individual similarities to derive a match decision. In our evaluation, we will always use the same attributes for comparability. Furthermore, all non-learning match approaches apply a threshold-based selection of the matching object pairs and require the similarity threshold to be provided as a parameter. For the similarity computation and the threshold-based match decision we used the implementation of the following non-learning match approaches:

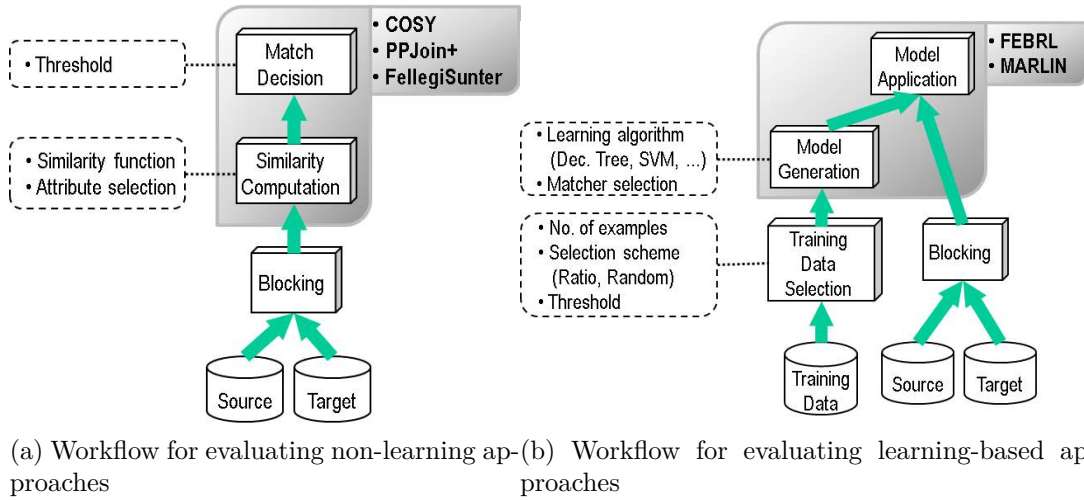


Figure 8.9: FEVER match workflows for evaluating existing object matching approaches

- **COSY**: This is the state-of-the-art commercial system for object matching as introduced in Section 8.2.
- **PPJoin+** [146] is a single-attribute match approach (similarity join) using sophisticated filtering techniques for improved efficiency. The approach has two parameters that need to be configured. The parameter function determines the similarity function used for the join. We will evaluate both supported implementations for the similarity function (Cosine, Jaccard). The parameter threshold determines the threshold for the similarity values above which entities are considered to match.
- **FellegiSunter** [54] is a non-learning approach from the FEBRL framework [33]. For similarity computation we evaluate three of the similarity measures provided by FEBRL (Winkler, Tokenset, Trigram). The approach has an lower and upper similarity threshold that can be adjusted. Object pairs with a similarity above the upper classification threshold are classified as matches, pairs with a combined value below the lower threshold are classified as non-matches, and those object pairs that have a matching weight between the two classification thresholds are classified as possible matches. For our evaluation, we set the lower threshold equal to the upper threshold as we only want a classification into matching and non-matching object pairs.

FEVER allows a systematic evaluation of operator trees for different parameter settings to help finding a suitable configuration. For this study we limit the number of parameters to be set by applying a fixed blocking approach and manually pre-selecting the attributes to be evaluated. We further evaluate the existing similarity functions either on one or two attributes of the input datasets. In both cases we have

to specify similarity thresholds on the single attribute or combined attribute similarity. For comparability, we evaluate every match approach for a fixed maximum number,  $N$ , of settings for the threshold parameters. FEVER supports several methods for selecting the parameter values such as manual (user-defined) and random. For this evaluation, we use the gradient descent strategy that iteratively refines a parameter setting by considering the quality of previously generated settings.

### Learning-based match approaches

Figure 8.9b shows the FEVER operator tree applied for the evaluation of learning-based approaches.

In our evaluation we will compare several existing training-based approaches for model generation and application offered by the following frameworks:

- **FEBRL [33] (Freely Extensible Biomedical Record Linkage)** provides a support vector machine (SVM) implementation for learning suitable matcher combinations. For attribute matching we will evaluate the same three similarity measures than for the non-learning matchers studied for FEBRL.
- **MARLIN [18] (Multiply Adaptive Record Linkage with INduction)** offers two string similarity measures (Edit Distance and Cosine) and several learners, specifically SVM and decision trees. The learners can be used in a single step approach or can be employed for a two-level learning approach. For the two-level approach string similarity measures are first trained for every selected attribute so that they can provide accurate estimates of string distance between values for that attribute. Next, a final decision is learned from similarity metrics applied to each of the individual attributes.

The effectiveness of machine learning approaches is known to depend on the provision of sufficient, suitable, and balanced training data. On the other hand, the number of object pairs to be labeled affects the manual tuning effort and should thus be small. To address these issues we build upon our evaluation experiences and only consider object pairs for labeling for which the similarity exceeds a specified threshold  $t$ . This ensures that the training is not dominated by trivial non-matching object pairs that are not useful to find effective matcher parameters and matcher combinations. We use our **Ratio** training selection approach. We found that setting  $r = 0.4$  and  $t = 0.4$  with TFIDF is a reliable and effective default configuration. Our evaluation for learning-based matching will thus be based on this configuration.

## 8.4.2 Evaluation results

We first present match quality and runtime results separately for non-learning and learning-based approaches. Afterwards we briefly compare the two kinds of matchers with each other. The runtime results are determined for a HP Z400 workstation with 2.66 GHz Intel Quad-Core Processor W3520 and 4GB of RAM running 64-bit Windows 7. The evaluated match approaches are implemented in different languages: PPJoin+ is implemented in C++, MARLIN in Java and FEBRL in Python.

### Non-learning approaches

Figures 8.10 and 8.11 show the match quality (precision, recall, F-measure) results for the four real world match tasks achieved with different non learning approaches. Figure 8.10 shows the results for approaches operating on just a single attribute, namely the first attribute listed in Table 8.1 (publication title for the bibliographic tasks, product name for the e-commerce tasks). Figure 8.11 shows the results for approaches combining the similarity for two attributes (the first two attributes listed in Table 8.1). In both cases we optimized the threshold for the final match decision while all other parameters of the approach were kept constant. Optimization was done with the GradientDescent approach on a test set of 500 object pairs for each match task. For the FellegiSunter approach from the FEBRL framework we considered three different similarity measures, namely Winkler, TokenSet, and Trigram. FEBRL's FellegiSunter approach sums the logarithms of the single similarities. For the COSY approach it is not known how similarities are combined.

All simple and combined approaches could effectively solve the simple bibliographic match tasks DBLP-ACM (F-measure > 91%), except for the FellegiSunter approach with the Winkler measure which did not even reach an F-measure of 50% because it suffers from a very low precision. The e-commerce match tasks turned out to be much more challenging so that no approach could achieve an F-measure of more than 62% (Amazon-GoogleProducts) or 70% (Abt-Buy). The COSY approach is the top or among the top performing approaches for all match tasks. From the FellegiSunter approaches the configuration using Trigram performed best for all match tasks. Using two attributes (first two of Table 8.1) is not always more effective than using one attribute because it is difficult to find a good similarity combination. All approaches become worse for the easy bibliographic match task DBLP-ACM. COSY becomes worse for DBLP-ACM, DBLP-Scholar, and ABT-BUY. PPJoin+ could not be evaluated on two attributes because no combination approach is provided.

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

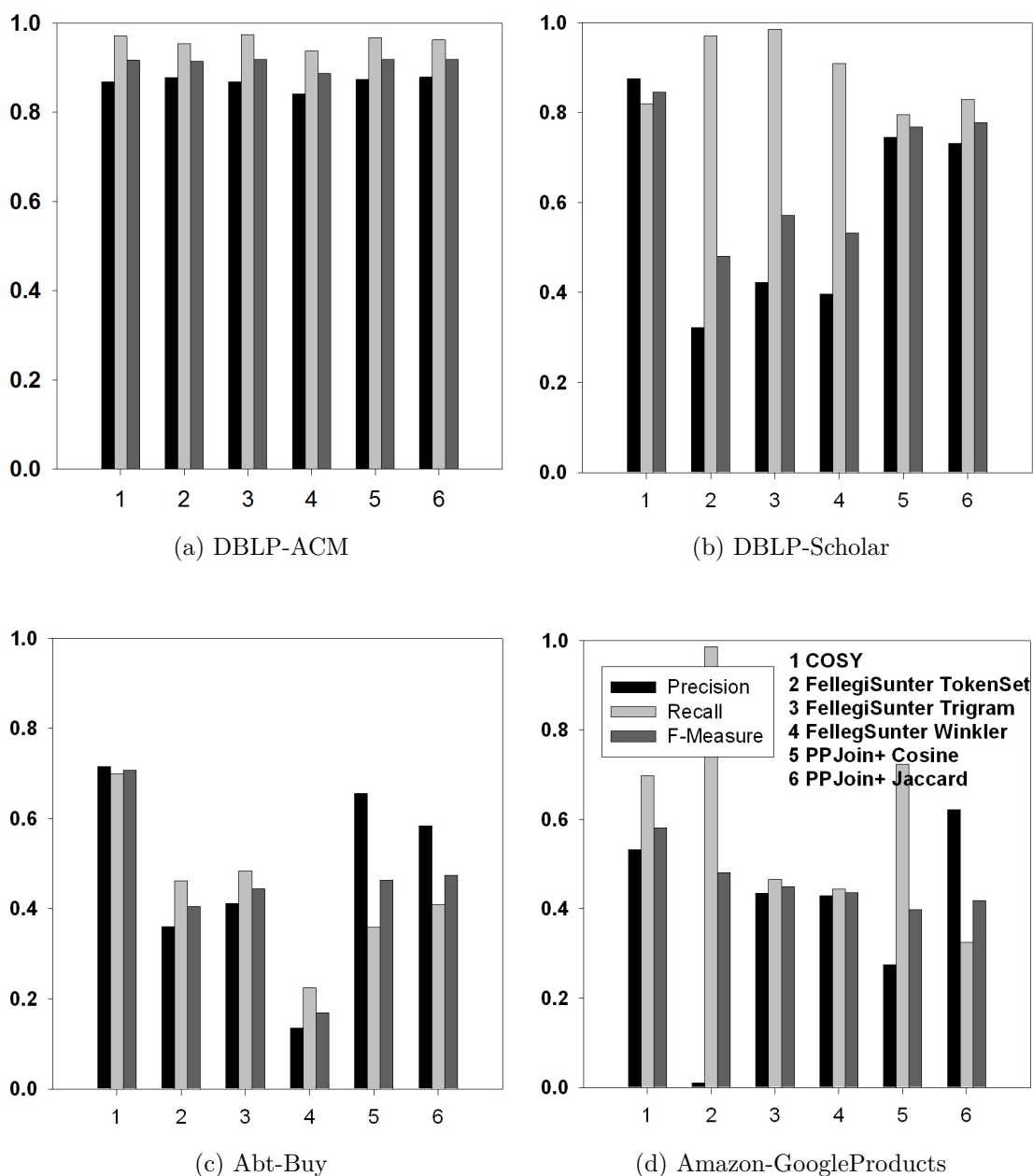


Figure 8.10: Performance results for simple non-learning approaches (1 attribute)

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

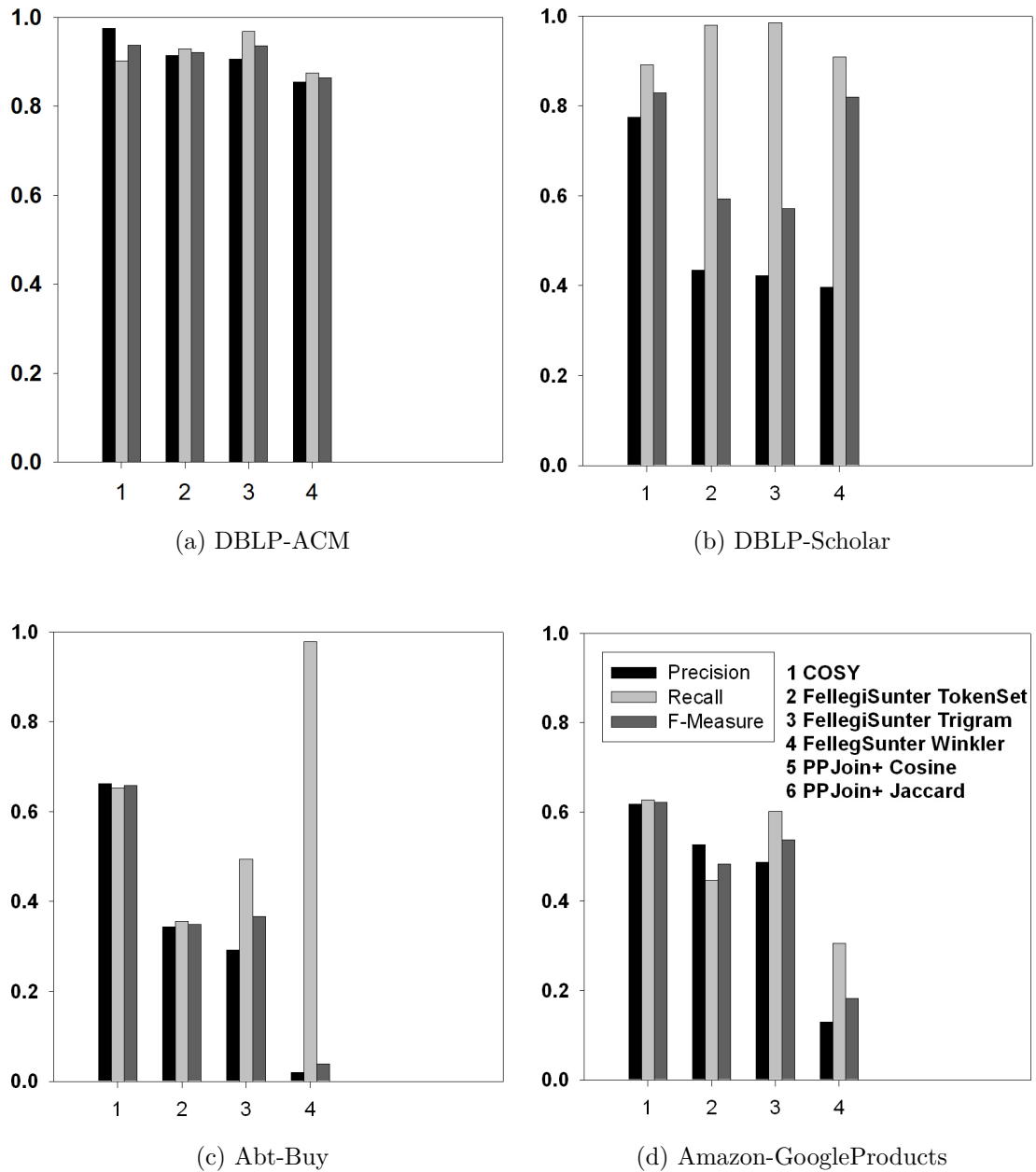


Figure 8.11: Performance results for non-learning combination approaches 2 attributes)

1 attribute	DBLP-ACM		DBLP-Scholar		Abt-Buy		Amazon-GoogleProducts	
	blocked	Cartesian	blocked	Cartesian	blocked	Cartesian	blocked	Cartesian
COSY	1	1.6	8.8	224.7	2.7	5.8	6.5	9.6
FellegiSumter TokenSet	2.1	170	10.2	64,057	2.5	17.2	4.6	84
FellegiSumter Trigram	2.5	655	44.7	243,060	25	105	34.1	320
FellegiSumter Winkler	5.7	1,601	164.6	277,200	53.5	364	96.2	1,065
PPJoin+ Cosine	0.4	0.9	3.4	6.9	0.6	2.5	0.5	0.9
PPJoin+ Jaccard	0.4	0.6	3.5	7	0.6	2.5	0.5	0.9
2 attributes								
COSY	35	56	17	434	44	94	28	41
FellegiSumter TokenSet	3	429	17.8	108,896	5.9	43	44	709
FellegiSumter Trigram	3.1	1,512	116	> 500,000	58	635	1,940	16,620
FellegiSumter Winkler	7.4	3,602	341	> 500,000	135	970	2,833	20,760

Table 8.3: Execution times (in seconds) for non-learning approaches



Table 8.3 lists the execution times for the considered non-learning approaches for the blocked input as well as the Cartesian product of the considered match tasks. The table shows significant differences between the approaches already for the blocked input. The evaluation of the Cartesian product tests the scalability and leads to huge differences. PPJoin+ and COSY achieved very fast execution times and could even achieve acceptable run times for the Cartesian product. PPJoin+ implements an intelligent pruning of the search space and is uniformly the fastest approach for all match tasks with execution times between less than a second to at most seven seconds. The small increase of at most a factor of 2 for evaluating the Cartesian product proves the excellent scalability of PPJoin+. In this respect it also outperforms COSY that noticeably slows down for the Cartesian product evaluation of DBLP-Scholar (almost 4 minutes vs. 9 seconds for the blocked input).

The considered FEBRL approaches were mostly much slower than COSY and PPJoin+, on the Cartesian products by orders of magnitude. This may be influenced by the Python-based implementation of FEBRL. FellegiSunter using the Winkler similarity turned out to be not only the least effective but also by far the slowest of all non-learning match approaches. On the blocked input, FEBRL with tokenset similarity is almost as fast as COSY.

### Learning-based approaches

Figures 8.12, 8.13 and 8.14 show the F-measure results for the four real-world match tasks achieved with different learning-based approaches from FEBRL and MARLIN and different labeling efforts (x-axis). The labeling effort varies between 20 and 500 object pairs, i.e., we consider only comparatively small training sizes and thus a limited amount of labeling effort. The F-measure results are averaged over 10 runs. All results in Figures 8.12, 8.13 and 8.14 refer to matching on the first or the first two attributes listed in Table 8.1 (publication title and authors for the bibliographic tasks, product name and product description for the e-commerce tasks) with different similarity functions. Figure 8.12 shows the results for the SVM learner of FEBRL that was applied for the same three similarity functions (TokenSet, Trigram, and Winkler) as for the non-learning case. In addition we use the SVM for two combined match strategies using all three similarity measures either on one or on two attributes. Figures 8.13 and 8.14 show the results for MARLIN separated by the employed learner, first for MARLIN’s decision tree implementation ADTree (Figure 8.13) followed by the SVM results (Figure 8.14). For both learners we applied the two similarity measures Edit Distance and Cosine. EditDistance was used in the single-step as well as the two-step learning approach. Cosine was just applied in the single-step approach as it has limitations in the two-step implementation as mentioned by the authors in [18]. We also tested combined match strategies using the two similarity measures either on one or on two attributes for single-step learning. In total, 15 different learning-based approaches are considered.

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

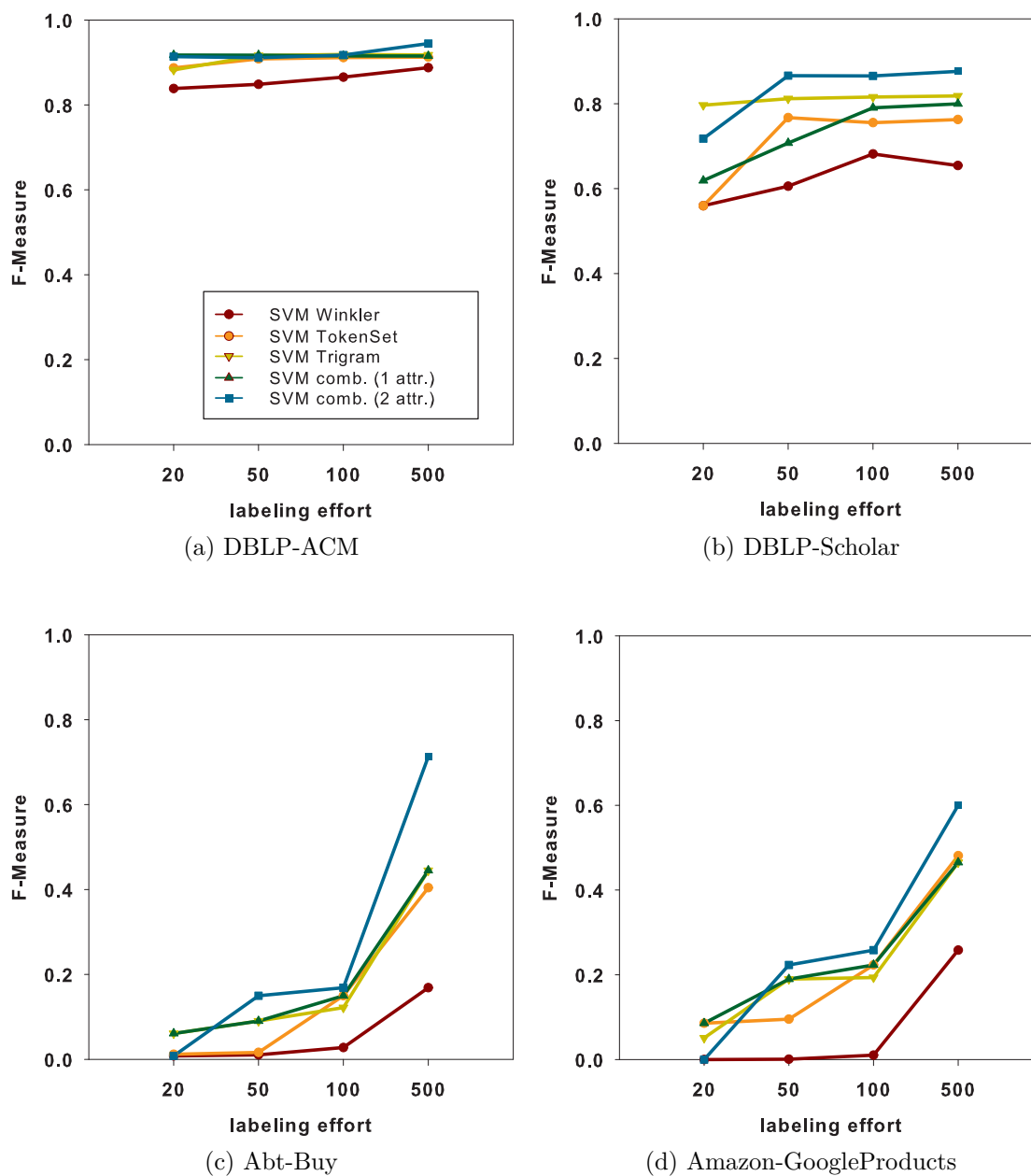


Figure 8.12: Evaluation results for learning-based approaches with SVM from FEBRL

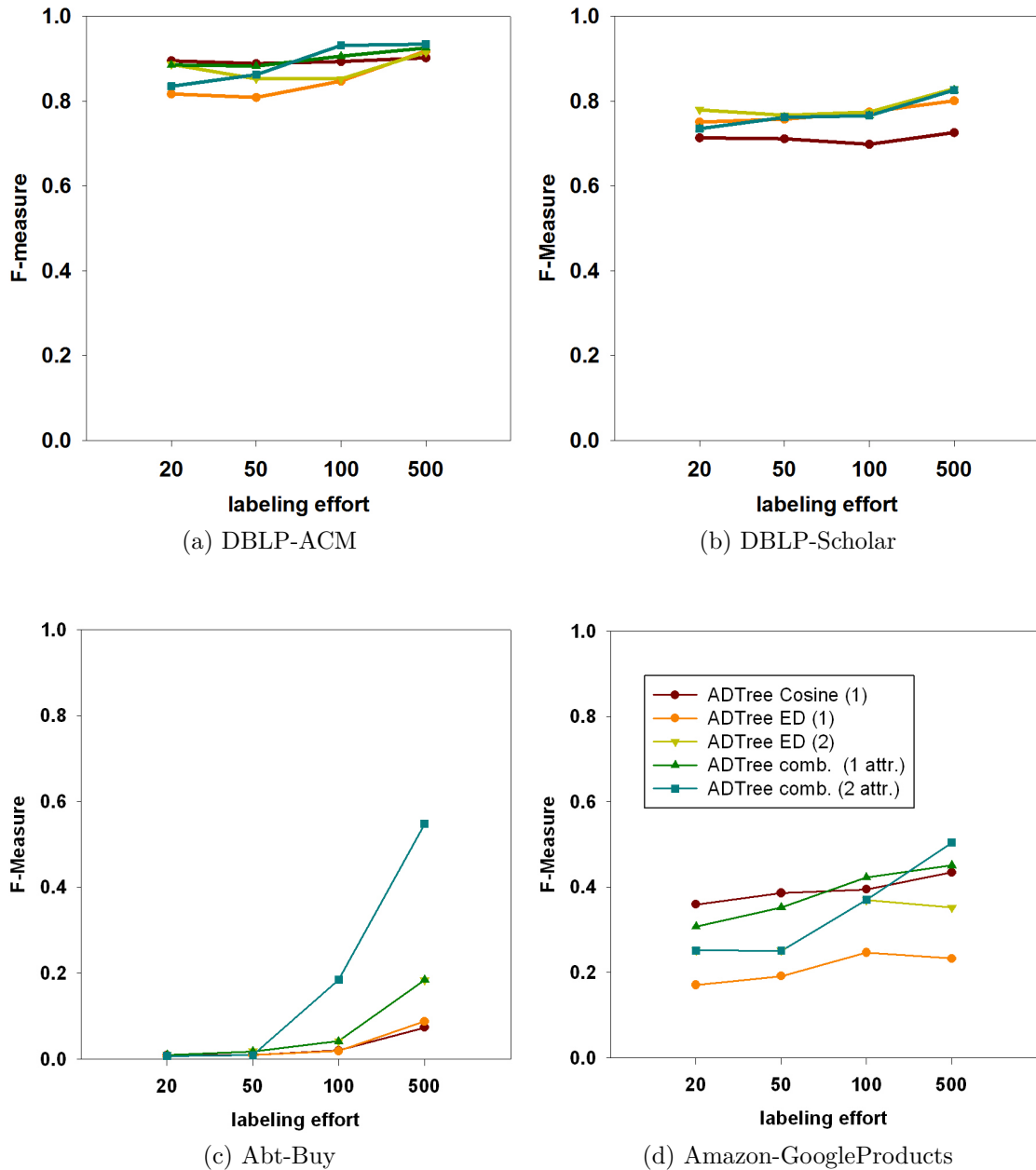


Figure 8.13: Evaluation results for learning-based approaches with decision tree from MARLIN

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING APPROACHES WITH FEVER

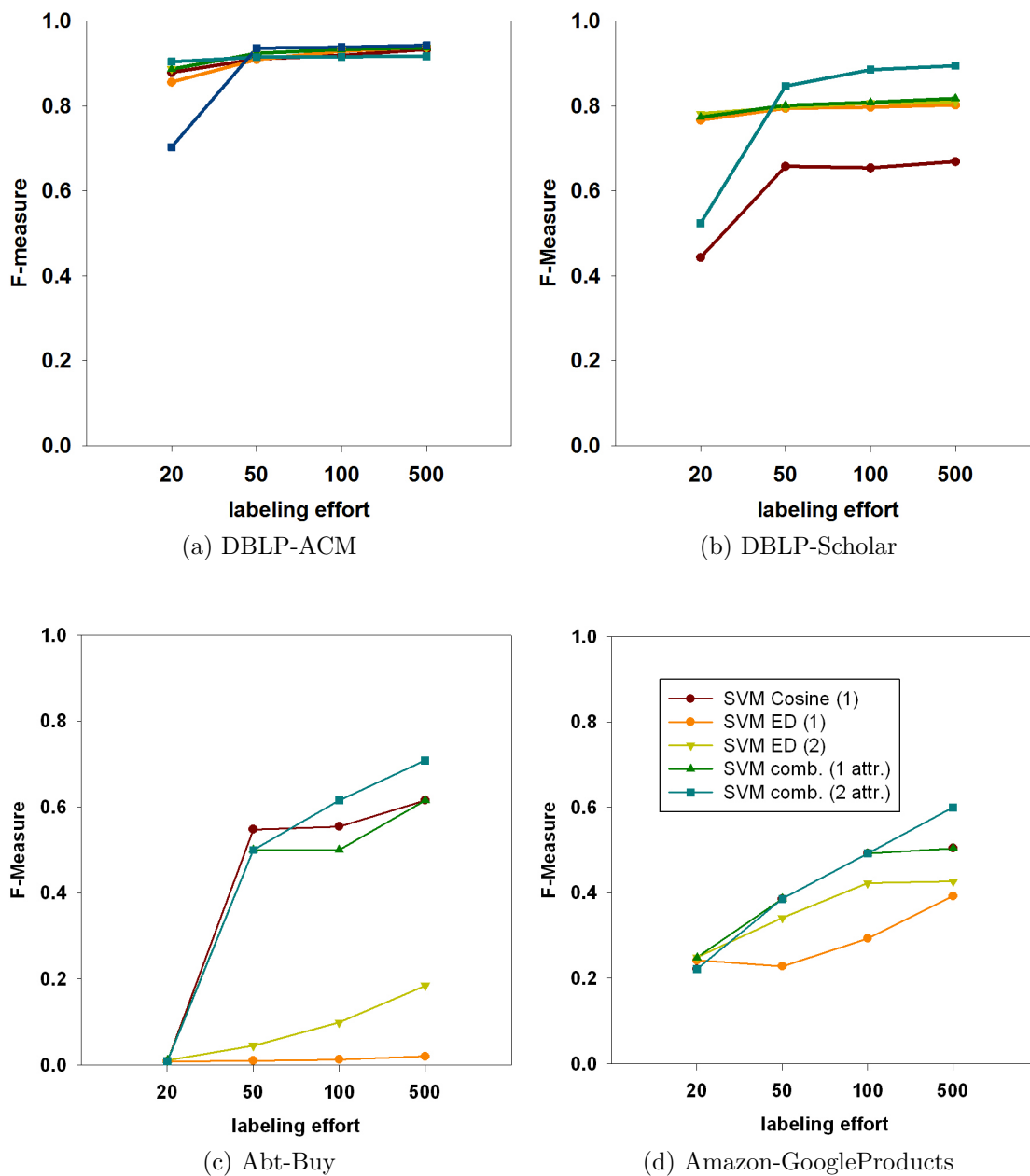


Figure 8.14: Evaluation results for learning-based approaches with SVM from MARLIN

For the easy bibliographic match task DBLP-ACM, we observe that both FEBRL and MARLIN are able to achieve stable results already for very small training sizes of 20 labeled object pairs with all evaluated approaches. For the more challenging bibliographic match task DBLP-Scholar, for both FEBRL and MARLIN the SVM strategies combining several matchers on two attributes perform best and achieve F-measure results of 88-89%. The best one-attribute strategies are the combined SVM approaches and SVM using trigram (for FEBRL) or EditDistance (MARLIN). All approaches have substantial difficulties with the e-commerce match tasks, especially for training sizes smaller than 500 object pairs. The best match quality is always achieved for the combined strategies using all similarity measures on two attributes, followed by the combined approach on one attribute. This underlines that the learners are able to effectively find a combination of several matchers. The decision tree learner of MARLIN is mostly inferior to the SVM-based results. The SVM learner of MARLIN performs slightly better than the one of FEBRL for smaller training sizes. However, for 500 training pairs both SVM learners perform similarly well and achieve a top F-measure of about 71% for Abt-Buy and (only) 60% for Amazon-GoogleProducts. From the single similarity approaches FEBRL with trigram and MARLIN with cosine similarity performed best for the e-commerce tasks. For MARLIN, the 2-step learning for Edit-Distance was always better than the 1-step approach but still too ineffective for the e-commerce tasks. Here the rather long product names and product descriptions tend to favor token-based similarity measures such as cosine, trigram, or the unsupported TF/IDF similarity.

There are huge differences between the approaches regarding execution time as can be seen in Table 8.4. In general, the execution times for the considered learning-based approaches are significantly worse than for the non-learning approaches. Nearly all learning-based approaches do not scale with larger input sets and are unable to match sufficiently fast on the Cartesian product. For the largest match task DBLP-Scholar execution times of hours to days are needed, the most effective combined approaches exceeded our limit of 500,000 seconds. On the blocked datasets, the approach with the fastest execution time for all match tasks is the FEBRL approach with the TokenSet Cosine measure. The combined match approaches on two attributes take the longest time for blocking, too. They are more than a factor 2 slower than the other learning-based approaches and (except for DBLP-ACM) requires execution times in the order of minutes to hours.

### **Non-learning vs. learning-based**

Table 8.5 shows a brief summary of the maximum F-measure results achieved for each of the considered non-learning as well as learning-based approaches. For three of four tasks the commercial COSY approach performs best for matching on one attribute. However for two match tasks its quality degrades when using two attributes. The learning-based approaches, on the other hand, always improve for matching on

	DBLP-ACM		DBLP-Scholar		Abt-Buy		Amazon-GoogleProducts		
	blocked	Cartesian	blocked	Cartesian	blocked	Cartesian	blocked	Cartesian	
<b>FEBRIL</b>	SVM TokenSet	3	244	20.0	249,364	8	23	14	124
	SVM Trigram	5	859	79.0	250,920	25	127	46	415
	SVM Winkler	8	2,022	196.5	295,800	62	409	110	1,225
	SVM comb. (1 attr.)	13	2,400	275	> 500,000	83	590	154	1,481
SVM comb. (2 attr.)	99	4,320	482	> 500,000	232	1,364	196	36,090	
<b>MARLIN</b>	ADTree ED (1)	3	329	76	10,090	22	64	41	161
	ADTree ED (2)	5	582	96	17,427	37	119	57	244
	ADTree Cosine (1)	5	157	71	301	1	89	2	98
	ADTree comb. (1 attr.)	7	951	104	28,476	40	340	95	373
	ADTree comb. (2 attr.)	12	1,553	324	46,501	551	3,456	10,299	41,615
	SVM ED (1)	5	633	117	257,982	28	231	66	333
	SVM ED (2)	7	979	146	445,575	44	192	80	465
	SVM Cosine (1)	4	267	41	7,696	10	143	26	186
	SVM comb. (1 attr.)	9	1,336	157	> 600,000	68	552	127	498
	SVM comb. (2 attr.)	20	2,196	375	> 900,000	324	3,747	13,768	55,632

Table 8.4: Execution times (in seconds) for non-learning approaches

CHAPTER 8. COMPARATIVE EVALUATION OF OBJECT MATCHING  
APPROACHES WITH FEVER

	DBLP-ACM		DBLP-Scholar		Abt-Buy		Amazon- GoogleProducts	
	1 attr	2 attr	1 attr	2 attr	1 attr	2 attr	1 attr	2 attr
COSY	91.7	93.8	<u>84.5</u>	82.9	<u>70.7</u>	65.8	<u>62.1</u>	<u>62.2</u>
FEBRL FellegiSunter	91.8	93.6	57.2	81.9	44.5	36.7	48.4	53.8
PPJoin+	91.9	-	77.8	-	47.4	-	41.9	-
FEBRL SVM comb.	91.5	<u>94.4</u>	81.9	87.6	44.5	<u>71.3</u>	46.5	60.1
MARLIN ADTree comb	<u>92.5</u>	93.4	82.6	82.9	18.4	54.8	45.0	50.5
MARLIN SVM comb.	91.6	94.2	82.6	<u>89.4</u>	54.8	70.8	50.5	59.9

Table 8.5: Summary of evaluation results (F-measure in %, top values are underlined) DBLP-ACM

two attributes compared to only one attribute underlining their potential to effectively combine different match criteria. SVM learning was most effective and the FEBRL and MARLIN implementations perform similarly well for training size 500. They achieve the top F-measure for three of the four match tasks for matching on two attributes. The learning-based approaches from FEBRL perform better than the non-learning FEBRL (FellegiSunter) approach, especially when considering two attributes. The good quality of the learning-based approaches on two attributes comes at the expense of significantly higher execution times. With a single matcher on just one attribute the learning-based approaches could not exploit their potential to combine several matchers and thus turned out to be inferior to the non-learning approaches considering both match quality and execution times. The relatively low match quality for the e-commerce task asks for further improvements, e.g., by considering additional similarity measures such as TFIDF and/or further attributes and spending more training effort on learning.

### Comparison with FEVER

Figure 8.15 compares the maximum F-measure results achieved with approaches using six and eight matchers with FEVER and the best learning-based approach on two attributes (six matchers) from the comparative evaluation summarized in Table 8.5. For the bibliographic match tasks we could achieve comparable results. For the two E-Commerce tasks we were able to clearly outperform all other evaluated competitive approaches. With our learning-based approaches we achieve significantly better match effectiveness results. For both E-Commerce task the obtained F-measure values are above the result of the best competitive approach. For Abt-Buy the F-measure improvement amounts to 3% for six matchers and to 14% for eight matchers. For Amazon-GoogleProduct the F-measure improvement amounts

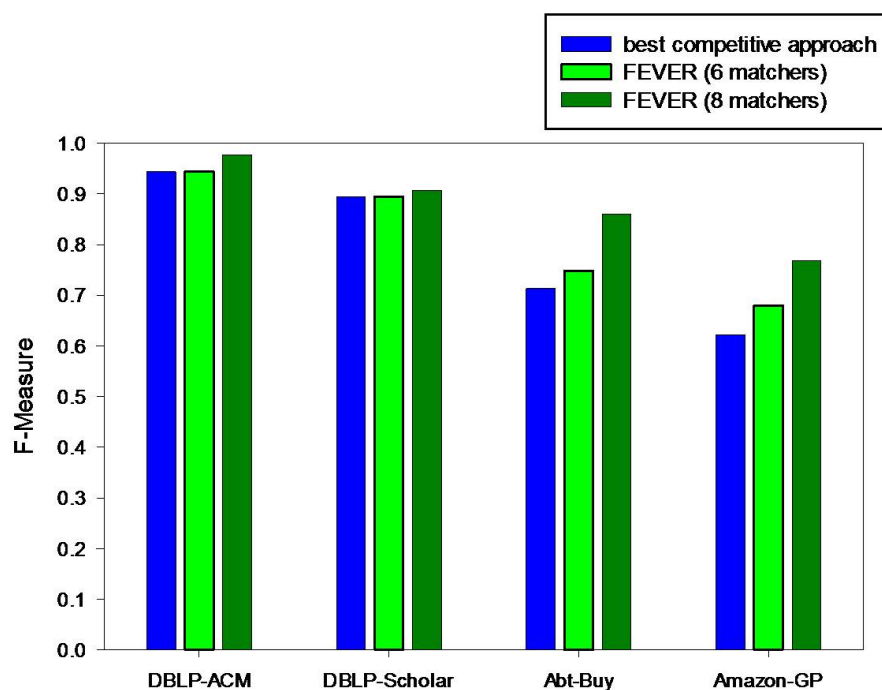


Figure 8.15: Effectiveness comparison of FEVER with best competitive learning-based approach on two attributes

to 5% for six matchers and to 14% for eight matchers. The improvements are achieved even for very small training sizes incurring a low manual effort.

Regarding efficiency, FEVER is on a competitive basis with the PPJoin+ approach for the non-learning approaches. For the learning-based approaches FEVER is more scalable compared to FEBRL and MARLIN. We illustrate that for the largest match task DBLP-Scholar on the Cartesian product. For simple strategies using the SVM on the title attribute with a single matcher FEVER requires between 1989 and 4002 seconds depending on the used matcher. That is more than 100 times faster than any of the approaches from FEBRL. With FEVER even combined approaches on one and two attributes are possible. The combined strategy on one attribute with four matchers required 8237 seconds.



## **Part IV**

# **Product offer matching**



# 9

## The product offer matching problem

Product offer matching is a special case of object matching that is needed to identify equivalent offers referring to the same real-world product.

After a short introduction to the problem of product offer matching in Section 9.1, Section 9.2 points out some specific challenges that have to be addressed.

### 9.1 Problem outline

As consumers all over the world increasingly use online shopping for consumer products and services, price comparison portals (e.g., Idealo.de), online marketplaces (e.g., Amazon.com), or product search engines (e.g., Google Product Search) have become popular. A challenge for all these providers of price comparisons is to aggregate offers for the same product.

Product offer matching is a special case of object matching. It deals with the identification of different descriptions or offers referring to the same real-world product. Given that many thousands of online shops sell millions of diverse products over the web, product offer matching has become of increasing importance [72], [110].

For the product offer matching problem we assume that there exists no catalogue of consolidated product offers. Hence, we have to match product offers against each other. Matching product offers against other offers is much more challenging than matching product offers to an existing catalogue of consolidated product descriptions (e.g., as done in [72]) due to the absence of structured and cleaned product

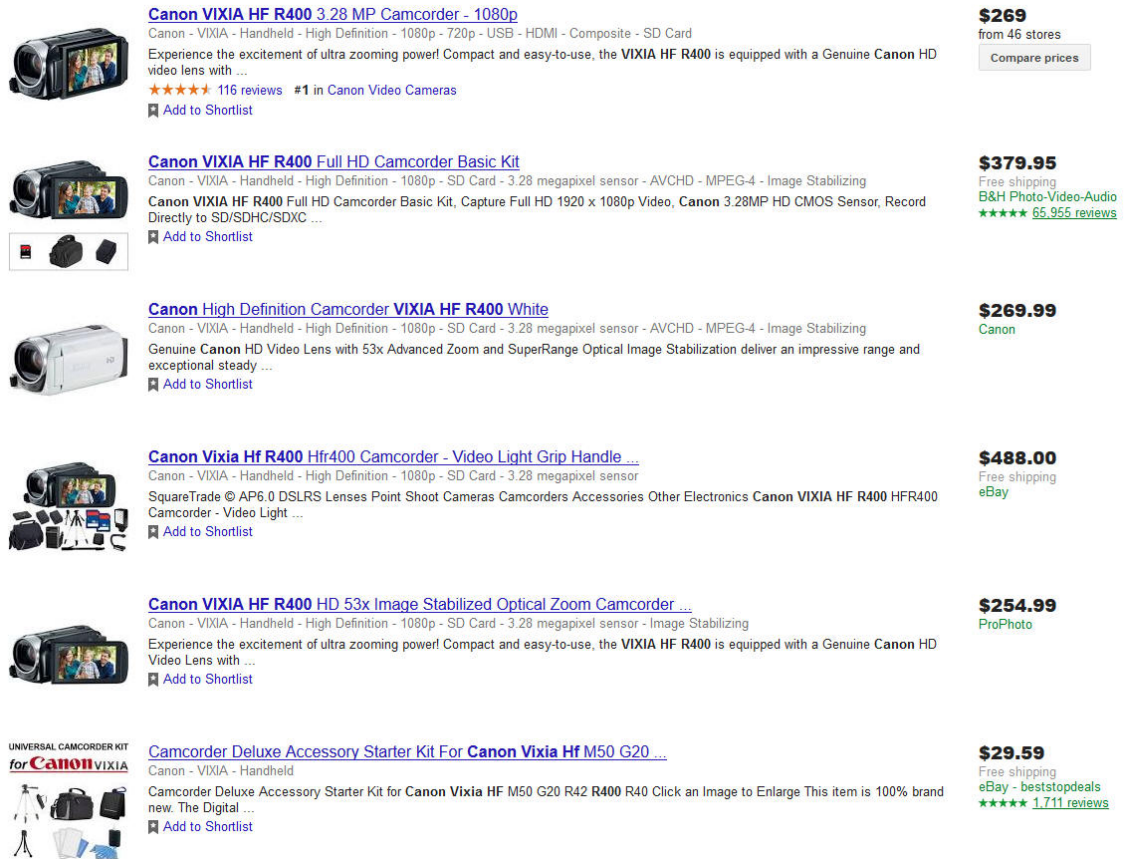
data. Directly matching product offers is highly relevant since in many scenarios a carefully maintained reference product catalogue is not available. For example, price monitoring applications that keep track of offers from different merchants or websites typically start without a product catalogue. Product offer matching is also useful to aggregate product information for an incremental construction of product catalogues.

Product offer matching for e-commerce websites introduces several specific challenges that make this problem much harder than other forms of object matching, e.g., to match records about publications. In particular, there is a huge degree of heterogeneity since product offers come from thousands of merchants using different names and descriptions of the products. Furthermore, offers frequently have missing or wrong values and are mostly not well structured but mix different product characteristics in text fields such as product name or description [72].

For illustration, we show in Figure 9.1 some result offers for the product search engine Google Product Search and a specific camcorder. The offers refer to different merchants that use heterogeneous names, descriptions, and other attributes for the same product and may also contain misspellings and other errors. For example, the product names for the considered product Canon Vixia HF R400 partially include specific technical details that may complicate product offer matching, e.g., to find out that (only) the first three entries refer to the same product.

As this example already illustrates there are several specific challenges that have to be mastered for product offer matching. We will discuss these challenges in more detail in the following section.

## CHAPTER 9. THE PRODUCT OFFER MATCHING PROBLEM



Product Title	Price	Seller
Canon VIXIA HF R400 3.28 MP Camcorder - 1080p	\$269	from 46 stores
Canon VIXIA HF R400 Full HD Camcorder Basic Kit	\$379.95	B&H Photo-Video-Audio
Canon High Definition Camcorder VIXIA HF R400 White	\$269.99	Canon
Canon Vixia Hf R400 Hfr400 Camcorder - Video Light Grip Handle ...	\$488.00	eBay
Canon VIXIA HF R400 HD 53x Image Stabilized Optical Zoom Camcorder ...	\$254.99	ProPhoto
Camcorder Deluxe Accessory Starter Kit For Canon Vixia Hf M50 G20 ...	\$29.59	eBay - beststopdeals

Figure 9.1: Product offers related to the Canon VIXIA HF camcorder in Google Product Search

### 9.2 Special challenges

For product offer matching there exist some specific challenges that have to be addressed. Some of them are illustrated in Figure 9.1.

- **Unstructured text:** Typically, an offer consists of a largely unstructured textual description. Different product characteristics are often mixed in text fields such as product name and technical specifications. For example, the product name "Canon VIXIA HF R400 3.28 MP Camcorder - 1080p" contains the resolution (3.28 MP) and the specification of the high-definition video mode (1080p).
- **Domain-specific labels and short forms:** For german fashion product offers "lg. A." is a short form for "lang Arm", whereas in electronics LG is a well-known brand manufacturer.
- **Heterogeneous labels for sizes and quantities:** Sizes and quantities can

be stated in different ways. For example for a bundle product the product offer might specify the total amount of pieces as a single number (e.g., 168 pieces). Another product offer on the other hand might specify the total amount of pieces as the product of the amount of pieces of all bundle parts (e.g., 3x56 pieces).

- **Heterogeneous string lengths:** Descriptions for product offer are highly heterogeneous. Some product offers are described in great detail with long product names and descriptions, whereas other product offers are scarcely described.
- **Synonyms:** For many products there exist alternative denominations. For example, hoodie, hoody and hooded sweatshirt are all denominations for a sweatshirt with a hood.
- **Missing values:** Often the representation of a product offer is fragmentary. Important attribute values like the brand value are often missing.
- **Wrong values:** Another problem are wrong values. The specifications of a product offer are often erroneous. This is even applies to product identification numbers, such as UPC (Universal Product Code) or EAN (European Article Number) values.

# 10

## Tailored product offer matching approaches

As a response to the challenges outlined in the previous section we propose the use of tailored approaches for product offer matching based on a preprocessing of product offers to extract and clean new attributes usable for matching. In particular, we propose a new approach to extract and use so-called product codes to identify products and distinguish them from similar product variations.

In Section 10.1 we outline our overall approach to matching product offers. It supports category-specific match strategies and is based on machine learning to semi-automatically determine a match strategy utilizing several attributes and similarity functions. In Section 10.2 we describe the Modified Naive Bayes approach according to [74] that we employ to assign product offers automatically to a category in a given taxonomy. Techniques to consolidate heterogeneous manufacturer denominations and complement missing values are introduced in section 10.3. Section 10.4 proposes a new approach for improving product offer matching based on a pattern-based extraction and web-based verification of product codes.

### **10.1 Overall product offer matching workflow**

Given a collection of product offers our goal is to identify corresponding offers, i.e., offers that refer to the same real-world product. Typically only few attribute values are available per offer, in particular product title, product description, manufacturer,

price, and perhaps a product identification such as UPC. While we consider the UPC values for evaluation, we are only concerned here with matching based on the remaining information.

Matching noisy real-world objects such as product offers requires the combined use of several matchers (e.g., different string similarity measures [38]) on several attributes to derive a match decision for every pair of entities. Given the availability of several relevant attributes and similarity measures, it becomes a very complex task to manually specify a reasonable strategy for the combination of matcher similarities. Therefore, we employ a learning-based approach and treat product offer matching as a classification problem.

The overall design of our system is illustrated in Figure 10.1. The match workflow runs in three phases: pre-processing, training, and model application. The **product code extraction** is one element of preprocessing and will be described in Section 10.4 in more detail. Further preprocessing tasks are the cleaning of manufacturer information and the categorization of product offers.

For **manufacturer cleaning** we cluster different variations of the same manufacturer based on a combination of string similarities, synonym lists, and existing lists of manufacturers. Since a significant number of product offers does not provide a manufacturer attribute value, we also analyze the product title and descriptions for manufacturer names from the manufacturer dictionary. We describe our approach in more detail in Section 10.3.

An important use case for product offer matching that we consider is the assignment of product offers to a given set of product categories. Such a product **categorization** [11] allows the restriction of matching to offers from the same category thereby improving match efficiency and likely also match precision. Furthermore, it is possible to devise category-specific match strategies to take characteristics of certain product types into account. We assign offers to one of the categories by using a Modified Naive Bayes approach according to [74] and utilizing already assigned offers. To classify an offer  $o$ , the Naive Bayes Classifier calculates the posterior probability  $P(c|o)$  for all categories  $c$ . To this end, attribute values (e.g., title, description, manufacturer) of already categorized offers are tokenized and the probabilities  $P(t|c)$  are computed for all tokens  $t$ . Tokens and their probabilities are weighted based on their attribute. The algorithm is described in more detail in the following section.

The training phase for the learning-based approach requires **selection of training data**, i.e., object pairs that are annotated whether they represent a match or a non-match. The manual labeling of training data is very time consuming and is typically done offline. In our approach match and non-match pairs are generated utilizing the ratio approach as described in Section 7.4.2. This strategy considers only object pairs for labeling, for which the similarity exceeds a specified threshold. This ensures that the training is not dominated by trivial non-matching object pairs that are not useful for finding effective matcher parameters and combinations. Furthermore it



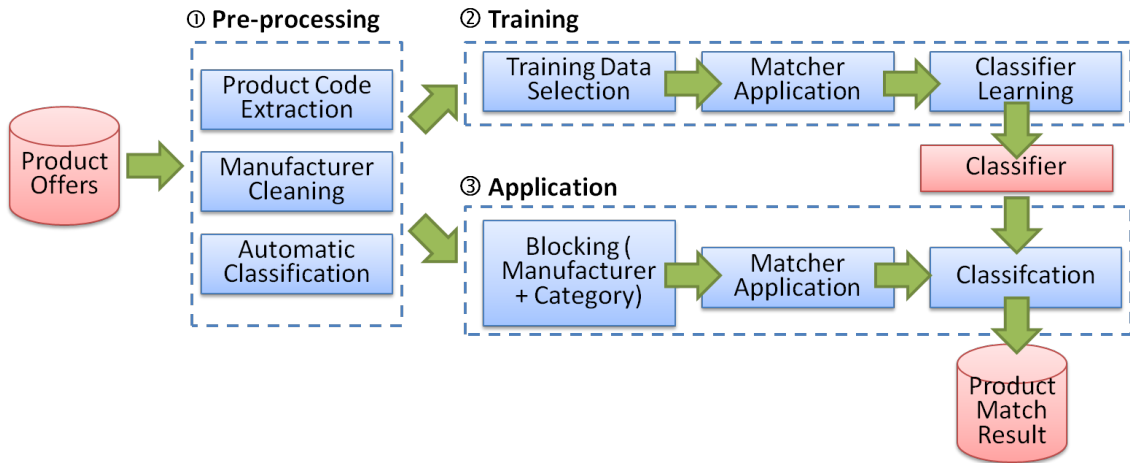


Figure 10.1: Overall workflow for matching product offers

aims at a certain ratio of matching and non-matching object pairs in the training data to enhance the training data’s discriminating value for learning effective match strategies.

As the characteristics of product data and the variety and distribution of errors across different categories can vary greatly we adopt an **adaptive learning approach** for training separate classifiers for each category. To this end we apply the learning individually for each category using disjoint subsets of training data according to the categories of the labeled training data. We also support learning a uniform model (match strategy) across all categories. In the evaluation we will compare the effectiveness of the uniform approach with category-specific match strategies. Learning is performed for several matchers, in particular for three string measures (TF/IDF, Trigram Jaccard) on the title and description attributes and a specific product code matcher (see Section 10.4). For the learner we decided on the Support Vector Machine (SVM).

During the last phase, **application**, the learned match strategies are applied to determine matching product offers. To reduce the search space we use the cleaned manufacturer value and the product category for blocking, i.e., we apply matchers only for pairs of product offers sharing the same manufacturer and category. The resulting match similarities are the input for the learned classification model that, in turn, provides the “match or non-match” decision.

## 10.2 Product offer classification

Merchant feeds may not have category information, or they may have categories under a taxonomy that is different from the one used in the catalog. To determine

the category for a given offer, we learn a categorization model utilizing a Naive Bayes approach similar to [74].

To classify an offer  $o$ , the Naïve Bayes Classifier calculates the posterior probability  $P(c|d)$  based on the Bayes Theorem and the independence assumption. Given a set of categories  $C$ , the attributes  $a_1, \dots, a_n$  and the values  $v_1, \dots, v_n$  that describe an input offer, the Naïve Bayes Classifier assigns the most probable category according to the following formula:

$$C = \arg \max_{c_j \in C} P(c_j) \prod_i P(a_i = v_i | c_j) \quad (10.1)$$

The value of an attribute is a set of terms  $v_i = \{t_{i1}, t_{i2}, \dots, t_{im}\}$ . The probability  $P(a_i = v_i | c_j)$  is calculated as follows:

$$P(A_i = v_i | c_j) = \prod_k P(t_{ik} \text{ appears in } a_i | c_j) = \prod_k \frac{n(c_j, a_i, t_{ik})}{n(c_j, a_i)} \quad (10.2)$$

where  $n(c_j, a_i, t_{ik})$  denotes the number of occurrences of  $t_{ik}$  in attribute  $a_i$  of category  $c_j$ . Similarly,  $n(c_j, A_i)$  denotes the sum of frequencies of all terms in  $a_i$  of category  $c_j$ .  $n(a_i)$  denotes the total number of terms appearing in attribute  $a_i$ .

Attributes are normalized with the geometric mean  $\sqrt[n]{t_1 \cdot \dots \cdot t_n}$

$$C_{NB} = \arg \max_{c_j \in C} P(c_j) \prod_i \left( \prod_k P(t_{ik} \text{ appears in } a_i | c_j) \right)^{\frac{1}{|v_i|}} \quad (10.3)$$

$$= \arg \max_{c_j \in C} \left\{ |c_j| \prod_i \left( \prod_k \frac{n(c_j, a_i, t_{ik})}{n(c_j, a_i)} \right)^{\frac{1}{|v_i|}} \right\} \quad (10.4)$$

Normalization enables us to weight each attribute  $a_i$  with a weight  $w_i$ . Attributes may have different effects on the classification accuracy although they are equally treated due to the independence assumption. The classification accuracy can be enhanced by giving more weights to more informative attributes. Weights have to be normalized as they are implicitly and arbitrarily weighted by the number of terms. The final classification model is based on the following formula:

$$C_{NB} = \arg \max_{c_j \in C} P(c_j) \prod_i \left( \prod_k P(t_{ik} \text{ appears in } a_i | c_j) \right)^{\frac{w_i}{|v_i|}} \quad (10.5)$$

$$= \arg \max_{c_j \in C} \left\{ |c_j| \prod_i \left( \prod_k \frac{n(c_j, a_i, t_{ik})}{n(c_j, a_i)} \right)^{\frac{w_i}{|v_i|}} \right\} \quad (10.6)$$

We illustrate the approach by means of some example offers depicted in Table 10.1. In the training phase, the string values of the attributes of the offers are tokenized

Name	Description	Manufacturer	Category
LCD SyncMaster100	High quality speaker is embedded	Samsung	LCD Monitors
Flatron LCD171	Effectively protects electromagnetic waves	LG	LCD Monitors
XNote LM50-DMP2	Compact size and slim-design	LG	Notebook Computers

Table 10.1: Example training data for categorization

and the term frequencies are counted to build the frequency table and total frequency table as illustrated in Table 10.2. Based on these two tables we can automatically assign a new product offer with product name "Flatron LCD 180" und product description "Compact size and slim design" based on formula 10.6.

$$\begin{aligned}
 P(c_A|d) &= P(c) \cdot P(a_1 = v_1|c_A) \cdot P(a_4 = v_4|c_A) \\
 &= 750 \cdot \left(\frac{1}{6} \cdot \frac{2}{6} \cdot \frac{1}{5000}\right)^{\frac{5}{3}} \cdot \left(\frac{1}{5000} \cdot \frac{1}{5000} \cdot \frac{1}{5000} \cdot \frac{1}{5000}\right)^{\frac{1}{4}} \\
 &= 750 \cdot (1.19 \cdot 10^{-19}) \cdot (2 \cdot 10^{-5}) \\
 &= 2.60 \cdot 10^{-18}
 \end{aligned}$$

$$\begin{aligned}
 P(c_B|d) &= P(c) \cdot P(a_1 = v_1|c_B) \cdot P(a_4 = v_4|c_B) \\
 &= 1100 \cdot \left(\frac{1}{5000} \cdot \frac{1}{5000} \cdot \frac{1}{5000}\right)^{\frac{5}{3}} \cdot \left(\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4}\right)^{\frac{1}{4}} \\
 &= 1100 \cdot (3.20 \cdot 10^{-24}) \cdot (2.5 \cdot 10^{-1}) \\
 &= 8.88 \cdot 10^{-22}
 \end{aligned}$$

Both  $n(a_1)$  and  $n(a_4)$  which are used for the probability estimation of the mismatching terms are set to 5000. The attribute  $a_1$ , product name, is weighted with 5 and the attribute  $a_4$ , product description, is weighted with 1. The final posterior probability of  $c_A$  is much higher although its probability from  $a_4$  is much lower than that of  $c_B$ . This means that the product is assigned to the category LCD Monitors. Without normalization and weighting, it would have been assigned to the category Notebook Computers.

Category	Attr	Term	Freq	Category	Attr	Total_Freq
LCD Monitors	1	lcd	2	LCD Monitors	1	6
LCD Monitors	1	syncmaster	1	LCD Monitors	2	4
LCD Monitors	1	100	1	LCD Monitors	3	2
LCD Monitors	1	flatron	1	LCD Monitors	4	8
LCD Monitors	1	171	1	Notebook Computers	1	3
LCD Monitors	2	cd	2	Notebook Computers	2	2
LCD Monitors	2	monitors	2	Notebook Computers	3	1
LCD Monitors	3	samsung	1	Notebook Computers	4	4
LCD Monitors	3	lg	1			
LCD Monitors	4	high	1			
LCD Monitors	4	...	...			
Notebook Computers	1	xnote	1			
Notebook Computers	1	lm50	1			
Notebook Computers	1	dmp2	1			
Notebook Computers	2	notebook	1			
Notebook Computers	2	computers	1			
Notebook Computers	3	lg	1			
Notebook Computers	4	compact	1			
Notebook Computers	4	...	...			

(a) Frequency Table

(b) Total Frequency Table

Table 10.2: Frequency table and total frequency table

## 10.3 Manufacturer cleaning

We provide several techniques to consolidate heterogeneous manufacturer denominations and complement missing values.

### 10.3.1 Automated look-up table construction with synonym maintenance

If there is a manufacturer attribute with values for at least some offers we start with those to automatically construct a look-up table with synonym maintenance. First of all, we clean values that represent missing values such as never, noname, not specified. Next, we consolidate manufacturer denominations if their values satisfy at least one of the following conditions:

1. The two strings are similar according to a similarity measure. In our approach we use normalized Levenshtein distance with a threshold of 0.75.

	Allied Telesis	Hewlett Packard	Hewlett Peckard	HP	
<b>Allied</b>	0.42	0.13	0.2	0.0	alliedtelesis.eu
<b>Allied Telesis</b>		0.13	0.2	0.0	alliedtelesis.eu
<b>Hewlett Packard</b>			<b>0.93</b>	0.13	hewlett-peckard.de
<b>Hewlett Peckard</b>				0.13	hewlett-peckard.de
<b>HP</b>					hp.com

(a) look-up table construction

ID	manufacturer	synonym
A1	Allied	Allied ; Allied Telesis
A2	Allied Telesis	Allied ; Allied Telesis
A3	Hewlett Packard	Hewlett Packard ; HewlettPackard ; HP
A4	HewlettPeckard	Hewlett Packard ; HewlettPackard ; HP
A5	HP	Hewlett Packard ; HewlettPackard ; HP

(b) look-up table

Table 10.3: Manufacturer cleaning using a look-up table

2. search engine evidence: A search engine return the same top url when queried with the manufacturer name as keyword

Tables 10.3 illustrates the approach for five manufacturer names that are eventually grouped into two clusters. Table 10.3a shows the Levenshtein distances and the search engine results for five manufacturer names. For the first two values, **Allied** and **Allied Telesis**, the search engine returned the same top domain (**alliedtelesis.eu**) and, thus, both are considered the same though their Levenshtein distance (0.42) is below the threshold (0.75). The remaining three manufacturer names (**Hewlett Packard**, **Hewlett Peckard**, **HP**) are also grouped together. The first two match because of their high string similarity; the first and the last share the same domain. From the result a look-up table as illustrated in table 10.3b is constructed.

### 10.3.2 Completion of missing values

The last phase, classification, assigns all product offers with an infrequent or missing manufacturer value to one of the existing manufacturer clusters. To this end, a manufacturer name is added to a cluster if it matches one of the cluster elements using the Levenshtein similarity and search engine results as described above. In contrast to the clustering phase no new clusters are created during the classification, i.e., if a manufacturer does not match any of the clusters its product offer is considered to have no manufacturer information for blocking.

Since a significant number of product offers does not provide a manufacturer attribute we apply two strategies to complement a value for offers with missing man-

ufacturer values. The first strategy uses the constructed look-up table to extract a possible value from the title or description of the offer. We employ an implementation of the Aho-Corasick algorithm [1]. The Aho-Corasick algorithm allows to find all matches against a dictionary in linear time independent of the number of matches or size of the dictionary. The second strategy is a heuristic based on the observation that the title of an offer quite often starts with the specification of the manufacturer. For this strategy we tokenize the title strings and extract the first token(s).

## 10.4 Product code extraction

One key observation is the frequent existence of specific product codes for certain product types that can help to differentiate similar but different products. A product code is a manufacturer-specific identifier that typically appears in the product title and description. In general, it can be any sequence consisting of alphabetic, special, and numeric characters split by an arbitrary number of white spaces. In the example of Figure 9.1 the term **HF R400** is a product code for the first three entries. A product code is under full control of the manufacturer and thus we observe very good data quality, i.e., the product code is usually correct if it is available. Unfortunately, product codes are generally not provided as a separate attribute but appear only within the product title or description.

The extraction of the product code of the offered product is non-trivial as the title and the description of the product offer contain several unstructured information. Furthermore, accessory products may also contain multiple product codes, e.g., one for the accessory itself and one for the target product. Product code extraction is a special case of product attribute extraction that identifies attribute-value pairs out of unstructured textual descriptions (e.g., [56]). However, such approaches typically require labeled (tagged) training data whereas our focused product code extraction does not need any training data but employs the rich knowledge of search engines. The product code extraction algorithm is illustrated in Algorithm 3 and will be described next. For illustration purposes Figure 10.2 demonstrates the extraction workflow for the sample accessory product **Hahnel HL-XF51 7.2V 680mAh for Sony NP-FF51** .

The first step, **feature extraction**, applies regular expressions to extract common features such as dimensions, weight specification, colors, etc. In our example the voltage (7.2V) and energy (680mAh) are extracted. The next step, **tokenization**, breaks the title string into words. Tokens are separated by white spaces and punctuations.

**Filtering** comprises the removal of stop words as well as other tokens that appear frequently in product offers of several different manufacturers. For this we calculate a

**Algorithm 3:** Product code extraction

---

```

1 getProductCode(offer, regularExpressions, threshold)
2   // remove common features, e.g., dimensions, weight, ...
3   offer ← removeFeatures(offer);
4   // tokenize and remove stop words and frequent
5   // category-specific tokens
6   tokens ← tokenize(offer);
7   tokens ← removeFrequentTokens(tokens);
8   // candidates (= up to 3 tokens) that satisfy
9   // specific patterns
10  candidates ← generateCandidates(tokens);
11  candidates ← filterCandidates(regularExpressions);
12  // get code with highest web score above threshold
13  code ← webVerification(candidates, threshold);
14  return code;

```

---

manufacturer-based frequency for each token  $t$  appearing in any offer representation. Let  $N(t, m)$  be the number of product offers of manufacturer  $m$  containing the token  $t$  and let  $N(t)$  be the overall number of product offers containing  $t$ . For any product offer of  $m$  only tokens  $t$  with a ratio  $N(t, m)/N(t)$  above a given threshold are considered for product code extraction. In our experiments we employ a threshold of 50%, i.e., at least 50% of all product offers containing  $t$  must be from manufacturer  $m$ . In the running example the term `for` will thus be excluded from further steps.

Afterwards we **generate candidates** for product codes. In general, a candidate consists of up to 3 consecutive tokens. To reduce the possibly large number of candidates regular expressions are employed to find “interesting” candidates, e.g., candidates that contain both letters and numbers. To this end we use a manually created list of regular expressions that captures knowledge on the syntactical structure of common product codes. Furthermore, string type frequencies can be computed to identify types that frequently occur with a particular manufacturer. For example, a significant number of candidates for the manufacturer **Hahnel** follows the pattern  $[A-Z]\{2\}\-[A-Z]\{2\}[0-9]\{2\}$  (“two capital letters, minus, two capital letters, two digits”) which indicates that such strings can be product codes.

Finally, a **web verification** step utilizes the web as an external knowledge source to verify the extracted candidates. For each of the determined candidates a query is submitted to a web search engine. The correctness of a code candidate is verified by the ratio of the results containing the corresponding manufacturer. Figure 10.2 illustrates for the two candidates **HL-XF51** and **NP-FF51** the retrieved top 2 query results. For the first candidate, **HL-XF51**, all results contain the manufacturer name **Hahnel** and thus giving an overlap of 100%. The term **HL-XF51** is therefore con-

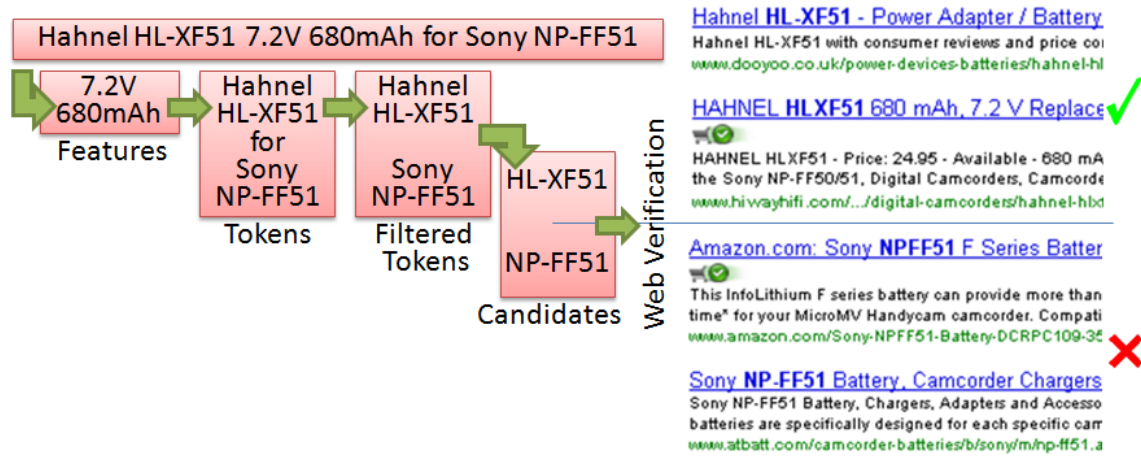


Figure 10.2: Example code extraction for Hahnel HL-XF51 7.2V 680mAh for Sony NP-FF51 (manufacturer: Hahnel).

sidered a valid product code. On the other hand, NP-FF51 is *not* a product code because none of the results contain the manufacturer name Hahnel.



# 11

## Evaluation of product offer matching

In this chapter we evaluate our approaches for product offer matching introduced in chapter 10.

The chapter is organized as follows: We first provide details on the used evaluation dataset in Section 11.1 We then evaluate in Section 11.2 our approach for product offer classification. This is followed in Section 11.3 by the evaluation of the proposed approach to derive product codes from the titles of product offers of different categories. We then analyze in Section 11.4 the effectiveness of different general and category-specific strategies for matching product offers and study the usefulness of using the extracted product codes. This evaluation is first done in Section 11.4.1 w.r.t. an UPC-based reference mapping assuming that only offers with the same UPC are matching. We then study in Section 11.4.2 the match quality w.r.t. the manually determined perfect mapping and discuss limitations of relying on UPC values only.

### 11.1 Evaluation dataset

To evaluate our product offer matching approaches we use a large real-world dataset provided by an e-commerce portal <sup>1</sup>. The evaluation dataset comprises a total of 102,182 offers for electronic products and accessory products that are associated to 71 given product categories of the portal. The offers are mostly limited to only a

---

<sup>1</sup>Because of a non-disclosure agreement we are not able to provide any details on the e-commerce portal.

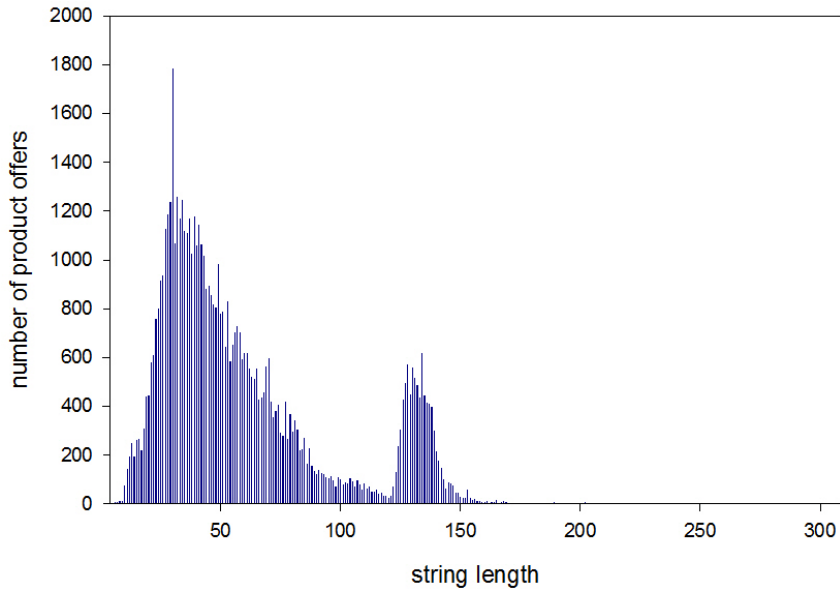


Figure 11.1: Distribution of string length for the product title attribute

few attributes, in particular product title, description, and manufacturer. For evaluation purposes, we also required that all offers contain an UPC value assuming that it permits the evaluation of the quality (recall, precision, F-measure) of different match strategies. In addition, we manually determined a perfect match mapping for two selected product categories. All product offers are preprocessed as described in chapter 10, i.e., we cleaned the manufacturer values, extracted product codes if possible, and automatically assigned offers to product categories. Since the evaluation of manufacturer cleaning and product categorization is beyond the scope of this paper, we corrected the results of these steps manually to avoid negative side effects.

Figure 11.1 shows the length distribution of the product titles for these offers. It illustrates that this important attribute is highly heterogeneous with many long, verbose strings. The puzzling second peak around string length 135 is due to numerous accessory products for digital cameras, mobile phones, and navigation systems whose titles contain a long list of models for which they are suitable. As a result, standard string measures alone are not able to achieve a sufficient match quality.

This is further confirmed by Figure 11.2 showing for two product match tasks and a publication match task the percentage of correspondences (matching object pairs) with a string (TF/IDF) similarity smaller or equal to a given similarity threshold  $t$ . For example, approx. 60% of all correspondences of the product category **Digital Cameras** have a title similarity below or equal to 0.5 making it difficult to identify matching offers. On the other hand, matching publications is much easier since 60%

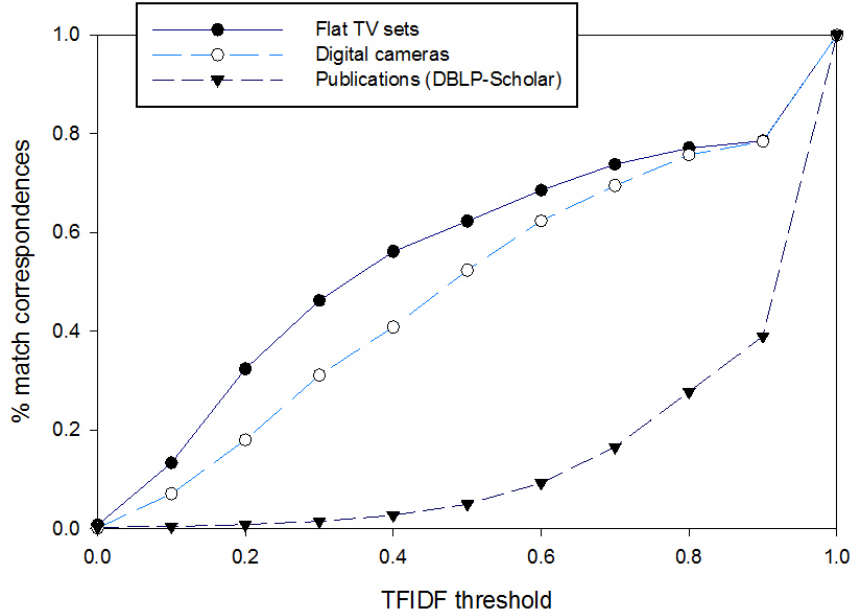


Figure 11.2: Cumulative distribution of TF/IDF similarity for match correspondences

of the correspondences have a title similarity of more than 90%.

## 11.2 Evaluation of product offer classification

We evaluated both effectiveness and efficiency of the approach for product offer classification. Effectiveness is evaluated in terms of classification accuracy (percentage of correctly classified product offers) and efficiency in terms of classification time. For training we selected a random sample of 1000 product offers from each of the 71 categories. The learned model was then applied to the remaining product offers.

Figure 11.4a illustrates the top 1 categorization accuracy for different attribute combinations and weights. The results reveal that weights have a high influence on categorization accuracy. For different weights the accuracy varies between 62% and 84%. Using only the title attribute achieves the highest accuracy of 89%. This is due to the fact that both the description and brand attribute are highly heterogeneous. Using these attributes decreases the categorization accuracy rather than increasing it.

Figure 11.4b illustrates the top  $N$  classification accuracy. Instead of selecting only the category with the highest probability we choose the  $n$  highest ranked categories according to their probability value. The top  $N$  classification accuracy is the per-

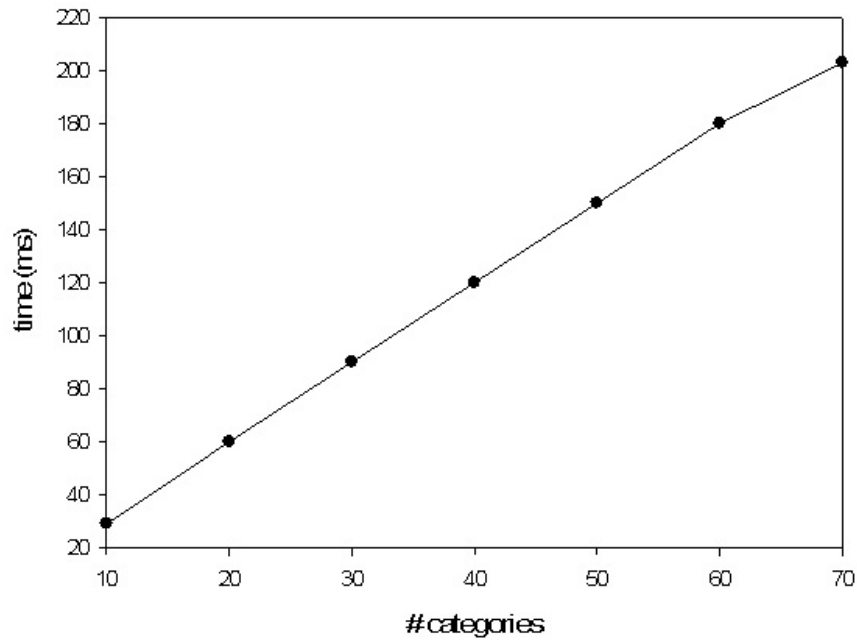
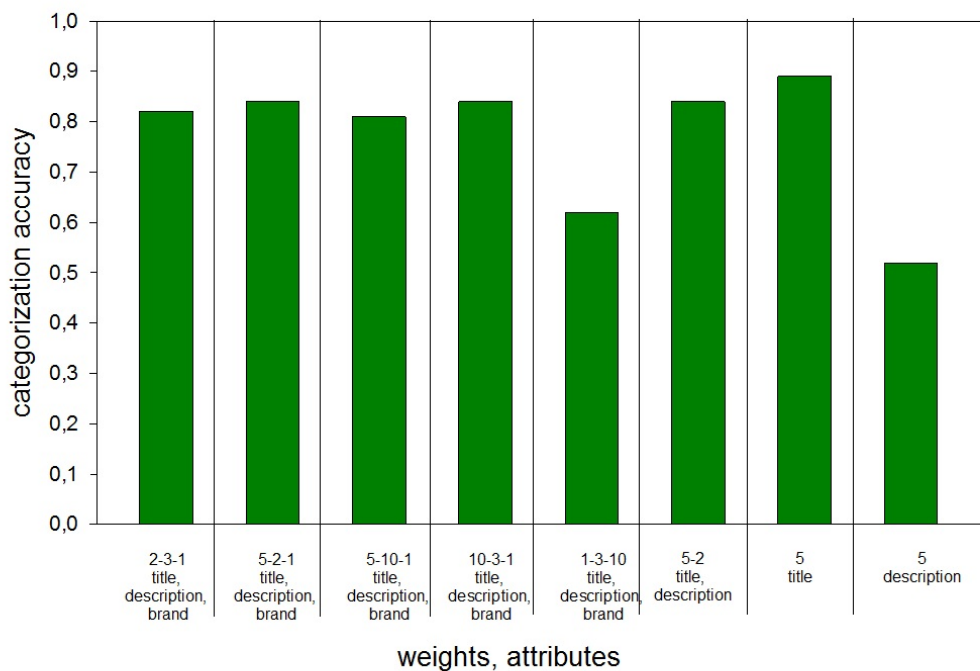


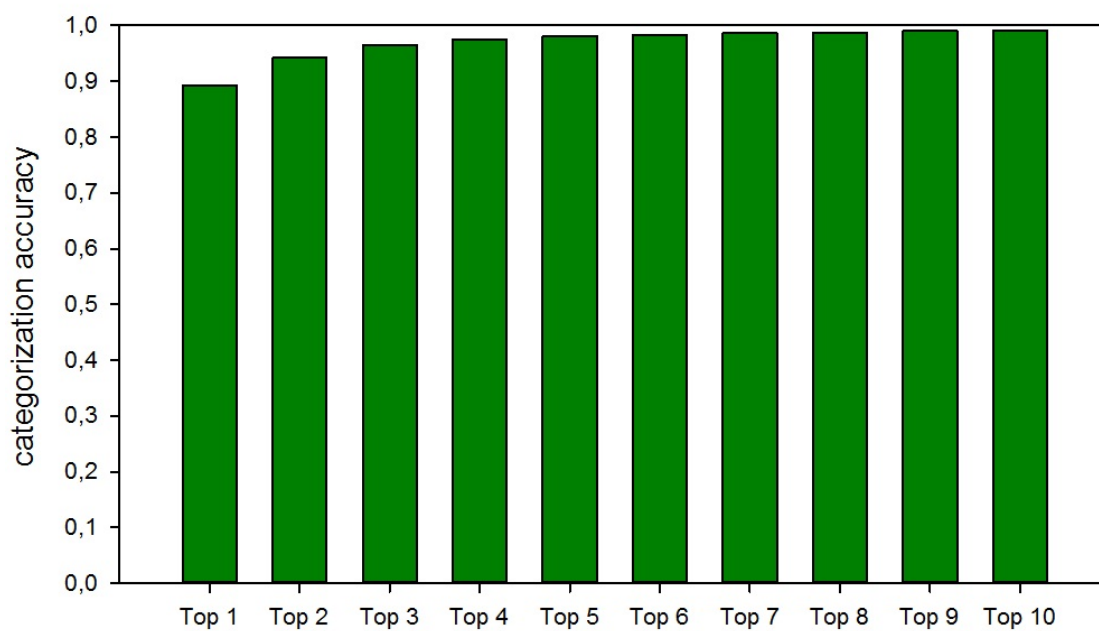
Figure 11.3: Efficiency of product offer classification

centage of times that the correct category is among the selected  $n$  categories. The results show that the right category is among the top 10 category in most cases. The categorization accuracy for top 10 is 99%. That is an improvement of 10% compared to top 1.

Figure 11.3 pictures the classification time for different numbers of categories. The figure illustrates that the classification time increases linearly with the number of target categories.



(a) Categorization accuracy for different attribute weights



(b) Top N categorization accuracy

Figure 11.4: Effectiveness of product offer classification

### 11.3 Evaluation of product code extraction

The effectiveness of the proposed approach to extract product codes depends on the product category, especially on whether the offers refer to accessory products or to the main, non-accessory products. In Figure 11.5 we show the number of product offers for the largest five accessory categories as well as the largest five non-accessory categories. Furthermore, we show for each category for how many offers a product code could be extracted by our approach. We observe significant differences between accessory and non-accessory products. For non-accessory product offers, we could mostly find a product code (in 85%, on average) in particular for the larger categories of mobile phones, cameras, and TV sets. By contrast, for the accessory product offers (which are much more frequent than non-accessory offers) a product code could be determined only in 34% on average.

To determine the quality of the extracted product codes we manually determined the correct product codes for a random subset of about 2% of the product offers over all categories.

Table 11.1 shows the quality results for product code extraction after the web-based verification. This verification proved to be an important step that helped to improve the results by up to 20% even for a low threshold value of 0.1. As shown, the average precision of the product codes is 79% over all categories, i.e., that almost four fifth of the found product codes are correct. The achieved recall values are smaller, especially for accessory products where less than half (48%) of the product codes could be found. The resulting F-measure is thus higher for non-accessory offers than for accessories. For mobile phones, product code extraction was most effective with an F-measure of 89%.

The results show that product code extraction works best for non-accessory products so that they are likely to benefit more than accessory products from using them for matching product offers.

	Precision	Recall	F-measure
Overall	79%	56%	66%
Non-Accessories	79%	64%	71%
Accessories	79%	48%	60%
Mobile Phones	93%	86%	89%

Table 11.1: Quality of product code extraction

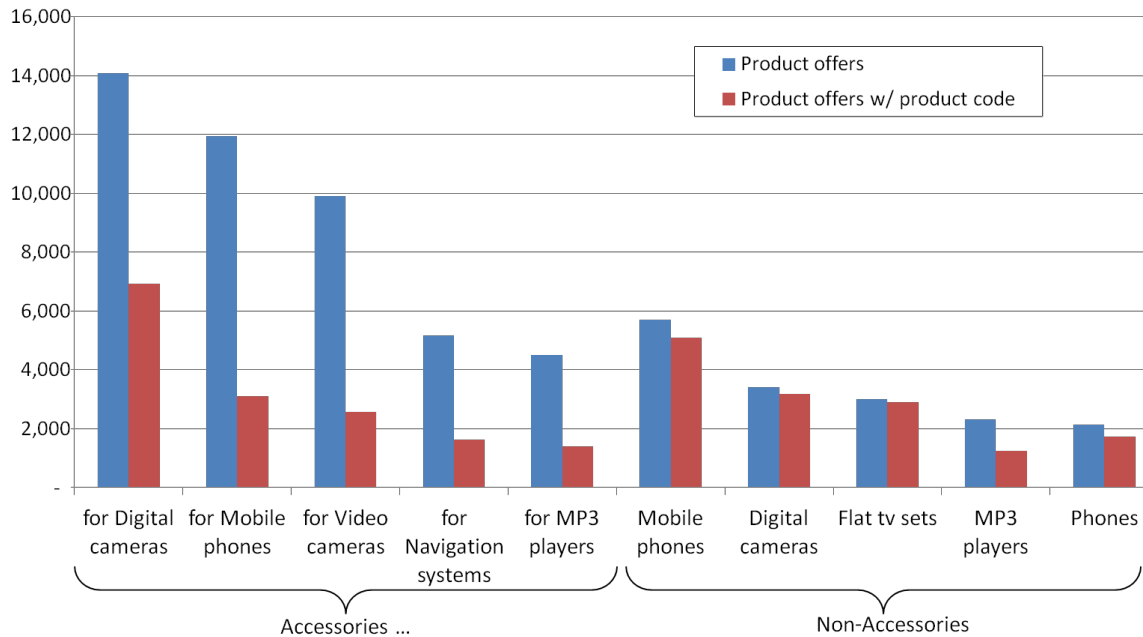


Figure 11.5: Number of product offers for 10 product categories (5 accessory and 5 non-accessory categories)

## 11.4 Evaluation of match quality

We evaluate the match quality for product offers of the five largest accessory categories and the five largest non-accessory ones. As described in Section 10.1, we apply SVM-based match strategies combining different string measures on the product title and product description. Optionally, we consider an additional product code matcher requiring equal product codes for product offers to match. For training we use 500 matching and 500 non-matching offer pairs per category respectively. The training data is used to determine both a common match strategy for all categories as well as category-specific match strategies.

### 11.4.1 Match quality against UPC mapping

We first evaluate match quality against an UPC-based reference mapping where only offers with the same UPC are considered to match (recall that we only evaluate offers for which the UPC is provided).

Figure 11.6 shows the resulting average F-measure results over the considered accessory and non-accessory categories for three match strategies: a common baseline strategy (not using product codes) for all categories, category-specific baseline match strategies, and category-specific match strategies including the product code matcher.

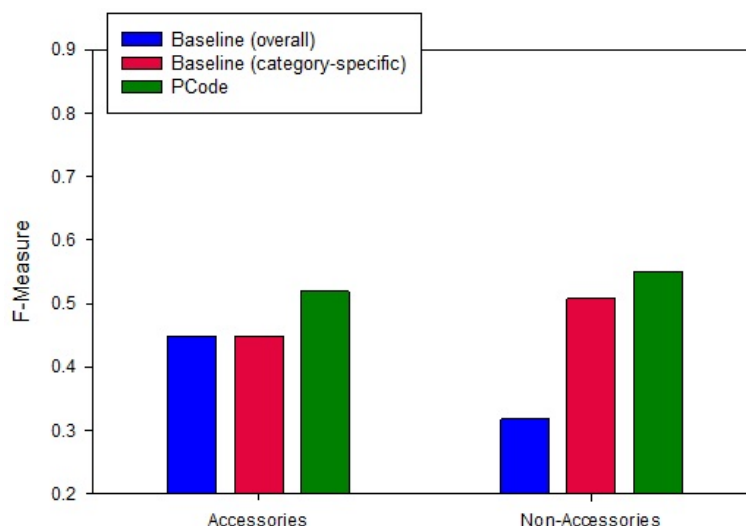


Figure 11.6: Quality of baseline and product code matching

We observe that the average F-measure values are generally rather low which is due to the high difficulty of matching offers against offers (and not against cleaned product descriptions). Furthermore, as we will see in the next subsection the comparison against the UPC reference mapping leads to pessimistic results.

We find that the use of category-specific match strategies generally outperforms the use of a common match strategy. This effect is especially pronounced for non-accessory product offers for which the average F-measure becomes higher than for accessory products. Similarly, the use of the product code matcher is most beneficial for non-accessory offers influenced by the improved coverage and quality of product code extraction as discussed above. Product code matching helps improve the average F-measure for non-accessory categories to 55%. The best match quality (F-measure 73%) is achieved for offers of mobile phones.

#### 11.4.2 Match quality against manually determined reference mapping

A closer inspection of the UPC values for the product offers revealed several anomalies that questioned the appropriateness of UPC-based match decisions. There are offers for the same or almost the same product that come with different UPC values. One of the reasons for this phenomenon is that products may come with different codes based on the manufacturer's country or target market. For example, there are three different UPCs for the Canon IXUS 90 IS camera in our dataset.

Furthermore, we observed the existence of offers for different products having the



same UPC. So we decided to manually determine a reference match mapping to evaluate the quality of the automatic match strategies. For this purpose, we employed a crowd-sourcing effort involving several local researchers. Due to the large number of offers we still had to restrict the determination of the reference match mapping to two categories (flat tv sets and digital cameras). The manual match decisions did not try to differentiate all minor variations of the same product (e.g., different colors) but had in mind what should be bundled in a comparison portal or product search engine to allow a meaningful price comparison, to utilize aggregated product reviews etc. Similar as in the example of Figure 9.1, the product code information turned out to be a useful indicator for match decisions.

Table 11.2 provides some base statistics about the manually determined and the UPC-based reference mappings.

The number of clusters indicates how many different products are distinguished per category, the cluster size indicates the average number of offers for the same product, and the number of correspondences specifies the number of matching pairs of offers. We observe that the manual mappings contain more correspondences than the UPC mappings by considering more offers to refer to the same product. The differences are especially pronounced for the camera category where the average number of offers per product more than doubles, i.e., the offers in most correspondences had different UPC values. To a much smaller extent, we observed that the same UPC value was assigned to non-matching offers with different product codes (for 90 correspondences of TV set offers and 331 pairs of camera offers).

Figure 11.7 compares the match quality (F-measure) for the two product categories w.r.t. both reference mappings.

We only consider category-specific match strategies. For the UPC-based reference mapping, the baseline match strategy (not using product codes) performs similarly for both categories. For the TV set category, product code matching helped to substantially improve F-measure already for the UPC reference mapping. Comparing against the manually determined reference mapping results in a further albeit relatively small improvement (to 69% F-measure) since the two reference mappings are relatively similar for this category. By contrast, for the camera category the UPC mapping did not enable taking much advantage from product code matching since many offers with the same product code had different UPC values. Here, using the manual reference mapping helped to almost double the F-measure result compared to the UPC mapping (to 81%) indicating the high value of product code matching.

While we could only evaluate the offers for two categories we expect similar trends for other categories. We conclude that UPC-based match evaluations tend to be too pessimistic and that UPC-based matching may leave many matching or comparable offers unmatched. The manual reference mappings also showed the high potential of the proposed product code matching.

Category	Mapping	#Clusters	Average Cluster size	#Correspondences
Flat TV sets	UPC	1,222	2.5	5,293
	manual	1,103	2.7	6,509
Digital cameras	UPC	1,087	3.1	8,375
	manual	504	6.8	32,571

Table 11.2: Reference mappings

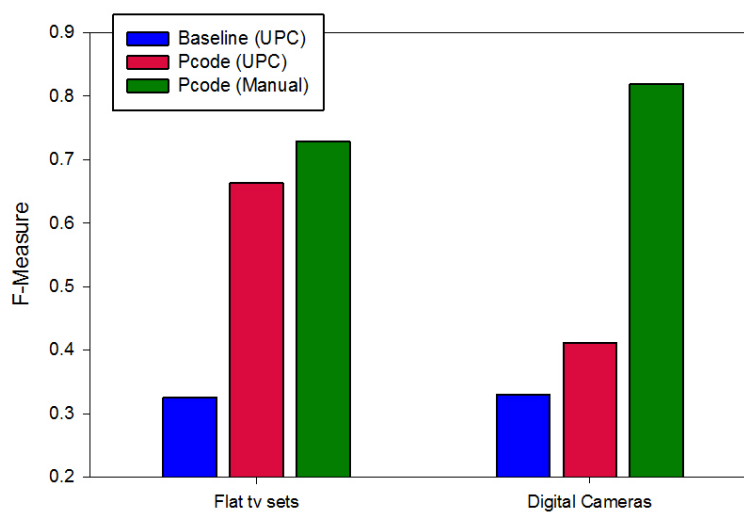


Figure 11.7: Match quality for different reference mappings

**Part V**

**Closing**



# 12

## Summary

This chapter summarizes the main contributions of the thesis.

Object matching is defined as the identification of different representations or versions of a same real-world object. Such duplicates are due to errors and inconsistencies in the data, such as typos and misspellings, missing information, or outdated data. As a consequence, duplicates are not exactly equal, which makes object matching a challenging task in data cleaning and data integration processes.

To reduce the manual effort as much as possible, semi-automatic approaches are needed to effectively assist the user in solving the match problem. This dissertation has advanced the state of the art of object matching by developing and implementing new solutions for object matching. In particular, it made four main contributions as follows:

- **Survey of object matching frameworks**

We performed a comprehensive survey of existing object matching approaches and frameworks. We analyzed and discussed various approaches for blocking, matching and the combination of several matchers. We then used the classification and evaluation criteria to comprehensively review previous prototypes and evaluations and thereby identified important issues to be addressed in the future.

- **Generic object matching framework**

With FEVER, we developed a new generic and customizable object matching system. In particular, FEVER is based on an open multi-component architecture, which offers high flexibility for extension and adaptation. FEVER

provides a comprehensive library of individual matchers and supports different mechanisms to combine and refine their results.

An important aspect of the FEVER framework is the support of training-based match approaches. FEVER includes several training-based approaches and methods to (semi-)automatically generate training data for object matching. Hence, FEVER can be used to compare non-learner and learner-based approaches for object matching. We specifically analyzed the effectiveness for small training sizes which incur only a modest effort for labeling.

The flexibility to customize the match operation allowed us to systematically evaluate the different match approaches supported by FEVER. This way we can achieve high quality and fast execution time for large real-world problems in different domains.

- **Comparative object matching evaluation**

We presented an evaluation of existing implementations on challenging real-world match tasks. We considered approaches both with and without using machine learning to find suitable parametrization and combination of similarity functions. In addition to approaches from the research community we also considered a state-of-the-art commercial object matching implementation. Our results indicate significant quality and efficiency differences between different approaches. We also find that some challenging resolution tasks such as matching product entities from online shops are not sufficiently solved with conventional approaches based on the similarity of attribute values.

- **Product offer matching**

Matching product offers is a challenging problem requiring sophisticated and tailored object matching approaches. We outlined and evaluated such an approach that is based on machine learning and a comprehensive preprocessing. In particular, we proposed a new approach for improving product offer matching based on a pattern-based extraction and web-based verification of so-called product codes. Our evaluation with a large real-life dataset showed the high benefit of product code matching, especially for non-accessory products. Furthermore, we found that category-specific match strategies should be applied to cope with the heterogeneity of the different categories. We also analyzed the use of UPC values for evaluating match strategies and for product matching and observed significant limitations. In particular, UPC-based match evaluations tend to be too pessimistic and UPC-based matching may leave many matching or comparable offers unmatched.

# 13

## Future directions

Although the thesis has made a number of contributions to improve the state of the art in object matching and object matching evaluation, it also raises several opportunities for improvement and further issues to be addressed in future work. We review some of them in the following.

- **Parallel object matching**

There exists an increasing amount of approaches that consider parallelized object matching [9, 36, 122]. In [36] the authors show how the match computation can be parallelized among available cores on a single node. Parallel evaluation of the Cartesian product of two sources is described in [122]. D-Swoosh [9] proposes different strategies for both evaluating the Cartesian product and the use of blocking. In [75] the authors propose a general model for parallel entity matching based on a balanced partitioning of the input data to create match tasks that can be evaluated in parallel.

A recent trend in parallel object matching is the use of MapReduce. MapReduce (MR) is a popular programming model for parallel processing on cloud infrastructures with up to thousands of nodes [42]. The availability of MR distributions such as Hadoop makes it attractive to investigate its use for the efficient parallelization of data-intensive tasks. MR has already been successfully applied to parallelize object matching workflows [134, 137, 80, 81, 77].

Dedoop (Deduplication with Hadoop ) [78, 79] is a tool for parallel object matching on cloud infrastructures. Dedoop supports a browser-based specification of complex object matching strategies and provides a large library of

blocking and matching approaches. To simplify the configuration of object matching strategies with several similarity metrics, training-based machine learning approaches can be employed with Dedoop. Specified object matching strategies are automatically translated into MapReduce jobs for parallel execution on different Hadoop clusters. For improved performance, Dedoop supports redundancy-free multi-pass blocking as well as advanced load balancing approaches.

Another direction in parallel object matching is to utilize the capabilities of modern graphics processing units (GPUs) to increase the efficiency of finding duplicates in very large datasets [8, 109].

- **Object matching for linked data**

The Linked Data paradigm has evolved into a powerful enabler for the transition from the document-oriented Web into the Semantic Web. While the amount of data published as Linked Data grows steadily and has surpassed 25 billion triples, less than 5% of these triples are links between knowledge bases. Thus, there is a strong need for powerful approaches for link discovery, especially for the discovery of sameAs-links. Over the last years, several link discovery approaches have been proposed [40, 65, 111, 113, 126, 136] .

Recently generic frameworks such as SILK [136] and LIMES [108] are being developed. With its modular architecture FEVER has the best possible prerequisites to incorporate methods for matching linked data.

- **Collaborative object matching**

Recently, crowdsourcing has attracted significant attention in both the industrial and academic communities (see [47] for a recent survey). Recent projects in the database community aim to embed crowdsourcing into database query processing. In [138] a first approach is proposed on how to improve object matching using hybrid human-machine techniques combining a generic micro-task crowdsourcing platform with machine-based techniques.



# Bibliography

- [1] AHO, A. V., CORASICK, M. J. Efficient string matching: An aid to bibliographic search. *Commun. ACM* 18, 6 (1975), 333–340.
- [2] AIZAWA, A. N., OYAMA, K. A fast linkage detection scheme for multi-source information integration. In *WIRI* (2005), pp. 30–39.
- [3] ANANTHAKRISHNA, R., CHAUDHURI, S., GANTI, V. Eliminating fuzzy duplicates in data warehouses. In *VLDB* (2002), pp. 586–597.
- [4] ARASU, A., GANTI, V., KAUSHIK, R. Efficient exact set-similarity joins. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)* (2006), pp. 918–929.
- [5] ARASU, A., RÉ, C., SUCIU, D. Large-scale deduplication with constraints using dedupalog. In *Proceedings of the 25th International Conference on Data Engineering (ICDE '09)* (2009), pp. 952–963.
- [6] BATINI, C., SCANNAPIECO, M. *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer, 2006.
- [7] BAXTER, R., CHRISTEN, P., CHURCHES, T. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD* (2003), vol. 3, pp. 25–27.
- [8] BENEDIKT FORCHHAMMER, THORSTEN PAPENBROCK, T. S. S. V. U. D. F. N. Duplicate detection on gpus. In *Proceedings of the 15th conference on Database Systems for Business, Technology, and Web (BTW)* (Magdeburg, Germany, 0 2013). Runner Up for Best Paper Award.
- [9] BENJELLOUN, O., GARCIA-MOLINA, H., GONG, H., KAWAI, H., LARSON, T. E., MENESTRINA, D., THAVISOMBOON, S. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *ICDCS* (2007), p. 37.
- [10] BENJELLOUN, O., GARCIA-MOLINA, H., MENESTRINA, D., SU, Q., WHANG, S. E., WIDOM, J. Swoosh: a generic approach to entity resolution. *The VLDB Journal* 18, 1 (2009), 255–276.
- [11] BERGAMASCHI, S., GUERRA, F., VINCINI, M. A data integration framework for e-commerce product classification. *ISWC* (2002).
- [12] BHATTACHARYA, I., GETOOR, L. Relational clustering for multi-type entity

## BIBLIOGRAPHY

---

- resolution. In *Proceedings of the 4th international workshop on Multi-relational mining* (New York, NY, USA, 2005), MRDM '05, ACM, pp. 3–12.
- [13] BHATTACHARYA, I., GETOOR, L. A latent dirichlet model for unsupervised entity resolution. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM '06)* (2006), pp. 47–58.
- [14] BHATTACHARYA, I., GETOOR, L. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data* 1, 1 (2007), 5.
- [15] BHATTACHARYA, I., GETOOR, L. Query-time entity resolution. *J. Artif. Intell. Res. (JAIR)* 30 (2007), 621–657.
- [16] BILENKO, M., BASU, S., SAHAMI, M. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05)* (2005), pp. 58–65.
- [17] BILENKO, M., KAMATH, B., MOONEY, R. J. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM '06)* (2006), pp. 87–96.
- [18] BILENKO, M., MOONEY, R. J. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)* (2003), pp. 39–48.
- [19] BILENKO, M., MOONEY, R. J. On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD-2003 workshop on data cleaning, record linkage, and object consolidation* (2003), pp. 7–12.
- [20] BILENKO, M., MOONEY, R. J., COHEN, W. W., RAVIKUMAR, P. D., FIENBERG, S. E. Adaptive name matching in information integration. *IEEE Intelligent Systems* 18, 5 (2003), 16–23.
- [21] BILKE, A., BLEIHOLDER, J., BÖHM, C., DRABA, K., NAUMANN, F., WEIS, M. Automatic data fusion with hummer. In *VLDB* (2005), pp. 1251–1254.
- [22] BITTON, D., DEWITT, D. J. Duplicate record elimination in large data files. *ACM Trans. Database Syst.* 8, 2 (1983), 255–265.
- [23] BOURNE, C. P., FORD, D. F. A study of methods for systematically abbreviating english words and names. *J. ACM* 8, 4 (1961), 538–552.
- [24] CARVALHO, J. C. P., DA SILVA, A. S. Finding similar identities among objects from multiple web sources. In *WIDM* (2003), pp. 90–93.
- [25] CHAUDHURI, S., CHEN, B.-C., GANTI, V., KAUSHIK, R. Example-driven design of efficient record matching queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)* (2007), pp. 327–338.

- [26] CHAUDHURI, S., GANTI, V., MOTWANI, R. Robust identification of fuzzy duplicates. In *ICDE* (2005), pp. 865–876.
- [27] CHAUDHURI, S., GANTI, V., XIN, D. Mining document collections to facilitate accurate approximate entity matching. *PVLDB* 2, 1 (2009), 395–406.
- [28] CHEN, Z., KALASHNIKOV, D. V., MEHROTRA, S. Exploiting relationships for object consolidation. In *Proceedings of the International Workshop on Information Quality in Information Systems (IQIS '05)* (2005), pp. 47–58.
- [29] CHEN, Z., KALASHNIKOV, D. V., MEHROTRA, S. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09)* (2009), pp. 207–218.
- [30] CHRISTEN, P. A comparison of personal name matching: Techniques and practical issues. In *ICDM Workshops* (2006), pp. 290–294.
- [31] CHRISTEN, P. Towards parameter-free blocking for scalable record linkage. Tech. Rep. TR-CS-07-03, Department of Computer Science, The Australian National University, Canberra, 2007.
- [32] CHRISTEN, P. Automatic training example selection for scalable unsupervised record linkage. In *Proceedings of 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '08)* (2008), pp. 511–518.
- [33] CHRISTEN, P. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the 2nd Australasian workshop on Health data and knowledge management (HDKM '08)* (Darlinghurst, Australia, Australia, 2008), Australian Computer Society, Inc., pp. 17–25.
- [34] CHRISTEN, P. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012.
- [35] CHRISTEN, P. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.* 24, 9 (2012), 1537–1555.
- [36] CHRISTEN, P., CHURCHES, T., HEGLAND, M. Febrl - a parallel open source data linkage system: <http://datamining.anu.edu.au/linkage.html>. In *PAKDD* (2004), pp. 638–647.
- [37] COHEN, W. W., KAUTZ, H. A., MCALLESTER, D. A. Hardening soft information sources. In *Proceedings of the 6th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '00)* (2000), pp. 255–259.
- [38] COHEN, W. W., RAVIKUMAR, P., FIENBERG, S. E. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb '03)* (2003), pp. 73–78.

## BIBLIOGRAPHY

---

- [39] COHEN, W. W., RICHMAN, J. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD* (2002), pp. 475–480.
- [40] CUDRÉ-MAUROUX, P., HAGHANI, P., JOST, M., ABERER, K., DE MEER, H. idmesh: graph-based disambiguation of linked data. In *WWW* (2009), pp. 591–600.
- [41] CULOTTA, A., MCCALLUM, A. Joint deduplication of multiple record types in relational data. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management (CIKM '05)* (2005), pp. 257–258.
- [42] DEAN, J., GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
- [43] DEY, D., SARKAR, S., DE, P. A distance-based approach to entity reconciliation in heterogeneous databases. *IEEE Trans. Knowl. Data Eng.* 14, 3 (2002), 567–582.
- [44] DICE, L. Measures of the amount of ecologic association between species. *Ecology* 26, 3 (1945), 297–302.
- [45] DOAN, A., LU, Y., LEE, Y., HAN, J. Object matching for information integration: A profiler-based approach. In *IIWeb* (2003), pp. 53–58.
- [46] DOAN, A., LU, Y., LEE, Y., HAN, J. Profile-based object matching for information integration. *IEEE Intelligent Systems* 18, 5 (2003), 54–59.
- [47] DOAN, A., RAMAKRISHNAN, R., HALEVY, A. Y. Crowdsourcing systems on the world-wide web. *Commun. ACM* 54, 4 (2011), 86–96.
- [48] DONG, X., HALEVY, A. Y., MADHAVAN, J. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)* (2005), pp. 85–96.
- [49] DRAISBACH, U., NAUMANN, F. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of QDB 2009 Workshop at VLDB* (2009).
- [50] DRAISBACH, U., NAUMANN, F. Dude - the duplicate detection toolkit. In *QDB 2010 Workshop* (2010).
- [51] DRAISBACH, U., NAUMANN, F. A generalization of blocking and windowing algorithms for duplicate detection. In *ICDKE* (2011), pp. 18–24.
- [52] ELFEKY, M. G., ELMAGARMID, A. K., VERYKIOS, V. S. TAILOR: A Record Linkage Tool Box. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)* (2002), pp. 17–28.
- [53] ELMAGARMID, A. K., IPEIROTIS, P. G., VERYKIOS, V. S. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.* 19, 1 (2007), 1–16.
- [54] FELLEGI, I. P., SUNTER, A. B. A theory for record linkage. *Journal of the*

- American Statistical Association* 64, 328 (1969), 1183–1210.
- [55] GALHARDAS, H., FLORESCU, D., SHASHA, D., SIMON, E. Ajax: An extensible data cleaning tool. In *Proceedings of the 2000 ACM SIGMOD international Conference on Management of Data (SIGMOD '00)* (2000), p. 590.
- [56] GHANI, R., PROBST, K., LIU, Y., KREMA, M., FANO, A. Text mining for product attribute extraction. *ACM SIGKDD Explorations Newsletter* 8, 1 (2006), 41–48.
- [57] GRAVANO, L., IPEIROTIS, P. G., JAGADISH, H. V., KOUDAS, N., MUTHUKRISHNAN, S., PIETARINEN, L., SRIVASTAVA, D. Using q-grams in a dbms for approximate string processing. *IEEE Data Eng. Bull.* 24, 4 (2001), 28–34.
- [58] GRAVANO, L., IPEIROTIS, P. G., JAGADISH, H. V., KOUDAS, N., MUTHUKRISHNAN, S., SRIVASTAVA, D. Approximate string joins in a database (almost) for free. In *VLDB* (2001), pp. 491–500.
- [59] GU, L., BAXTER, R., VICKERS, D., RAINSFORD, C. Record linkage: Current practice and future directions. Tech. rep., CSIRO Mathematical and Information Sciences, 2003.
- [60] GUHA, S., KOUDAS, N., MARATHE, A., SRIVASTAVA, D. Merging the results of approximate match operations. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04)* (2004), pp. 636–647.
- [61] HADJIELEFThERIOU, M., CHANDEL, A., KOUDAS, N., SRIVASTAVA, D. Fast indexes and algorithms for set similarity selection queries. In *Proceedings of the 24th International Conference on Data Engineering (ICDE '08)* (2008), pp. 267–276.
- [62] HARTUNG, M., GROSS, A., RAHM, E. Composition methods for link discovery. In *BTW* (2013), pp. 261–277.
- [63] HERNÁNDEZ, M. A., STOLFO, S. J. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD '95)* (1995), pp. 127–138.
- [64] HERNÁNDEZ, M. A., STOLFO, S. J. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.* 2, 1 (1998), 9–37.
- [65] HOGAN, A., POLLERES, A., UMBRICH, J., ZIMMERMANN, A. Some entities are more equal than others: statistical methods to consolidate linked data. In *Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS2010)* (2010).
- [66] JACCARD, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles* 37 (1901), 547–579.

## BIBLIOGRAPHY

---

- [67] JACCARD, P. The distribution of the flora in the alpine zone. 1. *New Phytologist* 11, 2 (1912), 37–50.
- [68] JAPKOWICZ, N., STEPHEN, S. The class imbalance problem: A systematic study. *Intell. Data Anal.* 6, 5 (2002), 429–449.
- [69] JONAS, J. Identity resolution: 23 years of practical experience and observations at scale. In *SIGMOD Conference* (2006), p. 718.
- [70] JURCZYK, P., LU, J., XIONG, L., CRAGAN, J., CORREA, A. FRIL: A tool for comparative record linkage. In *AMIA Annual Symposium Proceedings* (2008), vol. 2008, American Medical Informatics Association, p. 440.
- [71] KANG, H., GETOOR, L., SHNEIDERMAN, B., BILGIC, M., LICAMELE, L. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE Trans. Vis. Comput. Graph.* 14, 5 (2008), 999–1014.
- [72] KANNAN, A., GIVONI, I. E., AGRAWAL, R., FUXMAN, A. Matching unstructured product offers to structured product specifications. In *KDD* (2011), pp. 404–412.
- [73] KEMENTSIETSIDIS, A., ARENAS, M., MILLER, R. J. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD Conference* (2003), pp. 325–336.
- [74] KIM, Y., LEE, T., CHUN, J., LEE, S. Modified Naive Bayes Classifier for E-Catalog Classification. *Data Engineering Issues in E-Commerce and Services* (2006).
- [75] KIRSTEN, T., KOLB, L., HARTUNG, M., GROSS, A., KÖPCKE, H., RAHM, E. Data partitioning for parallel entity matching. In *QDB* (2010).
- [76] KITTLER, J., HATEF, M., DUIN, R. P. W., MATAS, J. On combining classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 3 (1998), 226–239.
- [77] KOLB, L., KÖPCKE, H., THOR, A., RAHM, E. Learning-based entity resolution with mapreduce. In *Proceedings of the third international workshop on Cloud data management* (New York, NY, USA, 2011), CloudDB '11, ACM, pp. 1–6.
- [78] KOLB, L., RAHM, E. Parallel entity resolution with dedoop. *Datenbank-Spektrum* 13, 1 (2013), 23–32.
- [79] KOLB, L., THOR, A., RAHM, E. Dedoop: Efficient deduplication with hadoop. *PVLDB* 5, 12 (2012), 1878–1881.
- [80] KOLB, L., THOR, A., RAHM, E. Load balancing for mapreduce-based entity resolution. In *ICDE* (2012), pp. 618–629.
- [81] KOLB, L., THOR, A., RAHM, E. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science - R&D* 27, 1 (2012), 45–63.
- [82] KÖPCKE, H., THOR, A., THOMAS, S., RAHM, E. Tailoring entity resolution

- for matching product offers. In *EDBT* (2012), pp. 545–550.
- [83] KÖPCKE, H., RAHM, E. Training selection for tuning entity matching. In *QDB/MUD* (2008), pp. 3–12.
- [84] KÖPCKE, H., RAHM, E. Frameworks for entity matching: A comparison. *Data Knowl. Eng.* 69, 2 (2010), 197–210.
- [85] KÖPCKE, H., THOR, A., RAHM, E. Comparative evaluation of entity resolution approaches with FEVER. *PVLDB* 2, 2 (2009), 1574–1577.
- [86] KÖPCKE, H., THOR, A., RAHM, E. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1 (2010), 484–493.
- [87] KÖPCKE, H., THOR, A., RAHM, E. Learning-based approaches for matching web data entities. *IEEE Internet Computing* 14, 4 (2010), 23–31.
- [88] LEE, M.-L., LING, T. W., LOW, W. L. Intelliclean: a knowledge-based intelligent data cleaner. In *Proceedings of the 6th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '00)* (2000), pp. 290–294.
- [89] LEITÃO, L., CALADO, P., WEIS, M. Structure-based inference of xml similarity for fuzzy duplicate detection. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM '07)* (2007), pp. 293–302.
- [90] LENGU, R., MISSIER, P., FERNANDES, A. A. A., GUERRINI, G., MESITI, M. Time-completeness trade-offs in record linkage using adaptive query processing. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT '09)* (New York, NY, USA, 2009), ACM, pp. 851–861.
- [91] LEVENSHTAIN, V. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, pp. 707–710.
- [92] LIM, E.-P., SRIVASTAVA, J. Query optimization and processing in federated database systems. In *CIKM* (1993), pp. 720–722.
- [93] MARZAL, A., VIDAL, E. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.* 15, 9 (1993), 926–932.
- [94] MCCALLUM, A., NIGAM, K., UNGAR, L. H. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2000), ACM, pp. 169–178.
- [95] MICHALOWSKI, M., THAKKAR, S., KNOBLOCK, C. A. Exploiting secondary sources for automatic object consolidation. In *Proceedings of the KDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation* (2003).
- [96] MICHALOWSKI, M., THAKKAR, S., KNOBLOCK, C. A. Exploiting secondary sources for unsupervised record linkage. In *Proceedings of the 2004 VLDB Workshop on Information Integration on the Web* (2004).

## BIBLIOGRAPHY

---

- [97] MICHALOWSKI, M., THAKKAR, S., KNOBLOCK, C. A. Automatically utilizing secondary sources to align information across sources. *AI Magazine* 26, 1 (2005), 33–44.
- [98] MICHELSON, M., KNOBLOCK, C. A. Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI '06)* (2006).
- [99] MIERSWA, I., WURST, M., KLINKENBERG, R., SCHOLZ, M., EULER, T. Yale: rapid prototyping for complex data mining tasks. In *KDD* (2006), pp. 935–940.
- [100] MINTON, S., NANJO, C., KNOBLOCK, C. A., MICHALOWSKI, M., MICHELSON, M. A heterogeneous field matching method for record linkage. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05)* (2005), pp. 314–321.
- [101] MITCHELL, T. M. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.
- [102] MONGE, A. E., ELKAN, C. The field matching problem: Algorithms and applications. In *KDD* (1996), pp. 267–270.
- [103] NAUMANN, F., HERSCHEL, M. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.
- [104] NAUMANN, F., ROTH, M. Information quality. In *Database Technologies: Concepts, Methodologies, Tools, and Applications*. 2009, pp. 2140–2156.
- [105] NAVARRO, G. A guided tour to approximate string matching. *ACM Comput. Surv.* 33, 1 (2001), 31–88.
- [106] NEILING, M., JURK, S., J. LENZ, H., NAUMANN, F. Object identification quality. In *Proceedings of the International Workshop on Data Quality in Cooperative Information Systems (DQCIS '03)* (2003), pp. 187–198.
- [107] NEWCOMBE, H. B., KENNEDY, J. M., AXFORD, S. J., JAMES, A. P. Automatic linkage of vital records. *Science* 130 (1959), 954–959.
- [108] NGOMO, A.-C. N., AUER, S. Limes a time-efficient approach for large-scale link discovery on the web of data. In *IJCAI* (2011), pp. 2312–2317.
- [109] NGOMO, A.-C. N., KOLB, L., HEINO, N., HARTUNG, M., AUER, S., RAHM, E. When to Reach for the Cloud: Using Parallel Hardware for Link Discovery. In *ESWC* (2013), pp. 275–289.
- [110] NGUYEN, H., FUXMAN, A., PAPANIZOS, S., FREIRE, J., AGRAWAL, R. Synthesizing products for online catalogs. *PVLDB* 4, 7 (2011).
- [111] NIKOLOV, A., UREN, V. S., MOTTA, E., ROECK, A. N. D. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *ASWC* (2009), pp. 332–346.



- [112] OOMMEN, B. J. Constrained string editing. *Inf. Sci.* 40, 3 (1986), 267–284.
- [113] PAPADAKIS, G., IOANNOU, E., NIEDERÉE, C., PALPANASZ, T., NEJDL, W. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL* (2011).
- [114] PASULA, H., MARTHI, B., MILCH, B., RUSSELL, S. J., SHPITSER, I. Identity uncertainty and citation matching. In *NIPS* (2002), pp. 1401–1408.
- [115] PINHEIRO, J. C., SUN, D. X. Methods for linking and mining massive heterogeneous databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98)*, (1998), pp. 309–313.
- [116] RAHM, E. Towards large-scale schema and ontology matching. In *Schema Matching and Mapping*. 2011, pp. 3–27.
- [117] RAHM, E., DO, H. H. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.
- [118] RAMAN, V., HELLERSTEIN, J. M. Potter’s wheel: An interactive data cleaning system. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB '01)* (2001), pp. 381–390.
- [119] SARAWAGI, S., BHAMIDIPATY, A. Interactive deduplication using active learning. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)* (2002), pp. 269–278.
- [120] SARAWAGI, S., KIRPAL, A. Efficient set joins on similarity predicates. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)* (2004), pp. 743–754.
- [121] SHEN, W., LI, X., DOAN, A. Constraint-based entity matching. In *AAAI* (2005), pp. 862–867.
- [122] SIK KIM, H., LEE, D. Parallel linkage. In *CIKM* (2007), pp. 283–292.
- [123] SINGLA, P., DOMINGOS, P. Multi-relational record linkage. In *Proceedings of the KDD-2004 Workshop on Multi-Relational Data Mining* (2004), pp. 31–48.
- [124] SINGLA, P., DOMINGOS, P. Object identification with attribute-mediated dependences. In *PKDD* (2005), pp. 297–308.
- [125] SINGLA, P., DOMINGOS, P. Entity resolution with markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM '06)* (2006), pp. 572–582.
- [126] SLEEMAN, J., FININ, T. Computing foaf co-reference relations with rules and machine learning. In *Proceedings of the Third International Workshop on Social Data on the Web* (2010).
- [127] SUTINEN, E., TARHIO, J. On using q-gram locations in approximate string matching. In *ESA* (1995), pp. 327–340.

## BIBLIOGRAPHY

---

- [128] TEJADA, S., KNOBLOCK, C. A., MINTON, S. Learning object identification rules for information integration. *Inf. Syst.* 26, 8 (2001), 607–633.
- [129] TEJADA, S., KNOBLOCK, C. A., MINTON, S. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)* (2002), pp. 350–359.
- [130] THOR, A., RAHM, E. MOMA - a Mapping-based Object Matching System. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR '07)* (2007), pp. 247–258.
- [131] UKKONEN, E. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.* 92, 1 (1992), 191–211.
- [132] ULLMANN, J. R. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *Comput. J.* 20, 2 (1977), 141–147.
- [133] VAN HALTEREN, H., ZAVREL, J., DAELEMANS, W. Improving accuracy in nlp through combination of machine learning systems. *Computational Linguistics* 27, 2 (2001), 199–229.
- [134] VERNICA, R., CAREY, M. J., LI, C. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD Conference* (2010), pp. 495–506.
- [135] VERYKIOS, V. S., MOUSTAKIDES, G. V., ELFEKY, M. G. A bayesian decision model for cost optimal record matching. *VLDB J.* 12, 1 (2003), 28–40.
- [136] VOLZ, J., BIZER, C., GAEDKE, M., KOBILAROV, G. Discovering and maintaining links on the web of data. In *International Semantic Web Conference* (2009), pp. 650–665.
- [137] WANG, C., WANG, J., LIN, X., WANG, W., WANG, H., LI, H., TIAN, W., XU, J., LI, R. Mapdupreducer: detecting near duplicates over massive datasets. In *SIGMOD Conference* (2010), pp. 1119–1122.
- [138] WANG, J., KRASKA, T., FRANKLIN, M. J., FENG, J. Crowder: Crowdsourcing entity resolution. *PVLDB* 5, 10 (2012), 1483–1494.
- [139] WEIS, M., NAUMANN, F. Dogmatix tracks down duplicates in xml. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)* (2005), pp. 431–442.
- [140] WEIS, M., NAUMANN, F., BROSZY, F. A duplicate detection benchmark for xml (and relational) data. In *SIGMOD 2006 Workshop on Information Quality for Information Systems (IQIS '06)* (2006).
- [141] WHANG, S. E., MENESTRINA, D., KOUTRIKA, G., THEOBALD, M., GARCIA-MOLINA, H. Entity resolution with iterative blocking. In *SIGMOD*

- '09: *Proceedings of the 35th SIGMOD international conference on Management of data* (New York, NY, USA, 2009), ACM, pp. 219–232.
- [142] WINKLER, W. E. Overview of record linkage and current research directions. Tech. rep., US Bureau of the Census, Washington, D.C., 2006.
- [143] WITTEN, I. H., MOFFAT, A., BELL, T. C. *Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition*. Morgan Kaufmann, 1999.
- [144] WOLPERT, D. H. Stacked generalization. *Neural Networks* 5, 2 (1992), 241–259.
- [145] XIAO, C., WANG, W., LIN, X. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB* 1, 1 (2008), 933–944.
- [146] XIAO, C., WANG, W., LIN, X., YU, J. X. Efficient similarity joins for near duplicate detection. In *WWW* (2008), pp. 131–140.
- [147] YAN, S., LEE, D., KAN, M.-Y., GILES, C. L. Adaptive sorted neighborhood methods for efficient record linkage. In *JCDL* (2007), pp. 185–194.
- [148] ZHAO, H., RAM, S. Entity identification for heterogeneous database integration—a multiple classifier system approach and empirical evaluation. *Inf. Syst.* 30, 2 (2005), 119–132.
- [149] ZHAO, H., RAM, S. Entity matching across heterogeneous data sources: An approach based on constrained cascade generalization. *Data & Knowledge Engineering* 66, 3 (2008), 368 – 381.



## List of Figures

3.1	Example objects with manufacturer values and the first two consonants as BKVs, and the corresponding inverted index data structure as used for traditional blocking . . . . .	32
3.2	Example execution of sorted neighborhood with window size $w = 3$ adapted from [81] . . . . .	33
3.3	Example execution of q-gram based indexing with $q = 2$ and $t = 0.8$ .	34
4.1	Example for the application of the neighborhood matcher . . . . .	47
5.1	Example of learning-based matcher combination with decision tree . .	51
5.2	Workflow-based combination following [116] . . . . .	54
7.1	Architecture of FEVER . . . . .	76
7.2	Fever match workflows. We can see (a) the operator tree and (b) its relevant configuration specification. . . . .	89
7.3	Graphical representation of an operator tree (left) incl. parameters for selected similarity join operator (right) . . . . .	90
7.4	Workflow specification in FEVER . . . . .	93
7.5	Result inspection in FEVER . . . . .	94
8.1	Duplicate paper entities in Google Scholar . . . . .	99
8.2	Match accuracy of manually configured single-attribute matcher for DBLP-ACM and DBLP-Scholar . . . . .	102
8.3	Comparison of non-learning match workflows for DBLP-ACM and DBLP-Scholar using Trigram . . . . .	104
8.4	Match accuracy for a state-of-the-art approach (COSY) with default and tuned configurations . . . . .	106

## LIST OF FIGURES

---

8.5	Comparison of Random and Ratio training selection using SVM with 8 matchers and 50 training pairs . . . . .	108
8.6	Comparison of Random and Ratio training selection using SVM with 8 matchers and 500 training pairs . . . . .	109
8.7	Comparison of different learners . . . . .	111
8.8	Evaluation of different matcher selections . . . . .	113
8.9	FEVER match workflows for evaluating existing object matching approaches . . . . .	115
8.10	Performance results for simple non-learning approaches (1 attribute) .	118
8.11	Performance results for non-learning combination approaches 2 attributes) . . . . .	119
8.12	Evaluation results for learning-based approaches with SVM from FEBRL . . . . .	122
8.13	Evaluation results for learning-based approaches with decision tree from MARLIN . . . . .	123
8.14	Evaluation results for learning-based approaches with SVM from MARLIN . . . . .	124
8.15	Effectiveness comparison of FEVER with best competitive learning-based approach on two attributes . . . . .	128
9.1	Product offers related to the Canon VIXIA HF camcorder in Google Product Search . . . . .	133
10.1	Overall workflow for matching product offers . . . . .	137
10.2	Example code extraction for Hahnel HL-XF51 7.2V 680mAh for Sony NP-FF51 (manufacturer: Hahnel). . . . .	144
11.1	Distribution of string length for the product title attribute . . . . .	146
11.2	Cumulative distribution of TF/IDF similarity for match correspondences . . . . .	147
11.3	Efficiency of product offer classification . . . . .	148
11.4	Effectiveness of product offer classification . . . . .	149
11.5	Number of product offers for 10 product categories (5 accessory and 5 non-accessory categories) . . . . .	151
11.6	Quality of baseline and product code matching . . . . .	152

11.7 Match quality for different reference mappings . . . . . 154





## List of Tables

2.1	Single source object matching problem for objects representing product offers . . . . .	22
2.2	Double source object matching problem for two bibliographic sources	24
3.1	Similarity matrix for Canopy Clustering example . . . . .	36
4.1	Example computation of TFIDF . . . . .	44
4.2	Example execution of compose operator . . . . .	47
6.1	Overview of frameworks without training (- means not present, ? means not clear from publication) . . . . .	63
6.2	Overview of learning-based and hybrid frameworks (- means not present, ? means not clear from publication) . . . . .	64
6.3	Overview of evaluations for frameworks without learning . . . . .	67
6.4	Overview of evaluations for learning-based and hybrid frameworks . .	72
7.1	Overview over FEVER operators . . . . .	80
8.1	Overview of real-world evaluation match tasks . . . . .	98
8.2	Execution times for non-learning matchers for DBLP-Scholar . . . . .	105
8.3	Execution times (in seconds) for non-learning approaches . . . . .	120
8.4	Execution times (in seconds) for non-learning approaches . . . . .	126
8.5	Summary of evaluation results (F-measure in %, top values are underlined) DBLP-ACM . . . . .	127
10.1	Example training data for categorization . . . . .	139
10.2	Frequency table and total frequency table . . . . .	140

## LIST OF TABLES

---

10.3	Manufacturer cleaning using a look-up table . . . . .	141
11.1	Quality of product code extraction . . . . .	150
11.2	Reference mappings . . . . .	154