

Learning Expressive Linkage Rules for Entity Matching using Genetic Programming

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Diplom-Informatiker Univ. Robert Isele
aus Waldshut-Tiengen

Mannheim, 2013

Dekan: Professor Dr. Heinz Jürgen Müller, Universität Mannheim
Referent: Professor Dr. Christian Bizer, Universität Mannheim
Korreferent: Professor Dr. Heiner Stuckenschmidt, Universität Mannheim
Tag der mündlichen Prüfung: 10. Juni 2013

Abstract

A central problem in data integration and data cleansing is to identify pairs of entities in data sets that describe the same real-world object. Many existing methods for matching entities rely on explicit linkage rules, which specify how two entities are compared for equivalence. Unfortunately, writing accurate linkage rules by hand is a non-trivial problem that requires detailed knowledge of the involved data sets. Another important issue is the efficient execution of linkage rules.

In this thesis, we propose a set of novel methods that cover the complete entity matching workflow from the generation of linkage rules using genetic programming algorithms to their efficient execution on distributed systems. First, we propose a supervised learning algorithm that is capable of generating linkage rules from a gold standard consisting of set of entity pairs that have been labeled as duplicates or non-duplicates. We show that the introduced algorithm outperforms previously proposed entity matching approaches including the state-of-the-art genetic programming approach by de Carvalho et al. and is capable of learning linkage rules that achieve a similar accuracy than the human written rule for the same problem.

In order to also cover use cases for which no gold standard is available, we propose a complementary active learning algorithm that generates a gold standard interactively by asking the user to confirm or decline the equivalence of a small number of entity pairs. In the experimental evaluation, labeling at most 50 link candidates was necessary in order to match the performance that is achieved by the supervised GenLink algorithm on the entire gold standard.

Finally, we propose an efficient execution workflow that can be run on cluster of multiple machines. The execution workflow employs a novel multidimensional indexing method that allows the efficient execution of learned linkage rules by reducing the number of required comparisons significantly.

Zusammenfassung

Ein zentrales Problem der Datenintegration und Datenbereinigung ist das Finden von Paaren von Entitäten in Datensets, welche das gleiche Realwelt-Objekt beschreiben. Viele bestehende Methoden für das Identifizieren solcher Paare basieren auf domänenspezifischen Verknüpfungsregeln, die festlegen, wie zwei Entitäten auf Äquivalenz verglichen werden. Allerdings ist das Schreiben solcher Verknüpfungsregeln von Hand in der Praxis aufwendig und erfordert eine detaillierte Kenntnis der verwendeten Datensets. Ein weiteres wichtiges Problem stellt das effiziente Ausführen der generierten Verknüpfungsregeln dar.

Die vorliegende Arbeit führt neuartige Algorithmen ein, die den gesamten Entity Matching Prozess abdecken, von der automatischen Erzeugung von effektiven Verknüpfungsregeln bis zu deren effizienten Ausführung. Zuerst wird ein genetischer Algorithmus eingeführt, der Verknüpfungsregeln aus einem Goldstandard lernt, welcher aus einer Menge von Paaren von Entitäten besteht, worin jedes Paar als Duplikat oder Nicht-Duplikat gekennzeichnet ist. Die experimentelle Evaluierung zeigt, dass der eingeführte Algorithmus eine höhere Genauigkeit als bestehende Verfahren einschließlich eines kürzlich eingeführten genetischen Algorithmus erreicht. Außerdem erreichten die gelernten Verknüpfungsregeln eine vergleichbare Genauigkeit als Regeln die für das gleiche Datenset von Hand geschrieben wurden.

Um auch Anwendungsfälle abzudecken, in denen kein Goldstandard verfügbar ist, wird anschließend ein komplementärer Active Learning Algorithmus eingeführt, welcher Verknüpfungsregeln interaktiv lernt indem er dem Nutzer eine kleine Anzahl von Beispielpaaren präsentiert, welcher dieser als Duplikat oder Nicht-Duplikat kennzeichnet. In den Experimenten erreichte der vorgestellte Algorithmus, nach dem der Nutzer maximal 50 Kandidaten manuell gekennzeichnet hat, eine vergleichbare Genauigkeit als der überwachte Algorithmus auf dem gesamten Goldstandard.

Abschließend führt die vorliegende Arbeit einen Algorithmus zur Ausführung der gelernten Verknüpfungsregeln auf verteilten Architekturen ein. Der eingeführte Algorithmus nutzt ein neuartiges Indexierungsverfahren, welches die Anzahl der notwendigen Vergleiche signifikant reduziert.

Contents

Abstract	i
1 Introduction	1
1.1 Use Cases	6
1.1.1 Data Integration	6
1.1.2 Matching Census Data	7
1.1.3 Bibliographic Databases	8
1.1.4 Publishing Linked Data	10
1.2 Contributions	15
1.3 Thesis Outline	17
1.4 Published Work	22
2 Linkage Rules	23
2.1 Problem Definition	24
2.2 Data Preparation	25
2.2.1 Standardization	26
2.2.2 Stop-Word Removal	27
2.2.3 Structural Transformations	27
2.3 Field Matching	29
2.3.1 Character-Based Measures	30
2.3.2 Token-Based Measures	32
2.3.3 Hybrid Measures	34
2.3.4 Other Measures	35
2.4 Previous Work on Linkage Rules.	38
2.4.1 Linear Classifiers	38
2.4.2 Threshold-based Boolean Classifiers	38
2.4.3 Other Representations	39
2.5 An Expressive Linkage Rule Representation	40
2.5.1 Example	42
2.5.2 Semantics	43
2.5.3 Discussion	45

2.5.4	Representing Common Classifiers	47
2.6	Summary	48
3	Supervised Learning of Linkage Rules	49
3.1	Problem Definition	50
3.2	Genetic Programming	51
3.2.1	Generating the Initial Population	51
3.2.2	Evolving the Population	52
3.2.3	Bloating Control	55
3.3	The GenLink Algorithm	55
3.3.1	Generating the Initial Population	57
3.3.2	Evolving the Population	59
3.3.3	Crossover Operators	60
3.3.4	Bloating Control	68
3.4	Previous Work on Supervised Learning	68
3.4.1	Linear Classifiers	69
3.4.2	Threshold-based Boolean Classifiers	71
3.4.3	Genetic Programming	74
3.4.4	Collective Approaches	75
3.4.5	Unsupervised Approaches	77
3.4.6	Discussion	78
3.5	Evaluation and Discussion	80
3.5.1	Data Sets	81
3.5.2	Experimental Setup	84
3.5.3	Evaluation Measures	85
3.5.4	Overall Results	86
3.5.5	Detailed Evaluation	94
3.6	Summary	99
4	Active Learning of Linkage Rules	103
4.1	Active Learning	104
4.1.1	Query Strategies	106
4.1.2	Uncertainty Sampling	106
4.1.3	Query-by-Committee	107
4.2	The ActiveGenLink Algorithm	109
4.2.1	Query Strategy	111
4.2.2	Building the Unlabeled Pool	115
4.3	Previous Work on Active Learning	116
4.3.1	Linear and Threshold-based Boolean Classifiers	116
4.3.2	Genetic Programming	118
4.3.3	Discussion	119

4.4	Evaluation and Discussion	122
4.4.1	Experiment Setup	122
4.4.2	Comparison with Supervised Learning	123
4.4.3	Scalability	127
4.4.4	Comparison of Different Query Strategies	131
4.5	Summary	132
5	Execution of Linkage Rules	135
5.1	Scalability Challenges	136
5.1.1	Quadratic Execution Time	136
5.1.2	Parallel Execution	136
5.1.3	Memory Constraints	137
5.2	Execution Data Flow	137
5.2.1	Indexing	137
5.2.2	Caching	139
5.2.3	Generating Comparison Pairs	140
5.2.4	Matching	140
5.2.5	Filtering	140
5.3	The MultiBlock Indexing Approach	141
5.3.1	Data Flow	142
5.3.2	Indexing Distance Measures	145
5.3.3	Indexing Aggregations	148
5.4	Previous Work on Indexing	149
5.4.1	Blocking	149
5.4.2	Sorted Neighborhood	150
5.4.3	Sorted Blocks	151
5.4.4	Q-Gram Indexing	151
5.4.5	Canopy Clustering	151
5.4.6	Metric Embedding Methods	152
5.4.7	StringMap	153
5.4.8	Discussion	155
5.5	Distributed Execution of Linkage Rules	156
5.5.1	Cluster Programming Models	156
5.5.2	MapReduce Data Flow	159
5.6	Evaluation and Discussion	163
5.6.1	Experiment Setup	163
5.6.2	Comparison with Other Methods	164
5.6.3	Scalability	167
5.6.4	Effectiveness	169
5.6.5	MapReduce	170
5.7	Summary	171

6	The Silk Link Discovery Framework	173
6.1	Silk Link Specification Language	174
6.2	Silk Workbench	177
6.2.1	Workspace Browser	177
6.2.2	Linkage Rule Learner	179
6.2.3	Linkage Rule Editor	181
6.2.4	Reference Links Manager	181
7	Conclusion	185
7.1	Summary	185
7.1.1	Lowering the Effort	185
7.1.2	Increasing the Accuracy	186
7.1.3	Increasing the Efficiency	187
7.2	Limitations and Future Work	188
7.2.1	Collective Matching	188
7.2.2	Distance Measures and Transformations	190
7.2.3	Learning of Functions Parameters	190
7.2.4	Population Seeding	190
7.2.5	Query Strategy	191
	List of Figures	193
	List of Tables	195
	Listings	197
	Bibliography	199

Chapter 1

Introduction

As companies are generating an ever increasing amount of data and more and more structured data is becoming available on the public Web, methods to integrate large data sources are moving into focus. An apparent issue in these data integration efforts is that data sources may use different representations for the same real-world object. For instance, different publication databases may contain citations to the same publication or different scientific databases may contain information about the same drugs. Identifying such duplicated entries constitutes an important issue in data integration. The process of identifying duplicates has been well studied in various research areas and is known as entity matching, instance matching, data matching, record linkage, coreference resolution, deduplication, duplicate record detection and merge/purge [Elmagarmid et al., 2007; Christen, 2012; Gu et al., 2003; Winkler, 1995; Hernández and Stolfo, 1995].

Table 1.1 illustrates the need for entity matching on an example database. In this example, the first two rows correspond to the same movie, even if

Title	Release Date	Director	Runtime
Frankenstein	July 20, 1958	Howard W. Koch	83 min
Frankenstein	1958	Howard Koch	83 min
Frankenstein	Nov. 21, 1931	James Whale	71 min
The Revenge of Frankenstein	June 1, 1958	Terence Fisher	89 min

Table 1.1: Excerpt of a movie database.

the dates and director names differ slightly. On the other hand, although the title of the third movie matches the titles of the previous movies, its distinct release date and director suggest that, in fact, it identifies a different movie than the previous ones. The last movie also identifies a distinct movie,

although its release data is very close to the release dates of the first two movies.

The goal of an entity matching task is to identify entities in data sets that denote the same real-world object [Köpcke and Rahm, 2010]. Entity matching is concerned with both identifying duplicates within a single data set (intra-source duplicates) as well as finding entities in multiple data sets that denote the same real-world object (inter-source duplicates) [Naumann and Herschel, 2010]. For each found pair of duplicates, the entity matching tasks generates a *link* that connects both matching entities.

A variety of approaches for solving the entity matching problem have been proposed in literature [Elmagarmid et al., 2007; Naumann and Herschel, 2010; Christen, 2012]. In the context of this thesis, we focus on rule-based approaches that employ domain-specific *linkage rules* [Winkler, 1995] that specify the conditions that two entities must fulfill in order to be considered a duplicate. For this purpose, linkage rules typically determine the overall similarity of two entities by comparing individual properties of both entities. For instance, in the above example of a movie database, a linkage rule may compare the titles and the release date of two entities in order to determine if both entities refer to the identical movie. As data sets may contain typographical errors or other kind of data heterogenities, properties of duplicated entities are not necessarily exact matches and thus need to be compared by using a distance measure. A distance measure determines the difference between two values based on a heuristic, such as, for instance, the number of characters that need to be changed in the first string value in order to transform it into the second string value. In addition, a linkage rule needs to define a distance threshold for each property that is compared that specifies the allowed maximum difference between two values. For instance, when comparing two release dates, a threshold could allow for a difference of 10 days for values of that particular property. Based on the distance of two properties and the corresponding distance threshold, a linkage rule is able to compute a similarity score for each property. The resulting similarity scores of all compared properties are then combined into a single similarity score. Different methods for aggregating multiple similarity scores can be used for that purpose, such as the weighted average, which, after assigning weights to each property that is compared, combines all values linearly. Furthermore, if the data sources use different data formats, property values need to be normalized by applying data transformations prior to the comparison.

The specific linkage rule that is needed for a particular entity matching task strongly depends on the data sources that are to be matched and the specific use case. Different data sources may use different data formats or can be structured differently, e.g., use different terms to express the same in-

formation. In addition, the notion of what is considered a duplicate depends on the concrete use case. For instance, a specific movie database may want to distinguish between different editions of the same movie (e.g., theatrical cut vs. director’s cut) in which case two entities that describe different editions of a specific movie should not be identified as duplicates, while other movie databases do not make that distinction.

Figure 1.1 depicts the overall workflow of a rule-based entity matching task. In the following, we describe each step in more detail and highlight the

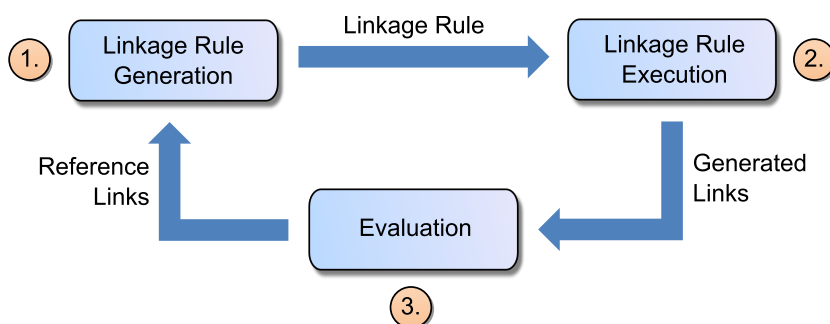


Figure 1.1: Entity Matching Workflow.

challenges that occur.

Linkage Rule Generation: Before executing the actual matching, a linkage rule needs to be built, which specifies how two entities are compared for equivalence. Linkage Rules can be created by a human expert based on the data sources that are to be matched. Writing good linkage rules by hand is a non-trivial problem as the rule author needs to have detailed knowledge about the structure of the data sets. We illustrate this point with the example of a data set about movies. Even within this simple example the linkage rule author faces a couple of challenges: First of all, a comparison solely by label fails for cases when movies with the same title actually represent distinct movies that have been released in different years. Therefore, the linkage rule needs to compare, at the very least, the titles of the movies as well as their release dates and combine both similarities with an appropriate aggregation function. As data sets can be noisy (e.g., the release dates might be off by a couple of days), the rule author also needs to choose suitable distance measures together with appropriate distance thresholds. The discovery of all data heterogeneities that are not covered by the linkage rule yet, is often not obvious and strongly dependent on the concrete data sets. For instance, a data source may contain some names that

are formatted as ”⟨first name⟩ ⟨last name⟩” while other names are formatted as ”⟨last name⟩, ⟨first name⟩”. Finding such heterogenities and adding the specific data transformations to avoid mismatches is often very tedious. Thus, writing a linkage rule is not only a cumbersome but also a time consuming task.

Linkage Rule Execution: The aim of the linkage rule execution is to identify all pairs of entities for which the provided linkage rule is fulfilled. Figure 1.2 illustrates a typical entity matching data flow for two data sources. As the naïve approach of evaluating a linkage rule for all possi-

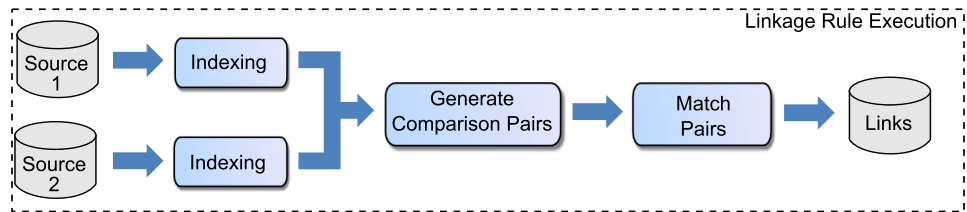


Figure 1.2: Execution Data Flow.

ble pairs of entities scales quadratically with the number of entities, an indexing method is usually employed to dismiss definitive non-matches early. Based on the index, a set of comparison pairs that contain two entities that are potential matches is generated. The linkage rule is only evaluated for each comparison pair. The output of the linkage rule execution is a set of *links* wherein each link connects two entities that have been found to match according to the linkage rule.

Various indexing methods have been proposed in literature to increase the efficiency by detecting and dismissing definitive non-matches early in the matching process.[Christen, 2012; Naumann and Herschel, 2010; Köpcke and Rahm, 2010; Köpcke et al., 2010]. Unfortunately, many of these methods may lead to a decrease of recall due to false dismissals.

In the previous example of a movie data set, an indexing method may for instance assign an index to each entity based on the release year of the movie that it describes. In this case, only entities about movies with the same release date need to be compared with the linkage rule. On the downside, entities that contain a wrong release date would be dismissed in this example. Developing indexing methods that reduce the number of comparisons while still generating a comparison pair for all matching entities is an ongoing research topic [Christen, 2011].

Evaluation: The purpose of the evaluation step is to measure the success of

the entity matching task and to find potential errors in the generated links. The success of the entity matching task can be determined by comparing the generated links with a gold standard consisting of a set of reference links. A set of *reference links* consist of positive reference links that identify pairs of entities that are known to match and negative reference links that identify pairs that that are known non-matches. If no reference links are known, a gold standard can be generated by a human expert who confirms or rejects a number of links. The creation of the reference links usually involves manual effort as a human expert is involved in confirming or declining candidate links. For that reason, it is desirable to reduce the number of link candidates that need to be labeled manually.

The challenge in reducing the labeling effort lies in choosing a set of link candidates for labeling that constitutes a representative sample of the entire data set. The reason why the naïve approach of choosing a few random links for labeling usually fails can be easily understood by looking at the example data set about movies that has been introduced in Table 1.1. In this example, a random selection of pairs of entities to be labeled as matches or non-matches would not guarantee that the rather rare case of a pair of movies that share the same title but still represent distinct movies due to different release dates is chosen.

After a set of reference links has been generated, two kinds of errors can be detected: False positive links, which are links that have been generated between entities that describe different real-world objects; and false negative links, which are positive reference links that have been missed by the entity matching task. The found errors can be used to improve the linkage rule, i.e., by going back to the linkage rule generation step of the entity matching workflow. This way, the linkage rule can be constantly improved by repeatedly improving the linkage rule, executing the linkage rule and identifying errors in the linkage rule.

In this thesis, we propose a set of novel methods that cover the complete entity matching workflow from the generation of linkage rules to their efficient execution on distributed systems. First of all, we present an expressive linkage rule representation, which may combine different similarity measures non-linearly and may employ chains of data translations to normalize data prior to comparison. In order to lower the effort of generating linkage rules, we propose the GenLink genetic programming algorithm to learn linkage rules that are based on the introduced representation. GenLink is a super-

vised learning algorithm that generates linkage rules from existing reference links.

As in many cases reference links are not available, we propose the ActiveGenLink active learning algorithm that generates reference links by requiring the user to confirm or decline a small number of link candidates. ActiveGenLink uses a novel query strategy that selects link candidates for labeling that yield a high information gain. By that it reduces the number of link candidates that need to be labeled by the user.

Finally, we present algorithms to efficiently execute the learned linkage rules based on the MultiBlock indexing method. MultiBlock uses a multidimensional index in which similar entities are located near each other in order to dismiss definitive non-matches early. All proposed algorithms have been implemented as part of the Silk Link Discovery Framework.

The electronic version of this dissertation along with links to data sets that have been used for evaluation in this work can be found at:

<http://dissertation.robertisele.com>

1.1 Use Cases

Entity matching has a abundance of use cases in a variety of application scenarios. In the following, we underline the motivation for entity matching by discussing four uses cases.


1.1.1 Data Integration

The purpose of data integration is to integrate multiple data sources into a single data set [Doan and Halevy, 2005]. A common problem in data integration efforts occurs when multiple entities from different data sources describe the same real-world object. In this case, duplicated entities from different data sources need to be identified and merged into a single entity. The essential task of identifying these duplicates is carried out by an entity matching process. Data integration problems are not bound to a specific data format and can be found, for instance, in relational databases or XML documents.

For instance, when integrating multiple data sources about persons, different entities in distinct data sources may refer to the same real person. We illustrate this problem by looking at two entries from two multi-domain data sources: Freebase and DBpedia. Freebase¹ [Bollacker et al., 2008] is a

¹<http://www.freebase.com>

collaboratively created knowledge base. DBpedia² [Bizer et al., 2009b] is a structured data set that has been extracted from Wikipedia articles. Figure 1.3 shows the entry for Woody Allen in both data sets. Both data sets



Woody Allen (born Allan Stewart Konigsberg, December 1, 1935) is an award-winning American screenwriter, director, actor, comedian, author, and playwright, whose career spans over half a century. He began as a comedy writer in the 1950s, penning jokes and scripts for television and also publishing several books of short humor pieces. In the early 1960s, Allen started performing as a stand-up comic, emphasizing monologues rather than traditional j...[More](#)

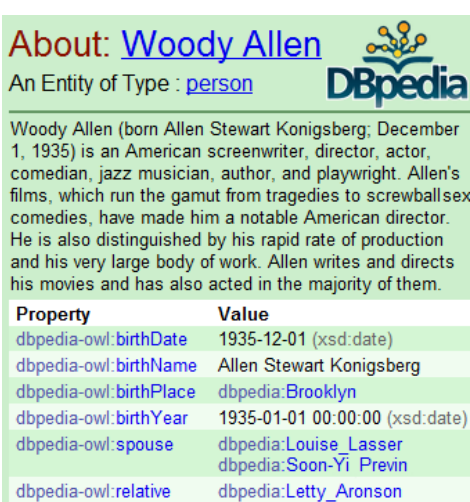
Date of birth: Dec 1, 1935 (age 76 years)

Place of birth: New York City, United States of America

Height: 1.65 m (5.41 ft)

Religion: Judaism, Atheism

Also known as: Allan Stewart Konigsberg, Allen Konigsberg, Allen Stewart Konigsberg



About: Woody Allen
An Entity of Type : [person](#)

Woody Allen (born Allen Stewart Konigsberg; December 1, 1935) is an American screenwriter, director, actor, comedian, jazz musician, author, and playwright. Allen's films, which run the gamut from tragedies to screwballsex comedies, have made him a notable American director. He is also distinguished by his rapid rate of production and his very large body of work. Allen writes and directs his movies and has also acted in the majority of them.

Property	Value
dbpedia-owl:birthDate	1935-12-01 (xsd:date)
dbpedia-owl:birthName	Allen Stewart Konigsberg
dbpedia-owl:birthPlace	dbpedia:Brooklyn
dbpedia-owl:birthYear	1935-01-01 00:00:00 (xsd:date)
dbpedia-owl:spouse	dbpedia:Louise_Lasser dbpedia:Soon-Yi_Previn
dbpedia-owl:relative	dbpedia:Letty_Aronson

(a) Freebase entry
(b) DBpedia entry

Figure 1.3: Woody Allen in Freebase and DBpedia. Retrieved June 18, 2012.

provided similar information, such as his name and his birth date. On the other hand, some properties, such as his height and information about his relatives, are only present in one data set. Therefore, merging both entities into a single entity would enrich the available data and provide a unified view of all information about Woody Allen.

1.1.2 Matching Census Data

A data integration challenge that is faced by many statistical institutes is the problem of integrating census data sets that have been collected from different sources [Jaro, 1989]. Table 1.2 shows three example census records:

Matching different census records can be difficult for a number of reasons, including:

- Persons may move to other locations or change their family name.
- Streets may be renamed.
- Administrative regions may be reorganized.

²<http://dbpedia.org>

Name	Birthdate	Street	Zip Code
J. Doe	2/12/1965	Main street 16	78701
Mr. John Doe	2/12/1965	Main st. 16	78701
Jack Smith	5/4/1986	Main street 18	78701

Table 1.2: Examples of census records.

- Persons names or addresses may be formatted differently.

In literature, entity matching has been applied to census data in a number of cases: Early work on applying entity matching for matching census records has been applied by Jaro [1989] on matching the 1985 Census of Tampa, Florida. Based on this preliminary work, the U.S. Bureau of the Census employed entity matching to merge administrative lists in the US census [Winkler and Thibaudeau, 1991]. Similar work has been conducted by the Australian bureau of statistics [Conn and Bishop, 2005].

All of these three efforts followed the same entity matching workflow, that has been proposed by Jaro [1989]:

Data Preparation Values such as names and addresses are standardized in order to normalize different spellings. For instance, different spellings of the name “John Doe”, such as “Mr. John Doe” and “J. Doe” can be normalized to “J. Doe” by removing common words (e.g., “Mr.”, “Dr.” etc.) and shorting the given name by leaving only its first character.

Field Matching The values of certain properties of pairs of entities are matched. The particular properties that are matched depends on the concrete data sets. Properties that qualify for matching include the name, gender, address or data of birth of the matched persons. For each of the matched properties, a similarity score is computed.

Match Decision All individual similarity scores are combined into a single similarity score. The scores are combined by taking the weighted average of all scores. Weights are specified so that properties that carry more information are weighted higher.

1.1.3 Bibliographic Databases

A bibliographic database collects citations to publications, such as research papers or books, together with corresponding metadata, such as article abstracts. A citation contains all information that is needed to identify a particular publication, including the name of the authors, the title of the paper,

the publication venue and the publication date. Applications of bibliographic databases are numerous [Lee et al., 2007; Christen, 2012]: They allow to measure the impact of a particular researcher. Furthermore, researchers can use them to search for papers that cover a particular research area or to find publications that are related to a specific paper. Examples of popular bibliographic databases are [Ford and O’Hara, 2008]: CiteSeer^{X3} [Li et al., 2006], DBLP⁴ [Ley, 2002, 2009], Google Scholar⁵ and Microsoft Academic Search⁶.

Figure 1.4 shows a screenshot of Google Scholar that illustrates the difficulty of finding citations that point to the same publication. The presented

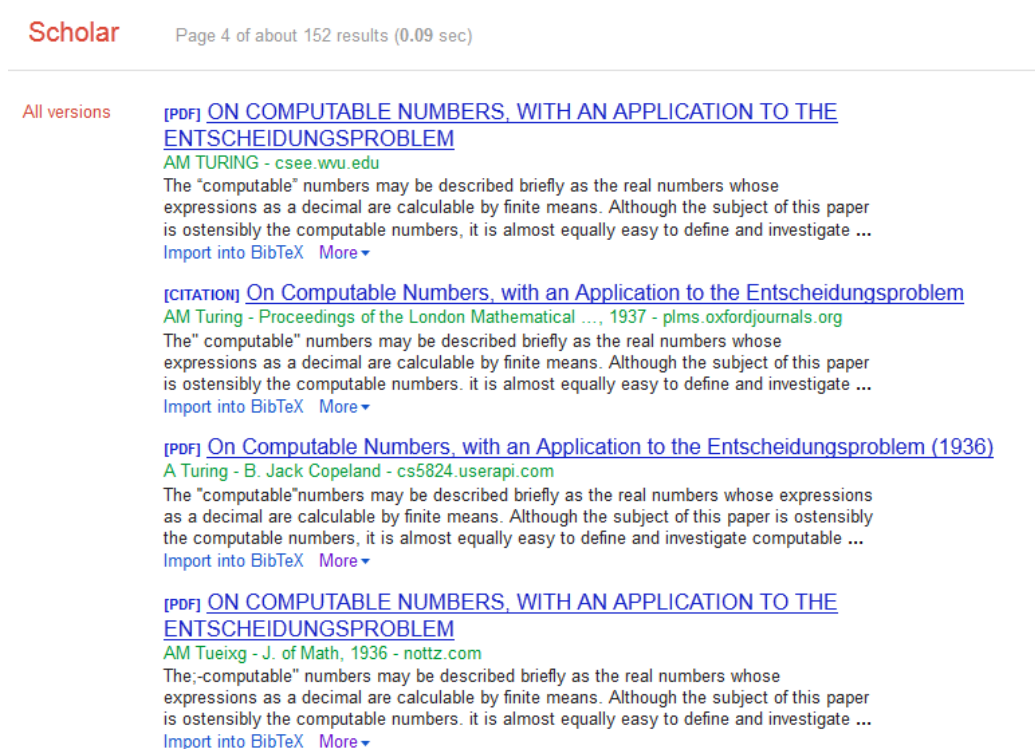


Figure 1.4: Screenshot of Google Scholar.

screenshot displays the result of a search for all versions of the paper “On Computable Numbers, with an Application to the Entscheidungsproblem” by Alan Turing. Each entry represents a citation to the paper that has been collected in the Web. The third entry represents a citation of a edited collection of papers published by Copeland [2004], which amongst other papers

³<http://citeseerx.ist.psu.edu>

⁴<http://dblp.uni-trier.de/>

⁵<http://scholar.google.com>

⁶<http://academic.research.microsoft.com/>

also contains the sought-after paper. The fourth entry has been identified as matching the sought-after paper in spite of the wrong spelling of the author.

A number of methods have been proposed that specifically focus on matching citations [Hylton, 1996; Cousins, 1998; Sitas and Kapidakis, 2008]. A data set that has been frequently used to evaluate the performance of different methods for matching citations is the Cora data set. The Cora data set contains citations that have been collected from the Cora Computer Science research paper search engine [McCallum et al., 2000b,a]. Table 1.3 shows a few entries from the Cora data set. This example demonstrates some of the

Author	Title	Venue	Date
D. Kibler and D. W. Aha.	Learning representative exemplars of concepts: an initial case study.	Proc. 4th International Workshop on Machine Learning	1987
D. Kibler and D. Aha.	Learning representative exemplars of concepts; an initial study.	In Proceedings of the Fourth International Workshop on Machine Learning	1987
Kibler, D., and Aha, D.W.	Learning Representative Exemplars of Concepts: An Initial Case Study.	In Proceedings of the Fourth International Workshop on Machine Learning	1987
Aha, D. W., Kibler, D., & Albert, M.	Instance-based learning algorithms	Machine Learning	1991

Table 1.3: Excerpt from the Cora data set. The first three citations refer to the same publication.

challenges that can occur when matching citations:

- Author names may be formatted differently.
- Journals and Conferences are sometimes abbreviated.
- Citations with spelling mistakes should still be identified as duplicates.

In Section 3.5.1, we will introduce the Cora data set in detail as one of the data sets that are used to evaluate the performance of the learning algorithms that are proposed in this thesis.

1.1.4 Publishing Linked Data

The central idea of Linked Data [Heath and Bizer, 2011] is to create a Web of Data by publishing and interlinking structured data on the Web. The term

Linked Data refers to a set of best practices for two main purposes: First, to make structured data available on the Web by following existing standards that have been defined by the World Wide Web Consortium (W3C). And secondly, to connect entities by setting typed RDF links, which can be followed in order to discover related entities.

In particular, the core principles of Linked Data, which have been proposed by Tim Berners-Lee in his Linked Data note [Berners-Lee, 2006], are the following:

(1) Use URIs as names for things

Uniform Resource Identifiers (URIs) [Berners-Lee et al., 1998] provide globally unique identifiers for entities, link types and concepts. URIs can be used to identify arbitrary things, such as:

Cities: For instance, the URI <http://dbpedia.org/resource/Berlin> identifies the city of Berlin in DBpedia⁷ [Bizer et al., 2009b]

Organizations: The URI <http://www.w3.org/data#W3C> identifies the W3C.

Persons: Tim Berners-Lee is identified in the Semantic Web Conference data set⁸ by the URI <http://data.semanticweb.org/person/tim-berners-lee>.

(2) Use HTTP URIs so that people can look up those names

URIs are not merely used as identifiers for things, they are also meant to be *dereferenceable*. A dereferenceable URI allows the retrieval of information about the referenced thing from the provider of the URI. The Hypertext Transfer Protocol (HTTP) [Fielding et al., 1999] provides the protocol for accessing information about the referenced entity, link or concept. For instance, a HTTP request on the URI <http://dbpedia.org/resource/Berlin>, returns structured data that describes the city of Berlin as provided by the DBpedia data source.

(3) Use standards to provide information

While the traditional Web relies on human readable documents in order to represent information, the idea of Linked Data is to provide machine readable structured data. In Linked Data, the Resource Description Framework

⁷<http://dbpedia.org>

⁸<http://data.semanticweb.org>

(RDF) [Manola and Miller, 2004] provides the common format for representing structured data. It follows a graph-based model wherein entities and concepts are represented as nodes, while links are represented as edges between nodes. Figure 1.5 depicts an example RDF graph. In the shown example, an

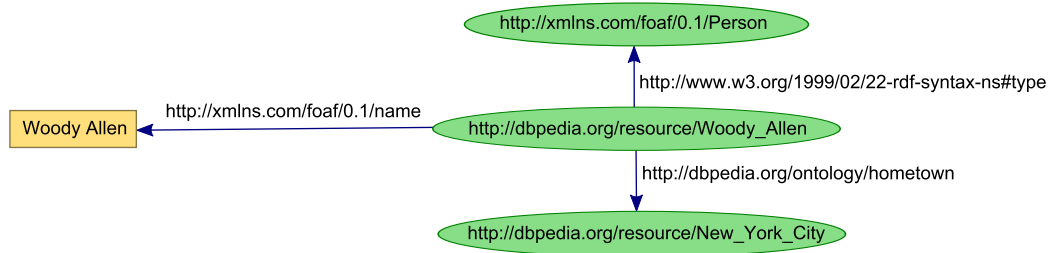


Figure 1.5: Excerpt of an RDF graph.

excerpt from the RDF graph that is returned when dereferencing the URI http://dbpedia.org/resource/Woody_Allen is shown. According to the shown RDF fragment, the given URI identifies an entity of type Person with the name “Woody Allen”. While the name is specified using a literal string value, the home town New York City is referred to with another URI. More information about New York City could be requested by dereferencing the corresponding URI.

An alternative way to access RDF data sources is to provide a SPARQL query service. SPARQL [Prud’hommeaux and Seaborne, 2008] is a query language to retrieve data that is structured using RDF.

(4) Include links to other URIs

While the first three principles are concerned with publishing isolated Linked Data sources on the Web, the fourth principle gives attention to connecting entities in different data sources. In the same way as entities inside a single data source can be connected using an internal RDF link, such as the hometown property in Figure 1.5, external RDF links refer to entities in another data source. Three types of external RDF links are commonly used [Heath and Bizer, 2011]:

Relationship Links relate two entities in different data sources. For instance, an entity about a music record in a music data source can point to the corresponding artist in a data source about persons using the RDF property <http://xmlns.com/foaf/0.1/maker> from the FOAF vocabulary [Brickley and Miller, 2005].

Identity Links connect two entities from different data sources that describe the same real-world object. By convention, the `http://www.w3.org/2002/07/owl#sameAs` property is usually used to set identity links.

Vocabulary Links are used to relate an entity to a term in a vocabulary or to connect two related terms. For instance, in Figure 1.5, the entity that describes Woody Allen is related to the term that identifies persons in the DBpedia.

Figure 1.6 shows an example with three RDF data sources. This ex-

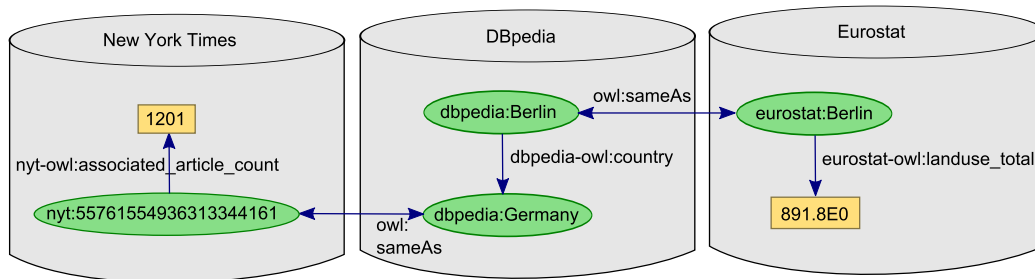


Figure 1.6: Interconnected RDF data sources. The original URIs have been shortened for illustration.

ample shows how the entity that describes the city of Berlin in DBpedia is connected to other data sources. A LinkedData consumer that dereferences the URI identifying Berlin in DBpedia receives an RDF graph that, among information about Berlin that is provided by DBpedia itself, also contains RDF links to other data sets. For instance, the Linked Data consumer can discover additional information about Berlin by following the `owl:sameAs` link to the corresponding entity in Eurostat⁹. On the other hand, following the `owl:sameAs` link to the New York Times¹⁰ data set gives access to information provided by the New York Times.

One of the most prominent efforts around Linked Data is the W3C *Linked Open Data* (LOD) community project¹¹. The primary aim of LOD is to make information freely available on the Web by using the Linked Data principles.

⁹<http://ec.europa.eu/eurostat>. An RDF version has been published at <http://wifo5-03.informatik.uni-mannheim.de/eurostat/>.

¹⁰<http://data.nytimes.com/>

¹¹<http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>

The Data Hub¹² is an effort to catalog Linked Open Data sources. The LOD cloud visualizes the current state of the Web of Linked Open Data (see Figure 1.7). The amount of data that is published as Linked Open Data

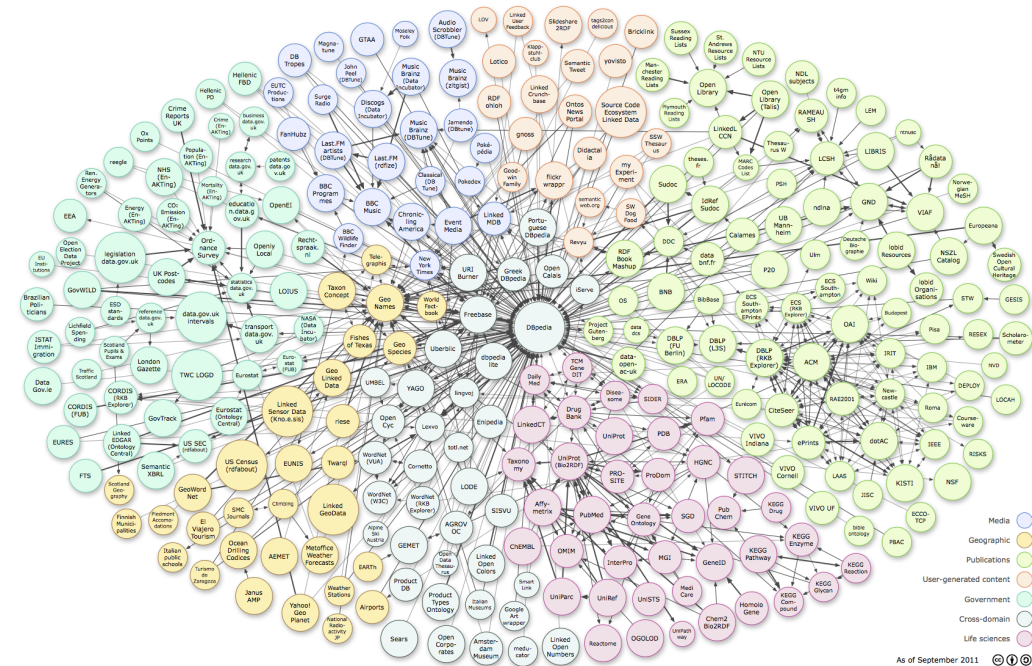


Figure 1.7: LOD cloud from 2011-09-19. The diagram is maintained by Richard Cyganiak and Anja Jentzsch.

has grown significantly over the last years and is estimated at over 31 billion RDF statements¹³.

Linked Data Publishers can follow the Linked Data principle of connecting data sources by setting RDF links from the entities in the data source to related entities in other Linked Data sources. In order to help data publishers to set RDF links pointing into other data sources, several link discovery tools - such as Silk or LIMES [Ngonga Ngomo and Auer, 2011] - have been developed. These tools employ entity matching techniques in order to compare entities in different Linked Data sources based on user-provided linkage rules, which specify the conditions that must hold true for two entities in order to be interlinked. The links are usually published together with the primary data set itself, but can also be published separately.

¹²<http://thedatahub.org/>

¹³<http://lod-cloud.net/>

1.2 Contributions

In this thesis, we propose novel algorithms for learning linkage rules that employ an expressive representation that can be executed efficiently. In particular, this thesis makes the following four contributions:

- (1) **An Expressive Linkage Rule Representation:** We propose an expressive linkage rule representation, which is capable of combining different distance measures non-linearly and may include chains of data transformations to normalize values prior to comparison. The proposed linkage rule representation is more expressive than previous work and subsumes threshold-based boolean classifiers and linear classifiers. Linkage Rules are represented as an operator tree and can be understood and modified by humans. The proposed linkage rule representation will be introduced in Chapter 2.
- (2) **Supervised Learning of Linkage Rules:** We propose the GenLink genetic programming algorithm to evolve linkage rules from a set of reference links. The learned linkage rules may utilize the full expressivity of the introduced representation. The basic idea of GenLink is to evolve the population of linkage rules by using a set of specialized crossover operators. Each of these operators only operates on one aspect of the linkage rule, e.g., one crossover operator builds chains of transformations while another operator only combines distance thresholds. The experimental evaluation shows that the proposed algorithm is capable of learning linkage rules which:
 - Select discriminative properties for comparison.
 - Select chains of data transformations to normalize property values prior to comparison.
 - Select appropriate distance measures combined with suitable distance thresholds.
 - Select appropriate aggregation functions that combine the result of multiple comparisons.

We show that GenLink outperforms the state-of-the-art genetic programming algorithm for entity matching recently presented by de Carvalho et al. [2012] and is capable of learning linkage rules that achieve a similar accuracy than the human written rule for the same problem. GenLink will be discussed in detail in Chapter 3.

- (3) **Active Learning of Linkage Rules:** For cases where no or only a small number of reference links are available, we propose the ActiveGenLink algorithm to generate additional reference links interactively. ActiveGenLink combines genetic programming with active learning in order to automate the generation of the linkage rule while the user only has to verify a set of link candidates. ActiveGenLink uses a novel query strategy that reduces the number of candidates, which need to be labeled by the user, and outperforms the query-by-vote-entropy strategy, which has been previously used for active learning of linkage rules. The idea of the proposed query strategy is to select link candidates for labeling that yield a high information gain. Within our experiments, by labeling a small number of links, ActiveGenLink was capable of finding linkage rules with a comparable performance than GenLink on a much bigger set of reference links. In addition, ActiveGenLink outperformed recent unsupervised algorithms after manually labeling a few link candidates. ActiveGenLink will be discussed in detail in Chapter 4.
- (4) **Efficient Execution of Linkage Rules:** It is essential that the learned linkage rules can be executed efficiently on local or distributed architectures. This thesis proposes the novel MultiBlock indexing method that allows the efficient execution of linkage rules. MultiBlock uses a multidimensional index in which similar objects are located near each other. In each dimension the entities are indexed by a different property increasing the efficiency of the index significantly. In addition, it guarantees that no false dismissals can occur. We show that, using MultiBlock, the linkage rules can be executed efficiently on distributed architectures, such as Hadoop clusters. The proposed execution data flow data that includes MultiBlock will be discussed in Chapter 5.

The proposed methods have been implemented as part of the Silk Link Discovery Framework [Isele et al., 2010]. Silk discovers matching entities within data sets that are represented as RDF. The main application area of the framework is to find matching entities within data sets that are accessible on the Web according to the Linked Data principles [Bizer et al., 2009a]. The discovered matches may be used to set RDF links (`owl:sameAs`) between the data sources. Silk Link Discovery Framework is available for download under the terms of the Apache License¹⁴ and all experiments that are presented in this thesis can thus be repeated by the interested reader. The Silk Link Discovery Framework will be introduced in Chapter 6.

¹⁴<http://silk.wbsg.de/>

1.3 Thesis Outline

Chapter 1: Introduction

This chapter introduces the reader to the main topics of this thesis and highlights its contributions.

Chapter 2: Linkage Rules

This chapter introduces the various parts of a linkage rule in detail and proposes an expressive linkage rule representation. The proposed representation may employ data preparation techniques prior to comparing different values of the entities that are to be matched. It allows to combine the resulting similarity scores non-linearly into a single similarity score.

Section 2.2: Data Preparation. As data sources may use different data formats, data needs to be normalized prior to comparison. This section introduces well-known data transformation and standardization methods. The presented methods can be used in the proposed linkage rule representation to normalize property values of entities that are to be compared.

Section 2.3: Field Matching. The similarity between different entities is usually assessed by comparing the values of individual properties of the entities. This section introduces distance measures that are commonly used to deal with different types of errors, such as typographical variations. The proposed linkage rule representation employs distance measures for comparing the normalized values of two entities.

Section 2.4: Previous Work on Linkage Rules. A linkage rule specifies the conditions that must hold true for two entities in order to be considered a match. For this purpose, a linkage rule may combine multiple comparisons of different properties of the entities that are to be matched. This section discusses previous work on representing linkage rules. It shows that popular representations include linear classifiers that combine multiple comparison scores by computing the weighted average and threshold-based boolean classifiers that combine multiple comparisons using logical operators. In contrast to the representation that is proposed in this thesis, linear and threshold-based boolean classifiers do not include data transformations into the linkage rule, but rely on a preceding data preparation stage instead [Elmagarmid et al., 2007].

Section 2.5: An Expressive Linkage Rule Representation. In this section, we propose an expressive linkage rule representation, which may combine different distance measures non-linearly and may include chains of data transformations to normalize values prior to comparison. We show that the introduced linkage rule representation is more expressive than previous work and subsumes threshold-based boolean classifiers as well as linear classifiers.

Section 2.6: Summary. This section summarizes the main points of this chapter.

Chapter 3: Supervised Learning of Linkage Rules

This chapter introduces the first key contribution of this thesis: The GenLink learning algorithm. It motivates machine learning of linkage rules by pointing out that creating linkage rules by hand is a non-trivial problem and requires a high level of expertise together with detailed knowledge of the data sets. We introduce supervised learning as a method to automate the creation of a linkage rule in cases where existing reference links are known.

Section 3.1: Problem Definition. We formalize the problem of learning a linkage rule from existing reference links.

Section 3.2: Genetic Programming. In this section, we introduce genetic programming to the reader. We describe the general genetic programming algorithm and highlight the capability of genetic programming to learn arbitrary operator trees.

Section 3.3: GenLink. In this section, we propose the GenLink supervised learning algorithm, which employs genetic programming in order to learn linkage rules from a set of existing reference links. Following the genetic programming paradigm, GenLink starts with an initial population of candidate solutions, which is iteratively evolved by applying a set of genetic operators.

Section 3.4: Previous Work. This section provides an overview of the current state-of-the-art in machine learning of linkage rules. We discuss supervised learning algorithms, which are concerned with learning linkage rules from reference links, as well as unsupervised learning algorithms, which are concerned with matching entities when no reference links are available. We show that most previous algorithms for learning linkage rules either employ threshold-based boolean classifiers or

linear classifiers for representing linkage rules. In addition, we discuss a previously proposed genetic programming algorithm for learning more expressive linkage rules. Finally, we compare the reported performance scores for different entity matching approaches that have been evaluated on two frequently used evaluation data sets.

Section 3.5: Evaluation and Discussion. We evaluate GenLink with six data sets: (1) We evaluate the learning performance on two frequently used record linkage data sets. The results show that the proposed approach outperforms the state-of-the-art genetic programming approach for entity matching by de Carvalho et al. [2012]. (2) We evaluate the performance with two data sets from the Ontology Alignment Evaluation Initiative and compare our results to the participating systems. (3) We compare the learned linkage rules with linkage rules created by a human expert for the same data set. The results show that our approach is capable of learning linkage rules, which achieve a similar accuracy than the human written rule for the same problem.

As we propose an approach that uses a linkage rule representation that is more expressive than the most common representations used in record linkage, we compared the learning performance to threshold-based boolean classifiers and linear classifiers. As GenLink further does not use subtree crossover, which is the de-facto standard in genetic programming, we evaluated the actual contribution to the learning performance of the introduced specialized crossover operators.

Section 3.6: Summary. This section summarizes the main points of this chapter.

Chapter 4: Active Learning of Linkage Rules

This chapter starts with highlighting the dependency of supervised learning on existing training data in the form of reference links. We motivate the use of active learning as a semi-supervised learning approach to overcome this limitation by generating additional reference links with minimal user-interaction. Finally, it presents the second key contribution of this thesis: The ActiveGenLink learning algorithm.

Section 4.1: Active Learning. This section introduces active learning to the reader.

Section 4.2: ActiveGenLink. In this section, we propose the ActiveGenLink algorithm, which combines genetic programming and active learn-

ing to learn linkage rules interactively. We introduce a novel query strategy that reduces user involvement by selecting the link candidates to be verified by the user that are the most informative.

Section 4.3: Previous Work. In this section, we discuss previous work in active learning in general and present existing applications of active learning to entity matching.

Section 4.4: Evaluation and Discussion. We evaluate the performance of ActiveGenLink on the same data sets as have been used to evaluate the supervised GenLink algorithm in the previous chapter. The experiments are executed by labeling a maximum of 50 link candidates for each data set and comparing the final performance to results that have been achieved by GenLink. We show that by labeling a small number of link candidates, ActiveGenLink achieves a comparable performance than GenLink on the full set of reference links.

Section 4.5: Summary. This section summarizes the main points of this chapter.

Chapter 5: Execution of Linkage Rules

The efficient execution of linkage rules is essential to scale to large data sets. This chapter introduces a complete workflow to efficiently execute linkage rules which have been manually written or learned by one of the algorithms previously proposed.

Section 5.2: Execution Workflow. This section introduces the overall data flow of executing a linkage rule.

Section 5.3: MultiBlock. We propose a novel indexing method called MultiBlock to efficiently execute linkage rules. The proposed method uses a multidimensional index in which similar objects are located near each other in order to improve the efficiency beyond existing methods.

Section 5.4: Previous Work on Indexing. The naive method of executing a linkage rule is to evaluate the rule for each pair of entities from the Cartesian product of all known entities. In record linkage, many indexing methods have been proposed that substantially reduce the number of required entity comparisons. This section discusses the most common methods.

Section 5.5: Distributed Execution of Linkage Rules. In this section, we discuss methods to scale the execution of linkage rules to a cluster of machines. We propose a distributed MapReduce-based data flow that employs MultiBlock to scale out the execution of a linkage rule.

Section 5.6: Evaluation and Discussion. We evaluate both the efficiency and the effectiveness of the proposed MultiBlock approach. Efficiency is demonstrated by showing that MultiBlock outperforms previously proposed indexing methods. Effectiveness is demonstrated by showing that MultiBlock reduces the number of comparisons significantly without any false dismissals, even when used with complex linkage rules. Finally, we demonstrate that MultiBlock can be run using the MapReduce paradigm to scale efficiently to a cluster of machines.

Section 5.7: Summary. This section summarizes the main points of this chapter.

Chapter 6: The Silk Link Discovery Framework

This section introduces the Silk Link Discovery Framework. Silk discovers matching entities within data sets that are represented as RDF. Silk implements all methods for learning and executing linkage rules that are proposed by this thesis.

Section 6.1: Silk Link Specification Language. The *Silk Link Specification Language* (Silk-LSL) is a declarative language for representing link specifications. A link specification encapsulates all information needed to interlink two data sets. This includes the type of link that is to be generated, information on how to access the data sets that are to be interlinked, the linkage rule including a link filter and finally the outputs to which the link should be written to.

Section 6.2: Silk Workbench. This section describes the Silk Workbench, a web application that provides a graphical user interface to the Silk Link Discovery Framework. The Silk Workbench provides a graphical user interface to the supervised learning as well as active learning methods implemented in the Silk Link Discovery Framework.

Chapter 7: Conclusion

This chapter summarizes the main contributions of this thesis and discusses known limitations and directions for future work.

1.4 Published Work

Parts of the work presented in this thesis have been published previously:

- Previous work on the GenLink genetic programming algorithm has been presented at two occasions:

Robert Isele, Christian Bizer: Learning Expressive Linkage Rules using Genetic Programming. Proceedings of the VLDB Endowment (PVLDB) 5(11):1638-1649, August 2012.

Robert Isele, Christian Bizer: Learning Linkage Rules using Genetic Programming. Proceedings of the Sixth International Workshop on Ontology Matching, October 2011.

- Previous work on the ActiveGenLink active learning algorithm has been presented at two occasions:

Robert Isele, Christian Bizer: Active Learning of Expressive Linkage Rules using Genetic Programming. Web Semantics: Science, Services and Agents on the World Wide Web, 2013.

Robert Isele, Anja Jentzsch, Christian Bizer: Active Learning of Expressive Linkage Rules for the Web of Data. Proceedings of the 12th International Conference on Web Engineering (ICWE), July 2012.

- Previous work on the MultiBlock indexing approach is covered by the following publication:

Robert Isele, Anja Jentzsch, Christian Bizer: Efficient Multi-dimensional Blocking for Link Discovery without losing Recall. Proceedings of the 14th International Workshop on the Web and Databases (WebDB), June 2011.

- The Silk Link Discovery Framework in general is covered in:

Robert Isele, Anja Jentzsch, Christian Bizer: Silk Server – Adding missing Links while consuming Linked Data. Proceedings of the First International Workshop on Consuming Linked Data (COLD), November 2010.

Chapter 2

Linkage Rules

A number of different approaches for solving the entity matching problem have been proposed in literature: *Unsupervised approaches* aim to match entities without any user-provided configuration [Köpcke and Rahm, 2010; Euzenat et al., 2011a; Aguirre et al., 2012]. An unsupervised approach identifies matching entities based on the characteristics of the data set. In addition, some unsupervised approaches have been proposed that exploit data set independent background knowledge [Michalowski et al., 2004; Doan et al., 2003]. In contrast to unsupervised approaches, *rule-based approaches* classify each pair of entities as match or non-match based on data set specific linkage rules Winkler [1995]. Within rule-based approaches, a linkage rule specifies how the similarity of a pair of entities is determined. In contrast to rule-based approaches that are local in the sense that they conduct match decisions for each pair of entities independently, *collective entity matching* approaches conduct match decisions of all related pairs of entities at once globally [Bhattacharya and Getoor, 2007] by exploiting relationships between different types of entities. In the context of this thesis, we focus on machine learning and execution of expressive linkage rules. Section 3.4 will discuss in more detail how our approach compares to unsupervised and collective approaches.

In this section, we will introduce the various components of linkage rules. At first, we will introduce data normalization techniques for preparing values for matching. After that, we will introduce common distance measures for comparing values for similarity. In addition, we will discuss previous work on representing linkage rules that combine multiple similarity measures. Finally, we will introduce the first main contribution of this thesis: a linkage rule representation that is more expressive than previous work and includes data transformations to normalize values prior to comparison.

2.1 Problem Definition

We consider the problem of matching entities between two sets of entities A and B . Each entity $e \in A \cup B$ can be described with a set of properties $e.p_1, e.p_2, \dots, e.p_n$. For instance, an entity denoting a person may be described by the properties *name*, *birthday* and *address*. The objective is to determine which entities in A and B identify the same real-world object.

The general problem of entity matching can be formalized as follows [Fellegi and Sunter, 1969]:

Definition 2.1 (Entity Matching) *Given two sets of entities A and B , find the subset of all pairs of entities for which a relation \sim_R holds:*

$$M = \{(a, b); a \sim_R b, a \in A, b \in B\}$$

Similarly, we define the set of all pairs for which \sim_R does not hold:

$$U = (A \times B) \setminus M$$

The purpose of relation \sim_R is to relate all entities that represent the same real-world object. Note that entities that are related by \sim_R are not necessarily equal (i.e., define properties with the same values). Instead of that, they may differ in some of their values, for instance, due to errors in the data set.

In some cases a subset of M and U is already known prior to matching in the form of *reference links*.

Definition 2.2 (Reference Links) *A set of positive reference links $R_+ \subseteq M$ contains pairs of entities for which relation \sim_R is known to hold (i.e., which identify the same real-world object). Analogously, a set of negative reference links $R_- \subseteq U$ contains pairs of entities for which relation \sim_R is known to not hold (i.e., which identify different real-world objects).*

Reference links can serve two purposes: Firstly, they can be used to evaluate the quality of a linkage rule. But more importantly, they can also be used to infer a linkage rule, which specifies how the similarity of a pair of entities is determined.

Definition 2.3 (Linkage Rule) *A linkage rule l assigns a similarity score to each pair of entities:*

$$l : A \times B \rightarrow [0, 1]$$

The set of matching entities is given by all pairs for which the similarity according to the linkage rule exceeds a threshold θ :

$$M_l = \{(a, b); l(a, b) \geq \theta, a \in A, b \in B\}$$

As the linkage rule may return arbitrary values in the range $[0, 1]$, we fix the value of the threshold θ to 0.5 without loss of generality. In literature, linkage rules are also known as identity rules [Lim et al., 1993], record matching rules [Fan et al., 2009] and record matching packages [Arasu et al., 2010]. By using the term *linkage rule*, we follow the terminology introduced by Fellegi et al. in their article *A Theory for Record Linkage* [Fellegi and Sunter, 1969].

Instead of using a single threshold, it has also been proposed to use two different thresholds in order to distinguish between three classes: *match*, *non-match*, and *possible match* [Fellegi and Sunter, 1969]. If the linkage rule assigns a similarity score that is between both thresholds, the pair is regarded as *possible match*. If the similarity score is higher than the bigger threshold the entity pair is regarded as *match*. Similarly, If the similarity score is lower than the smaller threshold the entity pair is regarded as *non-match*. The introduction of a third class enables the user to distinguish between entity pairs that are possible matches and thus need to be reviewed by a human expert and entity pairs which are considered certain matches and do not need to be manually confirmed.

2.2 Data Preparation

Data preparation is usually carried out as a separate step preceding the actual matching of the entities. Motivations for using a preparatory data normalization step are diverse:

Firstly, if two data sources that adhere to different schemata are to be matched, their schemata need to be standardized. For instance, one data source might represent person names in separate properties for the first and the last name, while another data source may use a single property for both. In cases like this, data transformation functions need to be applied in order to transform both data sources to a consistent schema.

Another motivation for data normalization is the presence of typographical errors. While some typographical errors cannot easily be removed and have to be dealt with in the comparison phase, many kinds of errors can be removed by a normalization step prior to matching. Examples include the use of an irregular letter case or the inconsistent use of separation characters, such as hyphens. Apart from typographical errors, mismatches also originate from the use of different data formats. In many cases, the reason for this is a lack of an established standard format. Examples of different data formats include the use of different units of measurements and the use of different formats for addresses and telephone numbers.

This section provides an overview of common data transformations func-

tions for the purpose of preparing the data for matching [Rahm and Do, 2000; Sarawagi, 2008; Pyle, 1999; Elmagarmid et al., 2007; Christen, 2012].

2.2.1 Standardization

Standardization transformations translate values that do not adhere to a consistent content format to a uniform representation [Elmagarmid et al., 2007].

Case Normalization

A common source of mismatches is the use of an irregular letter case (e.g., “eMail” vs. “EMAIL”). A way to address case inconsistency is to normalize all values to lower case prior to comparing them. Figure 2.1 presents a simple example.

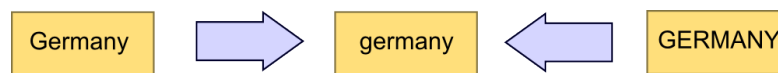


Figure 2.1: Case normalization.

Address Standardization

Address standardization is the task of normalizing street addresses into a uniform format [Winkler, 1995; Elmagarmid et al., 2007]. Address standardization is necessary if addresses are not following a consistent format.

For illustration, we have a look at two alternative spellings of the same address:

- 23 gary street, Berlin, DE
- 23 Gary St., 10999 Berlin, Germany

A typical approach to standardize such inconsistencies consists of at least three steps:

- (1) Segment each address into its components and store each component in a separate entity field (house number, street, city, state, zip code). This step can be skipped if the data set already provides each component separately.
- (2) Standardize synonyms and abbreviations, such as “Rd.” for “Road”.

- (3) Normalize the letter case, e.g., by capitalizing all words.

In literature, a number of more sophisticated methods for standardization have been proposed [Li et al., 2002; Guo et al., 2009]. While none of these methods typically achieves optimal results, they are usually sufficient to ensure that subsequent fuzzy string matching methods are capable of identifying potential duplicates.

2.2.2 Stop-Word Removal

A typical normalization operation is to remove characters or words that are very common or do not carry significant extra information [Christen, 2012]. For instance, when matching person names, prefixing titles, such as “Mr.” or “Dr.”, can usually be removed prior to matching. Another example is the removal of common words, such as “the” or “of”.

2.2.3 Structural Transformations

While many data transformations are only applied on the values of a single property, some other data transformations apply structural transformations on the data sets. For instance, a data source that uses the FOAF vocabulary [Brickley and Miller, 2005] may represent person names using the `foaf:firstName` and `foaf:lastName` properties, while a data source using the DBpedia ontology may represent the same names using just the `dbpedia:name` property. In order to compare entities expressed in different schemata or data formats, their values have to be normalized prior to comparing them for similarity. In this example, we could achieve this in two ways: We could concatenate `foaf:firstName` and `foaf:lastName` into a single name before comparing them to `dbpedia:name` using a character-based distance measure, such as the Levenshtein distance. Alternatively, we could split the values of `dbpedia:name` using a tokenizer and compare them to the values of `foaf:firstName` and `foaf:lastName` using a token-based distance measure, such as the Jaccard coefficient. Figure 2.2 illustrates the concatenation of two properties as well as the splitting of a single property into two properties. An overview of more complex structural transformations can be found in [Benjamin Braatz, 2008].

Segmentation

Segmentation is a special case of a structural transformation in which a string is split into a set of predefined properties [Sarawagi, 2008]. In literature, a

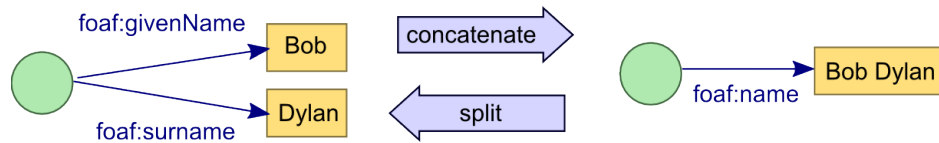


Figure 2.2: Concatenation and splitting of properties as examples of a structural transformations.

variety of approaches have been proposed to segment strings into structured entities such as product descriptions [Kannan et al., 2011; Ghani et al., 2006], postal addresses [Cai et al., 2005], bibliography records [Borkar et al., 2001] and person names [Christen, 2012].

Figure 2.3 shows an example of segmenting a product description into different properties, such as the product name and the manufacturer of the product.

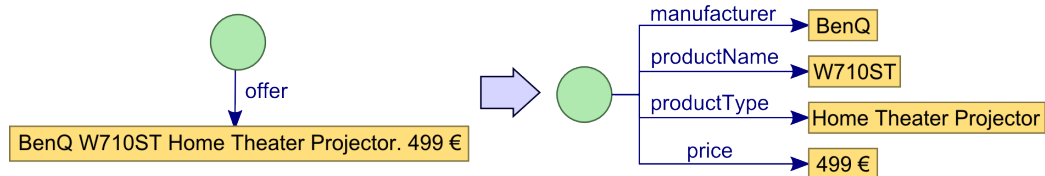


Figure 2.3: Segmentation of a product offer.

Popular approaches for segmentation in the context of entity matching include rule-based segmentation and statistical segmentation [Christen, 2012]. A typical rule-based segmentation approach consists of a set of rules in which each rule is of the form:

$$pattern \rightarrow action$$

The idea of a rule is that whenever the specified pattern is found in the input string, the corresponding action is executed. An action usually assigns the found pattern, or a part of it, to a specific output property. An example of a rule which extracts a price tag from a product description is:

$$\{\text{currency sign preceded by number}\} \rightarrow \{\text{Assign number to price property}\}$$

In contrast to rule-based systems, statistical segmentation approaches are not based on hard rules. Statistical segmentation approaches start by segmenting the input string into tokens or word chunks. A probability distribution is then used to assign segments of the input string to an output property.

In practice, rule-based methods are usually preferred to statistical methods when the input data is regular and thus strict rules work well [Sarawagi,

2008]. On the other hand, statistical methods are usually preferred if the input data is noisy.

Extraction of Values from URIs

Many data sets that are published as Linked Data encode information in the URI of each entity, which is not necessarily replicated in a property. Data transformations can be used to extract specific values from the URIs, which thereby can be used for matching the corresponding entities. Figure 2.4 illustrates the extraction of an identifier from a URI in a life science data set.

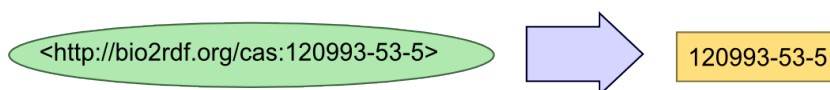


Figure 2.4: Extract a literal value from an URI.

2.3 Field Matching

The task of assessing the similarity of two string values is central to entity matching. As each entity provides a number of fields, matching two entities comes down to comparing the values of individual fields of both entities. For instance, when matching entities describing persons, field matching techniques can be employed to compare individual properties of the persons, such as their names, their birth date or their address.

Field matching measures can either be formulated as similarity measures or as distance measures [Naumann and Herschel, 2010]. Given two string values that are built from an alphabet Σ , a similarity measure returns a value from the range $[0, 1]$:

$$sim : \Sigma^* \times \Sigma^* \rightarrow [0, 1]$$

A similarity measure returns 1 if both strings are equivalent and 0 if both strings don't share any similarity. Analogously, we can define a distance measure as:

$$dist : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}$$

A distance measure returns 0 if both strings are equivalent and higher values for dissimilar strings. Note that, in contrast to similarity measures, in general, distance measures are not normalized (i.e., they might return values

bigger than one). In order to avoid any confusion, we formulate all measures as distance measures in the context of this work.

In literature, many distance measures have been proposed to match string values for entity matching. Each distance measure is typically targeted at handling a certain kind of variations in the string values. Which distance measure yields the optimal results depends on the specific entity fields that are to be compared and the particular data set. In order to compare two entity fields, a string measure needs to be chosen together with a distance threshold. The distance threshold indicates the maximum distance that is allowed between two values for them to be considered a match.

This section introduces distance measures that are commonly used to deal with different types of variations. We will describe the most well-known character-based measures, which assess the similarity of two strings by comparing them on the level of individual characters, as well as token-based measures, which compare both strings on the level of individual words. We will also introduce hybrid measures, which combine character-based measures and token-based measure in order to gain the advantages of both. More specific similarity measures, such as measures to compare numeric strings, are also discussed.

Comprehensive overviews of common distance measures can be found in [Elmagarmid et al., 2007; Christen, 2012; Naumann and Herschel, 2010]. While the performance of different distance measures depends on the characteristics of the data set, experimental performance comparisons of various distance measures can be found in [Cohen et al., 2003; Christen, 2006].

2.3.1 Character-Based Measures

Character-based distance measures assess the similarity of two string values on character level. They are well-suited for handling typographical errors.

Levenshtein Distance

Given two strings s_1 and s_2 , the *Levenshtein distance* [Levenshtein, 1966] (sometimes also called *edit distance*) is defined as the the minimum number of edit operations that are needed to transform s_1 into s_2 . Three edit operations are allowed:

- **Insertion** of a character into the string.
- **Deletion** of a character from the string.
- **Substitution** of one character in the string with another character.

We illustrate the computation of the Levenshtein distance on two simple examples: The Levenshtein distance between the strings “Table” and “able” is one as “Table” can be transformed into “able” by deleting the first character. Similarly, the distance between “Table” and “Cable” is also one as they can be transformed into each other by substituting one character.

Given a finite alphabet Σ and two strings $\sigma_1 \in \Sigma^*$ and $\sigma_2 \in \Sigma^*$, the (unnormalized) Levenshtein distance always returns a whole number between 0 and $\max(|\sigma_1|, |\sigma_2|)$. Based on this observation, the Levenshtein distance can be normalized in order to yield a value from the interval $[0, 1]$:

$$\text{NormalizedLevenshtein}(\sigma_1, \sigma_2) := \frac{\text{Levenshtein}(\sigma_1, \sigma_2)}{\max(|\sigma_1|, |\sigma_2|)} \quad (2.1)$$

Many variations and improvements, both to matching performance as well as execution performance, have been made to the original Levenshtein distance. A comprehensive overview and experimental evaluation of different algorithms for computing the Levenshtein distance can be found in [Navarro, 2001].

Jaro Distance

The *Jaro distance* metric [Jaro, 1989] has been originally developed to compare person names in the U.S. Census. The idea of the Jaro distance is to count the number of common characters within a specific distance. The Jaro distance is defined as [Winkler, 1990]:

$$\text{Jaro}(\sigma_1, \sigma_2) := \begin{cases} 0 & \text{if } c = 0 \\ \frac{1}{3} \left(\frac{c}{|\sigma_1|} + \frac{c}{|\sigma_2|} + \frac{c-\tau}{c} \right) & \text{otherwise} \end{cases} \quad (2.2)$$

The Jaro distance is based on two parameters:

c: The number of characters that are in common in both strings. Two characters are considered in common, if their position in both strings is no further apart than half the length of the longer string minus one.

τ : The number of transpositions of characters.

Jaro-Winkler Distance

The *Jaro-Winkler distance* measure [Winkler, 1990] is an extension of the Jaro distance that assigns a smaller distance to names with a common prefix. The Jaro-Winkler distance is defined as [Winkler, 1990]:

$$\text{JaroWinkler}(\sigma_1, \sigma_2) := \text{Jaro}(\sigma_1, \sigma_2) + \frac{\min(i, 4)}{10} (1 - \text{Jaro}(\sigma_1, \sigma_2)) \quad (2.3)$$

Wherein i denotes the length of the common prefix of both strings.

2.3.2 Token-Based Measures

Character-based measures work well for typographical errors, but fail when word arrangements differ. For instance, when comparing person names, changing the order of the first and the last name or adding a title prevents a character-based distance measure from identifying a duplicate. In the example of person names, changing 'John Doe' to 'Doe, John' would increase the Levenshtein distance from zero to six. Changing 'John Doe' to 'Mr. John Doe' has a similar effect.

The idea of token-based measures is to split the values into tokens before computing the distance. The distance is then computed by comparing the tokens for each string while ignoring their order. A string, such as 'Mr. John Doe', would be split into the token set {'Mr.', 'John', 'Doe'}. The method by which the strings are split into tokens can be chosen independently of a specific token-based metric. While simple approaches split the strings into tokens at each whitespace character, more sophisticated approaches also include punctuation characters in order to capture cases such as 'Doe, John'. If the strings are solely split by whitespace characters, data preparation may be needed to remove punctuation prior to comparison.

In pure token-based measures, the individual tokens are compared by equality, i.e., two tokens are only considered a match if they are exact duplicates. Later, we will also introduce hybrid measures which combine token-based measures with character-based measure to allow for near duplicates as well.

In the following, we introduce the most common token-based measures.

Jaccard Index

The intuition behind the *Jaccard index* [Jaccard, 1901]¹ is to measure the fraction of the tokens that are shared by both strings. More precisely, the Jaccard index is defined as the number of matching tokens divided by the total number of distinct tokens.

Definition 2.4 (Jaccard Distance) *The distance between two token sets A and B according to the Jaccard index is defined as:*

$$Jaccard(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

¹A translation of the French original of the cited article is provided in [Jaccard, 2006]

For instance, the Jaccard distance of the strings “Mr. John Doe” and “Doe, John” computes to:

$$Jaccard(\{\text{Mr.}, \text{John}, \text{Doe}\}, \{\text{Doe}, \text{John}\}) = 1 - \frac{2}{3} = \frac{1}{3}$$

Dice Coefficient

The *dice coefficient* [Dice, 1945] is very similar to the Jaccard index. The dice coefficient is defined as twice the number of tokens that are shared by both strings divided by the sum of the sizes of both token sets.

Definition 2.5 (Dice Distance) *The distance between two token sets A and B according to the Dice coefficient is defined as:*

$$Dice(A, B) = 1 - \frac{2|A \cap B|}{|A| + |B|}$$

For instance, the dice coefficient between the strings “Mr. John Doe” and “Doe, John” computes to:

$$Dice(\{\text{Mr.}, \text{John}, \text{Doe}\}, \{\text{Doe}, \text{John}\}) = 1 - \frac{4}{5} = \frac{1}{5}$$

In contrast to the Jaccard index, the dice coefficient is not a metric as it does not fulfill the triangle inequality.

SoftTFIDF

The idea of *SoftTFIDF* [Cohen et al., 2003] is to weight common terms lower than uncommon terms. For instance, given the string “Mr. Bertrand Russell”, “Mr.” will receive a lower weight if it is a common term in the data set while “Bertrand” and “Russell” will receive higher weights.

Definition 2.6 (SoftTFIDF) *For two term frequency vectors A and B , their SoftTFIDF distance is defined as:*

$$SoftTFIDF(A, B) = 1 - \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Both frequency vectors must be normalized to $\|A\| = 1$ and $\|B\| = 1$.

Because - if formulated as similarity measure - the SoftTFIDF similarity is equivalent to the cosine of the angle between both vectors, the SoftTFIDF distance is sometimes also called *cosine distance*.

2.3.3 Hybrid Measures

The aim of hybrid distance measures is to combine the advantages of character-based measures and token-based measures. We motivate the need for hybrid distance measures by discussing the main disadvantage of token-based distance measures: While token-based distance measures work well for strings that share many words they fail when typographical errors are present. For instance, a token-based distance measure successfully recognizes the equivalence of “John Doe” and “Doe, John” as both strings share the tokens “John” and “Doe”. On the other hand, the same token-based distance will assign a much higher distance to the pair “John Doe” and “Jon Doe” as due to a typographical error both strings only share one token. Hybrid measures employ character-based measures to allow tokens to be matched that are not perfectly equivalent but contain typographical errors.

A number of hybrid distance measures have been proposed in literature [Naumann and Herschel, 2010]. An experimental comparison of popular hybrid distance measures, token-based measures and character-based measures has been done by Cohen et al. [2003]. Cohen et al. found that, among the compared distance measures, the Monge Elkan distance achieved the best results on the evaluation data sets. The Monge Elkan distance measure combines character-based measures and token-based measures to improve the matching of strings under the presence of errors.

Monge Elkan Distance

The *Monge Elkan Distance* [Monge and Elkan, 1996] measures the distance of two strings by averaging the distance between the tokens in both strings. More precisely, for each token in the first string, the Monge Elkan distance determines the minimum distance to any other token in the second string. The distance between two tokens is determined using a user-defined character based distance measure, such as the Levenshtein distance. Based on that, the overall Monge Elkan distance is computed by taking the average of all found distances.

Definition 2.7 (Monge Elkan Distance) *For two token sets A and B and a character-based distance measure $dist$, the Monge Elkan distance is defined as:*

$$MongeElkan(A, B) = \frac{1}{|A|} \sum_{i=1}^{|A|} \min_{j=1}^{|B|} dist(A_i, B_j)$$

In order to illustrate the advantage of token-based distance measures, we have a closer look at a simple example using the Monge Elkan Distance measure. We compute the distance between two token sets that both denote the same person: $A = \{\text{'Mr.'}, \text{'Bertrand'}, \text{'Russell'}\}$ and $B = \{\text{'Dr.'}, \text{'Bertrant'}, \text{'Russell'}\}$. Using the normalized Levenshtein Distance as internal character-based distance measure the corresponding minimal distance scores for each token in A are:

$$\begin{aligned} \text{NormalizedLevensthein}(\text{Mr.}, \text{Dr.}) &= 1/3 \\ \text{NormalizedLevensthein}(\text{Bertrand}, \text{Bertrant}) &= 1/8 \\ \text{NormalizedLevensthein}(\text{Russell}, \text{Russell}) &= 0 \end{aligned}$$

Based on the internal distance scores, the Monge Elkan distance distance computes to:

$$\text{MongeElkan}(A, B) = \frac{1}{3} \left(\frac{1}{3} + \frac{1}{8} \right) \approx 0.153 \quad (2.4)$$

For comparison, the Jaccard distance between both strings computes to:

$$\text{Jaccard}(A, B) = 1 - \frac{1}{4} = 0.75 \quad (2.5)$$

In this example, the Jaccard distance failed to detect the similarity of both token sets, because both sets only share one exact duplicate. On the other hand, the Monge Elkan distance computed a much lower distance as it took near-duplicates, such as “Bertrand” and “Bertrant”, into account.

2.3.4 Other Measures

Phonetic Distance

The idea of phonetic distance measures is to take the pronunciation of the characters into account when comparing strings. Phonetic distance measures solely distinguish between characters that are pronounced different and ignore character differences between characters that are pronounced similarly. This is achieved by indexing characters in the string while characters with a similar pronunciations are assigned the same index. The resulting distance score is computed by comparing the resulting indices of both string values. As the pronunciation of characters varies between languages, specific phonetic distance measures are usually optimized for a particular language.

The most well-known phonetic distance measure is known as *Soundex* [Russell, 1918, 1922]. Soundex has been developed to compare person names in the United States Census and therefore is optimized for English words. However, variants for other languages have been proposed as well. A detailed description of Soundex and an overview of other widely-used phonetic distance measures can be found in [Elmagarmid et al., 2007].

Q-Grams Distance

While character-based distance measures usually compare strings at the level of single characters, the idea of q-grams distances is to compare multiple consecutive characters from both strings at once. For this purpose, a q-grams distance measure converts both strings to their q-grams before comparing them. Given a whole number q , the q-grams of a string value are generated by sliding a window of the size q over the characters in the string [Gravano et al., 2001a]. The actual comparison of the q-grams, which have been generated from both strings, can be done by using a token-based distance measure, such as the Jaccard distance. Q-grams distance measures can also be combined with phonetic distance measures by using a phonetic indexing scheme to index each q-gram individually [Zobel and Dart, 1995].

Longest Common Substring Distance

The *longest common substring distance* (LCS) [Friedman and Sideli, 1992] measures the distance between two strings based on the total length of all common substrings. The LCS distance is computed in two steps:

In the first step, all substrings that appear in both strings are collected. Usually, only substrings with a configured minimum length are considered, wherein Friedman and Sideli [1992] propose a minimum length of three. The result of the first step is held in a variable l_c , which denotes the summarized length of all found common substrings.

In the second step, the final distance is computed by comparing l_c to the length of the original strings. Friedman and Sideli [1992] proposes four alternatives for computing the final distance:

- (1) l_c divided by the average length of both strings:

$$LCS_{avg}(\sigma_1, \sigma_2) := \frac{2 \cdot l_c}{|\sigma_1| + |\sigma_2|} \quad (2.6)$$

- (2) l_c divided by the minimum length of both strings:

$$LCS_{min}(\sigma_1, \sigma_2) := \frac{l_c}{\min(|\sigma_1|, |\sigma_2|)} \quad (2.7)$$

- (3) l_c divided by the maximum length of both strings:

$$LCS_{max}(\sigma_1, \sigma_2) := \frac{l_c}{\max(|\sigma_1|, |\sigma_2|)} \quad (2.8)$$

- (4) The fourth proposed alternative is to divide l_c by the length of a related string, which is looked up in an external data set.

Variants of LCS that employ different functions to compute the final distance, have been proposed as well [Christen, 2012].

Numeric Distance

Most distance measures that have been presented so far are independent of the specific data format. That is, they can, for instance, work on person names as well as on book titles. However, distance measures have been proposed that are tailored to be used with string values that contain numeric values.

Traditional character-based or token-based distance measures usually achieve weak results on numeric values. The reason for this is that numbers that are close together may not share a single character. For instance, the numbers 999.9 and 1000.0 would not be considered similar by a character-based or token-based distance measure. Another case that is not covered by these measures is the fact that the same number may be represented using different notations (e.g., decimal notation and scientific notation). Numeric distance measures account for these circumstances when comparing different numeric string values.

Numeric distance can also be extended to cover specific numeric values, such as [Christen, 2012]:

Dates: Two dates can be compared by segmenting each date into the components day, month, and year. The total distance in days can be computed based on these three values.

Time: Similar to dates, time values can be compared by computing their distance in seconds.

Geographic Coordinates: Geographic coordinates are usually expressed in latitude and longitude. The distance between two geographic coordinates can be determined by measuring the distance between both points along the surface of the earth.

2.4 Previous Work on Linkage Rules.

In the previous section, we introduced various field matching techniques that can be used to determine the similarity of the values of specific entity properties. For instance, in a person data base, the Jaro-Winkler distance can be used to determine the similarity of the person names while the date distance can be used to assess the similarity of the birth dates. In addition, many other properties may also be suitable for comparison, such as the persons address, employers, the spouse etc. As in many cases the similarity of two entities cannot be determined by comparing a single property of the entities alone, methods to combine the similarity of multiple properties are essential.

A *linkage rule* assigns a similarity score to each pair of entities. In order to determine the overall similarity score, a linkage rule usually combines the individual similarity scores of multiple property comparisons. A number of models have been proposed to represent linkage rules. The remaining of this section will provide an overview of common linkage rule representations. A comprehensive overview of different classifier models, which have been used for entity matching, can be found in [Christen, 2012]. In the subsequent section, we will propose an extended representation that also includes data transformation functions in the linkage rule.

2.4.1 Linear Classifiers

A *linear classifier* (also called a *threshold-based classifier* [Christen, 2012]) combines multiple distance measures by computing the weighted sum of the individual distances [Dey et al., 1998]. The idea of the weights is to control the influence of a particular distance measure to the overall distance score. For instance, a discriminative distance measure may receive a higher weight.

A linear classifier can be formally defined as:

Definition 2.8 (Linear Classifier) *Given a vector of similarity scores \vec{s} and a vector of weights \vec{w} a linear classifier is defined as:*

$$L_{linear}(\vec{s}, \vec{w}) = \vec{s} \cdot \vec{w} = \sum_j w_j s_j$$

The optimal weights can be determined by employing machine learning algorithms (cf. Section 3.4.1)

2.4.2 Threshold-based Boolean Classifiers

A *threshold-based boolean classifier* (also called a *rule-based classifier* [Christen, 2012]) combines multiple similarity tests using boolean operators [Lim

et al., 1993]. A single similarity test consists of a distance measure and a distance threshold. Different models of threshold-based boolean classifiers differ in the set of boolean operators which they allow for combining similarity tests. While the original model by Lim et al. [1993] only allows conjunctions (logical *and*), some extensions also allow disjunctions (logical *or*) and negations (logical *not*) [Christen, 2012].

We now formalize threshold-based boolean classifiers as defined by Lim et al. [1993]:

Definition 2.9 (Threshold-based Boolean Classifier) *Given a vector of similarity scores \vec{s} and a vector of thresholds \vec{t} , a threshold-based boolean classifier is defined as:*

$$L_{boolean}(\vec{s}, \vec{t}) = \bigwedge_j (s_j \geq t_j)$$

2.4.3 Other Representations

While linear classifiers and threshold-based boolean classifiers are the most common linkage rule models, some more expressive representations have been proposed.

AJAX [Galhardas et al., 2001] is declarative language, which uses an SQL-like syntax for expressing various data cleaning tasks and allows the expression of linkage rules. *AJAX* is capable of expressing a linkage rule as a SQL query by proposing several extensions to SQL. Listing 2.1 shows an example of a linkage rule for matching authors in *AJAX* from Galhardas et al. [2001]. The shown linkage rule matches authors by their name. All pairs of

Listing 2.1: Example of a linkage rule in *AJAX* (from [Galhardas et al., 2001]).

```

1 CREATE MATCHING MatchDirtyAuthors
2 FROM DirtyAuthors a1, DirtyAuthors a2
3 LET distance = editDistanceAuthors(a1.name, a2.name, 15)
4 WHERE distance < maxDist(a1.name, a2.name, 15)
5 INTO MatchAuthors
```

author whose names differ at most by a Levenshtein distance of 15% of the maximum length of both names are returned. *AJAX* is capable of expressive linear and threshold-based boolean classifiers and also allows the integration of arbitrary transformation functions, although no learning algorithm has been proposed that allows learning linkage rules for *AJAX*.

Dedupalog [Arasu et al., 2009] is a Datalog-style language for expressing constraints between entities of different types. *Dedupalog* goes beyond the definition of a linkage rule that we used in this thesis. In particular, *Dedupalog* allows the linkage rule to include constraints between different types of entities, similar to the idea of collective entity matching approaches. Listing 2.2 shows two example constraints in *Dedupalog* for matching entities in a bibliographic data set from Arasu et al. [2009]. The first constraint specifies

Listing 2.2: Two example constraints in *Dedupalog* (from [Arasu et al., 2009]).

```

1 Paper*(id1,id2) <-> PaperRef(id1,title1,-),
    ↪ PaperRef(id2,title2,-), TitleSimilar(title1,title2)
2 ¬Author*(x, i, y, j) <- ¬(Wrote(x, i,-), Wrote(y, j,-), Wrote(x,
    ↪ p,-), Wrote(y, p,-), Author*(x, p, y, p))

```

that two citations likely refer to the same paper if their title is similar. The second constraint specifies that two entities that describe authors without any common co-authors unlikely refer to the same author. The complete example by Arasu et al. [2009] consists of eight such constraints. *Dedupalog* distinguishes between hard and soft constraints. The ideal result of an entity matching task on a set of constraints is a set of links that guarantee that no hard constraint is violated while the number of soft constraints that are violated is minimized.

The main disadvantage of AJAX and *Dedupalog* is that no machine learning methods have been proposed that cover their full expressivity. On the other hand, several methods have been proposed for learning linear classifiers and threshold-based boolean classifiers. Section 3.4 provides an overview of previously proposed machine learning methods in the context of learning linkage rules.

In the next section, we propose an expressive linkage rule representation, which subsumes linear and threshold-based boolean classifiers and for which Chapter 3 will propose a supervised machine learning algorithm that covers the full expressivity of the introduced representation.

2.5 An Expressive Linkage Rule Representation

In this section, we introduce an expressive linkage rule representation. We represent a linkage rule as a tree, which is built from four types of operators:

Property Operator: Retrieves all values of a specific property p of each entity, such as its label property. The purpose of the property operator is to enable the access of values from the data set that are used as input for other operators.

Transformation Operator: Transforms the values of a set of property or transformation operators \vec{v} according to a specific data transformation function f^t . Examples of transformation functions include case normalization, tokenization, and concatenation of values from multiple operators. Common transformation functions have been introduced in detail in Section 2.2. Multiple transformation operators can be nested in order to apply a chain of transformations.

Comparison Operator: Evaluates the similarity between two entities based on the values that are returned by two property or transformation operators v_a and v_b by applying a distance measure f^d and a distance threshold θ . Examples of distance measures include Levenshtein, Jaccard, or geographic distance. Common distance measures have been introduced in detail in Section 2.3.

Aggregation Operator: Due to the fact that, in most cases, the similarity of two entities cannot be determined by evaluating a single comparison, an aggregation operator combines the similarity scores from multiple comparison or aggregation operators \vec{s} into a single score according to a specific aggregation function f^a . Examples of common aggregation functions include the weighted average or yielding the minimum score of all operators.

A linkage rule tree is strongly typed [Montana, 1995], i.e., only specific combinations of the four basic operators are allowed. Figure 2.5 specifies the valid structure of a linkage rule. We group the linkage rule operators into two *similarity operators*, which return a similarity score for two given entities, and two *value operators*, which return a set of values for a given entity. The root of the linkage rule is built by a similarity operator, which can either be an aggregation operator or a comparison operator. An aggregation operator in turn may combine the scores of multiple similarity operators. A comparison operator compares the values that are returned by two value operators. Each value operator can either be a property operator or a transformation operator. Multiple transformation operators can be nested. The leaves of a linkage rule tree are always property operators.

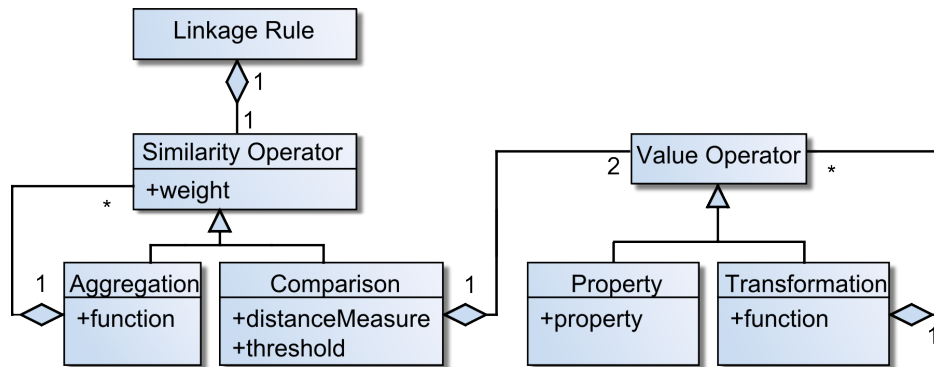


Figure 2.5: Structure of a linkage rule.

2.5.1 Example

Figure 2.6 shows an example of a linkage rule for interlinking geographic locations. In this example, the linkage rule compares the labels as well as the

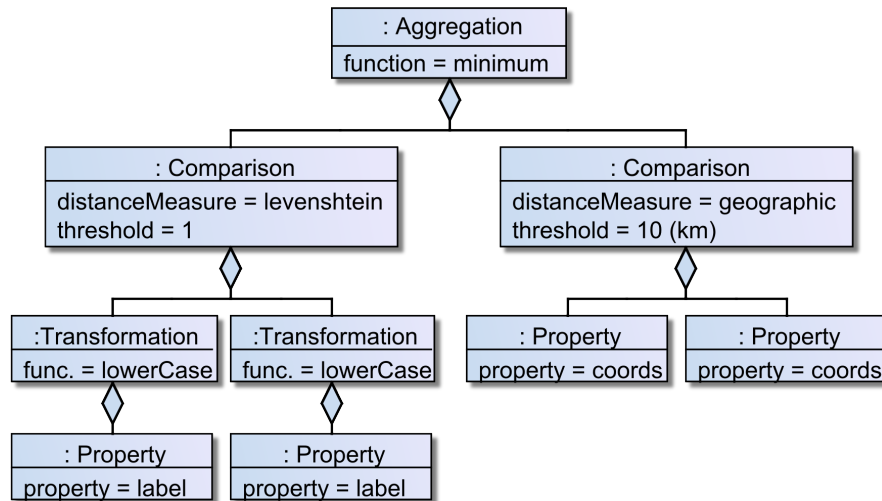


Figure 2.6: Example of a linkage rule that compares locations.

coordinates of the entities. The labels are normalized by converting them to lower case prior to comparing them with the Levenshtein distance. Labels may differ at most by a Levenshtein distance of one, i.e., the minimum number of edit operations that are required to transform the label of one entity into the label of the other entity must be one or less. The geographic coordinates of both entities may be at most 10 kilometers apart. The thresholds of the comparison operators normalize the similarity score to the range $[0, 1]$.

The similarity score of the labels is then aggregated with the geographic similarity score into a single score by using the minimum aggregation, i.e., both values must exceed the threshold of 0.5 in order to generate a link.

2.5.2 Semantics

We now define the semantics of the individual operators: We distinguish between two types of operators: *value operators* and *similarity operators*. While value operators provide a function that yields a discriminative value for a single entity, similarity operators provide a function which determines how similar two given entities are.

Given two data sources A and B , a *value operator* yields a function that returns a discriminative value for a given entity e by which it can be compared to other entities. Thus, it returns a value from the set:²

$$\mathcal{V} := [A \cup B \rightarrow \Phi]$$

Where Φ denotes a (possibly empty) set of values.

We now introduce two value operators: *property operators* and *transformation operators*:

Definition 2.10 (Property Operator) *A property operator retrieves all values of a specific property of an entity:*

$$\begin{aligned} v^p &: P \rightarrow \mathcal{V} \\ p &\mapsto (e \mapsto e.p) \end{aligned}$$

where p denotes the property to be retrieved by the operator.

Definition 2.11 (Transformation Operator) *A transformation operator transforms the input values according to a specific data transformation function:*

$$\begin{aligned} v^t &: (\mathcal{V}^* \times \mathcal{F}^t) \rightarrow \mathcal{V} \\ (\vec{v}, f^t) &\mapsto (e \mapsto f^t(v_1(e), v_2(e), \dots, v_n(e))) \end{aligned}$$

\vec{v} is a vector of operators: v_1, v_2, \dots, v_n . The transformation function f^t may be any function that transforms the value sets provided by the operators into a single value set:

$$f^t : \Phi^n \rightarrow \Phi$$

Transformations	
lowerCase	Converts all values to lower case
tokenize	Splits all values into tokens
stripUriPrefix	Strips the URI prefixes (e.g., <code>http://dbpedia.org/resource/</code>)
concatenate	Concatenates the values from two value operators

Table 2.1: Transformations used in the experiments discussed in Section 3.5 and Section 4.4.

Although in general we do not impose any restriction on the concrete transformation functions that can be used, Table 2.1 lists the functions that we employed in our experiments. An example of a transformation operator that concatenates the first and the last name of entities about persons is:

$$v^t((v^p(\text{firstName}), v^p(\text{lastName})), \text{concatenate})$$

Note that transformations also may be nested.

A *similarity operator* returns a function that assigns a value from the interval $[0,1]$ to each pair of entities:

$$\mathcal{S} := [A \times B \rightarrow [0, 1]]$$

We consider two types of similarity operators: *comparison operators*, which compare the result of two value operators, and *aggregation operators*, which aggregate multiple similarity operators.

Definition 2.12 (Comparison Operator) *Given two value operators v_a and v_b , a comparison operator is defined by:*

$$s^c : (\mathcal{V} \times \mathcal{V} \times \mathcal{F}^d \times \mathbb{R}) \rightarrow \mathcal{S}$$

$$(v_a, v_b, f^d, \theta) \mapsto \left((e_a, e_b) \mapsto \begin{cases} 1 - \frac{d}{\theta} & \text{if } d \leq \theta \\ 0 & \text{if } d > \theta \end{cases} \right)$$

with $d := f^d(v_a(e_a), v_b(e_b))$

f^d defines the distance measure that is used to compare the values of both value operators:

$$f^d : \Phi \times \Phi \rightarrow \mathbb{R}$$

² $[X \rightarrow Y]$ denotes Y^X i.e., the space of all functions $X \rightarrow Y$

Distance Measures	
levenshtein	Levenshtein distance
jaccard	Jaccard distance coefficient
numeric	The numeric difference
geographic	The geographical distance in meters
date	Distance between two dates in days

Table 2.2: Distance functions used in the experiments discussed in Section 3.5 and Section 4.4.

Table 2.2 lists the distance functions that we employed in our experiments. An example of a comparison operator that compares the name of the entities in the first data set with the lower cased labels of the entities in the second data set using the Levenshtein distance is:

$$s^c(v^p(\text{name}), v^t((v^p(\text{label})), \text{lowerCase}), \text{levenshtein}, 1)$$

Definition 2.13 (Aggregation Operator) *Given a set of similarity operators s an aggregation operator is defined as:*

$$\begin{aligned} s^a : (\mathcal{S}^* \times \mathbb{N}^* \times \mathcal{F}^a) &\rightarrow \mathcal{S} \\ (\vec{s}, \vec{w}, f^a) &\mapsto ((e_a, e_b) \mapsto f^a(s_e, w)) \\ \text{with } s_e &:= (s_1(e_a, e_b), s_2(e_a, e_b), \dots, s_n(e_a, e_b)) \end{aligned}$$

\vec{w} denotes the weights that are used by the aggregation function f^a to combine the values:

$$f^a : \mathbb{R}^n \times \mathbb{N}^n \rightarrow \mathbb{R}$$

The first argument contains the similarity scores returned by the operators of this aggregation while the second argument contains a weight for each of the operators.

Table 2.3 lists the aggregation functions that we employed in our experiments. Note that aggregations can be nested, i.e., non-linear hierarchies can also be expressed.

2.5.3 Discussion

Our representation of a linkage rule differs from other commonly used representations in a number of ways:

Aggregations	
max	$f^t(s, w) := \max(s)$
min	$f^t(s, w) := \min(s)$
wmean	$f^t(s, w) := \frac{\sum_{i=1}^n w_i s_i}{\sum_{i=1}^n w_i}$

Table 2.3: Aggregation functions used in the experiments discussed in Section 3.5 and Section 4.4. The *max* and *min* aggregations ignore the supplied weights.

Matching Between Different Schemata: It allows the matching between data sets that use different schemata. This is enabled by two additions: Firstly, by allowing two property operators for each comparison and secondly by introducing data transformations. For example, a data source that uses the FOAF vocabulary [Brickley and Miller, 2005] may represent person names using the `foaf:firstName` and `foaf:lastName` properties while a data source using the DBpedia ontology may represent the same names using just the `dbpedia:name` property. In order to compare entities expressed in different schemata or data formats, their values have to be normalized prior to comparing them for similarity. In this example we could achieve this in two ways: We could concatenate `foaf:firstName` and `foaf:lastName` into a single name before comparing them to `dbpedia:name` by using a character-based distance measure, such as the Levenshtein distance. Alternatively, we could split the values of `dbpedia:name` using a tokenizer and compare them to the values of `foaf:firstName` and `foaf:lastName` by using a token-based distance measure, such as the Jaccard coefficient.

Handling Noisy Data Sets: Another motivation for transformation operators is the matching of noisy data sets. A common example is data sources that contain values using an inconsistent letter case (e.g., “iPod” vs. “IPOD”). A way to address case inconsistency is to normalize all values to lower case prior to comparing them.

Representing Non-linear Classifiers: Arasu et al. [Arasu et al., 2010] categorize widely used approaches for representing linkage rules as *threshold-based boolean classifiers* and *linear classifiers*. In Section 3.5, we will show that the performance of entity matching can be improved with more expressive representations in that we allow aggregation operators to be nested in order to represent non-linear classifiers beyond pure boolean classifiers.

2.5.4 Representing Common Classifiers

We now show that the introduced linkage rule representation is capable of representing linear classifiers and threshold-based boolean classifiers. We further state in what respects our representation extends both models.

Linear Classifiers

Within the proposed representation, a linear classifier, as defined in Section 2.4.1, can be composed from a single aggregation that uses the weighted average aggregation function. More formally, given a vector of comparison operators \vec{s} and a vector of weights \vec{w} , a linear classifier can be represented as:

$$s_{linear}(\vec{s}, \vec{w}) := s^a(\vec{s}, \vec{w}, wmean) = (e_a, e_b) \mapsto \frac{\sum_{i=1}^n w_i s_i(e_a, e_b)}{\sum_{i=1}^n w_i}$$

An example of a linear classifier based on two comparisons is:

$$s^a((s^c(p_1, p_1, levenshtein, 1), s^c(p_2, p_2, levenshtein, 1)), (3, 1))$$

In this example, two properties p_1 and p_2 are compared using the Levenshtein distance measure together with a distance threshold of 1. Both comparisons are aggregated using a weight vector that assigns a weight of 3 to the first comparison and a weight of 1 to the second comparison.

Our representation subsumes linear classifiers and extends them in 3 ways:

- (1) It includes data transformations.
- (2) It generalizes the aggregation function, i.e., allows functions other than *wmean*.
- (3) It allows aggregations to be nested in order to express non-linear classifiers beyond pure boolean classifiers.

Threshold-based Boolean Classifiers

Within the proposed representation, a threshold-based boolean classifier, as defined in Section 2.4.2, can be composed from a single aggregation that uses the minimum aggregation function. Note that using the minimum function is equivalent to the conjunction of all comparisons. More formally, given a vector of comparison operators \vec{s} , where each comparison operator compares a pair of properties using a specific distance measure f^d and a threshold θ , a threshold-based boolean classifier can be represented as:

$$s_{boolean}(\vec{s}) := s^a(\vec{s}, \vec{0}, min) = (e_a, e_b) \mapsto min\{s_i(e_a, e_b) : 1 < i \leq n\}$$

The aggregation operator is provided with the vector of comparisons and the minimum aggregation function. As the minimum aggregation function ignores the weight vector, the zero vector $\vec{0}$ is provided as dummy.

Our representation subsumes threshold-based boolean classifiers and extends them in 2 ways:

- (1) It includes data transformations.
- (2) It generalizes the aggregation function, i.e., allows functions other than *min*.

2.6 Summary

In this chapter, we gave a detailed overview of linkage rules. In the course of this, we introduced well-known data transformation and standardization methods, which can be used to normalize values prior to matching individual entity pairs. For the purpose of matching entity pairs, we introduced distance measures that can be used to compare entity values. Further, we introduced different methods for combining multiple distance measures in order to yield a single similarity score.

The main subject of this chapter is the first contribution of this thesis: the proposal of of an expressive linkage rule representation. The proposed representation expresses linkage rules as operator trees that can be understood and modified by humans. The main contributions of the proposed representation include:

- (1) It may include chains of data transformations to normalize values prior to comparison.
- (2) It is capable of combining different distance measures non-linearly beyond pure boolean classifiers.
- (3) We showed how the proposed representation is capable of expressing threshold-based boolean classifiers and linear classifiers that have been previously used for entity matching.

By proposing an expressive linkage rule representation, we laid the foundation for the GenLink learning algorithm that will be introduced in the next chapter.

Chapter 3

Supervised Learning of Linkage Rules

The previous chapter has shown how linkage rules can be used to specify detailed conditions on how two entities are compared for equality. We also proposed an expressive linkage rule representation, which is capable of combining different distance measures non-linearly and may include chains of data transformations to normalize values prior to comparison. However, writing and fine-tuning these linkage rules manually for each type of entity that is to be matched is hard for the following reasons:

- (1) The author needs to choose discriminative properties of the entities to be interlinked together with a distance measure and an appropriate distance threshold.
- (2) For data sets that are noisy or use different data formats, property values need to be normalized by employing data transformations.
- (3) As comparing two entities by a single property usually is not sufficient to decide whether both entities describe the same real-world object, linkage rules commonly need to aggregate the similarity of multiple property comparisons using appropriate aggregation functions.

A way to reduce this effort is to use supervised learning in order to generate linkage rules from existing reference links. Creating reference links is much easier than writing linkage rules as it requires no previous knowledge about similarity computation techniques or the specific linkage rule representation that is used by the entity matching system at hand. Reference links can be created by domain experts who confirm or reject the equivalence of a number of entity pairs from the data sets. For instance, reference links

for locations in a geographical data set can be created by labeling pairs of locations as correct or incorrect. Figure 3.1 shows an example of such an entity pair. In the given example, the pair is to be declined as both entities

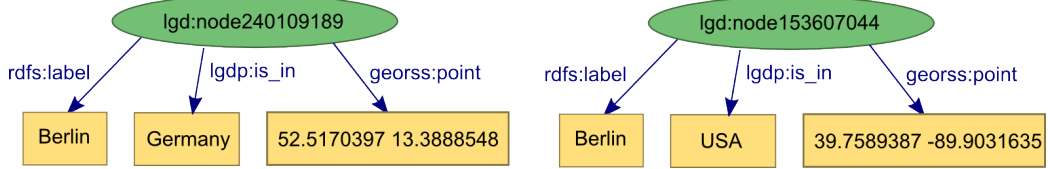


Figure 3.1: Example of an entity pair in a geographical data set.

represent different real-world locations. After a set of reference links has been created, a supervised learning algorithm may generate a linkage rule by using the reference links as training data.

In this chapter, we propose the GenLink algorithm for learning linkage rules, which is the second key contribution of this thesis. GenLink employs genetic programming in order to learn linkage rules from a set of existing reference links. Following genetic programming, GenLink starts with an initial population of candidate solutions, which is then iteratively evolved by applying a set of genetic operators.

3.1 Problem Definition

In this chapter, we consider the problem of learning a linkage rule from a set of reference links¹:

Definition 3.1 (Linkage Rule Learner) *The purpose of a learning algorithm for linkage rules is to learn a linkage rule from a set of reference links:*

$$m : 2^{(A \times B)} \times 2^{(A \times B)} \rightarrow (A \times B \rightarrow [0, 1])$$

The first argument denotes a set of positive reference links, while the second argument denotes a set of negative reference links. The result of the learning algorithm is a linkage rule that covers as many reference links as possible while generalizing to unknown pairs.

¹The shown definition is based on [Rastogi et al., 2011]. In contrast to the definition by Rastogi et al. [2011], the definition that is presented in this work returns a linkage rule instead of returning the final links directly, i.e., we distinguish between the learning of a linkage rule and its execution, while Rastogi et al. consider a more general case.

3.2 Genetic Programming

Genetic programming is an extension of genetic algorithms [Holland, 1975], which has been first proposed, in tree-based form, by Cramer [1985]. Genetic programming can be used whenever candidate solutions can be represented as operator trees, such as it is the case for the linkage rule representation that we propose in this thesis.

Figure 3.2 shows the basic control flow for genetic programming [Poli et al., 2008]. Genetic programming algorithms usually start with a random

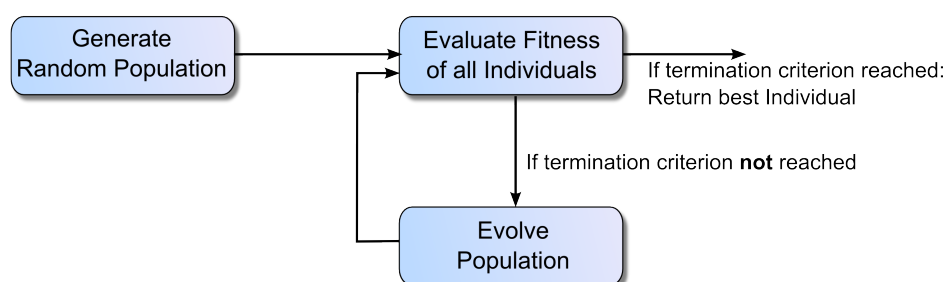


Figure 3.2: Control flow for genetic programming.

population that contains a set of initial candidate solutions (also called individuals). In each iteration, the genetic algorithm determines the fitness of all individuals in the current population and evolves a new population. The evolution of the population stops as soon as either the configured maximum number of iterations has been exceeded or an optimal solution has been found.

3.2.1 Generating the Initial Population

In genetic programming the initial population of individuals is usually generated randomly [Koza, 1993]. A genetic programming system may apply constraints on the structure of the trees that are generated (Also called strongly typed genetic programming [Montana, 1995]). For instance, when learning linkage rules, we do not want to allow arbitrary combinations of the different operators. Instead, we want to add a couple of constraints, such as to require that each aggregation operator is only allowed to have other aggregation operators or comparison operators as direct children. The valid structure of the linkage rule representation that is used in this work has been described previously in Section 2.5.

If there is already some knowledge about specific properties of the desired solution, a common improvement over generating a completely random

population is to only generate trees that possess the desired properties [Poli et al., 2008]. For instance, if specific parts of the linkage rule, such as the entity properties to be compared, can be inferred, the population can be seeded with linkage rules that include these comparisons.

3.2.2 Evolving the Population

In each iteration, the genetic algorithm evolves a new population from the individuals in the current population. The population is evolved using three genetic operators [Koza, 1993]:

- The *reproduction operator* copies an individual from the population without any modification.
- The *crossover operator* recombines two individuals from the population into a single individual.
- The *mutation operator* applies a random change to an individual from the population.

The evolved population is generated by repeatedly selecting individuals from the population, applying a genetic operator to each selected individual and adding the generated individual to the new population. Each individual in the evolved population is generated in three steps:

- (1) A genetic operator is chosen at random. As the crossover operator is often considered the primary operator in genetic programming [Koza, 1993], it is usually selected with a higher probability than the other operators.
- (2) One or two individuals are selected from the current population. The number of individuals that are selected depends on the chosen genetic operator. While the crossover operator requires two individuals, the reproduction and the mutation operators each require one individual. The individuals are selected according to a *selection method* that favors fitter individuals.
- (3) The genetic operator is applied to the selected individuals. The generated individual is added to the evolved population.

New individuals are generated until the evolved population reaches the size of the current population.

Selection Method

The idea of the selection method is that the individuals in the current population are not selected with equal probability. Instead, individuals with a higher fitness (i.e., which are closer to the desired solution) are more likely to be selected. In that respect, the selection method follows natural selection as fitter individuals are more likely to breed offsprings. In particular, the individuals are selected from the population based on two functions: The *fitness function* and the *selection method*.

The purpose of the *fitness function* is to assign a score to each individual that indicates how close the given individual is to the desired solution. In supervised learning, the fitness function is typically computed based on user-provided training data.

Based on the fitness score of each individual, the *selection method* selects the individuals to be evolved. Popular selection methods are *fitness proportionate selection* and *tournament selection* [Goldberg and Deb, 1991; Blickle and Thiele, 1996]. In fitness proportionate selection (in literature also called *roulette-wheel selection*) every individual is selected with a probability proportional to its fitness [Holland, 1975]. In tournament selection each individual is selected by choosing the fittest individual from a set of randomly selected individuals [Blickle and Thiele, 1995]. The number of individuals that are selected for each tournament must be specified.

Reproduction Operator

The reproduction operator copies a selected individual from the current population to the new population without applying any modification. Apart from using the reproduction operator in the same way as the crossover and mutation operators on individuals that are chosen by the selection method, it can also be used to perform an *elitist strategy* Goldberg [1989]. In an elitist strategy, the top-k individuals are reproduced prior to evolving the population, i.e., they are directly copied from the current population to the new population. In that case, the remaining individuals are evolved solely by the crossover and mutation operators. The motivation of using an elitist strategy is to retain good individuals from the current population in the evolved population.

Crossover Operator

The crossover operation generates a new individual by combining two selected individuals. The most commonly used crossover operator is *subtree crossover* [Koza, 1993]. Subtree crossover starts by randomly selecting a

node in both individuals. The new individual is then created by replacing the selected node in the first individual with the node that has been selected in the second individual including its complete subtree. Figure 3.3 illustrates a subtree crossover operation. In this example, subtree crossover randomly selected operator `Op3` in the first individual and operator `Op4` in the second individual. Both individuals are recombined by replacing `Op3` in the first individual with `Op4` in the second individual, including their subtrees. The modified first individual constitutes the result of the subtree crossover.

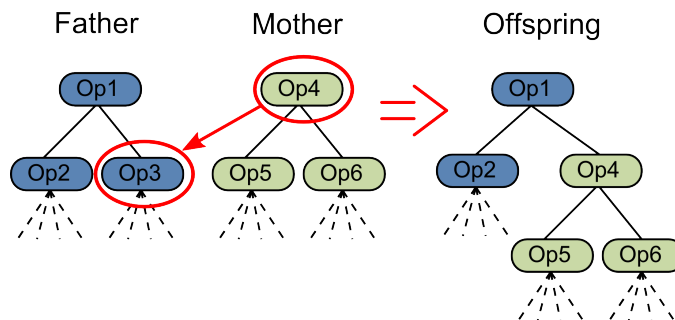


Figure 3.3: Example of subtree crossover.

Note that, although there are variants of the crossover operator that return two offsprings, as there is no added value in yielding two offsprings (the same result can be produced by executing the same crossover operator twice, interchanging the operands the second time), it is rarely used in practice. Thus, we will only consider crossover operators yielding one offspring.

If strongly typed genetic programming is used, subtree crossover must assert that the generated tree satisfies all type constraints. This is usually done by restricting the random selection of the node in the second individual to nodes that are of the same type as the already selected node in the first individual [Montana, 1995].

Mutation Operator

In genetic programming, mutation is often considered a minor genetic operator and some approaches do not employ mutation at all [Koza, 1993]. The most commonly used mutation operator is known as *headless chicken crossover*. Headless chicken crossover executes a crossover operation between the selected individual and a randomly generated new individual. If the root node is selected as the crossover point in the first individual this results in the complete replacement of the given individual with a randomly generated individual.

3.2.3 Bloating Control

In general, genetic programming does not restrict the size of the trees that represent the individuals, i.e., trees are allowed to grow arbitrarily large [Blickle and Thiele, 1994]. The uncontrolled growth of the individuals during the evolution of the population is a well-known problem in genetic programming. This phenomenon, known as *bloating* [Langdon and Poli, 1997] in literature, causes the individuals to develop redundant parts. Such redundant parts, which can be removed from the individual without changing its fitness, are called *introns* [Angeline, 1994].

While various methods have been developed to control bloating [Luke and Panait, 2006], the most popular method is *parsimony pressure* [Zhang and Mühlenbein, 1995]. The idea of parsimony pressure is to modify the fitness function to penalize big trees in order to force the algorithm to favor smaller trees over bigger ones.

3.3 The GenLink Algorithm

GenLink extends the genetic programming algorithm, as described in the previous section, in three notable ways:

- (1) The initial population is not generated completely on random. Instead, a seeding algorithm ensures that only linkage rules that compare properties that contain similar values are part of the initial population.
- (2) Rather than using subtree crossover, GenLink employs a set of specific crossover operators. Each crossover operator only modifies one aspect of a linkage rule, such as the distance thresholds or chains of transformations. The use of specific crossover operators constitutes one of the main contributions of GenLink and will be described in detail in Section 3.3.3.
- (3) In GenLink, we use an approach to prevent bloating of linkage rules that goes beyond parsimony pressure.

Section 3.5.5 will evaluate in detail how each of these extensions contributes to the overall performance.

The pseudocode of GenLink is given in Listing 3.1.

The algorithm starts by generating an initial population of linkage rules according to the method described in Section 3.3.1. After the initial population has been generated, it is iteratively evolved. In each iteration, a new

Listing 3.1: Pseudocode of the GenLink algorithm. The specific parameter values used in our experiments are listed in Section 3.5.2

```

1  $P \leftarrow$  generate initial population
2 while(max. iterations nor 100% F-measure reached) {
3    $P' \leftarrow \emptyset$ 
4   while( $|P'| < populationsize$ ) {
5      $r_1, r_2 \leftarrow$  select two linkage rules from  $P$ 
6      $op \leftarrow$  select random crossover operator
7      $p \leftarrow$  random number from interval  $[0,1]$ 
8     if( $p < mutationprobability$ ) {
9        $r_r \leftarrow$  generate random linkage rule
10       $P' \leftarrow P' \cup op(r_1, r_r)$ 
11    } else {
12       $P' \leftarrow P' \cup op(r_1, r_2)$ 
13    }
14  }
15   $P \leftarrow P'$ 
16 }
17 return best linkage rule from  $P$ 

```

population is generated by generating new linkage rules from selected rules in the existing population until the population size is reached.

A new linkage rule is generated according to the following steps: First, two linkage rules are selected from the population according to the selection method described in Section 3.3.2. In addition, a random crossover operator is selected from the set of available crossover operators. The basic idea of our approach is to provide a specific crossover operator for each aspect of a linkage rule. For instance, the threshold crossover operator only modifies the threshold of a comparison while the transformation crossover operator combines the transformations of both linkage rules. The set of crossover operators that have been employed is described in Section 3.3.3. The selected operator is used to either mutate one of the selected linkage rules or to combine both linkage rules into a new linkage rule. In the case of mutation, a headless chicken crossover [Jones, 1995] is performed, i.e., the chosen crossover operator is applied to the selected linkage rule and a randomly generated linkage rule.

The algorithm stops when either a predefined number of iterations is reached or when a linkage rule in the population reaches an F-measure of 100%. The best linkage rule in the final population is returned by the algorithm. A bloating control method, as described in Section 3.3.4, avoids that

linkage rules grow indefinitely.

3.3.1 Generating the Initial Population

In genetic programming algorithms, the initial population is usually generated randomly. Previous work has shown that starting with a fully random population works well on some record linkage data sets [Carvalho et al., 2008]. Allowing the entire range of possible linkage rules leads to a large search space as the population must include linkage rules for all possible comparisons of the available properties.

Two circumstances increase the search space (i.e., the set of all possible linkage rules) considerably: Firstly, if data sets that are represented using different schemata are to be matched the search space includes all possible property pairs from the source and target data set. Secondly, data sets with a high number of properties make it difficult to find discriminative properties. For instance, while the Cora and Restaurant data sets, which have been used by de Carvalho et al. to evaluate the performance of their genetic programming approach [de Carvalho et al., 2012], only provide four respectively five properties, many data sets in the Web of Data, such as DBpedia, provide over 100 properties².

In order to reduce the size of the search space, we employ a simple algorithm that preselects property pairs that hold similar values: Before the population is generated, we build a list of property pairs that hold similar values as described below. Based on that, random linkage rules are built by selecting property pairs from the list and building a tree by combining random data transformations, comparisons and aggregations.

Finding Compatible Properties

The purpose of this step is to generate a list of pairs of properties that share at least one token on any of their values. For each possible property pair, the values of the entities referenced by a positive reference link are analyzed. This is done by tokenizing and lowercasing the values and generating a new property pair of the form $(p1, p2)$ if there is a distance measure in a provided list of functions according to which two tokens are similar, given a certain threshold θ_d . The pseudocode is given in Listing 3.2.

Figure 3.4 illustrates a simple example with two entities. In this example, the following two property pairs are generated: $(label, label)$ and $(director, directorName)$.

²The number of properties for each data set that has been used for the experimental evaluation are listed in Table 3.4

Listing 3.2: Find compatible properties given a set of reference links R^+ , a list of distance measures F^d and a distance threshold θ

```

1 pairs  $\leftarrow \emptyset$ 
2 for all  $(e_a, e_b) \in R^+$  {
3   for all properties  $e_a.pi$  and  $e_b.pj$  {
4     for all distance measures  $f^d$  in  $F^d$  {
5        $v_a \leftarrow tokenize(lowerCase(e_a.pi))$ 
6        $v_b \leftarrow tokenize(lowerCase(e_b.pj))$ 
7       if  $(f^d(v_a, v_b) < \theta_d)$  add  $(pi, pj)$  to pairs
8     }
9   }
}
return pairs

```

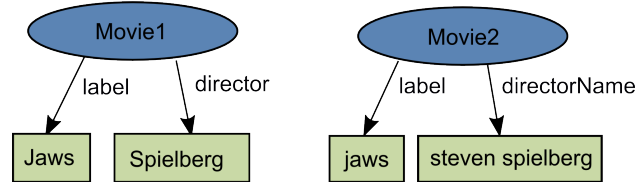


Figure 3.4: Finding compatible properties

Choosing the list of similarity measures F^d constitutes a trade-off between runtime performance and learning performance. In our experiments, we chose the Levenshtein distance with a threshold of one, but did not include other distance measures. While this yielded a high performance on the evaluation data sets, it may also fail to discover properties with similar values in some other data sets. For instance, by restricting the search to the Levenshtein distance, properties that use different units of measurements would not be discovered. In order to cover cases that are now missed, the list could be extended by adding more distance measures.

Generating a Random Linkage Rule

A random linkage rule is generated according to the following rules: First of all, a linkage rule is built consisting of a random aggregation and up to two comparisons. For each comparison, a random pair from the pre-generated list of compatible properties is selected. In addition, with a possibility of 50%, a random transformation is appended to each property.

Note that, although the initial linkage rule trees are very small, this does not limit the algorithm from growing bigger trees by recombining the initial linkage rules using the genetic operators.

3.3.2 Evolving the Population

Starting with the initial population, the genetic algorithm breeds a new population by evolving selected linkage rules using the genetic operators. In order to determine how linkage rules are selected from the population, a genetic programming algorithm needs to specify two functions: the *fitness function* and the *selection method*.

The purpose of the *fitness function* is to assign a value to each linkage rule that indicates how close the given linkage rule is to the desired solution. A disadvantage of using the F-measure as fitness function is that it may yield skewed results if the number of positive and negative reference links is unbalanced as it only takes the true negative rate into account [Powers, 2011]. We use *Matthews correlation coefficient* (MCC) as fitness measure. Matthews correlation coefficient [Matthews, 1975] is defined as the degree of the correlation between the actual and predicted classes:

$$\text{MCC} = \frac{n_{tp} \times n_{tn} - n_{fp} \times n_{fn}}{\sqrt{(n_{tp} + n_{fp})(n_{tp} + n_{fn})(n_{tn} + n_{fp})(n_{tn} + n_{fn})}}$$

n_{tp} , n_{tn} , n_{fp} and n_{fn} denote the number of true positives, true negatives, false positives and false negatives. All four values are computed based on the provided reference links (ignoring the remaining part of the data set).

Note that the MCC does assign an equal weight to the number of false positives and false negatives. In some use cases this may not be desirable because different costs are associated with each kind of error. For instance, in a medical application false positives may induce a higher cost than false negatives. A fitness measure that puts a bigger weight on one kind of error can be used instead to emphasize such a bias.

In order to prevent linkage rules from growing indefinitely, we extend the fitness function to penalize linkage rules based on their number of operators:

$$\text{fitness} = (1 - pf) \times \text{mcc} - pf \times \text{operatorcount}$$

The particular value of the penalty factor pf determines the extend to which large linkage rules are penalized. Choosing an appropriate value of the penalty factor is important as setting it to big decreases the learning performance as it prevents linkage rules from growing to their optimal size. On the other hand, small values may not punish linkage rules with redundant parts sufficiently. For our experiments we empirically determined the largest penalty factor that does not decrease the learning performance and fixed it to 0.05.

Based on the fitness of each linkage rule, the *selection method* selects the linkage rules to be evolved. As selection method, we chose tournament

selection as it has been shown to produce strong results in a variety of genetic programming systems [Koza et al., 2005] and is easy to parallelize.

3.3.3 Crossover Operators

A crossover operators accepts two linkage rules and returns an updated linkage rule that has been built by recombining parts of both linkage rules. Instead of using subtree crossover, which is commonly used in genetic programming, we use a set of specific crossover operators that are tailored to the structure of a linkage rule. The basic idea of our proposed crossover operators is that each operator learns a different aspect of a linkage rule. We propose seven crossover operators:

- The *transformation crossover operator* builds chains of transformations.
- The *distance measure crossover operator* selects appropriate distance measures.
- The *threshold crossover operator* learns the corresponding distance thresholds.
- The *combine operators crossover operator* learns combinations of multiple comparisons.
- The *aggregation function crossover operator* selects aggregation functions, which combine multiple comparisons.
- The *weight crossover operator* learns the weights, which determine how multiple comparisons and aggregations are aggregated based on the aggregation functions (e.g., weighted average).
- The *aggregation hierarchy crossover operator* builds hierarchies of aggregations and comparisons in order to assemble complex linkage rule trees.

The following subsections describe each of these operators in detail. For each crossover operation, an operator from this set is selected randomly and applied to two selected linkage rules. As mentioned earlier, we reduce mutation to a crossover operation with a randomly generated new linkage rule (i.e., *headless chicken crossover*). The contribution of the proposed operators to the learning performance over subtree crossover is evaluated experimentally in Section 3.5.5.

Transformation Crossover Operator

Entity matching systems usually provide a large collection of data transformation functions to normalize string values prior to comparison³. In Section 2.2, we provided an overview of commonly used transformation functions. The task of *transformation crossover* is to assemble chains of transformation functions that normalize string values that are to be compared. Chains of transformations are built by merging the transformation operators of both provided linkage rules.

Transformation crossover starts by randomly selecting an upper and a lower transformation operator in each linkage rule. The next step is to recombine the paths between the upper and the lower transformation by executing a two point crossover. Finally, duplicated transformations are removed. The pseudocode is given in Listing 3.3.

Listing 3.3: Pseudocode of the transformation crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   t1upper, t1lower ← random transformations from r1
3   t2upper, t2lower ← random transformations from r2
4
5   return r1 with:
6     t1upper replaced by t2upper
7     t2lower.v replaced by t1lower.v
8 }
```

Figure 3.5 illustrates an application of the transformation crossover operator. In this example, the `tokenize` operator was selected as both upper and lower transformation in the first linkage rule. In the second linkage rule, the `tokenize` operator was selected as upper transformation while the `stem` operator was selected as lower transformation. The `tokenize` operator in the first linkage rule is then replaced by the path between the upper and the lower transformation in the second linkage rule.

As both linkage rules have been selected based on natural selection, transformation crossover effectively tries a transformation path that already works well in the second linkage rule on the first linkage rule. In some cases, it will

³Note that although our representation of a linkage rule does include data transformations, traditional entity matching systems typically do not include data transformations into the linkage rule, but rely on a preceding data preparation stage instead [Elmagarmid et al., 2007].

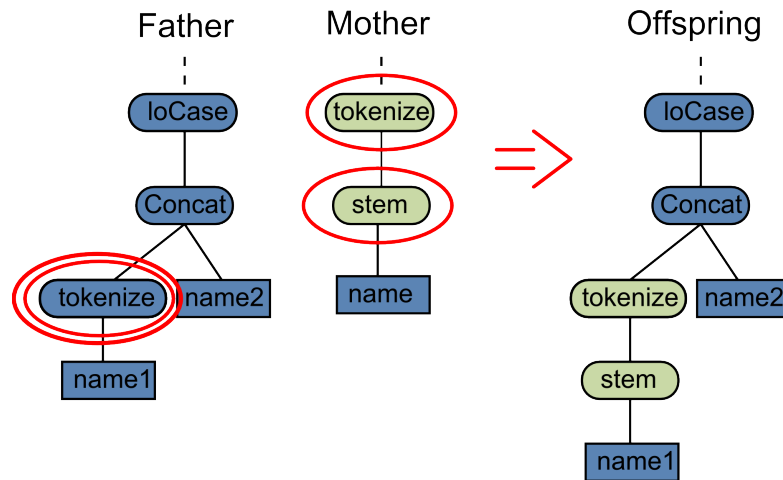


Figure 3.5: Example application of the transformation crossover operator.

recombine two transformation paths that both originate from the same property. In these cases, it is obvious that recombining two already well-proven transformation chains may yield improvements when comparing the given property values. However, there are clearly also cases when two transformation paths are recombined that originate from different properties. In these cases, transformation crossover effectively applies a chain of transformations that works on one property to another property. This may still yield improvements based on the assumption that multiple properties in a data set adhere to similar characteristics. For instance, if the values of one property do not follow a consistent letter case, it is likely that other properties in the same data set suffer from similar problems and thus can be normalized with the same transformations.

Distance Measure Crossover Operator

While the transformation crossover operator evolves chains of transformations to normalize input values, *distance measure crossover* selects distance measures to compare the normalized values. The task of distance measure crossover is to evaluate different distance measures.

Distance measure crossover selects one comparison at random in each linkage rule and interchanges the distance measures of both comparisons. For example, it may select a comparison that uses the Levenshtein distance measure in the first linkage rule and a comparison that uses the Jaccard distance measure in the second linkage rule and then interchange these two functions. The pseudocode for distance measure crossover is given in Listing 3.4.

Figure 3.6 illustrates a simple distance measure crossover on two linkage

Listing 3.4: Pseudocode of the distance measure crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   cmp1 ← random node of nodeType from r1
3   cmp2 ← random node of nodeType from r2
4
5   return r1 with cmp1.fd ← cmp2.fd
6 }

```

rules.

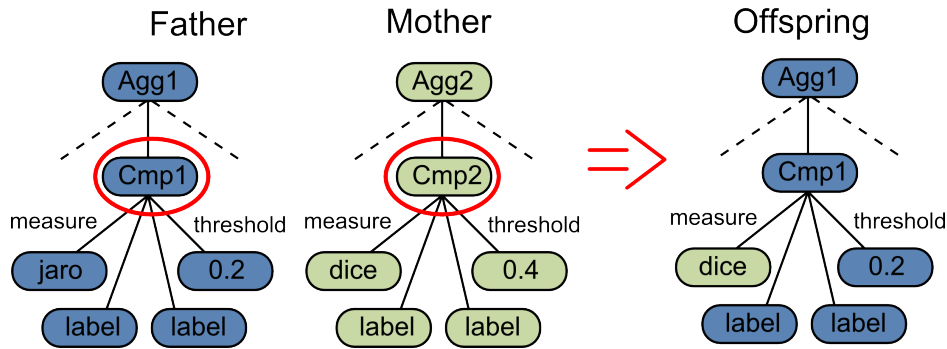


Figure 3.6: Example application of the distance measure crossover operator.

Threshold Crossover Operator

Finding the optimal distance threshold for each distance measure can be a difficult task. If the threshold is set to low, many near-duplicates will not be considered a match; however, if the threshold is set to high, many distinct entities with similar values may be found to be matching.

Threshold crossover accounts for the learning of distance thresholds by combining the distance thresholds of both linkage rules. For this purpose, one comparison operator is selected at random in each linkage rule. The new threshold is then set to the average of both comparisons. The pseudocode is given in Listing 3.5. Figure 3.7 illustrates a threshold crossover on two linkage rules.

Combine Operators Crossover Operator

A linkage rule usually uses an aggregation of multiple comparisons. For instance a linkage rule for matching persons may combine the similarities of

Listing 3.5: Pseudocode of the threshold crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   cmp1 ← random comparison from r1
3   cmp2 ← random comparison from r2
4
5   return r1 with (cmp1. $\theta$  ←  $0.5 \cdot (\text{cmp1}.\theta + \text{cmp2}.\theta)$ )
6 }

```

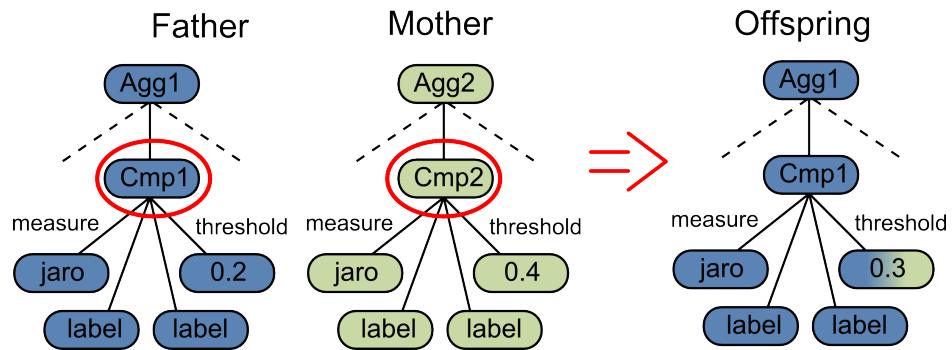


Figure 3.7: Example application of the threshold crossover operator.

their names and their birth dates into a single similarity score. Another linkage rule that combines the similarities of their birth names and their home addresses may do comparably well on the same data set. As it is often not clear which particular combination of comparisons yields good results, evaluating different combinations is essential for finding the optimal combination.

In order to build new combinations of comparisons, *combine operators crossover* combines aggregations from both linkage rules. For this, it selects two aggregations, one from each linkage rule and combines their comparisons. The comparisons are combined by selecting all comparisons from both aggregations and removing each comparison with a probability of 50%. Note that the comparisons are exchanged including the complete subtree, i.e., the distance measures as well as existing transformations are retained. For example, it may select an aggregation of a label comparison and a date comparison in the first linkage rule and an aggregation of a label comparison and a comparison of the geographic coordinates in the second linkage rule. In this case, the operator replaces the selected aggregations with a new aggregation that contains all four comparisons and then removes each comparison with a probability of 50%. The pseudocode is given in Listing 3.6. Figure 3.8 illustrates a simple application of combine operators crossover on two linkage rules.

Listing 3.6: Pseudocode of the combine operators crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   agg1 ← random aggregation from r1
3   agg2 ← random aggregation from r2
4
5   ops ← ∅
6   for all operators o in agg1 and agg2 {
7     p ← random number from interval [0,1]
8     if (p > 0.5)
9       add o to ops
10  }
11
12  return r1 with agg1.operators ← ops
13 }

```

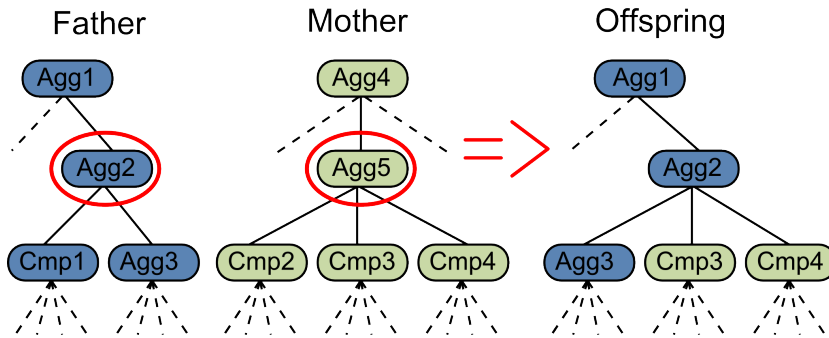


Figure 3.8: Example application of the combine operators crossover operator.

Note that, as a general principle of crossover operations genetic algorithms, the crossover operator is not blindly guessing new combinations. Both linkage rules have been selected according the selection function based on the premise that they are already doing well on the given data set. For that reason, the combine operators crossover operator is recombining comparisons that are already yielding good results on the current data set.

Aggregation Function Crossover Operator

An aggregation function combines the similarity scores of set of comparisons and aggregations. The way in which specific similarity scores are combined, depends on the particular aggregation function. In Section 2.5, we already presented commonly used aggregation functions, such as the weighted aver-

age, the selection of the minimum similarity score, or the selection of the maximum similarity score. The optimal aggregation function for a given set of comparisons and aggregations strongly depends on the concrete data set. Therefore, in order to find an optimal aggregation function, different functions have to be evaluated.

Aggregation function crossover selects one aggregation at random in each linkage rule and interchanges the aggregation functions. For example, it may select an aggregation that uses the minimum aggregation function in the first linkage rule and an aggregation with the weighted average function in the second linkage rule and then interchange these two functions. The pseudocode of aggregation function crossover is given in Listing 3.7.

Listing 3.7: Pseudocode of the aggregation function crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   cmp1 ← random node of nodeType from r1
3   cmp2 ← random node of nodeType from r2
4
5   return r1 with cmp1.fa ← cmp2.fa
6 }

```

Weight Crossover Operator

When aggregating the similarity scores of multiple operators, it is not always desirable that all operators contribute in the same way to the combined similarity score. For this purpose, some aggregation functions, such as the weighted average, may take a user-provided weight for each operator into account when combining the scores. The idea of the weight of an operator is to control the extend to which it contributes to the final similarity score. Finding the optimal weight for each operator is the subject of the *weight crossover operator*.

The weight crossover operator combines the weights of both linkage rules analogous to the threshold crossover. It selects a comparison or aggregation operator in each linkage rule and updates the weight in the first operator to the average of the weights of both operators. The pseudocode is given in Listing 3.8.

Note that in case the parent operator of the selected comparison or aggregation uses an aggregation function that does not consider the weights, such as the minimum aggregation, changing the weight does not have any effect.

Listing 3.8: Pseudocode of the weight crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   op1 ← random comparison or aggregation from r1
3   op2 ← random comparison or aggregation from r2
4
5   return r1 with (op1.weight ← 0.5 · (op1.weight + op2.weight))
6 }
```

In that case, the crossover operation degrades to a reproduction operation in which the fitness of the linkage rule is effectively unchanged.

Aggregation Hierarchy Crossover Operator

While for some data sets it is sufficient to use pure linear or boolean classifiers, for other data sets the accuracy can be improved by allowing non-linear aggregations (see Section 3.5.5). In order to learn aggregation hierarchies, *aggregation hierarchy crossover* selects a random aggregation or comparison operator in the first linkage rule and replaces it with a random aggregation or comparison operator from the second linkage rule. This way, the operator builds a hierarchy as it may select operators from different levels in the tree. For example, it may select a comparison in the first linkage rule and replace it with an aggregation of multiple comparisons from the second linkage rule. Note that aggregation hierarchy crossover is similar to subtree crossover, but only operates on aggregation and comparison nodes. The pseudocode is given in Listing 3.9. Figure 3.9 illustrates an aggregation hierarchy crossover on

Listing 3.9: Pseudocode of the aggregation hierarchy crossover operator.

```

1 def cross (r1: LinkageRule, r2: LinkageRule) = {
2   o1 ← random aggregation or comparison from r1
3   o2 ← random aggregation or comparison from r2
4
5   return r1 with o1 replaced by o2
6 }
```

two linkage rules.

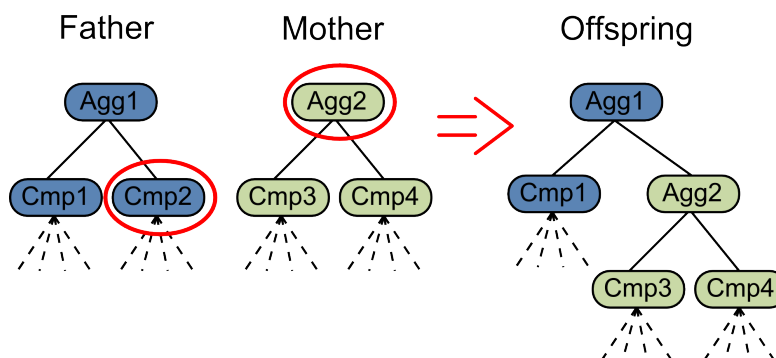


Figure 3.9: Example application of the aggregation hierarchy crossover operator.

3.3.4 Bloating Control

Section 3.2.3 already introduced parsimony pressure as a method to control bloating by penalizing the fitness of big trees in order to force the algorithm to favor smaller trees over bigger ones. In addition to parsimony pressure, we developed an heuristic to remove introns in linkage rules in order to further reduce the size of the learned linkage rules. The algorithm recursively traverses through each linkage rule and removes all operators that do not contribute to the overall fitness of a linkage rule. After all linkage rules have been simplified, there are usually many identical linkage rules. After each simplification run, all linkage rules that are duplicates of another linkage rule are removed from the population in order to increase its diversity. For each removed linkage rule, a new random linkage rule is generated and added to the population. The simplification algorithm is used in addition to parsimony pressure and is executed every 5 generations.

Section 3.5.5 evaluates different approaches to control bloating and measures their effect on the size of the linkage rules as well as on the learning performance.

3.4 Previous Work on Supervised Learning

There is a large body of work on supervised learning of linkage rules [Köpcke and Rahm, 2010] as well as on unsupervised entity matching [Euzenat et al., 2010, 2011a; Aguirre et al., 2012]. While supervised learning is concerned with learning linkage rules from reference links, unsupervised learning is concerned with matching entities when no labeled links are available. Supervised algorithms for learning linkage rules are discussed in three categories: ap-

proaches that learn *linear classifiers*, approaches that learn *threshold-based boolean classifiers*, and applications of *genetic programming* for learning more expressive linkage rules. In addition, we discuss *collective approaches* that focus on matching data sets by looking at the entire data sets at once instead of learning a linkage rule that compares pairs of entities independently. As this chapter proposes a supervised learning algorithm, the discussion of unsupervised methods is kept brief. We close this section with a comparison of the reported performance scores for various entity matching approaches on two frequently used evaluation data sets and a comparison of the employed linkage rule representations.

3.4.1 Linear Classifiers

A linear classifier combines a set of similarity comparisons by computing the weighted sum of the individual scores [Dey et al., 1998]. Linear classifiers have been introduced in Section 2.4.1. Two approaches for learning linear classifiers have been applied to entity matching [Köpcke and Rahm, 2010]: naive Bayes classifiers and support vector machines. As the original Fellegi-Sunter statistical model [Fellegi and Sunter, 1969] of record linkage is based on Bayesian statistics, *naive Bayes* classifiers have been applied to entity matching early on [Winkler, 2002].

Naive Bayes classifiers have been shown to perform worse than other supervised algorithms for a number of classification problems [Caruana and Niculescu-Mizil, 2006]. In particular, support vector machines and decision trees have been found to outperform naive Bayes classifiers when applied to entity matching [Sarawagi and Bhamidipaty, 2002].

Another popular method to model linear classifiers are *support vector machines* (SVM) [Cortes and Vapnik, 1995]. SVM is a binary linear classifier that maps the instances that are to be classified into a multi-dimensional space where the two classes are separated by a hyperplane. A requirement for using SVMs is that the input objects can be represented as vectors of numbers. In entity matching, this can be achieved by representing the input objects (i.e, the pairs of entities that are to be classified) as vectors of similarity scores. The selection of the properties that are to be compared together the distance measures that are used for computing the similarity scores, must be conducted prior to applying support vector machines.

An example of a support vector machine that has been applied on a data set that contains entities that describe companies is shown in Figure 3.10. In this example, each pair of entities is represented by two similarity scores: The similarity according the names of both companies as well as their similarity according their addresses. In this example, positive reference links are

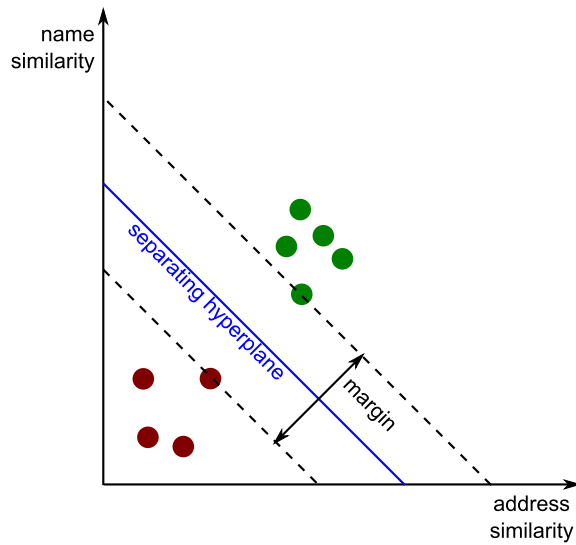


Figure 3.10: Example of a support vector machine. Green circles indicate positive reference links, while red circles indicate negative links.

represented by green circles, while negative reference links are represented by red circles. The SVM is represented by the blue line that separates positive and negative reference links. SVMs aim to maximize the margin, i.e., the distance to the closest training example. While SVMs can be extended to model non-linear classifiers, they are not suitable to learn chains of data transformations. In the following, we discuss previous applications of support vector machines to entity matching.

MARLIN

MARLIN (Multiply Adaptive Record Linkage with INduction) [Bilenko and Mooney, 2003] supports learning linear classifiers for entity matching using support vector machines. *MARLIN* assumes that the entities that are to be matched adhere to a consistent schema, i.e., if inter-source duplicates are to be found the schemata of both data sets need to be harmonized. *MARLIN* operates in two steps: In the first step, *MARLIN* learns a similarity measure for each entity property. For learning string similarity measures, *MARLIN* employs a stochastic model based on the edit distance with affine gaps. In the second step, it learns a support vector machine that combines the learned similarity measures linearly.

MARLIN has been evaluated on two data sets:

Restaurant: A set of restaurants that have been collected from the Fodor’s and Zagat’s restaurant guides.

Cora: A set of citations to research papers from the Cora Computer Science research paper search engine.

Both data sets have also been used for evaluation by other entity matching approaches including the GenLink algorithm, which we propose in this work. Section 3.4.6 will compare the results of a number of entity matching approaches that have been evaluated on these two data sets as well. MARLIN is evaluated with different configurations. Taking the best values for each data set, MARLIN achieves an F-measure of 92.2% on the Restaurant data set and an F-measure 86.7% on the Cora data set [Bilenko and Mooney, 2003].

FEBRL

FEBRL (Freely Extensible Biomedical Record Linkage) [Christen, 2008, 2009] is an open source system for entity matching. FEBRL supports various techniques for entity matching. Amongst these, it also provides a supervised learning algorithm based on support vector machines. Support vector machines are used to learn a vector weights to combine set of user-provided similarity measures linearly. Christen [2008, 2009] does not present evaluation results.

3.4.2 Threshold-based Boolean Classifiers

A *threshold-based boolean classifier* combines multiple similarity tests using boolean operators [Lim et al., 1993]. Threshold-based boolean classifiers have been introduced in Section 2.4.2. Many existing algorithms for learning threshold-based boolean classifiers use decision trees [Cochinwala et al., 2001; Tejada et al., 2001; Elfeky et al., 2002]. A major advantage of decision trees is that they provide explanations for each classification and thus can be understood and improved manually. As for linear classifiers, threshold-based boolean classifiers are not suitable for learning data transformations. A number of methods have been proposed for learning decision trees [Rokach and Maimon, 2008]. Popular decision tree learning methods include CART (Classification and Regression Trees) [Breiman et al., 1984], ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993]. In the following, we discuss different approaches for entity matching that are based on learning decision trees.

Cochinwala et al.

Cochinwala et al. present a method for learning decision trees [Cochinwala et al., 2001] using the CART method and compare it to two other approaches:

A linear discriminant analysis that generates a linear combination of the input parameters and an optimized version of nearest neighbor search [Clarkson, 1983]. All three approaches are evaluated on two data sets that contain customer records. The input parameters are manually chosen:

- (1) Levenshtein distance between the names of both entities.
- (2) Levenshtein distance between addresses of both entities.
- (3) Length of the names of both entities.
- (4) Length of the addresses of both entities.

On the evaluation data set, the proposed supervised algorithm achieved an accuracy of over 90%. However, the comparability of the presented results is limited as the origin of the used evaluation data sets is not stated.

Figure 3.11 shows a decision tree that has been learned by the proposed algorithm for the evaluation data set. The learned linkage rules classify each

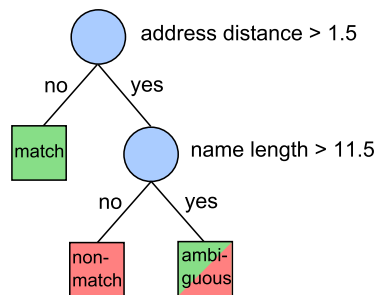


Figure 3.11: Decision tree that has been learned by the supervised algorithm proposed by Cochinwala et al. (adapted from [Cochinwala et al., 2001]).

pair of entities into three classes: match, non-match and ambiguous match.

Active Atlas

Active Atlas [Tejada et al., 2001] supports learning decision trees consisting of a combination of predefined similarity measures. It uses the C4.5 learning algorithm for learning the decision trees. Figure 3.12 shows an example of a decision tree that has been learned by Active Atlas [Tejada et al., 2001].

Active Atlas has been evaluated on three use cases:

- (1) Matching a data set that contains restaurants that have been collected from the Fodor's and Zagat's restaurant guides. This data set has also been used to evaluate the performance of the GenLink algorithm in

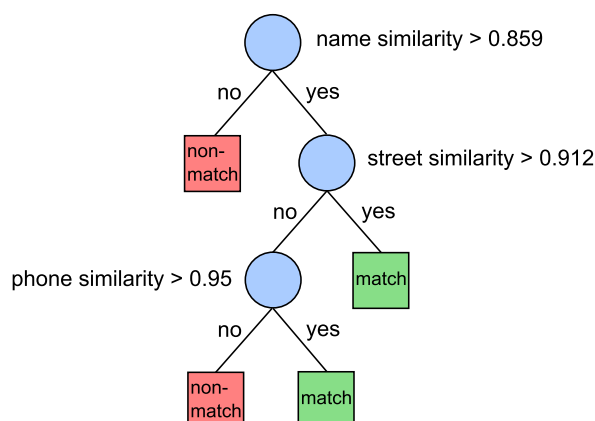


Figure 3.12: Example of a decision tree that has been learned by Active Atlas (adapted from [Tejada et al., 2001]).

Section 3.5. For the evaluation of Active Atlas, three properties have been used: name, street and phone.

- (2) Matching two data sets about companies that are provided by Cohen [1998]: 1163 companies that have been collected from HooversWeb⁴ and 957 companies that have been collected from IonTech⁵. For each company, three properties are used for evaluation: name, url and description.
- (3) Matching a data set containing 428 airports with the corresponding weather station in a data set containing 12,000 weather stations. Two properties are used for matching: The airport/weather station code and the location as plain string (e.g., “Kodiak, AK”). The origin of the employed data sets is not stated by Tejada et al. [2001].

No use case with more than three properties has been shown. Active Atlas has been executed on all three use cases using 5-fold cross-validation. It achieved an accuracy of 99.68% on the restaurant use case, 99.80% on the company use case and 99.51% on the airport use case. In addition to the supervised variant of Active Atlas, Tejada et al. [2001] also developed an active learning extension, which will be discussed in the next Chapter in Section 4.3.

⁴<http://www.hoovers.com/> (from: [Cohen, 1998])

⁵<http://www.iontech.com/> (from: [Cohen, 1998]. No longer active at time of writing.)

TAILOR

TAILOR [Elfeky et al., 2002] is a framework that implements different techniques for entity matching. *TAILOR* supports learning decision trees using the ID3 algorithm. *TAILOR* has been evaluated on two data sets:

- (1) A synthetic data set that has been generated using the DBGen Tool [Askarunisa A. et al., 2009].
- (2) A data set that has been generated from Wal-Mart data.

No references are provided to the original data set.

3.4.3 Genetic Programming

Genetic programming denotes a class of machine learning algorithms that are based on genetic algorithms and typically represent solutions as operator trees and thus are capable of learning much more expressive models than other learning methods [Koza, 1993]. The next section provides a detailed description of genetic programming.

To the best of our knowledge, genetic programming for supervised learning of linkage rules has only been applied by de Carvalho et al. so far [de Carvalho et al., 2006; Carvalho et al., 2008; de Carvalho et al., 2012]. Their approach uses genetic programming to learn how to combine a set of comparison pairs of the form `<property, distance measure>` (e.g., `<name, Jaro>`) into a linkage rule. It includes an automatic feature selection that chooses comparisons pairs automatically. Comparison pairs are combined by the genetic programming method to a linkage rule tree by using mathematical functions (e.g., `+`, `-`, `*`, `/`, `exp`) and constants. Their approach is very expressive although it cannot express data transformations. On the downside, using mathematical functions to combine the similarity measures does not fit any commonly used linkage rule model [Euzenat and Shvaiko, 2007] and may lead linkage rules that are difficult to interpret.

de Carvalho et al. evaluate their approach on the previously described Cora and the Restaurant data sets that both have also been used to evaluate the performance of the GenLink algorithm in Section 3.5. In addition, they evaluate the performance on three synthetic data sets. de Carvalho et al. show that their method produces better results than the state-of-the-art SVM based approach by MARLIN [de Carvalho et al., 2012] on both data sets.

Section 3.4.6 will discuss the differences between the approach by de Carvalho et al. and our approach in more detail.

3.4.4 Collective Approaches

Entity matching approaches that are based on linkage rules are local approaches that determine the similarity of each pair of entities independently. The idea of collective entity matching approaches is that match decisions of related pairs of entities are conducted holistically. A popular example of a domain for which collective approaches have been applied bibliographic data sets [Domingos, 2004; Dong et al., 2005; Bhattacharya and Getoor, 2007].

Common entity types in bibliographic databases include articles, authors and venues. The intuition is that given a set of articles that is to be matched, the matching decision of articles that share co-authors is not done independently. Within collective entity matching approaches, such relationships can be specified by building a relationship graph. We illustrate such a relationship graph on the previously introduced Cora data set. Figure 3.13 visualizes the relationship graph between two publications p_1 and p_2 . In this example,

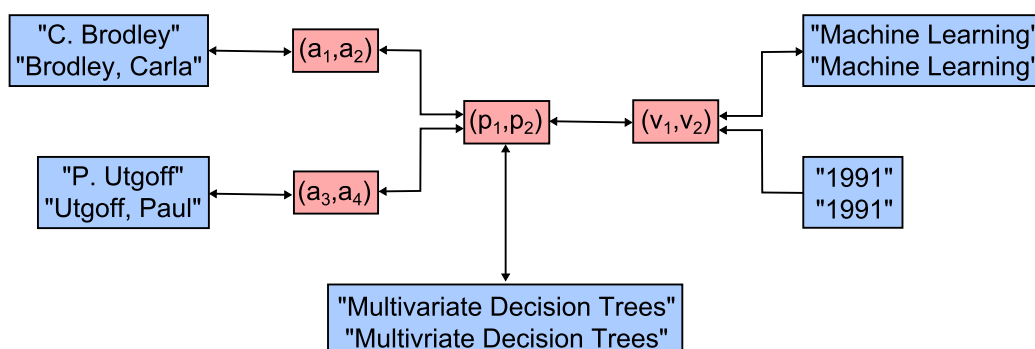


Figure 3.13: Relationship graph between two publications (adapted from: [Dong et al., 2005]).

the similarity of p_1 and p_2 directly depends on the similarity of their titles, the similarity of the authors as well as the similarity of the venues. The similarity of the venues itself depends on the similarity of the venue names and the date.

Domingos [2004] propose an collective entity matching approach that is based on conditional random fields [Lafferty et al., 2001]. Conditional random fields are used for propagation of the similarity values through the relationship graph (i.e., similarity propagation) Domingos [2004] evaluate their approach on the Cora data set and compare the results to a pairwise approach that does conduct matching decisions independently. Using two-fold cross-validation, the collective approach outperforms the pairwise approach achieving an F-measure of 87.0% on the Cora data set compared to 84.4% as achieved by the pairwise approach. While their approach outperforms

the chosen pairwise approach, it does not outperform some other proposed pairwise systems, such as the genetic programming approach by de Carvalho et al. [2012] that reports 91.0% or the GenLink algorithm, which is proposed in this thesis and achieves 96.9% on the same data set.

Dong et al. [2005] propose a similar collective entity matching approach that is also based on similarity propagation. They evaluate their approach on a personal data set and also on the Cora data set in order to compare their results to the reported results by Domingos [2004]. Their experimental evaluation shows that they significantly outperform the approach by Domingos [2004] achieving an F-measure of 95.4% when matching articles in the Cora data set.

Besides the approaches by Domingos [2004] and Dong et al. [2005], many other approaches for collective entity matching have been proposed. In the following we give a brief overview of other collective entity matching approaches. Unfortunately, none of the following approaches has been evaluated on a frequently employed evaluation data set, which hinders the comparability of the reported performance scores. A comprehensive discussion of various collective entity matching approaches can be found in [Christen, 2012].

Bhattacharya and Getoor [2007] propose a collective entity matching approach based on latent Dirichlet allocation [Blei et al., 2003]. They evaluate their approach on three bibliographic data sets extracted from CiteSeer [Giles et al., 1998], arXiv⁶ and the Elsevier BioBase data set⁷.

Ananthakrishna et al. [2002] and Weis and Naumann [2004] propose approaches for collective entity matching that exploit hierarchical relationships in the data sets. Ananthakrishna et al. [2002] use a customer data set for evaluation whose origin is not stated. Weis and Naumann [2004] evaluate their approach on the MONDIAL geographical data set⁸ that has been modified in order to add errors.

Kalashnikov et al. [Kalashnikov et al., 2005; Kalashnikov and Mehrotra, 2006] propose another collective entity matching approach, which is evaluated on a publication data set that contains entities from CiteSeer and HomePageSearch⁹ and a movie data set¹⁰.

⁶<http://arxiv.org/>

⁷<http://www.elsevier.com/bibliographic-databases/cabs>

⁸<http://www.dbis.informatik.uni-goettingen.de/Mondial/>

⁹<http://hpsearch.uni-trier.de/> (No longer active at time of writing.)

¹⁰<http://infolab.stanford.edu/pub/movies/doc.html>

3.4.5 Unsupervised Approaches

Unsupervised entity matching is concerned with matching data sets when no labeled training data (i.e., reference links) is available. In the recent years, a number of approaches have been proposed for unsupervised entity matching.

CODI (Combinatorial Optimization for Data Integration) [Noessner and Niepert, 2010] uses information from the schema in order to improve the instance matching. *ObjectCoref* [Hu et al., 2010, 2011] is a self-training interlinking approach, which starts with a kernel that consists of known equivalences and iteratively extends this kernel with discriminative property-value pairs. *RiMOM* [Wang et al., 2010] and *AgreementMaker* [Cruz et al., 2011] are approaches for ontology matching that have been extended with matchers for instance matching. *LN2R* [Sais et al., 2010] combines a logical part that translates schema information into first order logic and a numerical part for similarity computation.

SERIMI [Araujo et al., 2011] is another unsupervised interlinking approach, which matches entities based on their structural similarity in addition to matching their labels. *Zhishi.links* [Niu et al., 2011] employs an indexing technique on the labels of the entities as well as on discovered homonyms. The indexing technique allows it to scale to larger data sets than similar approaches. *SLINT* also employs an indexing technique to improve the matching performance.

LogMap [Jiménez-Ruiz et al., 2012] is a logic-based ontology and instance matching system that uses reasoning and inconsistency repair techniques. *SBUEI* [Taheri and Shamsfard, 2012] is another approach that combines schema level matching with instance level matching.

Ontology Alignment Evaluation Initiative

The Ontology Alignment Evaluation Initiative (OAEI) aims at evaluating different approaches for unsupervised vocabulary and instance matching [Euzenat et al., 2010, 2011a; Aguirre et al., 2012]. Table 3.1 lists the data sets that have been used for evaluation in the years 2010 to 2012 together with the systems that achieved the highest F-measure. The data sets for the ISLab Instance Matching Benchmarks (IIMB) have been created by automatically applying a number of transformations to the values in a base data set [Ferrara et al., 2011]. The base data set varies in each year, so values of different years cannot be compared.

Unfortunately, none of the data sets have been used in two subsequent years and the majority of the systems only participated in one year. In each year, the results of the participating systems varied with the data set and

Name	Description	F ₁	System
2010 [Euzenat et al., 2010]			
IIMB	Synthetic data set	82.1%	CODI
DI	Matching drugs in four data sets from the health-care domain.	≈14%	RiMOM
Persons	Matching persons	98.4%	RiMOM
Restaurants	Matching restaurants	81.1%	RiMOM
2011 [Euzenat et al., 2011a]			
IIMB	Synthetic data set	63.4%	CODI
NYT	Entities describing people, organizations, and companies	92.0%	Zhishi.links
2012 [Aguirre et al., 2012]			
IIMB	Synthetic data set	92.0%	LogMap
Sandbox	Entities from Freebase	96.3%	SBUEI

Table 3.1: Experiments in the OAEI instance matching challenges. The third column states the F-measure of the best system. For experiments that consist of multiple data sets the harmonic mean is stated.

most systems did not publish results for all data sets. The only data set that has also been used by any of the previously introduced supervised approaches is the restaurants data set that has been using in the 2010 challenge. The restaurant data set contains entities from the Fodor’s and Zagat’s restaurant guides that describe restaurants. We will compare the performance of the unsupervised systems on the restaurant data set to the supervised systems in the following section.

3.4.6 Discussion

Performance

For evaluation, many authors create their own data sets instead of reusing data sets that have been used for evaluation before by other authors. The comparability of the reported performance results, in these cases, is limited. Amongst the introduced approaches, only two data sets have been used by multiple authors for evaluation: The Cora citation data set and the restaurant data set. Table 3.2 compares the performance results that have been reported on these two data sets. For comparison, it also shows the F-measure that has been achieved by GenLink, which is evaluated in detail in Section 3.5.

Approach	Cora	Restaurant
Linear and Threshold-based Boolean Classifiers		
Active Atlas [Tejada et al., 2001]	(not available)	99.7% accuracy
MARLIN [Bilenko and Mooney, 2003]	86.7% F-measure	92.2% F-measure
Genetic Programming		
de Carvalho et al. [2012]	91.0% F-measure	98.0% F-measure
<i>GenLink</i>	96.6% F-measure	99.3% F-measure
Collective Approaches		
Domingos [2004]	87.0% F-measure	(not available)
Dong et al. [2005]	95.4% F-measure	(not available)
Unsupervised Approaches (Top 3)		
RiMOM [Wang et al., 2010]	(not available)	81% F-measure
LN2R [Sais et al., 2010]	(not available)	75% F-measure
ObjectCoref [Hu et al., 2010]	(not available)	73% F-measure

Table 3.2: Performance comparison of different approaches.

Learned Classifiers

The majority of the introduced supervised algorithms only support learning either linear classifiers or threshold-based boolean classifiers. Most supervised algorithms for learning linear classifiers employ support vector machines, while most supervised algorithms for learning threshold-based boolean classifiers learn decision trees.

In machine learning, a concern when choosing a learning algorithm is the interpretability of the generated classifiers [Cano et al., 2007]. A classifier that is interpretable can be understood and tuned by humans. In the field of entity matching, it is usually argued that learned decision trees, as compared to support vector machines, are in general easier to interpret by a human expert [Sarawagi and Bhamidipaty, 2002; Christen, 2012].

When it comes to genetic programming algorithms, the interpretability of the generated classifiers depends on the chosen representation. While genetic programming can be used to learn decision trees [Nikolaev and Slavov, 1997], the introduced genetic programming algorithm by de Carvalho et. al, as well as the *GenLink* algorithm, which is introduced in this chapter, employ a more expressive representation. Figure 3.14 compares two linkage rules that have been learned for the Cora data set. The first linkage rule has been learned by the genetic programming algorithm by de Carvalho et al. as reported in [de Carvalho et al., 2012]. The second linkage rule has been learned by *GenLink*. The approach that has been chosen by de Carvalho et al. is

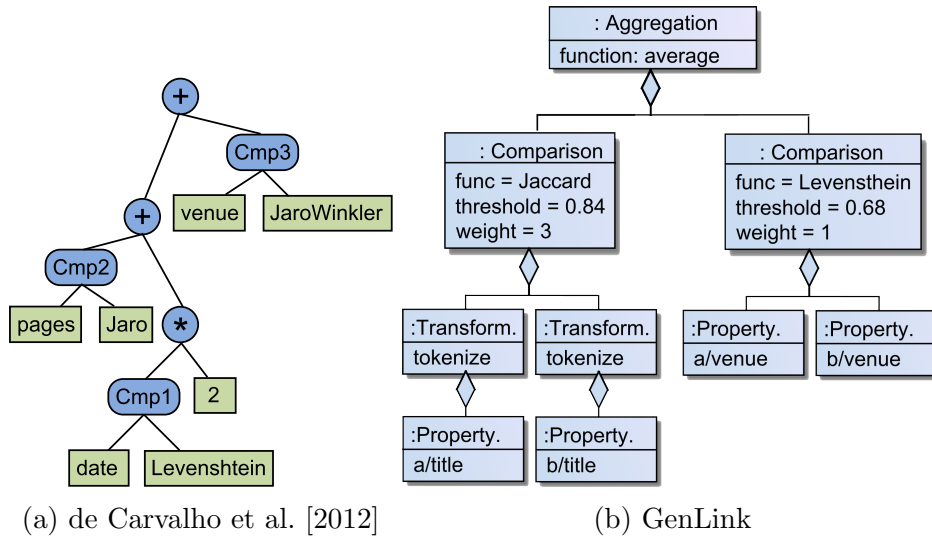


Figure 3.14: Comparison of linkage rules that have been learned for the Cora data set by two different genetic programming learning algorithms.

to combine comparisons using arithmetic operations, such as addition and multiplication. The shown linkage rule starts by multiplying the similarity of the dates of both entities by the factor two. The resulting value is then multiplied with the similarity of the pages that are given in the citations as well as the venues. The linkage rule that has been generated by GenLink returns the weighted average of the similarity of both titles and the similarity of the venues. Both representations allow the specification of the distance measure that is used for each comparison.

GenLink is the only system that is capable of learning linkage rules that include chains of transformations.

3.5 Evaluation and Discussion

In this section, we evaluate GenLink experimentally: At first, Section 3.5.1 will introduce the data sets that we used for the evaluation. Section 3.5.2 will describe the experimental setup. Section 3.5.3 will introduce evaluation measures. The overall learning results for several real-world data sets will be presented in Section 3.5.4. Finally, Section 3.5.5 will evaluate the contribution of specific parts of our algorithm to the accuracy of the learned linkage rules.

3.5.1 Data Sets

For evaluation, we used six data sets from three areas for which reference links are available that we could use as gold standard within our experiments:

- (1) We evaluated the learning performance on two well-known record linkage data sets and compared the performance with an existing state-of-the-art genetic programming approach. We chose two data sets, which have been used by a number of previous approaches for evaluation as well: the *Cora* data set and the *Restaurant* data set.

The **Cora** data set [McCallum et al., 2000b,a] contains citations to research papers from the Cora Computer Science research paper search engine. For each citation, it contains the title, the author, the venue as well as the date of publication.

The **Restaurant** data set [Tejada et al., 2001] contains a set of records from the Fodor’s and Zagat’s restaurant guides. For each restaurant, it contains the name, address, phone number as well as the type of restaurant. For both data sets, we used the XML version¹¹ that is provided by Draisbach et al.

- (2) We evaluated our approach with two data sets from the *Ontology Alignment Evaluation Initiative* (OAEI) [Euzenat et al., 2011b] and compared our results to the participating systems. The OAEI is an international initiative aimed at organizing the evaluation of different ontology matching systems. In addition to schema matching, OAEI also includes an instance matching track since 2009, which regularly evaluates the ability to identify similar entities among different RDF data sources.

The **SiderDrugBank** data set was selected from the OAEI 2010 data interlinking track¹² [Euzenat et al., 2010]. We chose this data set amongst the other drug related data sets, because it was the one for which the participating systems ObjectCoref [Hu et al., 2010] and RiMOM [Wang et al., 2010] performed the worst. This data set contains drugs from Sider, a data set of marketed drugs and their side effects, and DrugBank, containing contains drugs approved by the US Federal Drugs Agency. Positive reference links are provided by the OAEI.

¹¹http://www.hpi.uni-potsdam.de/naumann/projekte/dude_duplicate_detection.html

¹²<http://oaei.ontologymatching.org/2010/im/index.html>

The **NewYorkTimes** data set was selected from the OAEI 2011 data interlinking track¹³ [Euzenat et al., 2011a]. Amongst the seven data sets from this track, we chose the data set for which the participating systems performed the worst on average: Interlinking locations in the New York Times data set with their equivalent in DBpedia [Bizer et al., 2009b]. Besides other types of entities, the New York Times data set contains 5620 manually curated locations. In addition, it contains 1920 manually verified links between locations in the New York Times data set itself and the same location in DBpedia. The NewYorkTimes evaluation data set has been build by extracting all 5620 locations from the official New York Times data set¹⁴. Locations in DBpedia have been retrieved by requesting each interlinked DBpedia location by using the official SPARQL endpoint¹⁵.

- (3) On two data sets, we compared the learned linkage rules with linkage rules created by a human expert for the same data set. The original linkage rules for both data sets have been created as part of the LATC EU project¹⁶.

The **LinkedMDB** data set¹⁷ [Hassanzadeh and Consens, 2009] is an easy to understand data set about movies, which is non-trivial as the linkage rule cannot just compare by label (different movies may have the same name), but also needs to include other properties such as the date or the director.

The **DBpediaDrugBank** data set about interlinking drugs in DBpedia¹⁸ [Bizer et al., 2009b] and DrugBank¹⁹ is an example where the original linkage rule, which has been produced by humans, is very complex. In order to match two drugs, it compares the drug names and their synonyms as well as a list of well-known and used identifiers, which are provided by both data sets but are missing for many entities. In total, the manually written linkage rule uses 13 comparisons and 33 transformations. This includes complex transformations, such as replacing specific parts of the strings.

The first two data sets are frequently-used record linkage data sets while the following four sets are RDF data sets. While the record linkage data sets

¹³<http://oaei.ontologymatching.org/2011/instance/>

¹⁴<http://data.nytimes.com/>

¹⁵<http://dbpedia.org/sparql>

¹⁶<http://latc-project.eu/>

¹⁷<http://www.linkedmdb.org/>

¹⁸<http://dbpedia.org>

¹⁹<http://wifo5-04.informatik.uni-mannheim.de/drugbank/>

	Entities		Reference Links	
	$ A $	$ B $	$ R_+ $	$ R_- $
Cora	1879		1617	1617
Restaurant	864		112	112
SiderDrugbank	924	4772	859	859
NewYorkTimes	5620	1819	1920	1920
LinkedMDB	199	174	100	100
DBpediaDrugbank	4854	4772	1403	1403

Table 3.3: The number of entities in each data set as well as the number of reference links.

	Properties		Coverage	
	$ A.P $	$ B.P $	C_A	C_B
Cora	4		0.8	
Restaurant	5		1.0	
SiderDrugbank	8	79	1.0	0.5
NewYorkTimes	38	110	0.3	0.2
LinkedMDB	100	46	0.4	0.4
DBpediaDrugbank	110	79	0.3	0.5

Table 3.4: The total number of properties in each data set as well as the percentage of properties that are actually set on an entity.

are already adhering to a consistent schema, the RDF data sets are split into a source and a target data set that adhere to different schemata.

Table 3.3 lists the used data sets together with the number of entities as well as the number of reference links in each data set. As only positive reference links have been provided by the data set providers, we generated the negative reference links. For two positive links $(a, b) \in R^+$ and $(c, d) \in R^+$ we generated two negative links $(a, d) \in R^-$ and $(c, b) \in R^-$. For the Cora and Restaurant data set this is sound as the provided positive links are complete. Since the remaining data sources are split into source and target data sets, generating negative reference links is possible as entities in the source and target data sets are internally unique.

Table 3.4 shows the number of properties in the source and target data sets and their coverage, i.e., the percentage of properties that are actually set on an entity on average.

Parameter	Value
Population size	500
Maximum iterations	50
Selection method	Tournament selection
Tournament size	5
Probability of crossover	75%
Probability of mutation	25%
Stop condition	F-measure = 100%
Fitness Function	MCC with penalty factor
Penalty factor	0.05

Table 3.5: Parameters

3.5.2 Experimental Setup

The GenLink algorithm has been implemented in the Silk Link Discovery Framework. All experiments have been executed using Version 2.5.3 of the Silk Link Discovery Framework. A detailed description of the Silk Link Discovery Framework can be found in Chapter 6.

Because genetic algorithms are non-deterministic and may yield different results in each run, all experiments have been run 10 times. For each run the reference links have been randomly split into two folds for cross-validation. The results of all runs have been averaged and the standard deviation (σ) has been computed. For each experiment, we provide the evaluation results with respect to the training set as well as the test set. All experiments have been run on a 3GHz Intel(R) Core i7 CPU with four cores while the Java heap space has been restricted to 1GB.

Table 3.5 lists the parameters that have been used in all experiments. As it is the purpose of the developed algorithm to work on arbitrary data sets without the need to tailor its parameters to the specific data sets that should be matched, the same parameters have been used for all experiments. The parameters have been chosen based on the parameters used by Koza in [Koza, 1993, 1994, 1999; Koza et al., 2005], which have been shown to work well on a variety of use cases. The population size determines the number of linkage rules in the population. In each iteration of the genetic algorithm, a new population is build using the tournament selection method. Tournament selection randomly selects five linkage rules from the population and chooses the rule with the highest fitness. The fitness is determined by computing Matthews correlation coefficient and penalizing linkage rules with many operators with a penalty factor of 0.05 as described in Section 3.3.2. With a probability of 75%, a crossover operation is performed on two chosen

linkage rules. In the remaining cases a mutation operation is performed. The genetic algorithm stops if either a maximum number of 50 iterations have been performed or if a linkage rule with an F-measure of 100% has been found.

3.5.3 Evaluation Measures

This section introduces methods to evaluate the performance of a linkage rule based on the supplied positive and negative reference links. Given the output of a matching task and a set of reference links, two types of errors can be detected:

- **Type I error:** A link is generated for a specific negative reference link (false positive).
- **Type II error:** No link is generated for a specific positive reference link (false negative).

Based on these two types of errors, we distinguish four cases:

	predicted class	
	match	no match
positive ref. link	true positive (tp)	false negative (fn)
negative ref. link	false positive (fp)	true negative (tn)

Based on these counts, various measures can be defined: First of all, we define the precision of a linkage rule:

Definition 3.2 (Precision) *The precision of a linkage rules is defined as the fraction of generated links that are correct:*

$$precision = \frac{tp}{tp + fp}$$

While precision is a measure of correctness, the recall of a linkage rule is a measure of its completeness:

Definition 3.3 (Recall) *The recall of a linkage rule is defined as fraction of generated correct links from all positive reference links:*

$$recall = \frac{tp}{tp + fn}$$

Usually, there is a trade-off between maximizing precision on the one hand and maximizing recall on the other. Increasing precision usually comes at the cost of a reduced recall. On the contrary, increasing recall may increase the number of false dismissal and thereby reduce the precision.

The trade-off between recall and precision for a particular data set can be visualized using a recall-precision diagram. Figure 3.15 shows a typical recall-precision diagram.

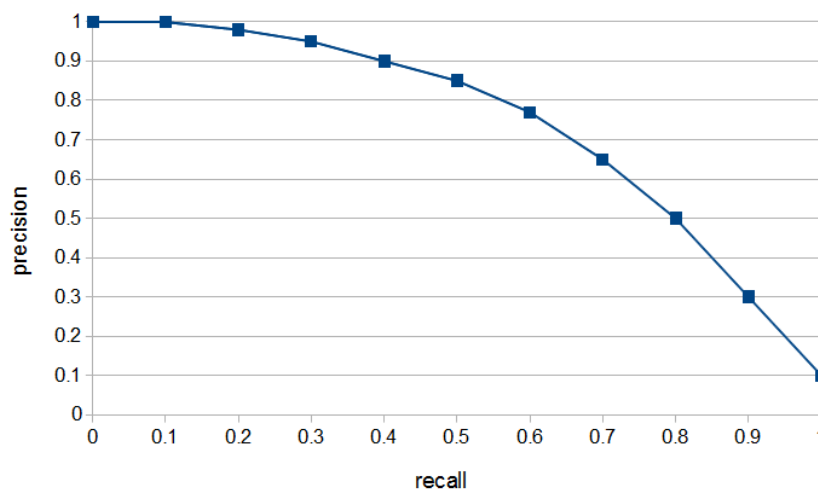


Figure 3.15: Typical precision-recall diagram.

The F-measure combines both recall and precision:

Definition 3.4 (F-measure) *The F-measure of a linkage rules is defined as the harmonic mean of its recall and precision:*

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

A disadvantage of the F-measure is that it may yield skewed results if the number of positive and negative reference links is unbalanced. A popular measure that avoids this problem is the previously introduced *Matthews correlation coefficient* that we use in the GenLink approach as fitness measure.

3.5.4 Overall Results

In this section we evaluate the overall performance of GenLink.

Frequently Used Record Linkage Datasets

A number of data sets have been used frequently to evaluate the performance of different record linkage approaches. Following this practice, we compared the overall learning performance of our approach with the approach proposed by de Carvalho et al. [2012] on the Cora citation data set and the Restaurant data set. In their experiments, de Carvalho et al. report to produce better results as the state-of-the-art SVM based approach by MARLIN. The related work section provides more details about how their approach compares to ours technically.

Table 3.6 summarizes the cross validation results for the Cora citation data set. On average, our approach achieved an F-measure of 96.9% against

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
0	5.5 (0.7)	0.880 (0.030)	0.877 (0.031)
10	28.6 (2.7)	0.949 (0.018)	0.945 (0.021)
20	60.1 (4.1)	0.965 (0.005)	0.962 (0.005)
30	93.6 (6.1)	0.968 (0.003)	0.965 (0.004)
40	129.4 (9.7)	0.968 (0.002)	0.965 (0.004)
50	185.8 (26.7)	0.969 (0.003)	0.966 (0.004)
Ref.		0.900 (0.010)	0.910 (0.010)

Table 3.6: Results for the Cora data set. The last row contains the best results of de Carvalho et al. for comparison.

the training set and 96.6% against the test set and needed about three minutes to perform all 50 iterations on the test machine. The learned linkage rules compared by title, author and venue. Figure 3.16 shows an example of a linkage rule that has been learned after 10 generations. Figure 3.17 shows an example of a linkage rule that has been learned after 40 generations, which reached the top F-measure.

For the same data set, de Carvalho et al. report an F-measure of 90.0% against the training set and 91.0% against the test set [de Carvalho et al., 2012]. We suspected the main reason for the better performance of our method on this data set to be found in the inclusion of data transformations in our learning approach. To confirm this claim we re-executed our method with one limitation: No data transformations were allowed to be used in a linkage rule. With this limitation, the performance of our methods declined to an F-measure of 91.2% against the training set and 90.5% against the test set approximately matching the numbers of de Carvalho et al. Figure 3.18 shows a learned linkage rule without transformations.

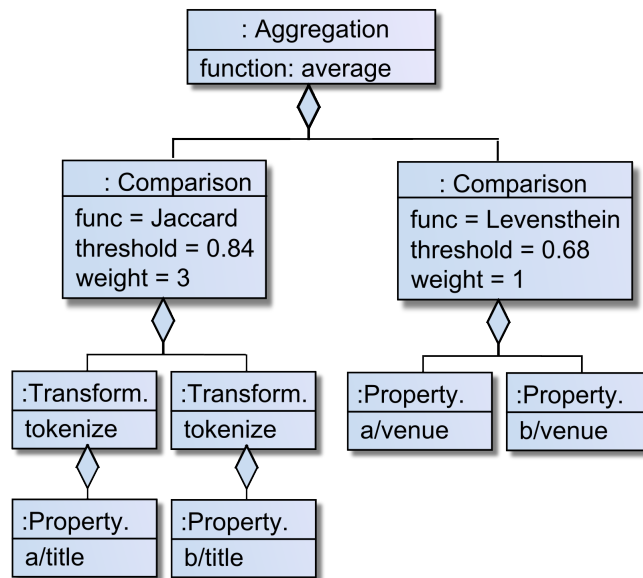


Figure 3.16: Cora: Learned linkage rule after 10 generations

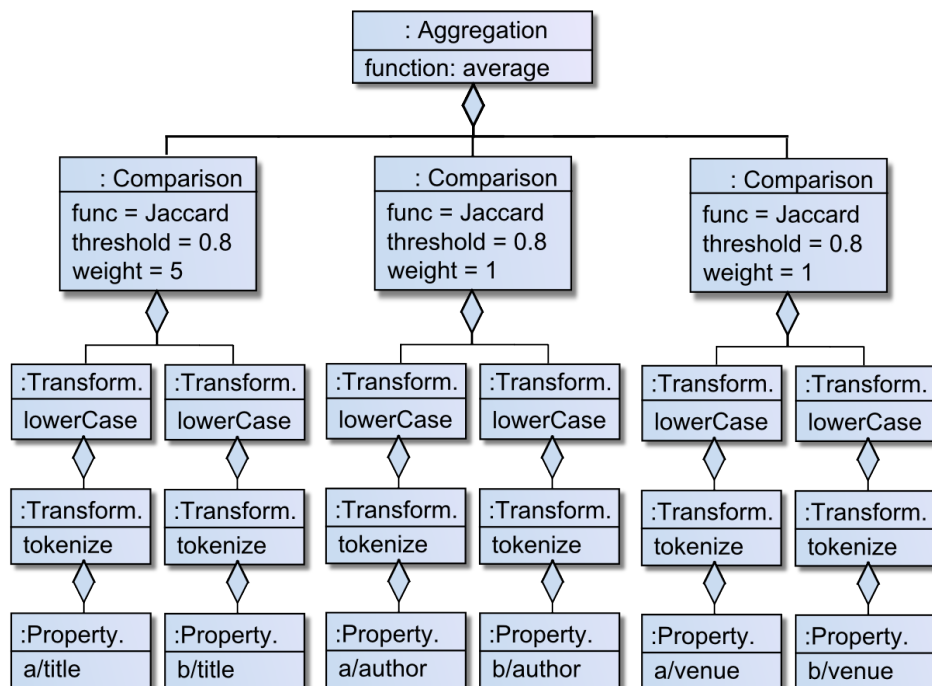


Figure 3.17: Cora: Learned linkage rule after 40 generations

While GenLink outperforms the method by de Carvalho et al., it still

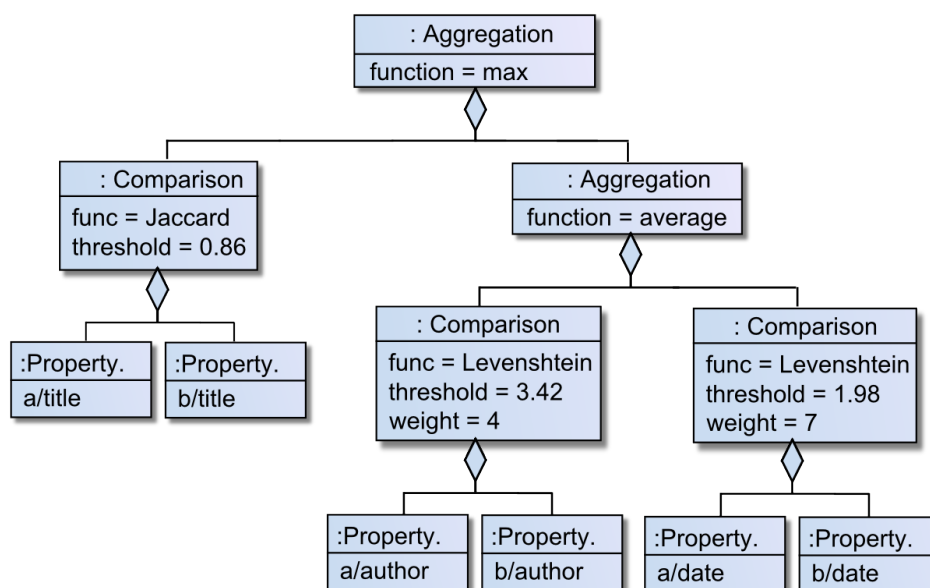


Figure 3.18: Cora: Learned linkage rule without transformations

misses a couple of reference links. Table 3.7 shows two examples of false positives. The first entry shows a border case in which both citations are very

Author	Title	Venue	Date
D. Aha and D. Kibler.	Noise-tolerant instance-based learning algorithms	In Proceedings of IJCAI-89	-
D. Aha D. Kibler and M. Albert.	Instance-based learning algorithms	Machine Learning	-
M. Pazzani and D. Kibler.	The utility of knowledge in inductive learning	Machine Learning	-
P. Utgoff.	Shift of bias for inductive concept learning	Machine Learning	-

Table 3.7: Two examples of false positives. '-' indicates missing values.

similar, while the second entry shows two citations that are clearly referring to different publications. The difficulty in distinguishing between both pairs of citations lies in the fact that the employed Jaccard similarity measure yields high similarity scores as in both cases the titles contain similar words. As the Jaccard comparison also includes common words, such as 'of' and 'the' in the comparison, the perceived overlap is even higher. The performance

in this case could be further improved by including transformations that remove stop-words (e.g., 'the' and 'of') and by normalizing the author names. Section 7.2 will discuss this extension for future work.

Table 3.8 summarizes the cross validation results for the Restaurant data set. On average, our approach achieved an F-measure of 99.6% against the

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
0	0.4 (0.1)	0.953 (0.038)	0.951 (0.039)
10	2.0 (0.9)	0.996 (0.004)	0.992 (0.006)
20	3.1 (1.9)	0.996 (0.004)	0.993 (0.006)
30	4.1 (3.0)	0.996 (0.004)	0.993 (0.006)
40	5.2 (4.0)	0.996 (0.004)	0.993 (0.006)
50	6.3 (5.3)	0.996 (0.004)	0.993 (0.006)
Ref.		1.000 (0.000)	0.980 (0.010)

Table 3.8: Results for the Restaurant data set. The last row contains the best results of de Carvalho et al. for comparison.

training set and 99.3% against the test set. For the same data set, de Carvalho et al. report an F-measure of 100.0% against the training set, but only 98.0% against the test set [de Carvalho et al., 2012].

Ontology Alignment Evaluation Initiative

For two data sets from the Ontology Alignment Evaluation Initiative (OAEI), we compared our results with the results of the participating systems in the instance matching track. In the OAEI, the systems were asked to identify similar entities in a data set without being allowed to employ existing reference links for matching. Note that as the OAEI only compares unsupervised systems and does not consider supervised systems (i.e., systems that are supplied with existing reference links), our approach has an advantage over the participating systems. For that reason, we used the official OAEI results merely as a baseline for our approach.

Table 3.9 summarizes the cross validation results for the SiderDrugBank data set. After 30 iterations, our approach reached an F-measure of 97.2% for the training set and 97.0% for the test set. ObjectCoref and RiMOM achieved only around 50% percent, which indicates the difficulty in matching this data set.

Table 3.10 summarizes the cross validation results for the NewYorkTimes data set. After 50 iterations, our approach reached an F-measure of 97.7%

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
0	21.7 (0.3)	0.840 (0.018)	0.837 (0.018)
10	38.8 (2.4)	0.943 (0.025)	0.939 (0.030)
20	83.1 (11.1)	0.970 (0.007)	0.969 (0.008)
30	147.2 (20.9)	0.972 (0.006)	0.970 (0.007)
40	215.6 (28.0)	0.972 (0.006)	0.970 (0.007)
50	301.5 (39.0)	0.972 (0.006)	0.970 (0.007)

Reference System	F1
ObjectCoref	0.464 [Hu et al., 2010]
RiMOM	0.504 [Wang et al., 2010]

Table 3.9: Results for the SiderDrugBank data set.

for the training set and 97.4% for the test set. With an F-measure of 92%, Zhishi.links achieved the best result of the participating systems.

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
0	85.2 (1.7)	0.703 (0.064)	0.709 (0.048)
1	107.7 (11.0)	0.803 (0.037)	0.803 (0.036)
5	260.7 (76.8)	0.844 (0.048)	0.846 (0.048)
10	344.5 (86.2)	0.854 (0.052)	0.854 (0.053)
20	496.7 (95.0)	0.907 (0.074)	0.906 (0.074)
30	652.8 (108.1)	0.927 (0.069)	0.928 (0.067)
40	804.8 (132.4)	0.965 (0.039)	0.963 (0.041)
50	975.4 (141.1)	0.977 (0.024)	0.974 (0.026)

Reference System	F1 [Euzenat et al., 2011a]
AgreementMaker	0.69
SEREMI	0.68
Zhishi.links	0.92

Table 3.10: Results for the NewYorkTimes data set.

Comparison With Manually Created Linkage Rules

In addition, we evaluated how the learned linkage rules compare to linkage rules that have been manually created by a human expert for the same data set.

LinkedMDB: The manually written linkage rule for the LinkedMDB data set compares movies by their label as well as their release date. For the evaluation we used a manually created set of 100 positive and 100 negative reference links. Special care was taken to include relevant corner cases such as movies which share the same title but have been produced in different years.

Table 3.11 summarizes the cross validation results for the LinkedMDB data set. Figure 3.19 depicts a learned linkage rule.

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	4.2 (1.2)	0.981 (0.011)	0.959 (0.023)
10	19.6 (2.1)	0.998 (0.001)	0.921 (0.007)
20	40.3 (3.3)	1.000 (0.000)	0.974 (0.004)
30	59.2 (4.2)	1.000 (0.000)	0.999 (0.002)
40	74.2 (7.2)	1.000 (0.000)	0.999 (0.002)
50	99.7 (12.8)	1.000 (0.000)	0.999 (0.002)

Table 3.11: Results for the LinkedMDB data set.

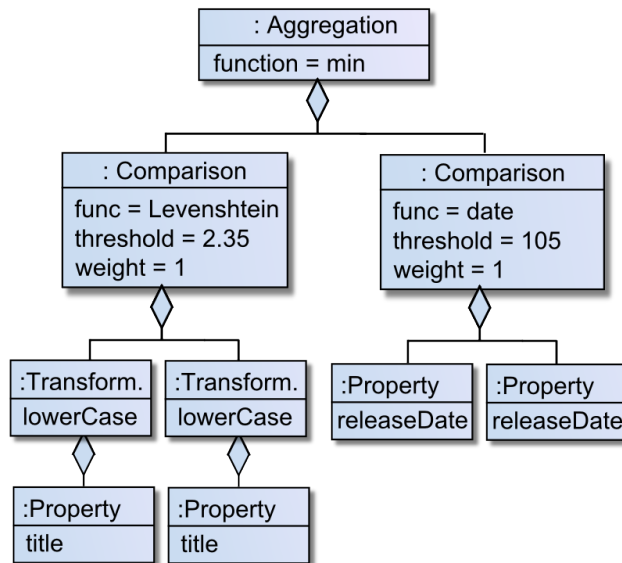


Figure 3.19: Learned linkage rule for the LinkedMDB data set

In all runs, the learning algorithm needed no more than 12 iterations in order to achieve the full training F-measure. The learning algorithm learned linkage rules that compare movies by their title and their release date just as the original human-created linkage rule did.

DBpediaDrugBank: While the vast majority of linkage rules commonly used in the Linked Data context are very simple, the original linkage rule for interlinking drugs in DBpedia and DrugBank that has been developed in the course of the LATC EU project is an example of a complex linkage rule. In order to match drugs in DBpedia and DrugBank, it is necessary to compare various identifiers, such as the CAS number²⁰), in addition to the names of the drugs and known synonyms. In total, the manually written linkage rule uses 13 comparisons and 33 transformations, including transformations that replace specific parts of the strings.

Figure 3.20 shows a slightly simplified version of the original linkage rule. All 1,403 links that have been generated by executing the original linkage rule have been used as positive reference links.

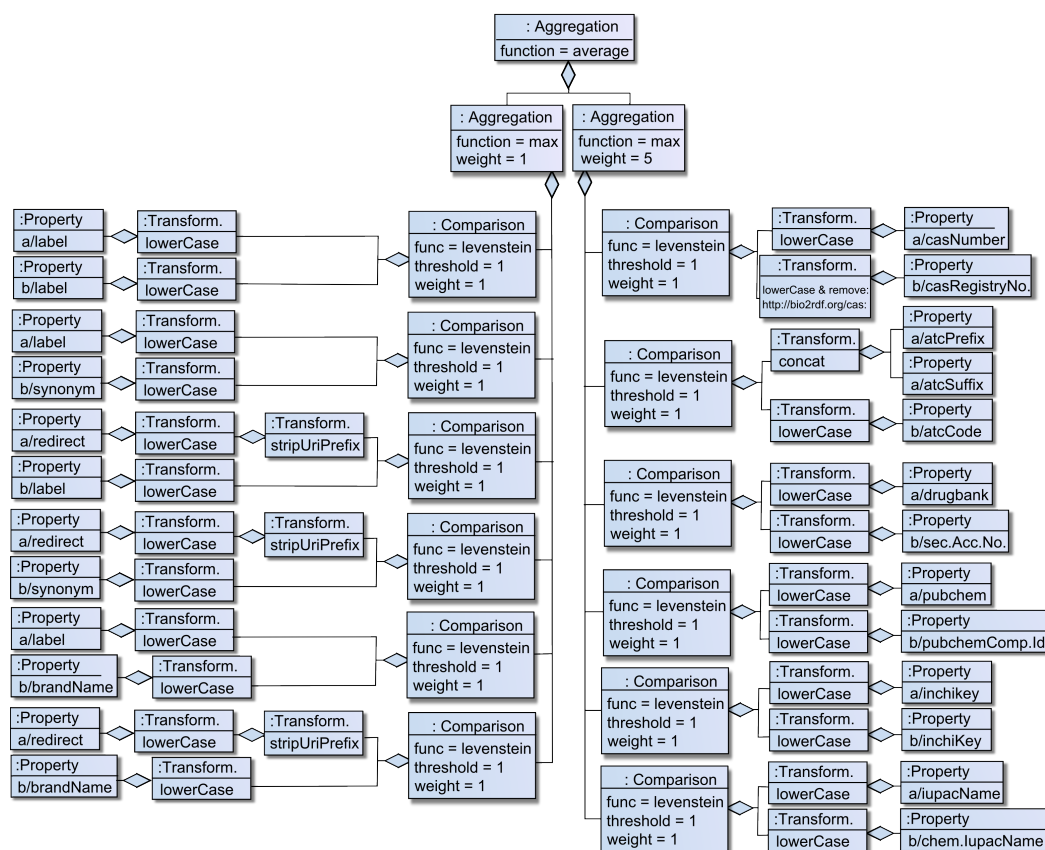


Figure 3.20: Original linkage rule for the DBpediaDrugBank data set

Table 3.12 summarizes the cross validation results for the DBpediaDrugBank data set. The learned linkage rules yield an F-Measure of 99.8% for

²⁰A unique numerical identifier assigned by the "Chemical Abstracts Service"

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	67.5 (2.2)	0.929 (0.026)	0.928 (0.029)
10	334.1 (157.4)	0.994 (0.002)	0.991 (0.003)
20	1014.1 (496.8)	0.996 (0.001)	0.988 (0.010)
30	1829.7 (919.3)	0.997 (0.001)	0.985 (0.016)
40	2685.4 (1318.9)	0.998 (0.001)	0.994 (0.002)
50	3222.2 (1577.7)	0.998 (0.001)	0.994 (0.002)

Table 3.12: Results for the DBpediaDrugBank data set

the training data and 99.4% for the test data.

From the 30th iteration the generated linkage rules on average only use 5.6 comparisons and 3.2 transformations and the parsimony pressure successfully avoids bloating in the subsequent iterations. Thus, the learned linkage rules use less than half of the comparisons and only one-tenth of the transformations of the human written linkage rules. Figure 3.21 shows how the linkage rule increased in size during the execution of the learning algorithm.

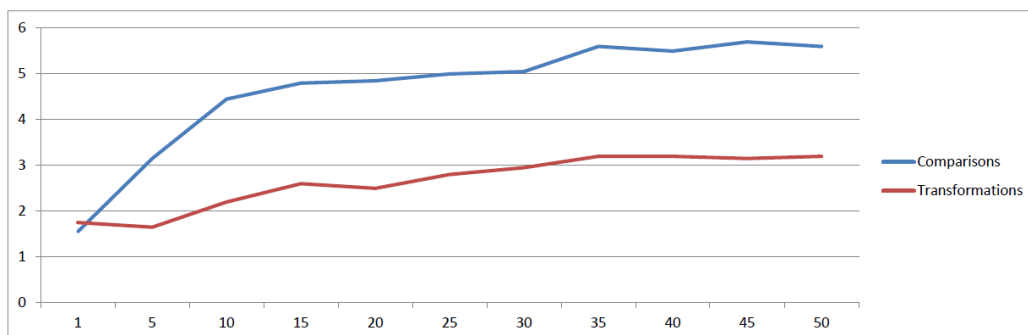


Figure 3.21: Average number of comparisons and transformations

3.5.5 Detailed Evaluation

While the previous section focused on the evaluation of the performance of the overall algorithm this section focuses on evaluating specific parts of our approach. One of the main claims of this thesis is that using the expressive linkage rule representation presented in Section 2.5 allows the algorithm to learn rules with higher accuracy. We evaluate this claim by comparing the performance of learned linkage rules using the proposed representation versus common representations in record linkage. After that, we show how our

	Boolean	Linear	Nonlin.	Full
Cora	0.900	0.896	0.898	0.965
Restaurant	0.954	0.959	0.951	0.992
SiderDrugBank	0.931	0.956	0.966	0.970
NewYorkTimes	0.714	0.716	0.724	0.916
LinkedMDB	0.973	0.986	0.987	0.997
DBpediaDrugBank	0.990	0.981	0.991	0.993
Average	0.910	0.916	0.920	0.972

Table 3.13: Representations: F-measure in round 25

approach to generate the initial population improves the average accuracy of the initial linkage rules. Finally, we evaluate how the proposed specialized crossover operators improve the learning performance over subtree crossover.

Comparison with Other Linkage Rule Representations

We presented an approach that uses a linkage rule representation that is more expressive than other representations used in record linkage. Our model includes chains of transformations and is also able to represent non-linear classifiers.

In order to measure the effect of this extended representation, we evaluated the learning performance of four representations:

- **Boolean:** Boolean classifiers without transformations
- **Linear:** Linear classifiers without transformations
- **Non-linear:** Non-linear classifiers without transformations
- **Full:** Our Approach with full expressivity

Table 3.13 shows the F-measure on the test set after 25 iterations. We now review how the introduction of non-linearity and transformations in our approach improves the learning performance: On the SiderDrugBank and the NewYorkTimes data sets, using non-linear classifiers yields better results than either boolean or linear classifiers. On all six data sets, the introduction of transformations improved the performance.

Seeding

In this experiment we evaluated if our approach of generating the initial population improves over the completely random generation of linkage rules,

	Random	Seeded
Cora	0.849 (0.045)	0.865 (0.018)
Restaurant	0.963 (0.010)	0.985 (0.012)
SiderDrugBank	0.624 (0.181)	0.848 (0.013)
NewYorkTimes	0.178 (0.164)	0.701 (0.072)
LinkedMDB	0.719 (0.175)	0.975 (0.008)
DBpediaDrugBank	0.702 (0.217)	0.957 (0.013)

Table 3.14: Seeding: Initial F-measure

	Random	Seeded
Cora	0.928 (0.023)	0.961 (0.006)
Restaurant	0.997 (0.004)	0.997 (0.004)
SiderDrugBank	0.932 (0.048)	0.962 (0.007)
NewYorkTimes	0.370 (0.114)	0.814 (0.002)
LinkedMDB	0.960 (0.022)	0.991 (0.012)
DBpediaDrugBank	0.962 (0.007)	0.994 (0.001)

Table 3.15: Seeding: F-measure in round 10

which is usually used in genetic programming. Table 3.14 compares the average F-measure of the linkage rules in the initial population for each data set. The table shows that for data sets with only a few properties, such as the Cora and the Restaurant data set, the seeding does not yield a significant improvement. However, for data sets with many properties, the improvement over the complete random generation is significant and increases the average F-measure of the linkage rules in the population considerably.

Note that seeding not only improves the average F-measure of the linkage rules in the initial population, but also reduces the number of generations that are needed to learn a linkage rule. Table 3.15 compares the average F-measure after 10 iterations. The seeding improves the F-measure after 10 iterations in all cases except for the Restaurant data set for which after 10 iterations the maximum F-measures is almost reached in both cases.

Crossover Operators

Subtree crossover is the de-facto standard in genetic programming [Koza, 1993; Koza et al., 2005]. We evaluated the actual contribution to the learning performance of using specialized crossover operators as described in Section 3.3.3 instead. Table 3.16 compares the performance of both configura-

	10 Iterations	
	Subtree Crossover	Our Approach
Cora	0.943 (0.015)	0.951 (0.013)
Restaurant	0.997 (0.004)	0.997 (0.004)
SiderDrugBank	0.919 (0.013)	0.963 (0.013)
NewYorkTimes	0.814 (0.015)	0.834 (0.016)
LinkedMDB	0.985 (0.012)	0.991 (0.009)
DBpediaDrugBank	0.992 (0.002)	0.994 (0.002)
	25 Iterations	
	Subtree Crossover	Our Approach
Cora	0.959 (0.007)	0.967 (0.003)
Restaurant	0.997 (0.004)	0.997 (0.004)
SiderDrugBank	0.974 (0.004)	0.987 (0.003)
NewYorkTimes	0.814 (0.005)	0.916 (0.006)
LinkedMDB	0.996 (0.007)	0.998 (0.003)
DBpediaDrugBank	0.994 (0.001)	0.997 (0.002)

Table 3.16: Crossover experiment: F-measure after 10 and 25 iterations.

tions after executing 10 iterations and again after 25 iterations.

In all data sets our approach either matches the subtree crossover results or outperforms them. In addition, the specialized crossover operators in our approach have the advantage that each operator only covers a specific aspect of a linkage rule. Thus, the operators can be selectively enabled to control which aspects of a linkage rule are learned.

Bloating Control

Two properties are essential for a bloating control method:

- (1) It does not reduce the learning performance.
- (2) It does avoid the unrestricted growth of the individuals.

This section evaluates both properties on four different configurations:

None: No bloating control strategy.

Penalty (Pen.): Parsimony pressure as described in Section 3.2.3.

Pruning (Prun.): Pruning trees as described in Section 3.3.4.

	None	Pen.	Prun.	Comb.
Cora	0.946	0.944	0.946	0.945
Restaurant	0.993	0.992	0.991	0.993
SiderDrugbank	0.970	0.964	0.969	0.970
NewYorkTimes	0.813	0.807	0.812	0.812
DBpediaDrugbank	0.993	0.992	0.992	0.994
LinkedMDB	0.995	0.994	0.996	0.996

Table 3.17: F-measure after 10 iterations for different bloating control strategies.

	None	Pen.	Prun.	Comb.
Cora	7.83	7.33	7.17	6.50
Restaurant	9.33	5.83	6.00	5.17
SiderDrugbank	21.50	15.33	20.00	14.83
NewYorkTimes	15.17	5.33	11.17	5.27
DBpediaDrugbank	15.67	9.83	14.17	9.50
LinkedMDB	8.67	8.17	7.83	6.83
Average	13.03	8.64	11.06	8.02

Table 3.18: Number of comparisons and transformations in a learned linkage rule size after 10 iterations for different bloating control strategies.

Combine (Comb.): Both strategies combined.

Table 3.17 evaluates if any of the tested configurations affects the learning performance by comparing the F-measure on the test set after the 10th iteration. Table 3.18 compares the total number of comparisons and transformations of the learned linkage rules after 10 iterations.

While the learning performance is unaffected by the choice of the bloating control strategy, all three methods reduce the size of the linkage rules. The biggest reduction is achieved by combining both methods, i.e., using pruning in addition to parsimony pressure further reduces the linkage rule size. Compared to using no bloating control, the combined strategy reduces the number of comparison and transformations from 13 to 8 on average.

We will now evaluate the bloating control in more detail for the Cora data set. Figure 3.22 shows how the test F-measure develops for all four configurations. None of the three tested bloating control methods decrease the performance of the learning algorithm. Figure 3.23 shows how the number of comparisons grow during learning while Figure 3.24 shows the development

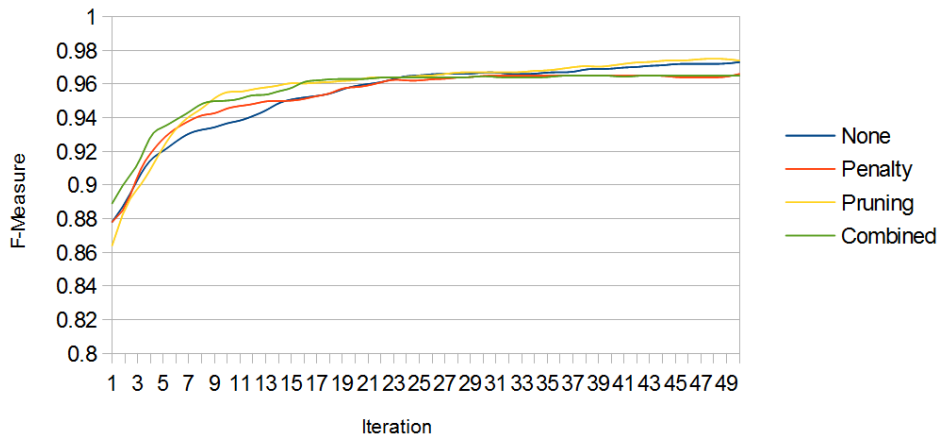


Figure 3.22: Learning performane

of the number of transformations. All three bloating control strategies

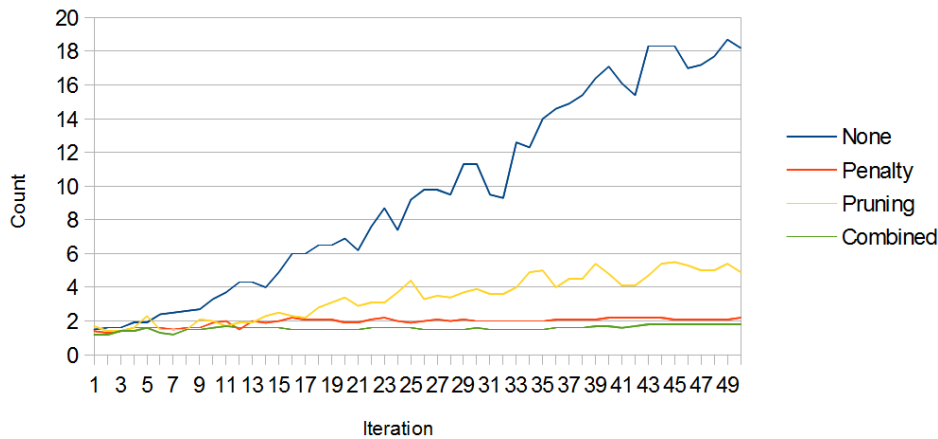


Figure 3.23: Number of comparisons

reduce the size of the linkage rules significantly. Figure 3.25 and 3.26 zoom on the parsimony pressure strategy alone and the same strategy combined with pruning. Pruning further reduces the size of the linkage rules without sacrificing learning performance.

3.6 Summary

In this chapter, we motivated machine learning of linkage rules by pointing out that creating linkage rules by hand is a non-trivial problem and requires a high level of expertise together with detailed knowledge of the data sets.

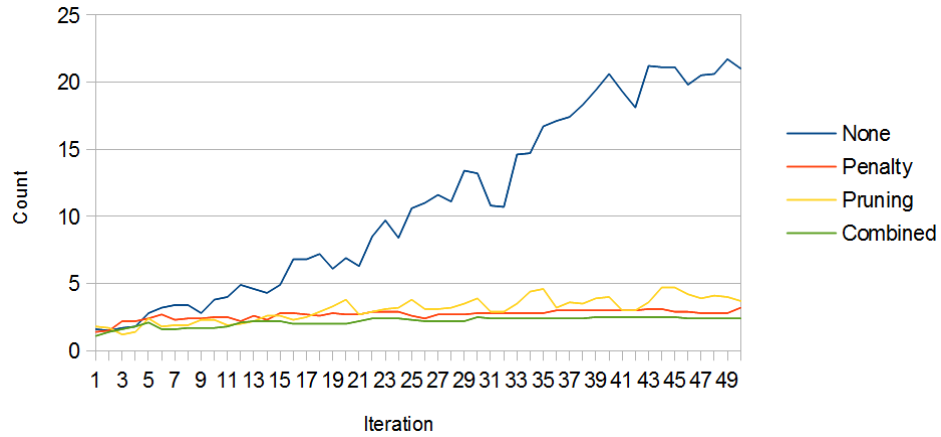


Figure 3.24: Number of transformations

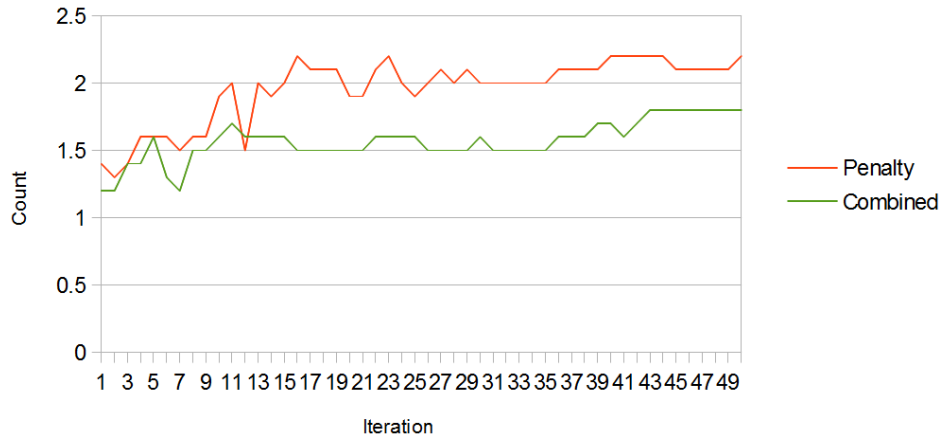


Figure 3.25: Number of comparisons

We introduced supervised learning as a method to automate the creation of a linkage rule in cases where existing reference links are known.

As one of the main contributions of this thesis, we proposed the GenLink supervised learning algorithm. GenLink employs genetic programming in order to learn linkage rules from a set of existing reference links by using a set of specialized crossover operators. GenLink is capable of learning linkage rules covering the full expressivity of the linkage rule representation that has been introduced in the previous chapter.

GenLink has been evaluated on six data sets from three areas:

- (1) We evaluated GenLink on two frequently used record linkage data sets and compared the performance to previously proposed learning algorithms that have been evaluated on the same data sets. On both data

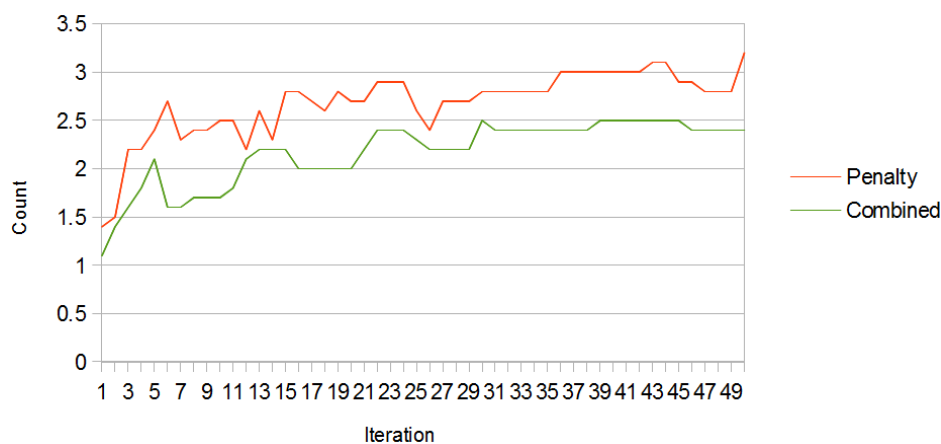


Figure 3.26: Number of transformations

sets, GenLink outperformed the *MARLIN* [Bilenko and Mooney, 2003] entity matching system that supports learning linear classifiers using support vector machines. Likewise, GenLink outperformed the state-of-the-art genetic programming approach for entity matching by de Carvalho et al. [2012] on both data sets. Finally, we also showed that GenLink outperforms two collective approaches on both data sets [Domingos, 2004; Dong et al., 2005].

- (2) We evaluated the performance on two data sets from the Ontology Alignment Evaluation Initiative and compared our results to the participating systems. By using the reference links as training data, GenLink outperformed the participating unsupervised systems and achieved an F-measure of about 97% on both data sets.
- (3) We evaluated how the learned linkage rules compare to linkage rules that have been manually created by a human expert for the same data set. On a movie data set, GenLink successfully reproduced a manually written linkage rule that compares movies by their title and their release date. On a more complex life science data set, GenLink was capable of learning linkage rules that achieve a similar accuracy than the human written rule that uses 13 comparisons and 33 transformations in order to deduplicate drugs. In addition, the employed bloating control strategy successfully avoids the uncontrolled growth of the learned linkage rules. The learned linkage rules on average only consisted of 5.6 comparisons and 3.2 transformations and thus have been smaller than the original linkage rule.

In addition to the evaluation of the overall performance, we evaluated by

which amount specific parts of GenLink contribute to the overall performance of the learned linkage rules:

- (1) First, we evaluated how the expressive linkage rule representation, which is proposed in this thesis and constitutes one of its main contributions, increases the learning performance over previously proposed common representations. On two data sets, the capability to represent non-linear classifiers improved the performance over linear and threshold-based boolean classifiers. The introduction of transformations further improved the performance on all data sets.
- (2) In the second experiment, we showed that the seeding strategy that is used by GenLink improves the performance of the initial population and subsequent generations.
- (3) GenLink uses a set of specialized crossover operators wherein each operator is targeted at learning a specific aspect of a linkage rule. Compared to subtree crossover, which is the de-facto standard in genetic programming, using specialized crossover operators has the advantage that each operator can be selectively enabled or disabled in order to control which aspects of the linkage rules are learned. The experimental evaluation showed that on all evaluation data sets, the performance of the specialized crossover operators either matches the subtree crossover results or outperforms them.

While the proposed GenLink algorithm depends on existing reference links, the next chapter will introduce the ActiveGenLink active learning algorithm, which extends GenLink with an interactive generation of reference links and thereby makes GenLink usable in cases where no previous reference links are available.

Chapter 4

Active Learning of Linkage Rules

In the previous chapter, we presented the supervised GenLink algorithm for learning linkage rules. GenLink is capable of learning expressive linkage rules in cases where positive and negative reference links are available. As existing reference links are often not available, they need to be created prior to executing GenLink.

As also noted in the previous chapter, reference links can be generated by asking a human expert to confirm or reject the equivalence of a number of entity pairs from the data sets. When selecting a subset of pairs of entities for labeling from the data sets, an obvious problem is to decide which entity pairs are chosen to be labeled by the human annotator. As manually labeling entity pairs is time-consuming and thus costly, it is desirable to minimize the number of pairs which are chosen.

Deciding which pairs of entities are to be labeled by the human annotator is an important task, but it is also a challenging one. The reason for this is that in order for a supervised learning algorithm to perform well on unlabeled data, the reference links need to include all relevant corner cases. We illustrate this point by having a look at the example of interlinking two cities as depicted in Figure 4.1: While for many cities a comparison by label

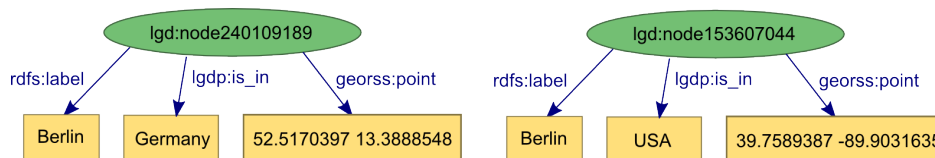


Figure 4.1: Example of an entity pair in a geographical data set.

is sufficient to determine if two entities represent the same real-world city,

the given example shows the rare corner case when distinct cities share the same name. If the entity pairs to be labeled by the user are just selected randomly from data sets, the user has to label a very high number of pairs in order to include these rare corner cases reliably. As manually labeling link candidates is time-consuming, methods to reduce the number of candidates that need to be labeled are desirable.

The fundamental idea of active learning is to reduce the number of link candidates that need to be labeled by actively selecting the most informative candidate for labeling [Settles, 2009]. Active learning can work on top of any existing supervised learning algorithm.

In this chapter, we present ActiveGenLink, an algorithm for learning linkage rules interactively using active learning and genetic programming. ActiveGenLink learns a linkage rule by asking the user to confirm or reject a number of link candidates, which are actively selected by the algorithm. Compared to writing linkage rules by hand, ActiveGenLink lowers the required level of expertise as the task of generating linkage rules is automated by the genetic programming algorithm while the user only has to verify a set of link candidates. The employed query strategy reduces user involvement by selecting the link candidate with the highest information gain for manual verification.

4.1 Active Learning

The idea of active learning is to actively guide the process of labeling examples by the user while learning a model from the set of already labeled examples. Active learning is an iterative algorithm. In each iteration, the algorithm selects the most informative unlabeled example to be labeled by the user and uses a machine learning algorithm to learn a model based on the extended labeled training data.

An active learning algorithm bootstraps by initializing three sets:

Unlabeled Pool: Contains a set of training examples for which the label is yet unknown. Note that in the context of this work, we focus on *pool-based sampling*, i.e., the examples are selected from a pool of unlabeled examples. Other sampling methods, such as *membership query synthesis* and *stream-based selective sampling*, are not covered. An overview of common sampling methods can be found in [Settles, 2009].

Labeled Training Set: Contains the set of training examples that have been labeled by the human annotator. Initially this set is empty.

Model: The model that represents the solution. During the execution of the active learning algorithm, the model will be trained according to the labeled examples. If no labeled training examples are available, the active learning algorithm can either employ unsupervised learning to generate an initial model or start with a random model.

Figure 4.2 summarizes the three steps that are involved in each iteration:

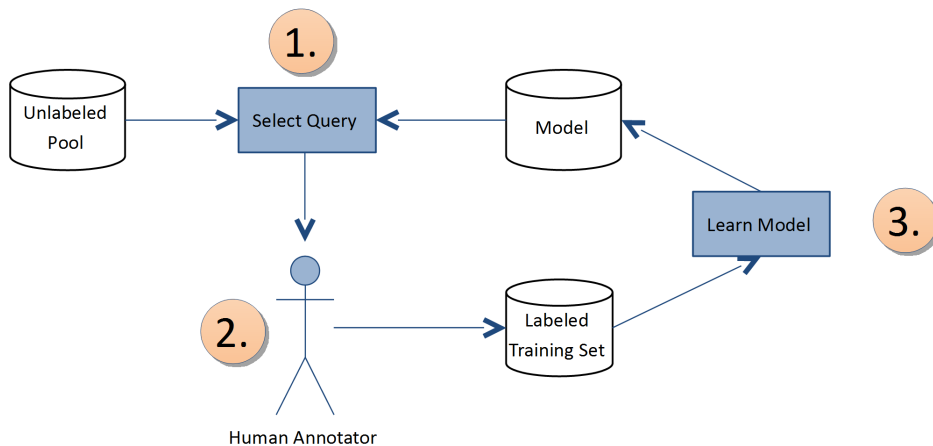


Figure 4.2: Active learning workflow.

- (1) A query strategy selects an example for labeling from the unlabeled pool, which is forwarded to the human annotator. The fundamental idea of the query strategy is to minimize the number of necessary queries by selecting the most informative example. Section 4.1.1 introduces common query strategies.
- (2) A human expert labels the selected example. In the context of this work, we view labeling as a binary decision (i.e., confirming or declining an example), although active learning in general is not restricted to learning binary classifiers.
- (3) A supervised learning algorithm learns a model based on the updated labeled training examples. Active learning is independent of any specific machine learning method and can for instance be used with support vector machines, decision trees or genetic programming. As Section 3.4 already discussed popular approaches for learning linkage rules, this chapter does not provide a separate discussion of supervised learning algorithms.

4.1.1 Query Strategies

The goal of the query strategy is to reduce the number of examples that need to be labeled in order to learn a model that generalizes to the entire set of unlabeled examples. Most existing query strategies can be categorized in three variants [Muslea et al., 2006]:

Uncertainty sampling¹: The query strategy selects the example for which the current model is the least certain.

Expected-error minimization: The query strategy selects the example that most likely reduces the expected generalization error the most.

Version space reduction: The query strategy selects the example that yields the biggest reduction of the version space. The version space is the set of all models that are consistent with the current training examples.

A comprehensive overview of frequently used query strategies can be found in [Settles, 2009].

In the following we will give an overview of uncertainty sampling and introduce a class of version space reduction strategies known as query-by-committee. We do not cover expected-error minimization strategies as they are computationally expensive [Settles, 2009].

4.1.2 Uncertainty Sampling

One of the most commonly used query frameworks is known as *uncertainty sampling*. The rationale behind uncertainty sampling is that the example is selected for which the current model is the least certain. While there are different variants of uncertainty sampling, we are considering the variant when the example with the maximum entropy is selected:

Definition 4.1 (Uncertainty Sampling) *Given an unlabeled pool \mathcal{U} , uncertainty sampling selects the example with the maximum entropy:*

$$x_{LC}^* = \operatorname{argmax}_{u \in \mathcal{U}} H(P_r(u))$$

where H denotes the entropy, which - for the binary case - is defined as:

$$H(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

¹Also called uncertainty reduction [Muslea et al., 2006].

For each example u from the unlabeled pool \mathcal{U} , $P_r(u)$ denotes the probability that the given example is correct according to the learned model. By maximizing the entropy, uncertainty sampling selects the example for which the probability is closest to 50%.

4.1.3 Query-by-Committee

While uncertainty sampling only takes a single model into account, genetic algorithms train a population of alternative models at the same time. In order to take advantage of the population of models, we continue by introducing version space reduction strategies known as *query-by-committee*. Query-by-committee methods select the query based on the voting of a committee of candidate solutions. By that, they aim to reduce the version space by selecting the example for which the members in the committee disagree the most. Ideally, the committee is a subset of the version space, which is the set of candidate solutions that are consistent with the current training examples. Two major approaches have been proposed in literature [Settles, 2009]: *query-by-vote-entropy* and *query-by-Kullback-Leibler-divergence*.

The *query-by-vote-entropy strategy* [Dagan and Engelson, 1995] selects the example with the maximum disagreement in the committee.

Definition 4.2 (Query-By-Vote-Entropy) *Given an unlabeled pool \mathcal{U} , the query-by-vote-entropy strategy selects the example with the maximum vote entropy:*

$$x_V^* = \operatorname{argmax}_{u \in \mathcal{U}} H(P_C(u))$$

Where P_C represents the committee voting, which is defined for an unlabeled example $u \in \mathcal{U}$ and a committee \mathcal{C} as:

$$P_C(u) = \frac{\sum_{l \in \mathcal{C}} P_l(u)}{|\mathcal{C}|}$$

The query-by-vote-entropy strategy aims to reduce the version space by selecting the unlabeled example for which the members in the committee disagree the most. The idea of the vote entropy is that unlabeled examples for which either most committee members confirm or decline the example receive a low score. On the other side, unlabeled examples for which distribution of committee members that confirm the example and the members that decline it is balanced receive the highest disagreement score. In the ideal case, i.e., 50% of the committee members confirm the example while the other 50% decline it, following the goal of reducing the version space, the version space is cut in half.

Instead of just measuring the disagreement among the committee members, the idea of the *query-by-Kullback-Leibler-divergence strategy* [McCallum and Nigam, 1998] is to measure the average divergence between the voting of one committee member and the consensus voting. The Kullback-Leibler divergence is used to evaluate the divergence from the mean.

Definition 4.3 (Query-By-Kullback-Leibler-Divergence) *Given an unlabeled pool \mathcal{U} and existing training data R , query-by-Kullback-Leibler-divergence selects the example with the maximum Kullback-Leibler divergence from any existing example:*

$$x_{KL}^* = \operatorname{argmax}_{u \in \mathcal{U}} \sum_{r \in R} \frac{D_{KL}(P_r(u) \parallel P_R(u))}{|R|}$$

The Kullback-Leibler divergence [Kullback and Leibler, 1951] (sometimes also called information gain or relative entropy), which is a measure of the difference between two probability distributions, is defined as:

$$D_{KL}(p \parallel q) = p \log_2 \left(\frac{p}{q} \right) + (1 - p) \log_2 \left(\frac{1 - p}{1 - q} \right)$$

Using the Kullback-Leibler divergence in an active learning approach imposes two practical problems: While Kullback-Leibler divergence is always positive, it is not finite (i.e., no upper bound can be assumed for its value). In addition, if $p > 0$, it is only defined for $q > 0$.

To overcome these two problems, many approaches rely on the related Jensen-Shannon divergence instead:

$$D_{JS}(p \parallel q) = \frac{1}{2} D_{KL}(p \parallel m) + \frac{1}{2} D_{KL}(q \parallel m) \quad \text{where} \quad m = \frac{p + q}{2}$$

Which can be simplified to:

$$D_{JS}(p \parallel q) = H \left(\frac{p + q}{2} \right) - \frac{H(p) + H(q)}{2}$$

Figure 4.3 compares Kullback-Leibler divergence and Jensen-Shannon divergence. In contrast to Kullback-Leibler divergence, Jensen-Shannon divergence is both symmetric and bounded to the interval $[0, 1]$.

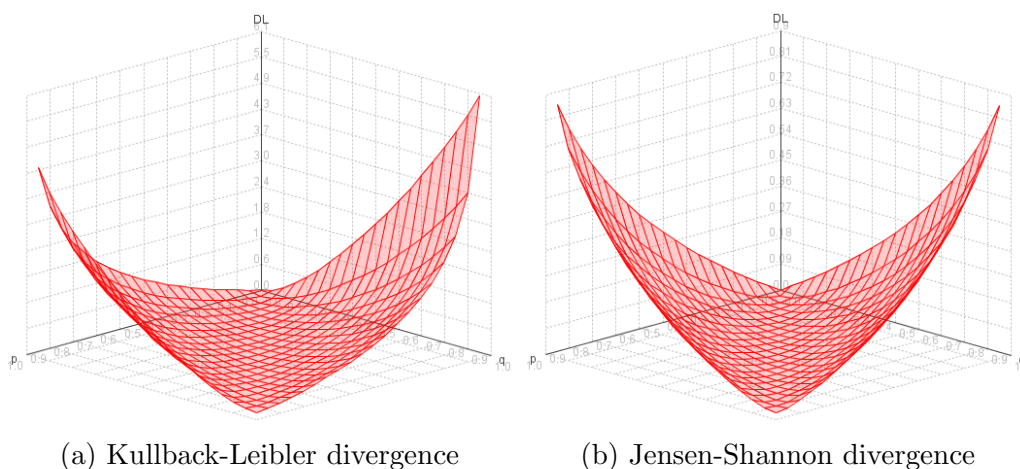


Figure 4.3: Kullback-Leibler divergence and Jensen-Shannon divergence.

4.2 The ActiveGenLink Algorithm

This section describes the proposed ActiveGenLink algorithm, which combines genetic programming and active learning to learn linkage rules interactively.

The main idea of ActiveGenLink is to evolve a population of candidate solutions iteratively while building a set of reference links. The algorithm starts with a random population of linkage rules and an initially empty set of reference links. In each iteration, it selects a link candidate for which the current population of linkage rules is uncertain from a pool of unlabeled links. After the link has been labeled by a human expert, it evolves the population of linkage rules based on the extended set of reference links. Figure 4.4 summarizes the three steps that are involved in each iteration:

- (1) The query strategy selects link candidates to be labeled by the user from an unlabeled pool of entity pairs (i.e., *pool-based sampling* [Settles, 2009]). Entity pairs are selected according to a query-by-committee [Seung et al., 1992] strategy, i.e., the selected link candidate is determined from the voting of all members of a committee, which consists of the current linkage rules in the population. As the linkage rules from the population are all trained on the current reference links they represent competing hypotheses. Thus, they are part of the version space which is the space of all linkage rules that correctly match the known reference links. The aim of the query strategy is to select link candidates which reduce the version space as much as possible.
- (2) A human expert labels the selected link as correct or incorrect. Con-

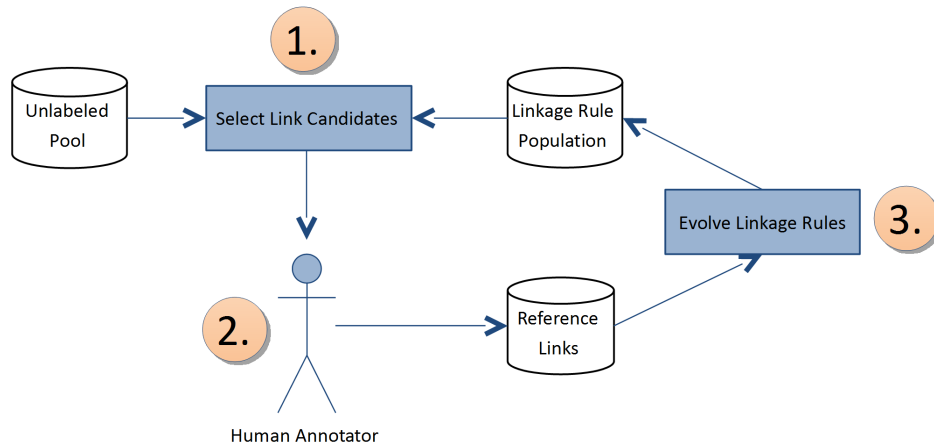


Figure 4.4: Overview of the ActiveGenLink workflow.

firmed links are added to the positive reference link set and declined links are added to the negative reference link set.

- (3) The GenLink genetic programming algorithm evolves the population of linkage rules. The goal is to find linkage rules which cover the current set of reference links. The population is evolved until either a maximum number of iterations is reached or a linkage rule has been found which covers all reference links.

The pseudocode of ActiveGenLink is provided in Algorithm 4.1.

Listing 4.1: Pseudocode of the ActiveGenLink algorithm. The specific parameter values used in our experiments are listed in Section 3.5.2.

```

1   $P \leftarrow$  generate initial population
2   $\mathcal{U} \leftarrow$  generate unlabeled pool
3   $R \leftarrow$  empty set of reference links
4
5  while( $|R| <$  maximum links to be labeled) {
6     $u \leftarrow$  query strategy selects link candidate from  $\mathcal{U}$ 
7     $r \leftarrow$  ask user to label  $u$ 
8     $R \leftarrow R \cup r$ 
9     $P \leftarrow$  learn linkage rules from  $R$  based on population  $P$ 
10 }
11
12 return best linkage rule from  $P$ 

```

4.2.1 Query Strategy

The goal of the query strategy is to reduce the number of links that need to be labeled. While many query strategies, such as uncertainty sampling, assume that a single model is being trained by the learning method, genetic algorithms train a population of alternative models at the same time. Section 4.1.1 already introduced *query-by-committee* methods, which can take advantage of this set of competing models based on the voting of a committee of candidate solutions. Ideally, the committee is a subset of the version space, which is the set of candidate solutions that correctly match the current reference links. In our case, the committee is built by the linkage rules in the current population. Query-by-committee strategies usually aim to reduce the version space by selecting the unlabeled candidate that reduces the version space the most.

Given the query-by-vote-entropy strategy as a baseline, we now present an improved strategy which is based on two observations:

- **Link Distribution:** The unlabeled links are usually not distributed uniformly across the similarity space but build clusters of links which convey similar information concerning the characteristics according to which both interlinked entities are similar.
- **Suboptimal Committee:** The voting committee, which is built from the population of linkage rules, may contain suboptimal linkage rules that do not cover all reference links. As a result, the query-by-vote-entropy strategy allows linkage rules to vote that only cover a small part of the set of reference links.

The next paragraphs will take a more detailed look at both observations and show how we account for them in the proposed query strategy.

Accounting for Link Distribution

Usually multiple link candidates convey similar information concerning the characteristics according to which both interlinked entities are similar. For this reason, labeling a single link candidate can be representative of a high number of related link candidates. Thus, it is not necessary to label the entire pool of possible link candidates, but only a subset thereof and it is the goal of the query strategy to minimize the size of this subset of candidates which need to be labeled.

In order to get a better understanding of this observation, we look at a simple example using a data set about movies. For illustration we only consider two dimensions: The similarity of the movie titles as well as the

similarity of the release date. Figure 4.5 shows the distribution of a set of link candidates between movies from the LinkedMDB data set used in Section 4.4 according to these two characteristics. For instance, the cluster

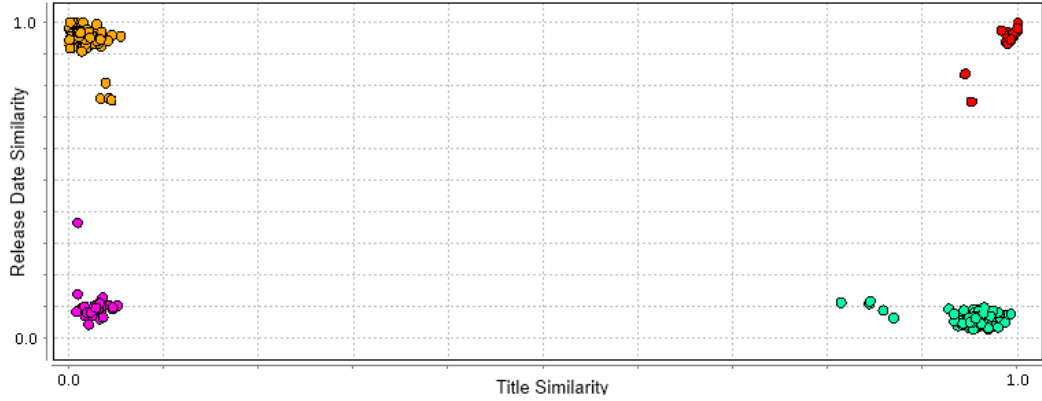


Figure 4.5: Distribution of movies in the similarity space

in the top-right corner represents link candidates between movies that share a similar title as well as a similar release date, while the cluster in the bottom-right corner represents link candidates between movies that share a similar title, but have been released at different dates. We can easily see that a query strategy needs to label link candidates from all four clusters in order to include all necessary information to learn an appropriate linkage rule.

The idea of our proposed query strategy is to distribute the links onto different clusters by only selecting links for labeling that are different from already labeled links. For measuring the extent to which a specific unlabeled link differs from an existing reference link we use the Jensen-Shannon divergence, which has been introduced in Section 4.1.1. Intuitively, we can interpret the Jensen-Shannon divergence between an unlabeled link candidate and a reference link as the amount of information that is needed to encode the label of the unlabeled link, given that the label of the reference link is known. If the divergence is zero, we expect every linkage rule to return the same label for the unlabeled link as it did for the reference link. In that case the unlabeled link would already be perfectly represented by the given reference link and there is no need in labeling it. On the other hand, if the divergence to all existing reference links is large, we expect that the information gained by labeling the link candidate is not yet contained in the set of reference links.

Based on the Jensen-Shannon divergence, we propose the *query-by-divergence* strategy:

Definition 4.4 (Query-By-Divergence) *We define query-by-divergence*

as the strategy that selects the unlabeled link candidate that has the maximum divergence from any existing reference link:

$$x_I^* = \underset{u \in \mathcal{U}}{\operatorname{argmax}} \underset{r \in R}{\operatorname{argmin}} D_{JS}(P_C(u) || (P_C(r)))$$

We now take a closer look at how the result of the query-by-divergence strategy is computed: At first, the Jensen-Shannon divergence is used to determine the distance of each unlabeled link to every reference link. For each unlabeled link, the inner *argmin* expression selects the divergence from the closest reference link. By that we get a list of all unlabeled links together with a divergence value for each link. From this list, the outer *argmax* expression selects the unlabeled link for which the highest divergence has been found.

Accounting for Suboptimal Committee

Both the query-by-vote-entropy as well as the proposed query-by-divergence are based on the committee voting of the entire committee. When active learning is used together with genetic algorithms, the committee is built from the members of the evolved population. The population typically contains suboptimal linkage rules that do not cover all existing reference links. For instance, if a data set about movies is deduplicated, the population may contain linkage rules that solely compare by the movie titles. While these linkage rules may cover most reference links, they do not fulfill reference links which relate movies that have the same title, but have been released in different years. By allowing linkage rules to vote that only cover a subset of the reference links, they distort the final voting result.

We account for the suboptimal committee by introducing a modified version of the committee voting which is based on 3 factors:

- An unlabeled pool \mathcal{U} .
- A reference link set R , which is divided into positive reference links R^+ and negative reference links R^- .
- A committee of linkage rules C . The committee is formed by all linkage rules in the current population.

We now define the subset of the committee which is fulfilled by a specific reference link:

Definition 4.5 (Restricted Committee) *Give a reference link r , we define the subset of the committee that fulfills r as:*

$$\mathcal{C}(r) = \begin{cases} \{l \in \mathcal{C} | l(r) > \theta\} & \text{if } r \in R^+ \\ \{l \in \mathcal{C} | l(r) < \theta\} & \text{if } r \in R^- \end{cases}$$

, where θ is the global threshold and l a linkage rule in the committee \mathcal{C} .

We define the restricted committee voting similarly to the committee voting $P_{\mathcal{C}}(l)$, which we already defined for the query-by-vote-entropy strategy:

Definition 4.6 (Restricted Committee Voting) *Given a link u and a reference link r , the restricted committee voting is defined as:*

$$\bar{P}_{\mathcal{C}}(u, r) = \frac{\sum_{c \in \mathcal{C}(r)} P_c(u)}{|\mathcal{C}(r)|}$$

The idea of the restricted committee voting is that only linkage rules which fulfill a specific reference link are allowed to vote.

Combined Strategy

Based on the query-by-divergence strategy and the restricted committee voting, we can now define the query strategy used by the ActiveGenLink algorithm.

Definition 4.7 (Proposed Query Strategy) *The proposed query strategy selects the unlabeled link candidate that has the maximum divergence from any existing reference link:*

$$x_I^* = \underset{u \in \mathcal{U}}{\operatorname{argmax}} \underset{r \in R}{\operatorname{argmin}} D_{JS}(\bar{P}_{\mathcal{C}}(u, r) || (\bar{P}_{\mathcal{C}}(r, r)))$$

When assessing the divergence between a unlabeled link candidate and a reference link, using the previously defined restricted committee voting guarantees that only linkage rules that fulfill the given reference link are allowed to vote. Thus, if a linkage rule from the population does not fulfill a specific reference link, it does not distort the computation of the divergence from that particular reference link.

4.2.2 Building the Unlabeled Pool

The overall goal of the active learning algorithm is to create a linkage rule that is able to label all possible entity pairs as matches or non-matches with high confidence. The number of possible entity pairs can be very high for large data sets and usually far exceeds the number of actual matches. For this reason, we use an indexing approach to build a sample that does not include definitive non-matches.

Given two data sets A and B , the initial unlabeled pool $\mathcal{U} \subset A \times B$ is built according to the following sampling process: The sampling starts by querying for all entities in both data sets. Instead of retrieving all entities at once, a stream of entities is generated for each data set. For each property in the streamed entities, all values are indexed according to the following scheme:

- (1) All values are normalized by removing all punctuation and converting all characters to lower case.
- (2) The normalized values are tokenized into words.
- (3) A set of indices is assigned to each token. The indices are generated so that tokens within an edit distance of 1 share at least one index. The MultiBlock blocking algorithm is used to generate the index. MultiBlock will be described in detail in Section 5.3.
- (4) The indices of all tokens of a value are merged. If in total more than five indices have been assigned to a value, 5 indices are randomly selected while discarding the remaining indices.

After the index has been generated, all pairs of entities that have been assigned the same index are added to the unlabeled pool until a configured maximum size is reached. Figure 4.6 illustrates the sampling of a single property.

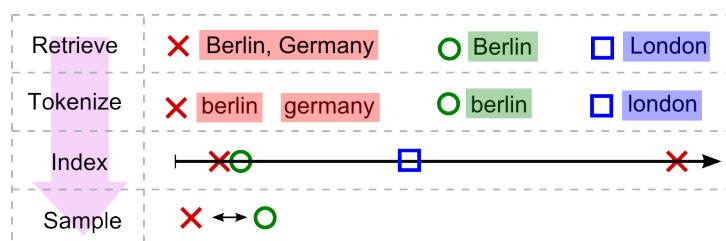


Figure 4.6: Sampling of entities by label

4.3 Previous Work on Active Learning

Section 3.4 already discussed related work in supervised learning of linkage rules as well as on unsupervised entity matching. The field of related work that applies active learning to entity matching is significantly smaller. In this section, we describe previous active learning approaches for learning linear and threshold-based boolean classifiers as well as active learning approaches that employ genetic programming. We close this section by discussing the introduced approaches and comparing them to the ActiveGenLink algorithm that is proposed in this thesis.

4.3.1 Linear and Threshold-based Boolean Classifiers

Active Atlas

Active Atlas [Tejada et al., 2001] is an entity matching system that integrates supervised and active learning techniques. The supervised learning algorithm that is provided by Active Atlas has been introduced in Section 3.4.2. In addition, Active Atlas integrates an active learning technique for generating linkage rules based on decision trees. Active Atlas uses the query strategy that selects link candidates based on the disagreement of a committee. As Active Atlas learns a single linkage rule at once, the committee is created by using the bagging technique [Breiman, 1996] to generate multiple versions of the current linkage rule. The active learning algorithm that is integrated into Active Atlas has been evaluated on the same data sets as the supervised variant, as described in Section 3.4.2. The evaluation of the active learning algorithm shows that the active learning version of Active Atlas needs significantly fewer reference links in order to achieve similar or better results than the chosen baseline.

ALIAS

ALIAS [Sarawagi and Bhamidipaty, 2002] is an active learning algorithm that can be used together with various supervised machine learning algorithms for generating linkage rules. ALIAS has been evaluated with three supervised learning algorithms:

- (1) Naive Bayes classifiers as implemented in $\mathcal{MLC}++$ [Kohavi et al., 1994].
- (2) The C4.5 algorithm for learning decision trees [Quinlan, 1993].
- (3) The SVM Torch algorithm for learning support vector machines [Collobert and Bengio, 2001].

For each of the chosen supervised learning algorithms, ALIAS has been evaluated on two data sets: A data set that contains citations from CiteSeer [Giles et al., 1998] and an address data set that has been provided by a telephone company. Figure 4.7 shows an example of a decision tree that has been learned by ALIAS for the bibliographic data set.

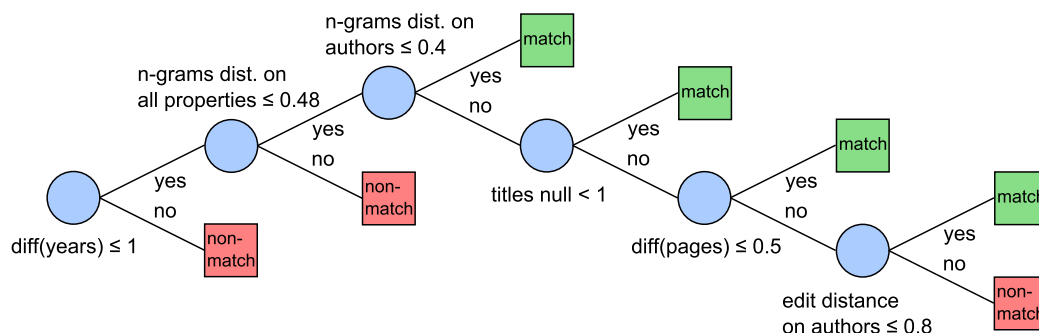


Figure 4.7: Example of a decision tree that has been learned by ALIAS for a bibliographic data set (adapted from [Sarawagi and Bhamidipaty, 2002]).

The evaluation results show that while decision trees outperform support vector machines on both data sets when only a few reference links are available, support vector machines outperform decision trees from the point the user labeled about 10 to 30 link candidates. Naive Bayes classifiers are outperformed by both decision trees and support vector machines.

When comparing the results with a random baseline (i.e., the query strategy selects random examples from the unlabeled pool for labeling), ALIAS managed to achieve the peak F-measure of 97% for the bibliographic data set and 98% for the address data set after labeling less than 100 link candidates, while the random baseline only achieved less than 50% on both data sets after labeling the same number of link candidates.

Arasu et al.

Arasu et al. [2010] propose a scalable active learning algorithm for entity matching by introducing the assumption of *monotonicity of precision*. While they show that their algorithm can scale to large data sets, it is only able to learn simple linear or boolean classifiers. The proposed active learning algorithm has been evaluated on two use cases:

- (1) Matching two data sets, both containing 10^6 entities that describe companies.
- (2) Matching a set of 10^5 citations with another set of 10^6 citations.

The origin of the data sets is not specified. In addition to evaluating their proposed approach, Arasu et al. also conduct a comparison with Active Atlas and ALIAS. On the evaluation data sets, the proposed approach achieved the peak F-measure after labeling about 100 link candidates and outperformed both Active Atlas and ALIAS. The time needed by the query strategy to select a link candidate for labeling is reported as less than a second on average.

RAVEN

RAVEN [Ngonga Ngomo et al., 2011] is an algorithm for active learning of linear or boolean linkage rules that is specifically targeted at RDF data sets. RAVEN has been evaluated on three use cases:

- (1) Matching diseases from DBpedia and Diseasesome².
- (2) Matching drugs from Sider³ and DrugBank⁴.
- (3) Matching side-effects from Sider and Diseasesome.

For each use case, the links that are generated by a manually written linkage rule have been used as reference links. The corresponding linkage rules are not provided in the paper. On the evaluation data sets, RAVEN was able to generate linkage rules that achieve an F-measure of over 90% after the user labeled 4 to 12 link candidates.

4.3.2 Genetic Programming

Freitas et al.

Freitas et al. [de Freitas et al., 2010] present an approach that combines genetic programming and active learning for learning linkage rules for entity matching. Their algorithm is based on the supervised genetic programming algorithm proposed by de Carvalho et al. [2012] as described in Section 3.4. In order to choose linkage candidates for labeling, it uses a query strategy that is based on the disagreement of the committee members. Similar to the query-by-vote-entropy strategy discussed earlier, the query strategy selects the link candidate for which the committee is the most uncertain.

Freitas et al. evaluated their active learning algorithm on three data sets:

- A synthetic data set that contains person data.

²<http://wifo5-03.informatik.uni-mannheim.de/diseasome/>

³<http://wifo5-03.informatik.uni-mannheim.de/sider/>

⁴<http://wifo5-03.informatik.uni-mannheim.de/drugbank/>

- The data set that contains citations from CiteSeer that also has been used by ALIAS for evaluation.
- The Restaurant data set that has been used for evaluation of a number of supervised approaches. The Restaurant data set has been described in detail in Section 3.5.1.

On all three evaluation data sets, Freitas et al. showed that their proposed algorithm outperforms ALIAS and achieves similar F-measures than the corresponding supervised approach by Carhalho et al. when labeling at most 170 link candidates.

EAGLE

EAGLE [Ngonga Ngomo and Lyko, 2012] is another approach that applies genetic programming and active learning to the problem of learning linkage rules interactively. *EAGLE* learns linkage rules that are represented as a tree, which allows it to learn rules with a similar complexity as ActiveGenLink, but does not support transformations. Similar to the previously introduced approach by Freitas et al., *EAGLE* uses a query strategy that is based on the disagreement of the linkage rules in the population.

EAGLE has been evaluated on three data sets:

- Interlinking drugs in Dailymed⁵ and DrugBank⁶.
- Interlinking movies in DBpedia and LinkedMDB⁷.
- Interlinking publications from ACM and DBLP [Köpcke et al., 2010].

On the drugs data set, *EAGLE* achieved an F-measure of 99.9% after labeling 10 link candidates. On the movie data set, *EAGLE* achieved an F-measure of 94.1% after labeling 50 link candidates. On the ACM-DBLP data set, *EAGLE* compared the achieved results with two other systems: FEBRL and MARLIN. While FEBRL and MARLIN outperform *EAGLE* slightly in F-measure, the runtime performance of *EAGLE* is significantly higher.

4.3.3 Discussion

In the following, we will discuss various aspects of the previously introduced active learning algorithms. For each algorithm, we will discuss the used

⁵<http://wifo5-03.informatik.uni-mannheim.de/dailymed/>

⁶<http://wifo5-03.informatik.uni-mannheim.de/drugbank/>

⁷<http://www.linkedmdb.org>

query strategy, enumerate the learned features of the linkage rules, and state the classifier that is used to carry out the match decision. In each part, we will discuss how the presented active learning algorithms compare to ActiveGenLink.

Query Strategy

With the exception of the approach by Arasu et al., all previously introduced active learning approaches employ a query strategy closely based on either uncertainty sampling or the query-by-vote-entropy strategy as introduced in Section 4.1.1.

ActiveAtlas, ALIAS, Freitas et al., EAGLE: The query strategies used in ActiveAtlas, ALIAS, the approach by Freitas et al., and EAGLE select the link candidate for which a committee of candidate linkage rules disagree the most similar to the query-by-vote-entropy strategy. They only differ in the method that is used to build the committee: Because Active Atlas and ALIAS learn a single linkage rule instead of a population of linkage rules, they create a committee by generating multiple different versions of the currently learned linkage rule. For this purpose, Active Atlas employs the bagging technique [Breiman, 1996], while ALIAS uses parameter randomization [Seung et al., 1992]. For the genetic programming algorithms EAGLE and the one proposed by Freitas et al., the current population of linkage rules, which has been evolved by the genetic programming algorithm, serves as committee.

RAVEN: In order to choose a linkage candidate for labeling, RAVEN uses uncertainty sampling: The linkage candidate for which the current linkage rule is the least certain (i.e., the predicted probability that the given link candidate is a true link is close to 50%) and which has not been labeled already, is chosen by the query strategy.

Arasu et al.: Arasu et al. is the only introduced method that uses a novel query strategy that has not been proposed in literature before. The proposed query strategy assumes that a similarity space (i.e., each dimension specifies the similarity according to a specific property) is given. The goal is to identify a similarity threshold for each dimension of the similarity space. The assumption of the query strategy is that whenever a similarity threshold is increased, the precision of the classifier over the similarity space is increased as well. Based on this assumption, the query strategy is capable of efficiently selecting link candidates. The downside of the proposed query strategy is that it is only capable of

learning a set of learning linear classifiers or threshold-based boolean classifiers without transformations.

ActiveGenLink uses a novel query strategy that aims at distributing the link candidates that are selected for labeling uniformly across the version space by selecting link candidates that exhibit a maximum divergence from any existing reference link. In Section 4.4.4 we will show that the query strategy that is used by ActiveGenLink outperforms query-by-vote-entropy on the evaluation data sets.

Learned Features

In the following, we compare which parts a linkage rule are learned by the introduced approaches and which parts have to be specified manually by the user.

Active Atlas, ALIAS, Arasu et al.: Active Atlas, ALIAS, and the approach by Arasu et al. require the user to specify comparisons pairs that consist of two parts: The property that is to be compared and the distance measure that is used to compare values of the specified property. As a single property is specified instead of a pair of properties, these approaches also require that the schema of the matched data sets is normalized when finding intra-source duplicates between sets entities.

Freitas et al.: The approach by Freitas et al. includes an automatic feature selection that chooses comparisons pairs, consisting of a property and a distance measure, automatically.

RAVEN, EAGLE: RAVEN and EAGLE use a property matching algorithm that finds pairs of matching properties that are later used by the learning algorithm for building the linkage rule. RAVEN does not learn appropriate distance measures, but instead always uses the n-gram distance measure on properties with string values and a numeric distance measure for properties that carry numeric values. EAGLE does include the distance measures into the linkage rule representation and by that enables the genetic programming algorithm to learn appropriate distance measures for each compared property.

None of the introduced active learning algorithms learns chains of transformations for normalization of the property values that are to be compared. Active Atlas allows the user to include a list of manually defined transformations, but does not build chains of transformations by itself.

ActiveGenLink includes the definition of discriminative properties for comparison, the construction of chains of data transformations and the selection of appropriate distance measures into the lineage rule representation. ActiveGenLink is the only algorithm that supports learning linkage rules that include chains of data transformations to normalize values prior to comparison.

Supported Classifiers

Active Atlas supports learning threshold-based boolean classifiers ALIAS, the approach by Arasu et al., and RAVEN support learning linear classifiers as well as learning threshold-based boolean classifiers. The genetic programming approaches EAGLE, the approach by Freitas et al., and ActiveGenLink use an expressive linkage rule representation that enables them to generate linkage rules that combine comparisons non-linearly beyond pure threshold-based boolean classifiers.

4.4 Evaluation and Discussion

In this section, we evaluate ActiveGenLink experimentally: Section 4.4.1 will describe our experimental setup. Section 4.4.2 will evaluate if by labeling a small number of links, the proposed active learning algorithm is capable of learning linkage rules with a similar accuracy than the supervised learning algorithm GenLink on all available reference links. As in contrast to the supervised learning algorithm, the active learning needs to take all unlabeled data into account in order to generate the queries, we evaluated the scalability of the active learning algorithm in Section 4.4.3. Finally in Section 4.4.4, we will evaluate the contribution of the proposed query strategy compared to the query-by-vote-entropy strategy.

4.4.1 Experiment Setup

For evaluation of ActiveGenLink, we used the identical six data sets as we used to evaluate the supervised GenLink algorithm. Section 3.5.1 already introduced all six evaluation data sets.

Each experiment has been executed by loading all entities in the corresponding data set, but no reference links, and running ActiveGenLink on the loaded entities. Instead of using a human annotator, the link candidates that have been selected by the query strategy have been automatically labeled as correct if the link candidate has been found in the positive reference links

and as incorrect otherwise. The positive reference links for each data set are complete, i.e., for each pair of matching entities there is a positive reference link. Each time a link has been labeled and after the approach updated the learned linkage rule, we evaluated the performance of the learned linkage rule using the complete set of reference links. We executed all experiments until either the learned linkage rule fully covered all reference links or 50 iterations have been reached. For each experiment we also compared the final performance to the performance of the supervised learning approach when being trained on the complete set of reference links.

Each experiment has been run 10 times, averaging the final results. All experiments have been run on a 3GHz Intel(R) Core i7 CPU with four cores while the Java heap space has been restricted to 1GB.

Table 4.1 lists the parameters that have been used in all experiments for the active learning algorithm. It also lists the parameters that have been used for the genetic programming algorithm.

Parameter	Value
Unlabeled pool size $ \mathcal{U} $	10,000
Maximum links to be labeled	50
Population size	500
Max. iterations per labeled link	25
Selection method	Tournament selection
Tournament size	5
Probability of crossover	75%
Probability of mutation	25%
Stop condition	F-measure = 1.0
Penalty factor	0.05

Table 4.1: ActiveGenLink Parameters

4.4.2 Comparison with Supervised Learning

In this section, we evaluate the performance of the ActiveGenLink approach on the same data sets as have been used to evaluate the supervised GenLink algorithm. We show that by labeling a small number of links, ActiveGenLink achieves a comparable performance as GenLink on the complete reference link set.

Note that the interpretation of the training F-measure is different for the supervised and the active learning evaluation: In the supervised evaluation in each run all available reference links have been split into a training set

and a equal-sized test set for the cross validation. Thus, the training F-measure denotes the performance on the training set on which the algorithm has been trained on while the test F-measure denotes the performance on the test set. In the active learning evaluation, the training F-measure denotes the performance on the links that have been labeled so far. Here, the test F-measure denotes the performance on the complete reference link set.

Ontology Alignment Evaluation Initiative

We evaluated the performance of the ActiveGenLink algorithm on the OAEI data sets in detail. Note that, as the OAEI only compares unsupervised systems and does not consider systems that are supplied with existing reference links, ActiveGenLink has an advantage over the participating systems. For that reason, we used the official OAEI results merely as a baseline for our approach and evaluated the number of links that had to be labeled in order to outperform the participating unsupervised systems.

Table 4.2 summarizes the active learning results for the SiderDrugBank data set. After labeling two links, the test F-measure outperforms the un-

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	4.7 (1.1)	1.000 (0.000)	0.393 (0.091)
2	7.5 (1.7)	1.000 (0.000)	0.484 (0.136)
3	10.7 (2.5)	1.000 (0.000)	0.714 (0.130)
4	15.4 (4.6)	1.000 (0.000)	0.725 (0.141)
5	20.8 (7.4)	1.000 (0.000)	0.725 (0.141)
10	68.4 (2.1)	1.000 (0.000)	0.793 (0.073)
15	117.8 (7.5)	1.000 (0.000)	0.947 (0.006)
20	189.9 (5.7)	1.000 (0.000)	0.962 (0.024)
25	240.1 (13.5)	1.000 (0.000)	0.962 (0.024)
30	308.0 (40.3)	1.000 (0.000)	0.971 (0.011)
GL	301.5 (39.0)	0.972 (0.006)	0.970 (0.007)
Reference System		F1	
ObjectCoref		0.464 [Hu et al., 2010]	
RiMOM		0.504 [Wang et al., 2010]	

Table 4.2: Results for the SiderDrugBank data set. The GL row contains the F-measure that is achieved by the supervised algorithm on the entire set of reference links. The results of the participants of the OAEI 2010 challenge are included for comparison.

supervised ObjectCoref system and after labeling a third link the F-measure already outperforms the RiMOM system. About 30 links had to be labeled until a linkage rule could be learned that achieved a similar F-measure than the ones learned by GenLink by using all 1718 reference links.

The time column states the time in seconds that was needed to execute both the query strategy that selects the link candidates to be labeled and the learning of the linkage rules according to the updated reference links. The time needed to confirm or decline the link candidates is not included.

In order to get a better idea on how the learned linkage rule evolves during the iterations, we illustrate one active learning run by showing three linkage rules that have been learned after different number of iterations: Figure 4.8 shows a learned linkage rule after 10 links have been labeled, Figure 4.9 after labeling 20 links and finally Figure 4.10 after labeling 30 links. The first

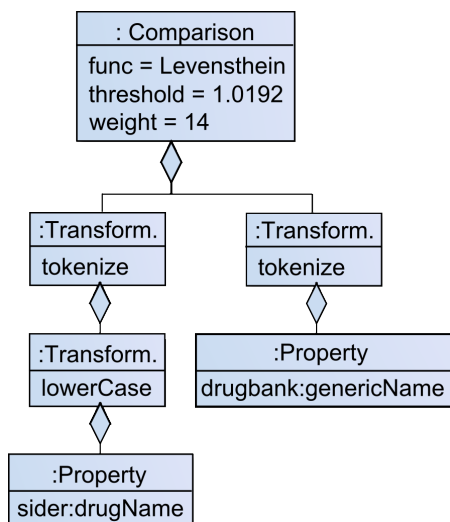


Figure 4.8: Example of a learned linkage rule after labeling 10 links.

linkage rule is very simple and only consists of one comparison of the drug names in both data sets. By looking at the two subsequent linkage rules, we can see that this specific comparison, with minor modifications, was carried through all learned linkage rules. The second linkage rule introduces a second comparison with the brand name of the drug in DrugBank. This comparison is also carried over to the third linkage rule, which adds a third comparison that compares the PubChem Compound Identifiers, which identify unique chemical structures.

Table 4.3 summarizes the active learning results for the NewYorkTimes data set. AgreementMaker and SEREMI have been outperformed after labeling four links. 30 links had to be labeled in order to outperform the

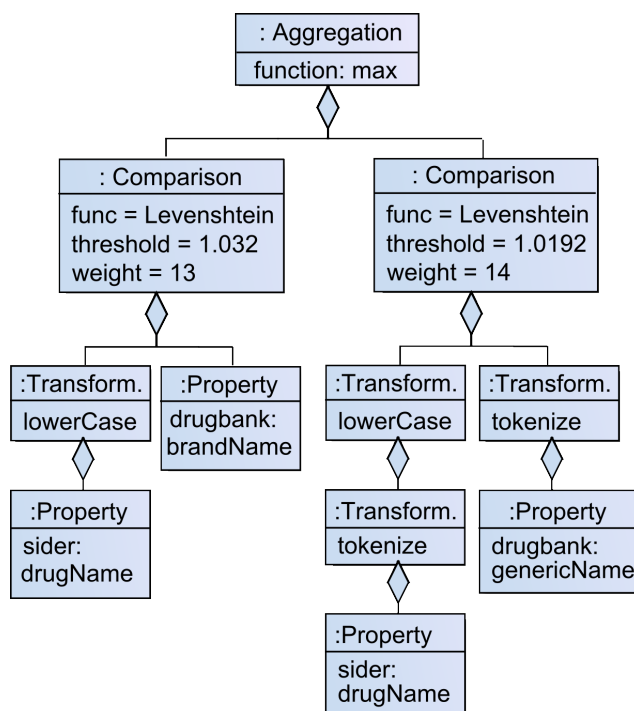


Figure 4.9: Example of a learned linkage rule after labeling 20 links.

Zhishi.links system after an F-measure of 90% has been reached by labeling 25 links. The NewYorkTimes data set is the only data set in which the maximum number of 50 links was not sufficient to achieve the same F-measure as GenLink achieved on the full reference link set.

Frequently Used Record Linkage Data sets

Table 4.4 summarizes the active learning results for the Cora data set. The results show that after labeling five links, the learned linkage rules already achieves a F-measure of over 90% on the complete reference link set. After labeling 40 links, the active learning algorithm achieves similar results as the supervised learning algorithm on all 3234 reference links.

Table 4.5 summarizes the active learning results for the Restaurant data set. The results show that labeling nine links was sufficient to achieve the same performance as GenLink on all reference links.

Comparison With Manually Created Linkage Rules

Table 4.6 summarizes the active learning results for the LinkedMDB data set. The results show that the active learning algorithm achieves an F-measure

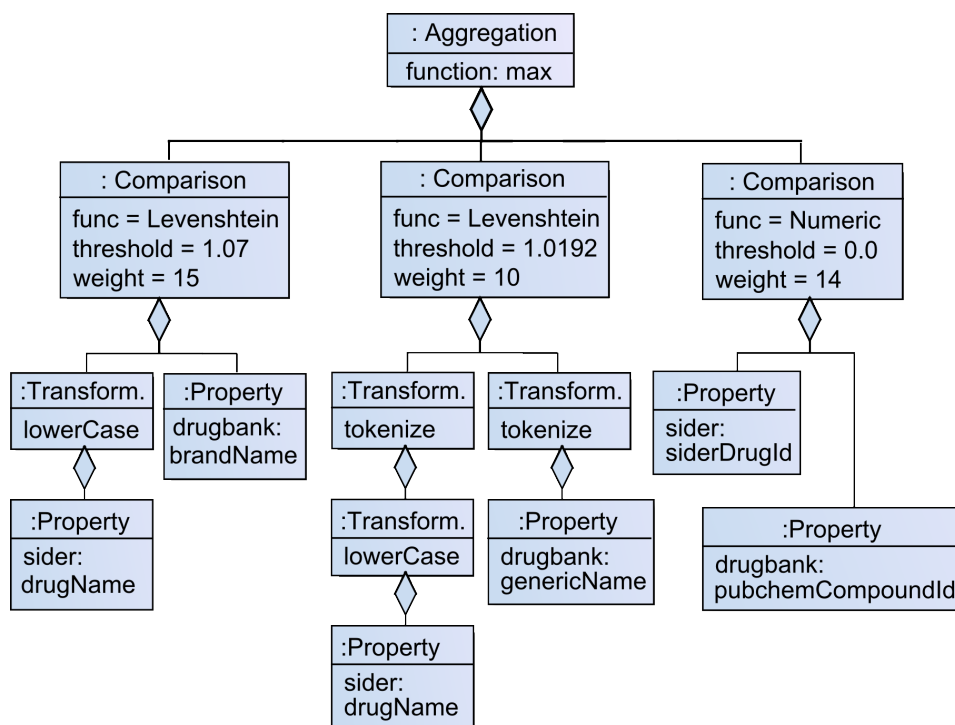


Figure 4.10: Example of a learned linkage rule after labeling 30 links.

of over 90% by labeling five links. After labeling 15 links the learned linkage rules almost reached an F-measure of 100%.

Table 4.7 summarizes the active learning results for the DBpediaDrug-Bank data set. The results show that the active learning algorithm achieves an F-measure of over 90% by labeling 10 links. After labeling 50 links the learned linkage rules achieve the same performance as the supervised algorithm on all reference links.

4.4.3 Scalability

In this experiment we show that ActiveGenLink is able to scale to large data sets. For evaluation we use the example of learning a linkage rule for interlinking settlements in DBpedia and LinkedGeoData⁸ [Stadler et al., 2011]. At the time of writing, DBpedia contains 323,257 settlements while LinkedGeoData contains 560,123 settlements. The execution of the learned linkage rules generates over 70,000 links. While, in the case of passive learning the learning algorithm only needs to take the provided reference links into

⁸<http://linkedgeo.org/>

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	7.9 (27.8)	0.500 (0.500)	0.480 (0.128)
2	11.8 (69.5)	1.000 (0.000)	0.501 (0.107)
3	21.1 (159.3)	1.000 (0.000)	0.694 (0.112)
4	29.4 (227.5)	1.000 (0.000)	0.791 (0.023)
5	39.2 (322.1)	1.000 (0.000)	0.727 (0.087)
10	69.7 (563.4)	0.967 (0.033)	0.814 (0.000)
15	104.5 (778.2)	0.978 (0.022)	0.845 (0.035)
20	151.6 (1025.3)	1.000 (0.000)	0.850 (0.037)
25	216.0 (1247.7)	0.989 (0.011)	0.901 (0.087)
30	370.5 (1524.0)	0.958 (0.000)	0.923 (0.125)
35	1046.5 (6080.1)	0.990 (0.010)	0.929 (0.128)
40	1514.3 (5768.6)	0.992 (0.017)	0.931 (0.081)
50	2803.2 (8492.2)	0.993 (0.016)	0.931 (0.080)
GL	975.4 (141.1)	0.977 (0.024)	0.974 (0.026)
Reference System		F1 [Euzenat et al., 2011a]	
AgreementMaker		0.69	
SEREMI		0.68	
Zhishi.links		0.92	

Table 4.3: Results for the NewYorkTimes data set. The GL row contains the F-measure that is achieved by the supervised algorithm on the entire set of reference links. The results of the participants of the OAEI 2011 challenge are included for comparison.

account, active learning also needs to take the pool of unlabeled data into account in order to generate the queries. As the unlabeled data consists of the complete Cartesian product, which in this example amounts to over 180 billion pairs, the active learning algorithm clearly cannot work on the whole set. For this reason, this experiment evaluates if the sampling algorithm managed to build a reduced unlabeled pool that is still representative enough to cover the relevant cases. In order to evaluate the learned linkage rules we used a manually collected set of 100 positive and 100 negative reference links. Special care has been taken to include rare corner cases, such as for example cities that share the same name but do not represent the same city, and different cities that are located very closely to each other.

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	2.2 (0.6)	0.500 (0.500)	0.700 (0.023)
2	3.5 (0.1)	0.500 (0.500)	0.840 (0.010)
3	5.1 (0.2)	1.000 (0.000)	0.849 (0.083)
4	8.1 (1.3)	1.000 (0.000)	0.886 (0.065)
5	12.0 (1.2)	1.000 (0.000)	0.916 (0.018)
10	72.2 (3.5)	1.000 (0.000)	0.925 (0.053)
15	154.4 (4.6)	1.000 (0.000)	0.932 (0.038)
20	262.3 (11.3)	1.000 (0.000)	0.946 (0.019)
25	375.5 (3.8)	1.000 (0.000)	0.960 (0.009)
30	492.5 (15.7)	1.000 (0.000)	0.960 (0.010)
35	834.6 (184.6)	0.991 (0.009)	0.960 (0.010)
40	1099.4 (326.9)	1.000 (0.000)	0.964 (0.003)
GL	185.8 (26.7)	0.969 (0.003)	0.966 (0.004)

Table 4.4: Results for the Cora data set. The last row contains the results of the supervised algorithm.

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	1.2 (0.4)	1.000 (0.000)	0.489 (0.037)
2	1.5 (0.5)	1.000 (0.000)	0.637 (0.111)
3	2.2 (0.1)	1.000 (0.000)	0.776 (0.075)
4	2.6 (0.1)	1.000 (0.000)	0.778 (0.029)
5	3.4 (0.1)	1.000 (0.000)	0.761 (0.035)
6	4.2 (0.5)	1.000 (0.000)	0.870 (0.075)
7	5.0 (0.2)	1.000 (0.000)	0.932 (0.059)
8	5.6 (0.2)	1.000 (0.000)	0.935 (0.061)
9	6.2 (0.0)	1.000 (0.000)	0.993 (0.002)
10	6.8 (0.0)	1.000 (0.000)	0.993 (0.003)
GL	6.3 (5.3)	0.996 (0.004)	0.993 (0.006)

Table 4.5: Results for the Restaurant data set. The last row contains the results of the supervised algorithm.

Passive Learning

Table 4.8 summarizes the cross validation results. In all runs, an F-measure of 100% has been reached before the 25th iteration.

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	2.3 (0.9)	0.500 (0.500)	0.643 (0.151)
2	3.3 (1.0)	1.000 (0.000)	0.706 (0.122)
3	4.3 (1.0)	1.000 (0.000)	0.815 (0.100)
4	6.7 (2.4)	1.000 (0.000)	0.818 (0.065)
5	7.9 (2.5)	1.000 (0.000)	0.911 (0.020)
10	26.9 (12.1)	0.955 (0.045)	0.987 (0.013)
15	40.1 (19.0)	1.000 (0.000)	0.999 (0.002)
GL	99.7 (12.8)	1.000 (0.000)	0.999 (0.002)

Table 4.6: Results for the LinkedMDB data set. The last row contains the results of the supervised algorithm

Iter.	Time in s (σ)	Train. F1 (σ)	Test F1 (σ)
1	23.4 (3.3)	1.000 (0.000)	0.740 (0.124)
2	38.0 (8.4)	1.000 (0.000)	0.748 (0.118)
3	52.0 (12.6)	1.000 (0.000)	0.646 (0.017)
4	85.3 (29.7)	1.000 (0.000)	0.797 (0.134)
5	101.3 (32.7)	1.000 (0.000)	0.813 (0.150)
10	250.9 (56.1)	1.000 (0.000)	0.945 (0.030)
15	522.8 (259.9)	1.000 (0.000)	0.983 (0.007)
20	700.7 (354.7)	1.000 (0.000)	0.983 (0.007)
25	2558.8 (2107.6)	1.000 (0.000)	0.984 (0.009)
30	4461.2 (3916.9)	1.000 (0.000)	0.984 (0.009)
35	6832.3 (6200.4)	1.000 (0.000)	0.986 (0.010)
40	9885.8 (9104.3)	0.993 (0.007)	0.925 (0.072)
45	14951.6 (13845.9)	0.994 (0.006)	0.989 (0.008)
50	21387.5 (19937.3)	1.000 (0.006)	0.993 (0.008)
GL	3222.2 (1577.7)	0.998 (0.001)	0.994 (0.002)

Table 4.7: Results for the DBpediaDrugBank data set. The last row contains the results of the supervised algorithm

Active Learning

Next we evaluated if the proposed method is able to build a reference link set interactively. We employed the same setup as used in the previous experiment. Table 4.9 shows the results for each five iterations.

The runtimes only include the time needed by the algorithm itself and not the time needed by the human to label the link candidates. It does

Iter.	Time (σ)	Train. F1 (σ)	Test F1 (σ)
1	2.6s (1.0)	0.984 (0.025)	0.932 (0.059)
10	3.8s (2.1)	0.996 (0.007)	0.932 (0.059)
20	3.9s (2.3)	0.998 (0.004)	0.964 (0.032)
25	4.0s (2.4)	1.000 (0.000)	1.000 (0.000)

Table 4.8: Passive learning

Iter.	Time	Train. F1	Test F1
5	7.3s	1.000 (0.000)	0.982 (0.023)
10	15.6s	1.000 (0.000)	1.000 (0.000)

Table 4.9: Active learning

further not include the time needed to build the initial unlabeled pool. As the public endpoints of DBpedia and LinkedGeoData have been used, which offer very restricted query performance, loading the initial unlabeled pool required about two hours.

In all three runs, the algorithm managed to learn a linkage rule with an F-measure of 100% after the second iteration. In the first iteration, it missed the case that two entities with the same name may in fact relate to different cities. In the second iteration, it managed to include this rare case in the proposed link candidates.

On average, ActiveGenLink needed about 1.5 seconds for each iteration, which includes learning a linkage rule from the existing reference links and selecting a link from the unlabeled pool. Thus, ActiveGenLink successfully sampled an unlabeled pool from the original data set that was representative of the entire data set.

4.4.4 Comparison of Different Query Strategies

In this section, we compare the performance of the proposed query strategy with two other strategies:

- **Random:** Selects a random link from the unlabeled pool for labeling (baseline).
- **Entropy:** Selects a link according to the query-by-vote-entropy strategy.

Table 4.10 compares the test F-measure after labeling 10 links. In all

	Random	Entropy	Our Approach
Cora	0.604 (0.222)	0.841 (0.041)	0.917 (0.055)
Restaurant	0.568 (0.195)	0.888 (0.029)	0.993 (0.002)
SiderDrug.	0.309 (0.189)	0.666 (0.007)	0.795 (0.044)
NewYorkTimes	0.467 (0.174)	0.756 (0.080)	0.809 (0.039)
LinkedMDB	0.774 (0.235)	0.948 (0.035)	0.988 (0.005)
DBpediaDrug.	0.654 (0.146)	0.902 (0.076)	0.953 (0.011)

Table 4.10: Query Strategy: F-measure after 10 iterations

cases, the query-by-vote-entropy strategy as well as our proposed query strategy outperformed the random baseline. Our approach outperforms the query-by-vote-entropy strategy on all data sets. For the restaurant data set, our approach already achieved the maximum F-measure that can be achieved with GenLink on this data set as shown in Table 4.5.

Table 4.11 compares the test F-measure after labeling 20 links. For the

	Random	Entropy	Our Approach
Cora	0.762 (0.176)	0.938 (0.026)	0.945 (0.024)
Restaurant	0.707 (0.185)	0.994 (0.001)	0.993 (0.001)
SiderDrug.	0.615 (0.191)	0.926 (0.035)	0.954 (0.043)
NewYorkTimes	0.543 (0.182)	0.741 (0.102)	0.859 (0.084)
LinkedMDB	0.885 (0.209)	0.973 (0.125)	0.998 (0.003)
DBpediaDrug.	0.788 (0.156)	0.973 (0.007)	0.989 (0.003)

Table 4.11: Query Strategy: F-measure after 20 iterations

restaurant data set, the query-by-vote-entropy also reaches the maximum F-measure. For the remaining data sets, our approach outperforms query-by-vote-entropy strategy.

4.5 Summary

In this chapter, we presented the third main contribution of this thesis: the ActiveGenLink learning algorithm. ActiveGenLink is an algorithm for learning linkage rules interactively using active learning and genetic programming. ActiveGenLink learns a linkage rule by asking the user to confirm or reject a number of link candidates, which are actively selected by the algorithm. ActiveGenLink lowers the required level of expertise as the task of generating the linkage rule is automated by the genetic programming algorithm while the

user only has to verify a set of link candidates. The proposed query strategy reduces user involvement by selecting the link candidates to be verified by the user that are the most informative. ActiveGenLink employs the GenLink algorithm for learning linkage rules and thus is capable of learning linkage rules with the same expressivity, i.e., it chooses which properties to compare, it chooses appropriate distance measures, aggregation functions, and thresholds, as well as data transformations, which are applied to normalize data prior to comparison.

Within our experiments, ActiveGenLink outperformed state-of-the-art unsupervised approaches after manually labeling a few link candidates. In addition, ActiveGenLink usually required the user to label less than 50 link candidates in order to generate linkage rules with the same performance performance as the supervised GenLink algorithm on the entire set of reference links. The proposed query strategy required the user to label fewer links than the query-by-vote-entropy strategy.

Chapter 5

Execution of Linkage Rules

The growing number and size of available data sets demands efficient methods for entity matching. A number of indexing methods have been proposed to improve the efficiency of entity matching by reducing the number of required entity comparisons by dismissing definitive non-matches prior to comparison [Elmagarmid et al., 2007]. Unfortunately, many indexing methods may lead to a decrease of recall due to false dismissals [Draisbach and Naumann, 2009]. Therefore, increasing the efficiency usually represents a trade-off between reducing the execution time of the entity matching task on the one hand and retaining the effectiveness of the entity matching task by avoiding a significant decrease of recall on the other hand.

While the previous chapters have been concerned with algorithms for learning linkage rules that are represented using the model that has been introduced in Section 2.5, the practical value of the introduced linkage rule representation also depends on the availability of efficient methods for executing learned linkage rules. In this chapter, we propose a data flow to efficiently execute linkage rules using a multidimensional indexing approach that guarantees that no false dismissals, and thus no decrease of recall, can occur. The proposed indexing approach, is called MultiBlock and constitutes the fourth main contribution of this thesis. The basic idea of MultiBlock is to map entities to a multidimensional index that preserves the distances of the entities, i.e., similar entities will be located near to each other in the index space. While standard blocking techniques block in one dimension, MultiBlock concurrently blocks by multiple properties using multiple dimensions. Thereby it increases its efficiency significantly.

We further propose a distributed data flow for executing linkage rule efficiently on a cluster of machines. For this purpose, we employ the MultiBlock approach to segment the data sets into partitions, which can be executed on remote machines. MultiBlock enables the parallel indexing of entities on

different machines as it generates indices for each entity independently and does not require any global preprocessing. In order to scale to large data sets, we apply the MapReduce paradigm for distributing and executing the partitions on multiple machines.

5.1 Scalability Challenges

When executing linkage rules on local or distributed system, three challenges hinder scalability:

Quadratic Execution Time: Evaluating linkage rules for all pairs of entities scales quadratically.

Parallel Execution: The increasing hardware parallelism demands data flows that can be parallelized in order to utilize multiple computation cores at the same time.

Memory Constraints: Data sets may exceed the size of the available memory and thus cannot be held in memory at once.

In the following paragraphs, we describe each of these challenges and state how the proposed workflow accounts for each of them.

5.1.1 Quadratic Execution Time

The naïve solution for generating links is to evaluate the linkage rule for all pairs of entities, creating a link for each pair that the rule classifies as a match. The disadvantage of the naïve solution is that executing a linkage rule scales quadratically with the number of entities in the data sets. In order to improve the efficiency, indexing approaches have been developed, which aim to dismiss pairs of entities that are definitive non-matches prior to evaluating a linkage rule. Section 5.4 will provide an overview of existing approaches for discussion. In Section 5.3, we propose the MultiBlock indexing approach to reduce the number of required entity comparisons.

5.1.2 Parallel Execution

In addition to reducing the number of comparisons, the efficiency of entity matching can further be improved by executing multiple comparisons concurrently. Two approaches are commonly used for this purpose [Michael et al., 2007]: Firstly, to scale vertically to multi-core processors (scale-up); and secondly to scale horizontally to distributed systems (scale-out).

In order to allow parallel execution of the entity matching process, the workflow splits the required entity comparisons into tasks that can be executed concurrently. While this naturally enables the workflow to scale-up to multiple cores on the same machine, in Section 5.5 we show how the proposed workflow can also be scaled-out to distributed systems.

5.1.3 Memory Constraints

Big data sets may exceed the size of the available memory and thus cannot be held in memory entirely for entity matching. For this reason, we distribute entities into partitions which are stored in a file-backed cache. Each partition can be loaded separately and matched against another partition. This reduces the size of the working set¹ of the entity matching task. In addition, it enables the distribution of partitions to other machines in a distributed system.

5.2 Execution Data Flow

This section introduces the basic data flow that we propose to execute linkage rules efficiently. Figure 5.1 illustrates the overall data flow of executing a linkage rule, which will be detailed in the following paragraphs. The execution of a linkage rule begins by retrieving all entities from the two data sources. Each retrieved entity is indexed and written to the cache. From the cache, pairs of entities that are potential matches are generated from the index. For each pair, the linkage rule is evaluated and a link is generated between each pair of entities that is found matching.

5.2.1 Indexing

Since the evaluation of the linkage rule is computationally expensive, the goal of the indexing phase is to dismiss definitive non-matches prior to comparison. This is achieved by assigning a set of indices to each entity by which later phases can identify definitive non matches. We represent an index as a vector of natural numbers, i.e., as an element from the set \mathbb{N}^n . The overall data flow is independent from a concrete indexing function. Given a linkage rule l , a suitable indexing function assigns a set of indices to each entity from the

¹The working set of a process is the data which must be held in memory at once for efficient execution [Denning, 1968]

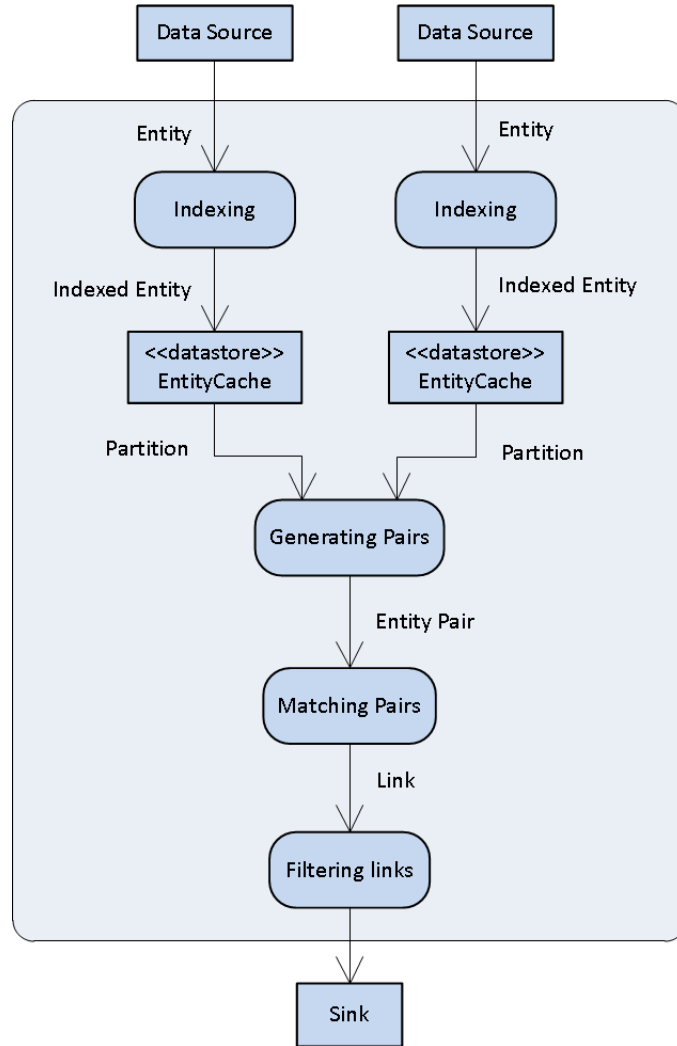


Figure 5.1: Data Flow for executing linkage rules.

input data sets A and B^2 :

$$index_l : A \cup B \rightarrow \mathcal{P}(\mathbb{N}^n)$$

A suitable indexing function further adheres to the property that entities which are matches according to the linkage rule share at least one index:

$$l(e_1, e_2) \geq 0.5 \iff index_l(e_1) \cap index_l(e_2) \neq \emptyset$$

given two entities e_1 and e_2 .

² $\mathcal{P}(\mathbb{N}^n)$ denotes the power set of \mathbb{N}^n , which is the set of all subsets of \mathbb{N}^n

In the context of this work, we propose the MultiBlock approach for indexing. The basic idea of MultiBlock is to generate an index for each entity with the goal of assigning the same index to entities that are potential matches and a different index to entities that are definitive non-matches. Section 5.3 will describe MultiBlock in detail.

5.2.2 Caching

After an entity has been retrieved and indexed it is written to a cache. Based on their index, entities are distributed into blocks. The idea is that entities that share the same index are written to the same block. The number of blocks can be configured and is usually smaller than the number of possible indices. Thus, entities with different indices might end up in the same block. Entities are assigned a number of blocks based on the following function³:

$$block_l(e_1, e_2) = \{flatten(b) \bmod numBlocks | b \in index_l(e_1, e_2)\}$$

Blocks that are bigger than a configured maximum size are further split into partitions. In our experiments, we set the maximum number of blocks to 100 and split blocks into multiple partitions if they exceeded a size of 10,000 entities. These parameters worked well for data sets of different size.

Each pair of partitions from the same block is now sent to next phase in order to generate the comparison pairs. As pair of partitions can be held in memory, the comparison pairs can be generated efficiently as explained in the next section. On a distributed system a pair of partitions can also be send to another machine for matching.

Figure 5.2 shows an example cache.

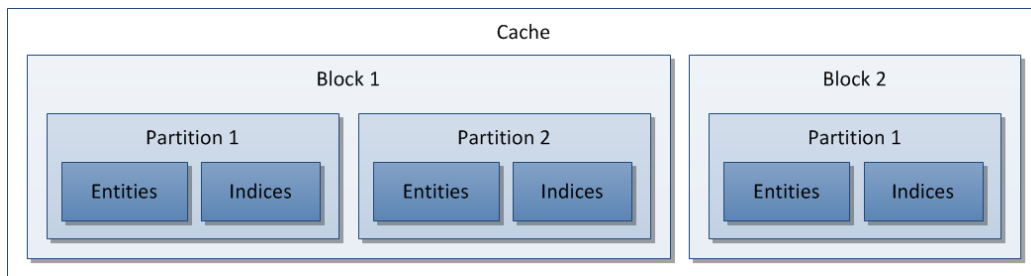


Figure 5.2: Example cache.

³ $flatten : \mathbb{N}^n \rightarrow \mathbb{N}$ flattens an index vector into a scalar value. Every injective function is suitable, that is, every function which preserves the distinctness of the indices.

5.2.3 Generating Comparison Pairs

In order to generate all comparison pairs for which the linkage rule is evaluated, we select all pairs of partitions from the same block. For each of these partition pairs, a comparison pair is generated for each pair of entities which share the same index. More formally, for a pair of partitions P_a and P_b , the comparison pairs are generated according to:

$$\{(e_a, e_b) | i_a = i_b, i_a \in \text{index}_l(e_a), i_b \in \text{index}_l(e_b), e_a \in P_a, e_b \in P_b\}$$

These pairs are then evaluated using the linkage rule to compute the exact similarity and determine the actual links.

Figure 5.3 illustrates how two caches are compared.

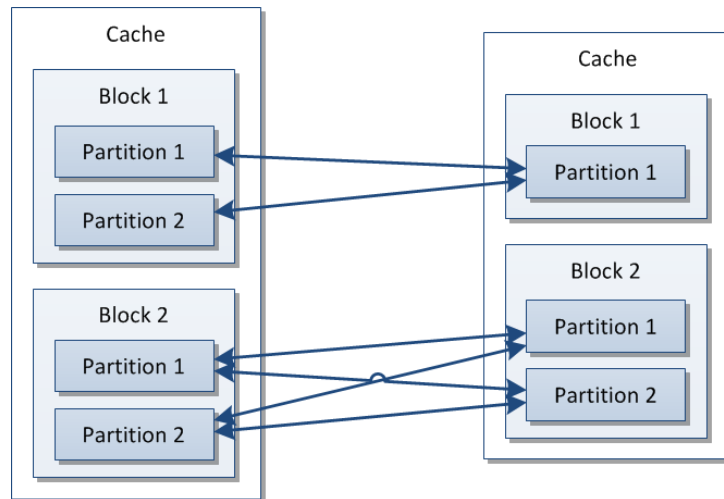


Figure 5.3: Example cache

5.2.4 Matching

The matching phase evaluates the linkage rule for each comparison pair. For each pair of entities for which the similarity according to the linkage rule is above a certain threshold, a link is generated.

5.2.5 Filtering

In many data sources the assumption can be made that there are no duplicates inside a single data source itself, i.e., for each real-world object the data source contains no more than one entity. In that case, generating more than

one links between two data sources that all share the same source entity but target different entities in the other data source means that at least one link is incorrect as the transitive closure of the links would imply that both target entities are referring to the same real-world entity.

In order to handle case like this, a link limit can be supplied. The link limit defines the number of links originating from a single entity. Only the n highest-rated links per source data item will remain after the filtering. If no limit is provided, all links per entity will be returned.

5.3 The MultiBlock Indexing Approach

The basic idea of MultiBlock is to build a multidimensional index for each entity that preserves the distances of the entities, i.e., similar entities are located near each other in the index space. The indexing is built based on the linkage rule and does not need any additional configuration. MultiBlock supports the full expressivity of the proposed linkage rule representation. MultiBlock is organized in two phases:

- (1) In the *index generation* phase, an index is built for each comparison. The basic idea of the indexing method is that it preserves the distances of the entities, i.e., similar entities will be located near each other in the index. The specific indexing method depends on the employed similarity measure. For instance, for each numeric property a one-dimensional index is built and for each property that contains a geographic coordinate a two dimensional index is built.
- (2) In the *index aggregation* phase, all indexes are aggregated into one multidimensional index, preserving the property of the indexing that the indexes of two entities within a given distance share the same index.

We illustrate the indexing by looking at the example of interlinking geographical entities based on their label and geographic coordinates: In this case, the index generation phase would generate 2 indices: A 1-dimensional index of the labels and a 2-dimensional index of the coordinates. The index aggregation phase would then aggregate both indexes into a single 3-dimensional index. Figure 5.4 visualizes the index generation and aggregation in this example. Note that each similarity measure may create multiple index values for a single entity, which for simplicity is not considered in the Figure. Figure 5.5 shows the aggregated index for 1,000 cities in DBpedia.

The next section describes each of the two phases of MultiBlock in detail. The subsequent two sections will then provide an overview of various

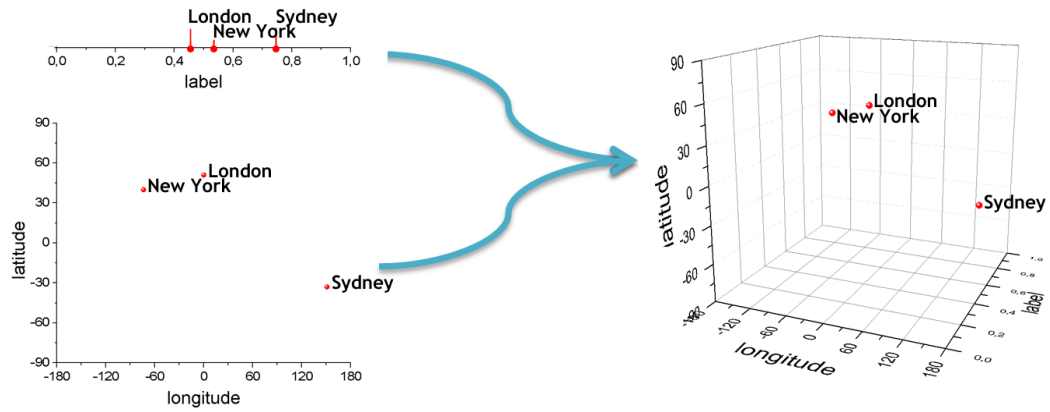


Figure 5.4: Aggregating a geographic and a string similarity

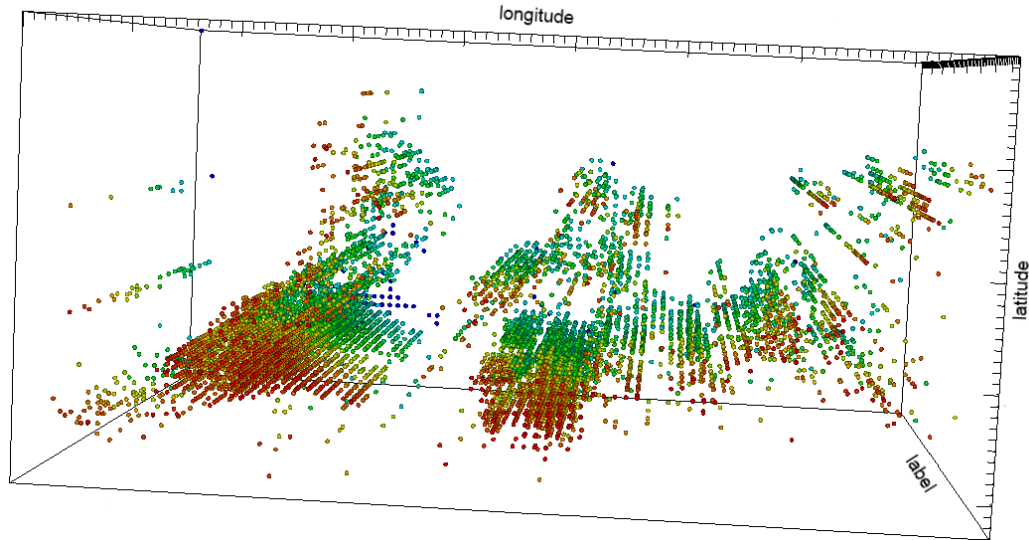


Figure 5.5: Index of 1,000 cities in DBpedia

distance measures and aggregations that can be used in conjunction with our approach.

5.3.1 Data Flow

In this section, we lay down the general framework of the MultiBlock approach, which is independent of a specific similarity measure or aggregation. At the same time, we define which properties a similarity measure or aggregation must adhere to in order to be used in our approach. The subsequent section will specify various similarity measures and aggregations that can be

plugged into the framework in order to provide a complete indexing method.

MultiBlock is organized in two phases: At first, for each distance measure, an index is generated. Afterwards, all distance values are aggregated into a single multidimensional index.

The indexing phases are executed following the components of the linkage rule. Analogous to the computation of the similarity of two entities, the index of a single entity is computed by successively applying the components of the given linkage rule. Table 5.1 summarizes how computing the similarity between two entities compares to computing the index of a single entity.

Step	Phase	
	Compute Similarity	Compute Index
Input	Two entities	A single entity
Retrieve	Retrieve property paths of the given entity(s)	
Transform	Transform values	
Compare	Similarity of two values	Index one value
Aggregate	Aggregate similarity scores	Aggregate indexes

Table 5.1: Steps involved in computing the similarity of two entities compared to computing the index of a single entity.

Index Generation

For each similarity measure in the linkage rule, an index is built, which consists of a set of vectors that define locations in the Euclidean space. The basic idea of the indexing method is that it preserves the distances of the entities, i.e., similar entities will be located near each other in the index. The index generation is not restricted to a specific distance measure.

In order to be used for MultiBlock, for each distance measure must define an additional indexing function. The indexing function has to be defined in addition to the distance function been introduced in Section 2.5.

In total, for each distance measure, two functions have to be defined:

- The distance function:

$$f^d : \Phi \times \Phi \rightarrow \mathbb{R} \quad (5.1)$$

- While the distance function computes the distance between a pair of values, the indexing function generates an index for a single entity:

$$index : \Phi \times [0, 1] \rightarrow \mathcal{P}(\mathbb{N}^n) \quad (5.2)$$

where \mathcal{P} denotes the power set (i.e., it might generate multiple indices for a single entity) and n is the dimension of the index. The first argument denotes the entity to be indexed, which may be either in the source or target set. The second argument denotes the similarity threshold. *index* includes two modifications of the standard *block* function presented in the preliminaries. Firstly, it does not map each entity to a one-dimensional block, but to a multi-dimensional index. This way increases the efficiency as the entities are distributed in multiple dimensions. Secondly, it does not map each entity to a single index, but to multiple indices at once, similar to multi-pass blocking. This avoids losing recall if an entity cannot be mapped to a definite index, such as in some string similarity measures (see Section 5.3.2).

The *index* function must adhere to the property that two entities whose distance according to f^d is below the threshold must share a block. More formally, given two values v_1, v_2 and a threshold θ , f^d and *index* must be related by the following equivalence:

$$f^d(v_1, v_2) \leq \theta \iff \text{index}(v_1) \cap \text{index}(v_2) \neq \emptyset \quad (5.3)$$

Section 5.3.2 gives an overview over the most common distance measures that can be used in MultiBlock.

Index Aggregation

In the index aggregation phase, all indexes that have been built in the index generation phase are aggregated into one compound index. The aggregation function preserves the property of the index that two entities within a given distance share the same index vector. Generally, aggregating the indexes of multiple similarity measures will lead to an increase in dimensionality, but the concrete aggregation function depends on the specific aggregation type. For instance, when aggregating a 2-dimensional geographic index and an 1-dimensional string index using an average aggregation, the resulting index will be 3-dimensional. Section 5.3.3 outlines the concrete aggregation functions for the most common aggregation types.

In order to be used for MultiBlock, each aggregation must define the following functions:

- A function that aggregates multiple similarity values:

$$f^a : \mathbb{R}^n \times \mathbb{N}^n \rightarrow \mathbb{R} \quad (5.4)$$

where n is the number of operators to be aggregated. The first argument denotes the similarity values to be aggregated. As an aggregation may weight the results of the underlying operators, such as the weighted average aggregation, the second argument denotes the weight of the specific operator. This function has already been introduced in Section 2.5.

- A function that aggregates multiple s:

$$\text{aggIndex} : \mathcal{P}(\mathbb{N}^n) \times \mathcal{P}(\mathbb{N}^n) \rightarrow \mathcal{P}(\mathbb{N}^n) \quad (5.5)$$

where \mathcal{P} denotes the power set and n is the dimension of the index. Note that while aggIndex_a only aggregates two sets of indices at once, it can also be used to aggregate multiple sets by calling it repeatedly.

- A function that updates the threshold of the underlying operators in order to retain the condition that two entities within the threshold share an index.

$$\text{modifyThreshold} : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R} \quad (5.6)$$

The first argument denotes the threshold on the aggregation. As in the similarity aggregation function, the second argument denotes the weight of the specific operator.

5.3.2 Indexing Distance Measures

As data sets may make use of a variety of different data types, many distance measures have been proposed to match their values. Section 2.3 provides an overview of the most common distance measures. In this section, we show how various distance measures can be integrated into MultiBlock. For each similarity measure, we specify the distance function and provide the corresponding indexing function.

Character-Based Similarity

A number of character-based similarity measures have been developed in literature (cf. Section 2.3.1). We now show how the Levenshtein distance can be integrated into MultiBlock by adding an indexing function.

Given a finite alphabet Σ and two strings $\sigma_1 \in \Sigma^*$ and $\sigma_2 \in \Sigma^*$, we define the distance function to compute the normalized Levenshtein distance as (cf. Section 2.3):

$$f^d(\sigma_1, \sigma_2) := \frac{\text{levenshtein}(\sigma_1, \sigma_2)}{\max(|\sigma_1|, |\sigma_2|)} \quad (5.7)$$

The basic problem of indexing string values under the presence of typographical errors is the potential loss of recall. We define an indexing function that avoids false dismissals by indexing multiple q -Grams of the given input string. For this purpose, we first define a function that assigns a single index to a given q -Gram $\sigma_q \in \Sigma^q$:

$$\text{index}_q(\sigma_q) := \sum_{i=0}^q |\Sigma|^i \cdot \sigma_q(i) \quad (5.8)$$

index_q assigns each possible letter combination of the q -Gram to a different block.

In order to increase the efficiency, we do not want to index all q -Grams of a given string, but just as many needed to avoid any false dismissals. We can make the following observation [Gravano et al., 2001b]: Given a maximum Levenshtein distance k between two strings, they differ by at most $(k \cdot q + 1)$ q -Grams. Given a threshold θ , we can compute the maximum Levenshtein distance as $k := \max(|str1|, |str2|) \cdot (1.0 - \theta)$. Consequently, the minimal number of q -grams that must be indexed in order to avoid false dismissals is:

$$c(\theta) := \max(|str1|, |str2|) \cdot (1.0 - \theta) \cdot q + 1 \quad (5.9)$$

By combining both functions, we can define the indexing function as:

$$\text{index}(\sigma, \theta) := \{\text{index}_q(\sigma_q); \sigma_q \in \text{qgrams}(\sigma)[0..c(\theta)]\} \quad (5.10)$$

The function starts with decomposing the given string into its q -Grams. From this set, it takes as many q -Grams as needed to avoid false dismissals and assigns an index to each. Finally, it returns the set of all indices.

Token-Based Similarity

We now show how the Jaccard distance can be integrated into MultiBlock by adding an indexing function. Given two token sets A and B , Section 2.3.2 already defined the Jaccard distance as:

$$\text{Jaccard}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \quad (5.11)$$

From the definition of the Jaccard distance, we can see that it is not necessary to index all tokens in order to avoid any false dismissals. Given a token set A and a local distance threshold θ , the number of indices that need to be indexed at the very least is:

$$c(A, \theta) := \lceil |A| * \theta \rceil \quad (5.12)$$

This results in the final indexing function:

$$index(A, \theta) := \bigcup_{i=1}^{c(A, \theta)} \{hashcode(A)\} \quad (5.13)$$

where *hashcode* is a function which returns a non-negative integer for each token.

Numeric Similarity

For numbers in the range $[min, max]$, we can measure the similarity of two numbers using the normalized difference:

$$f^d(d_1, d_2) := \frac{|d_1 - d_2|}{d_{max} - d_{min}} \text{ where } d_1, d_2 \in [d_{min}, d_{max}] \quad (5.14)$$

Using standard blocking without overlapping blocks may lead to false dismissals. For that reason, we use overlapping blocks as proposed by [Draisbach and Naumann, 2009]. The overlapping factor specifies to what extent the blocks overlap. In addition to [Draisbach and Naumann, 2009]. which uses a user-specified overlapping factor, we provide a function which computes the maximum number of indices that does not lead to false dismissals. Using an overlapping factor of 0.5, the maximum number of indices can be computed with:

$$size_s(\theta) := \frac{1}{\theta} \cdot overlap \quad (5.15)$$

Based on the index size we can define the indexing function as follows:

$$index_s(d, \theta) := \begin{cases} \{0\} & \text{if } scaled(d) \leq 0.5 \\ \{size_s(\theta) - 1\} & \text{if } scaled(d) \geq size_s(\theta) - 0.5 \\ \{i(d), i(d) - 1\} & \text{if } scaled(d) - i(d) < overlap \\ \{i(d), i(d) + 1\} & \text{if } scaled(d) - i(d) + 1 < overlap \\ \{i(d)\} & \end{cases} \quad (5.16)$$

$$\text{with } scaled(d) := size_s(\theta) \cdot \frac{d - d_{min}}{d_{max}}$$

$$i(d) := floor\left(\frac{d - d_{min}}{d_{max}}\right)$$

Geographic Distance

Blocking geographic coordinates can be reduced to indexing numbers by using the numeric distance functions on both the latitude and the longitude of the coordinate, which results in 2-dimensional indices.

5.3.3 Indexing Aggregations

In this section, we focus on the most common aggregations: Computing the average similarity and selecting the minimum or maximum similarity value. For each aggregation, we define the specify the similarity aggregation function and define the index aggregation function.

Average Aggregation

The average aggregation computes the weighted arithmetic mean of all provided similarity values.

$$f^a(v, w) := \frac{v_0 * w_0 + v_1 * w_0 + \dots + v_n * w_n}{n} \quad (5.17)$$

Two indexes are combined by concatenating their index vectors:

$$aggIndex(A, B) := \{(a_1, \dots, a_n, b_1, \dots, b_m), a \in A, b \in B\} \quad (5.18)$$

In order to preserve the condition that two entities within the given threshold share an index, the local threshold of the underlying distance measures is modified according to:

$$modifyThreshold(\theta, w) := 1 - (1 - \theta) \frac{1}{w} \quad (5.19)$$

Minimum Aggregation

The minimum aggregation selects the lowest similarity score and thus is equivalent to the logical conjunction of the individual classifications of all operators:

$$f^a(v, w) := \min(v_0, v_1, \dots, v_n) \quad (5.20)$$

Two indexes are combined by concatenating their index vectors:

$$aggIndex(A, B) := \{(a_1, \dots, a_n, b_1, \dots, b_m), a \in A, b \in B\} \quad (5.21)$$

For the minimum aggregation we can leave the threshold unchanged:

$$modifyThreshold(\theta, w) := \theta \quad (5.22)$$

Maximum Aggregation

The maximum aggregation selects the highest similarity score and thus is equivalent to the logical disjunction of the individual classifications of all operators:

$$f^a(v, w) := \max(v_0, v_1, \dots, v_n) \quad (5.23)$$

In this case, we cannot just aggregate the indices to separate dimensions in the same way as in the minimum aggregation. The reason for this is that if one similarity value exceeds the threshold, the remaining similarity values may be arbitrary low while the entities are still considered as matches. For this reason, the index vectors of all indexes are mapped into the same index space:

$$\text{aggIndex}(A, B) := \{(a_1, \dots, a_n), (b_1, \dots, b_n); a \in A, b \in B\} \quad (5.24)$$

In case the dimensionality of the two indexes does not match, the vectors of the lower dimensional index are expanded by setting the values in the additional dimensions to zero.

For the maximum aggregation we can also leave the threshold unchanged:

$$\text{modifyThreshold}(\theta, w) := \theta \quad (5.25)$$

5.4 Previous Work on Indexing

In entity matching [Elmagarmid et al., 2007] a number of indexing methods to improve the efficiency by reducing the number of required comparisons are often applied. In this section, we describe previously proposed indexing methods in detail and conclude with a discussion and comparison with MultiBlock.

5.4.1 Blocking

Traditional *blocking* methods work by partitioning the entities into blocks based on the value of a specific property [Baxter et al., 2003]. In the subsequent comparison phase, only entities from the same block are compared reducing the number of comparisons significantly at the cost of a loss in accuracy. Especially in cases where the data is noisy as it is often the case in Linked Data, similar entities might be assigned to different blocks and thus not compared in the subsequent comparison phase. For instance, in a data set about books, a possible choice for the blocking key is the publisher of

each book. Using this key, entities that describe books with the same publisher would be assigned to the same block. This would reduce the number of required comparisons as entities describing books from different publishers would not need to be compared. On the downside, in cases when the publisher is misspelled, the corresponding entity would not be identified as duplicate of entities for which the publisher has been spelled correctly.

In order to further reduce the number of comparisons, multiple blocking keys can be combined conjunctively into a composite key by concatenating the values of multiple blocking keys. For instance, in a data set about books, the publisher and author of each book could both be combined into a single blocking key by concatenating both values. This reduces the number of comparisons as only entities that describe books that share the same publisher and author will be compared. On the other hand, a misspelling in either property can result in a false dismissal.

Another approach that combines multiple blocking keys is known as *multi-pass blocking* [Hernández and Stolfo, 1998]. Multi-pass blocking aims at reducing the number of false dismissals by combining multiple blocking keys disjunctively. In multi-pass blocking, the blocking is run several times, each time with a different blocking key. Each run assigns a single block to the entity resulting in multiple blocks for each entity. For instance, in a data set about books, the publisher and author of each book could both be used as blocking keys. Using multiple keys guarantees that even if the value of one blocking key is misspelled, matching entities will still be found as long as the values of other keys are spelled correctly. A disadvantage of using multi-pass blocking is that it usually increases the number of comparisons compared to using a single blocking key. The reason for this is that, multi-pass blocking generates multiple blocks for each entity, i.e., one for each blocking key.

Another way of reducing the number of false dismissals is to use phonetic encoding in order to lessen the effects of spelling errors. The idea of phonetic encoding is that characters or syllables that are pronounced similar are encoded to the same value. Phonetic encoding has already been introduced in Section 2.3.4. Their usage for blocking is detailed in Christen [2012].

5.4.2 Sorted Neighborhood

The *sorted neighborhood* method has been proposed in order to improve the handling of fuzzy data [Hernández and Stolfo, 1995]. The sorted neighborhood method works on the list of entities that has been sorted according to a user-defined key. The entities that are selected for comparison are determined by sliding a fixed-size window along the sorted list. Only entities inside the window are selected for comparison. The biggest problem of the

sorted neighborhood Method lies in the choice of the window size. A small size may miss entities if many similar entities share the same key. A big size will lead to a decrease in efficiency. A solution for this is to adapt the windows size while sliding through the list [Yan et al., 2007].

5.4.3 Sorted Blocks

The *sorted blocks* [Draisbach and Naumann, 2009] method generalizes blocking and sorted neighborhood in order to overcome some of their individual disadvantages. It uses overlapping blocks and is both easy to parallelize and more stable in the presence of noise.

5.4.4 Q-Gram Indexing

The idea of *q-gram indexing* is to index different permutations of the q-grams of each value [Baxter et al., 2003]. For each value, a list of q-grams is generated by sliding a window of the size q over the characters in the string [Gravano et al., 2001a]. The value for q must be specified. A typical value for q is two, in which case q-gram indexing is also referred to as *bigram indexing* [Baxter et al., 2003]. For instance, bigram-indexing on the value “clyde” results in the following q-grams list: {'cl', 'ly', 'yd', 'de'}.

Given a q-gram list, all possible sub-lists of different length are generated. In the previous example, the following sub-lists of length three would be generated: {'ly', 'yd', 'de'}, {'cl', 'yd', 'de'}, {'cl', 'ly', 'de'}, {'cl', 'ly', 'yd'}. Given the generated sub-lists, each list is indexed separately. By that, q-gram indexing assigns multiple indices to each string value.

In order to limit the number of sub-lists that are generated for each value, a user-provided threshold determines the minimum length of the sub-lists. Given a threshold t_q and a value that generates k q-grams, the minimum length of the generated sub-lists is computed with [Christen, 2012]:

$$l = \max(1, \lfloor k \cdot t_q \rfloor) \quad (5.26)$$

Only sub-lists up to the determined minimum length are generated.

5.4.5 Canopy Clustering

Canopy clustering [McCallum et al., 2000a] is an indexing method that uses clustering to group similar entities. The basic idea of canopy clustering is the following: Given a linkage rule that is potentially expensive to compute, in order to use canopy clustering an additional cheap distance measure that

can be computed faster than the original linkage rule needs to be defined. By using the cheap distance, all entities are grouped into overlapping clusters. After clustering is done, only entities from the same cluster need to be compared with the linkage rule. A common choice for the cheap distance measure that is used for clustering is the TFIDF (term frequency - inverse document frequency) measure [McCallum et al., 2000a; Baxter et al., 2003].

5.4.6 Metric Embedding Methods

While blocking and sorted neighborhood methods usually map the property keys to a single dimensional index, *metric embedding methods* map the similarity space to a multidimensional space [Hjaltason and Samet, 2003a]. The fundamental idea of the employed mapping is to preserve the distance of the entities, i.e., similar entities are located close to each other in the Euclidean space. Metric embedding methods operate in two steps:

Map: All entities are mapped into a multi-dimensional space by assigning a location in n -dimensional space to each entity, where n is specified in advance.

Join: All pairs of entities which are located close to each other in the mapped space are identified.

More formally, given two sets of entities A and B together with a distance metric $d : A \times B \rightarrow \mathbb{R}^+$, a metric embedding method defines a mapping into a k -dimensional space [Hjaltason and Samet, 2003a]:

$$map : A \cup B \rightarrow \mathbb{R}^k \quad (5.27)$$

The essential property of the embedding space is that an efficient distance function $\delta : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is defined. The reasoning behind the mapping function is that the distance between each pair of entities $a \in A$ and $b \in B$ is preserved through the mapping:

$$d(a, b) \approx \delta(map(a), map(b)) \quad (5.28)$$

The amount by which a metric embedding approach diverts from the ideal case in which the distances in the original space and the mapped space are identical for all pairs of entities, can be measured by its distortion. The distortion of a metric embedding approach is specified by two constant factors $c_1 \geq 1$ and $c_2 \geq 1$ [Hjaltason and Samet, 2003a]:

$$\frac{1}{c_1} \cdot d(a, b) \leq \delta(map(a), map(b)) \leq c_2 \cdot d(a, b) \quad (5.29)$$

A metric embedding approach for which c_2 has an upper bound of 1 is called contractive [Hjaltason and Samet, 2000] and thereby guarantees that:

$$\delta(\text{map}(a), \text{map}(b)) \leq d(a, b) \quad (5.30)$$

Thus, a contractive embedding approach guarantees that for any given pair of entities the distance in the mapped space is equal or smaller than the original distance between both entities.

Estimating an upper bound for c_2 is essential in order to avoid false dismissals. Given an upper bound c_2 and a distance threshold θ , the comparison pairs in the mapped space that are located within the threshold can be identified with:

$$d(a, b) \leq \theta \Rightarrow \delta(\text{map}(a), \text{map}(b)) \leq c_2 \cdot \theta \quad (5.31)$$

Thus, given a threshold θ , the comparison pairs can be identified by searching for all entities in the mapped space that are located within a distance of $c_2 \cdot \theta$.

Metric embedding methods that have been proposed for entity matching include *FastMap* [Faloutsos and Lin, 1995], *MetricMap* [Wang et al., 1999], *SparseMap* [Gabriela and Martin, 1999], and *StringMap* [Jin et al., 2003]. Unfortunately, in general, these methods do not guarantee that no false dismissals will occur, with the exception of SparseMap for which a variant has been proposed by Hjaltason and Samet [2003a] that guarantees that no false dismissals can occur. All of these approaches require the similarity space to form a metric space, i.e., the similarity measure must respect the triangle inequality [Hjaltason and Samet, 2003a]. This implies that they cannot be used with non-metric similarity measures, such as Jaro-Winkler [Winkler, 1990]. A detailed comparison of FastMap, MetricMap and SparseMap can be found in [Hjaltason and Samet, 2003a].

Another recent approach that uses the characteristics of metric spaces, in particular the triangle inequality, to reduce the number of similarity computations, has been implemented in LIMES [Ngonga Ngomo and Auer, 2011].

5.4.7 StringMap

We now describe the general algorithm of a metric embedding method in more detail on the example of StringMap. The StringMap algorithm is organized in three phases:

Map: An n -dimensional space is iteratively constructed from the entities. For each coordinate axis, StringMap chooses two pivot points by using a heuristic to find a pair of entities that are as far apart as possible. The position of all other entities along the axis is determined by projecting

each entity onto the axis based on its distance to the pivot points of that axis. Figure 5.6 illustrates the projection of an entity e onto an coordinate axis that is spanned by two pivot points p_1 and p_2 . This

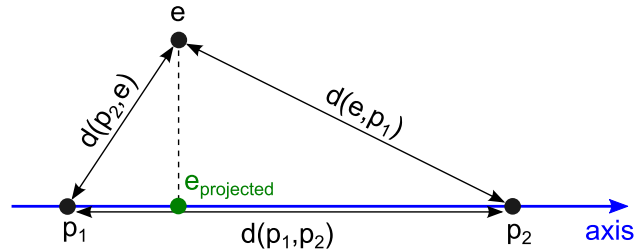


Figure 5.6: Projection of an entity onto an axis in StringMap.

process is repeated until all n coordinate axis have been build. The overall runtime of the mapping phase scales linear with the number of entities. Jin et al. [2003] show that StringMap achieves its optimal performance if n is between 15 and 25.

Determine Threshold: StringMap, as a variant of FastMap, is not contractive [Hjaltason and Samet, 2000]. For this reason, the distance between a pair of entities in the mapped space may be bigger than the original distance between both entities. StringMap proposes a heuristic to estimate the threshold in the mapped space that corresponds to the original threshold. The proposed heuristic selects a random subset of the entire set of entities and finds all pairs of entities in the chosen subset that are within the original threshold. The threshold in the mapped space is then set to the maximum distance between the found entities. This heuristic does not guarantee that an upper bound is found for the mapped threshold that covers all entities that are within the original threshold.

Join: Given the mapped space together with the determined threshold, the target is to find all pairs of mapped entities that are within the threshold. In literature, a number of data structures have been proposed that allow an efficient search for pairs of entities that are located within a specified distance [Hjaltason and Samet, 2003b]. StringMap constructs an R-tree for that purpose [Guttman, 1984]. All pairs of entities which are located close to each other in the mapped space are identified by querying the R-tree.

StringMap has been evaluated on three data sets:

- (1) A set of 54,000 actor names from the IMDB movie database⁴.
- (2) A set of 133,101 person names from the *Die Vorfahren* Database⁵.
- (3) A set of 20,000 publications from DBLP⁶.

On the evaluation data sets, StringMap identified close to 100% of all duplicates while outperforming a naïve method that compared all possible pairs.

5.4.8 Discussion

In the following we will introduce previous studies that compare the performance of different indexing methods. After that, we will compare properties of previous indexing methods with MultiBlock.

Performance

Draisbach and Naumann [2009] compare the performance of blocking, sorted neighbourhood and sorted blocks on a set of 9,763 entities describing audio records that have been extracted from freeDB⁷. On the evaluation data set, sorted neighbourhood and sorted blocks generated fewer false dismissals than blocking. In addition, sorted blocks generated slightly fewer comparison pairs than sorted neighbourhood and significantly fewer pairs than blocking.

Baxter et al. [2003] compare the performance of four indexing methods: blocking, sorted neighbourhood, q-gram indexing and canopy clustering. All methods have been compared on a synthetic data set that has been generated using the DBGen Tool [Askarunisa A. et al., 2009]. On the evaluation data set, q-gram indexing and canopy clustering outperformed blocking and sorted neighbourhood. The evaluation also showed that the performance of all indexing methods strongly depends on the chosen parameters. For instance, the performance of blocking varies with the number of characters that are extracted from each value in order to determine the block.

A more comprehensive comparison of different indexing methods is provided by Christen [2011]. Six different indexing methods are compared while each indexing method is evaluated with different parameters: Blocking, sorted neighbourhood, q-gram indexing, canopy clustering, StringMap and suffix-array based indexing. All indexing methods are evaluated on four

⁴<http://www.imdb.com/>

⁵The link provided by Jin et al. [2003] (<http://feefhs.org/dpl/dv/indexdv.html>) is no longer active at time of writing.

⁶<http://www.informatik.uni-trier.de/~ley/db/>

⁷<http://www.freedb.org/>

real data sets and two synthetic data sets. In terms of runtime performance, blocking and sorted neighbourhood outperformed all other approaches on average. On the opposite side, StringMap and q-gram indexing achieved the worst runtime performance and required longer to execute than the other methods. In terms of recall, the performance of all indexing methods varied with the data set as well as the used parameters and thus no clear winner could be shown. Blocking stands out as it achieved the highest F-measure on two of the four real data sets.

An experimental comparison of the performance of MultiBlock and existing indexing methods will be presented in Section 5.6.

Properties

In the following, we compare properties of previous indexing methods with MultiBlock. We compare three properties:

Lossless: The indexing method guarantees that no false dismissals occur.

Non-Metrics: The indexing method can be used with non-metric distance measures.

Local: Each entity is indexed independently and thus no global processing of the entire data set is required. Local methods can be parallelized in a straightforward way.

Table 5.2 summarizes how MultiBlock compares to existing indexing methods. The main advantage of MultiBlock over the majority of previous approaches is that, while significantly reducing the number of comparisons, MultiBlock guarantees that no false dismissals and thus no loss of recall can occur and does not require the similarity space to form a metric space. In addition, it uses a multidimensional index increasing its efficiency significantly. Another advantage of MultiBlock is that it can work on a stream of entities as it does not require to preprocess the whole data set.

5.5 Distributed Execution of Linkage Rules

In this section, we propose a data flow for executing linkage rules on a distributed architecture.

5.5.1 Cluster Programming Models

Before we describe the proposed data flow, we discuss popular approaches for developing distributed systems [Begoli, 2012].

Method	Property		
	Lossless	Non-Metrics	Local
Blocking	no	yes	yes
Sorted-Neighborhood	no	yes	no
Sorted Blocks	no	yes	no
Q-grams	no	yes	yes
Canopy Clustering	no	yes	no
FastMap	no	no	no
MetricMap	no	no	no
SparseMap	no	no	no
StringMap	no	no	no
Modified SparseMap	yes	no	no
MultiBlock	yes	yes	yes

Table 5.2: Comparison of different blocking methods

Message Passing

In message-passing systems the processes communicate by exchanging messages. Each process that runs on the distributed system may send a message to another process as well as receive messages from other processes. Message-passing systems are typically *shared nothing systems*, i.e., conceptually the processes don't share memory but exchange data with messages.

A well-known message-passing system is the Message Passing Interface (MPI) [Walker and Dongarra, 1996; Snir et al., 1995]. MPI is a standard that specifies the syntax and semantics of the interfaces for which multiple implementations for different programming languages have been developed.

A related message-passing model for concurrent programming is the actor model. In systems using the actor model, each actor encapsulates a specific behaviour, which decides how that actor reacts to incoming messages. After receiving a message, an actor modifies its behavior for future messages, which allows state to be represented. Each actor may also create additional actors. If an actor creates another actor, it is automatically the supervisor of this actor. This means that if the actor offloads work to actors it created, it is responsible for handling failures in these actors. An actor system usually starts with a single root actor that creates new actors, which in turn may create other actors. Thereby, actors are organized in a supervision hierarchy.

A couple of programming languages have been developed that are based on the actor model. An example for this is the Erlang programming lan-

guage⁸ [Armstrong, 2007], which has been developed in 1986. For programming languages that do not support the actor model natively, such as Java, a number of libraries have been developed. An example is Scala⁹, which includes support for the actor model in the standard library. Many of these systems also allow actors to be distributed across different machines.

Besides MPI and the actor model, many other systems for message-passing have been developed: The Common Object Request Broker Architecture (CORBA) [Object Management Group, 2012] is a standard by the Object Management Group¹⁰ for inter-system communication. Java RMI [Downing, 1998] is an interface for remote method calls on the Java Platform and is interoperable with CORBA. The Simple Object Access Protocol (SOAP) [Gudgin et al., 2007] is a W3C recommendation for exchanging structured messages based on existing standards, such as XML and HTTP.

MapReduce

MapReduce [Dean and Ghemawat, 2004] is a data flow model for processing large data sets on a distributed cluster of machines. A popular framework that supports the MapReduce paradigm is Apache Hadoop¹¹. In addition to MapReduce, Hadoop also provides a complete framework for distributed computing. Although, at the time of writing, Hadoop is the most widely used framework for MapReduce, we will describe the MapReduce paradigm in general instead of focusing on a specific MapReduce framework.

The data flow of a single MapReduce task consists of 5 steps:

Input Reader: The input reader reads the input files and splits the data into logical segments. Each segment is sent to a separate map task for processing.

Map: A map task receives a data segment from the input reader. The data segment is transformed by a user-provided mapping function. For each data segment, the mapping function returns a set of key-values pairs, which are forwarded to the next phase.

Shuffle and Sort: All key-values pairs are grouped by key. For each unique key, all values with the given key are transferred to a separate reduce task. In contrast to the map and reduce phase, the shuffle and sort phase is usually fixed and not user-defined.

⁸<http://www.erlang.org/>

⁹<http://www.scala-lang.org/>

¹⁰<http://www.omg.org/>

¹¹<http://hadoop.apache.org/>

Reduce: A reduce task receives a key and a list of all values with the given key. Based on that, a user-provided reduce function computes the output.

Output Writer: Finally, the output is written to the file system. As each reducer may be running on a separate machine, typically every reducer generates a separate file on a distributed file system.

Note that a few simplifications have been made for illustration. For many real-world problems, a single MapReduce task is not enough. For these cases, multiple MapReduce tasks can be chained.

Resilient Distributed Data Sets

A recently introduced model for distributed computing are Resilient Distributed Data Sets (RDDs) [Zaharia et al., 2012]. The basic idea of RDDs is to allow to derive new data sets from existing data sets by applying a chain of transformations. These transformations are coarse-grained, i.e., they are applied to an entire data set instead of updating individual entities in a data set. Examples of such transformations are the map and reduce operations, which are already known from MapReduce. However, in contrast to MapReduce, RDDs are not limited to these two operations and may also define other transformations, such as join or count operations. A derived data set can either be recomputed on-the-fly based on the original data set from which it is derived or it can be persisted on stable storage. RDDs that are recomputed on-the-fly are very similar to views in databases, while persistent RDDs are comparable to materialized views.

RDDs can be used to efficiently express other cluster programming models, such as Hadoop [Zaharia et al., 2012].

5.5.2 MapReduce Data Flow

In this section, we propose a data flow to execute a linkage rule based on the MapReduce paradigm. We chose MapReduce over message-passing systems, because MapReduce is simpler to apply as it provides a pre-defined data flow and automatically distributes processing to multiple machines without requiring manual specification of the communication between the machines. We did not use RDDs, because RDDs are a very recent model, which, so far, is not widely used in the software industry, while MapReduce is a well-established model.

The overall data flow consists of three MapReduce tasks: One indexing task for each of the two data sets and a match task. The purpose of the

indexing tasks is to generate the MultiBlock index for each entity and to build partitions of entities that share a similar index. The purpose of the match task is to match pairs of partitions and to generate a link for each matching pairs of entities. Figure 5.7 shows an overview of the overall data flow.

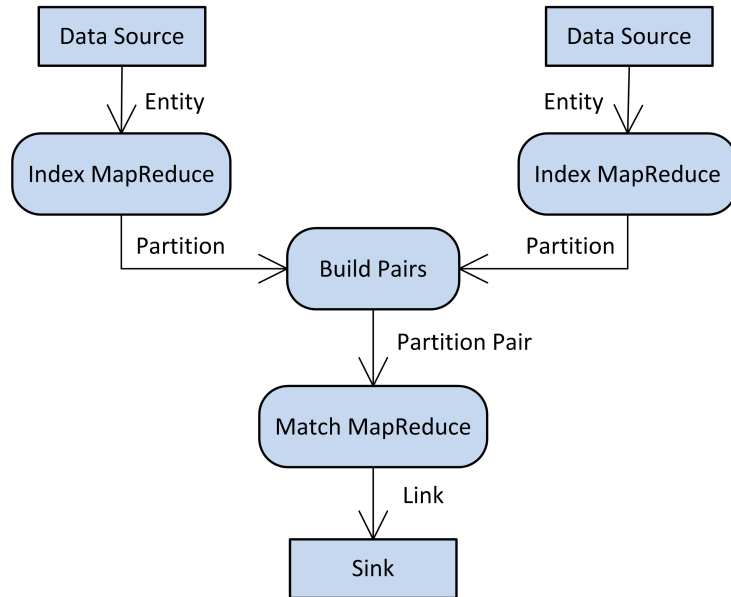


Figure 5.7: Overall MapReduce Data Flow

The subsequent sections describe the indexing and matching tasks in more detail.

Indexing

Figure 5.8 shows the steps involved in the indexing phase.

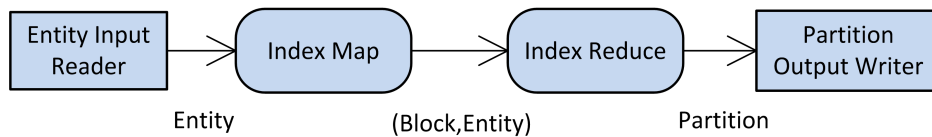


Figure 5.8: Indexing MapReduce Task

The **Entity Input Reader** reads all entities in a specific data set. A map task is created for each entity.

The **Index Map** function computes the MultiBlock index for each entity. After that, for each index it computes a block number. This is done by

flatten each multidimensional index into a scalar value. Listing 5.1 shows the pseudocode for the Index Map function. Based on the output of the

Listing 5.1: Index map function

```

1 function map(entity) {
2   //Compute a set of index vectors
3   entity.index ← index(entity)
4   //Compute a set of scalar indices
5   flatIndex ← flatten(entity.index)
6   //Compute output key–value pairs
7   output ← ∅
8   for(i in flatIndex) {
9     block ← i mod numBlocks
10    Add (block, entity) to output
11  }
12  return output
13 }
```

Index Map function, the MapReduce framework groups all entities by their block.

The subsequent **Index Reduce** function receives all entities that share the same block. All entities that share one block are segmented into partitions of a specified maximum size. Listing 5.2 shows the pseudocode for the index reduce function.

Finally, the **Partition Output Writer** writes the generated partitions to the output.

Matching

Figure 5.9 shows the steps involved in the matching phase.

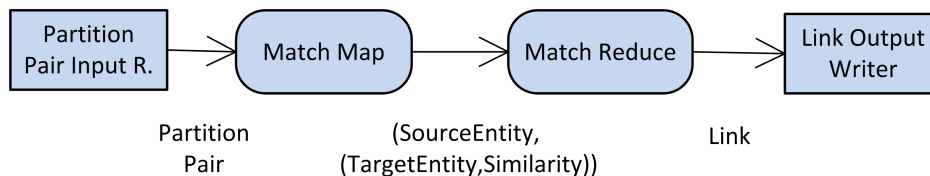


Figure 5.9: Matching MapReduce Task

The **Partition Pair Input Reader** iterates through all blocks and for each block builds the Cartesian product of all pairs of partitions from this block. A map task is created for each pair of partitions.

Listing 5.2: Index reduce function

```

1 function reduce(block, entities) {
2   output ← ∅
3   partition ← ∅
4   for(entity in entities) {
5     if(size(partition) < maxPartitionSize) {
6       Add entity to partition
7     }
8     else {
9       Add partition to output
10      partition ← ∅
11    }
12  }
13  return output
14 }

```

The **Match Map** function reads the supplied pair of partitions and iterates through all pairs of entities that share an index. For each pair of entities that share an index, it evaluates the linkage rule. Listing 5.1 shows the pseudocode for the Match Map function. The result of the Match Map function

Listing 5.3: Match map function

```

1 function map(sourcePartition, targetPartition) {
2   Get pairs of entities (sourceEntity, targetEntity) from sourcePartition
3   ↔ and targetPartition that share an index
4   similarity ← evaluate rule for (sourceEntity, targetEntity)
5   return (sourceEntity, (targetEntity, confidence))
6 }

```

is a set of key-value pairs. The keys are represented by the source entities while the values are represented by the target entities and the similarity of both entities according to the linkage rule.

The **Match Reduce** function receives all target entities that share a specific source entity. For each source entity it yields a user-specified maximum number of links. For this it selects the top-k links with the highest similarity.

Listing 5.4 shows the pseudocode for the Match Reduce function.

Finally, the **Link Output Writer** writes all generated links to an output.

Listing 5.4: Match reduce function

```
1 function reduce(sourceEntity, targetEntitiesWithConfidences) {  
2   return up to limit entities from targetEntitiesWithConfidences with the  
   ↪ highest confidence  
3 }
```

5.6 Evaluation and Discussion

MultiBlock has been evaluated regarding scalability and effectiveness. This section reports on the results of four experiments in which we used Silk with MultiBlock to generate links between different data sets.

5.6.1 Experiment Setup

In each experiment, we computed four measurements that are usually used to compare the experimental performance of different indexing methods [Christen, 2011, 2012; Baxter et al., 2003]:

Number of Comparisons: The number of pair comparisons that are generated by a given indexing method. The quality of the generated pairs is not assessed, i.e., the number of comparisons also includes pairs of entities that are non-matches. The *reduction ratio* of an indexing method can be computed by dividing the size of the Cartesian product of all entities by the number of comparisons that have been generated. The number of comparisons provides an indication for the runtime of the subsequent matching phase in which the linkage rule is evaluated for each comparison pair.

Pair Completeness: The pair completeness is the number of matching pairs of entities for which a comparison pair has been generated divided by the number of all matching pairs including the ones not found by the indexing method. An ideal pair completeness of 100% means that a comparison pair has been generated for all matching pairs and thus no loss of recall due to false dismissals will occur. Pair completeness does not consider the quality of the generated comparison pairs. For instance, generating the full Cartesian product achieves a pair completeness of 100% as all matching pairs of entities are also included in the Cartesian product.

Pair Quality: The pair quality is the number of comparison pairs that represent matching entities divided by the total number of generated comparisons. The pair quality is related to the reduction ratio. While the reduction ratio measures the reduction in number comparisons without considering the correctness of the generated comparisons, the pair quality only measures the reduction of comparison pairs which correspond to matching entities.

Pair completeness is a measure of completeness, while pair quality is a measure of correctness. In that sense, pair completeness and pair quality correspond to recall and precision as discussed in Section 3.5.3.

As the experiments have been carried out on different machines, the specification of the machines that have been used will be stated in each experiment separately.

5.6.2 Comparison with Other Methods

In this experiment, we compared MultiBlock to other common indexing methods. We used the person names data set that is included in the reference implementation of StringMap [Li and Jin, 2010]. The reference implementation of StringMap is part of the FLAMINGO package for approximate string matching [Behm et al., 2010]. The evaluation data set consists of two sets of entities, each containing 2001 person names. No other properties, apart from the names, are provided. Experiments with data sets that are matched based on multiple properties for each entity are regarded later. This experiment has been executed on a virtual machine that has been assigned two cores of a four core Intel Xeon E5-2609 CPU and 8GB of RAM.

The used linkage rule compares the names of persons in both data sets. Two person names are considered matching if their Levenshtein distance is two or less. We compared the performance of six different configurations. A detailed description of the compared indexing methods is available in Section 5.4.

Blocking: Standard blocking on person names. As person names in both data sets contain spelling errors, we used phonetic encoding to normalize different spellings based on their pronunciation. We compared three different phonetic encoding algorithms: Soundex [Russell, 1918, 1922], the New York State Identification and Intelligence System (NYSIIS) encoding [Taft, 1970] and Metaphone [Philips, 1990].

Sorted Blocks: For the Sorted Blocks method, all person names have been converted to lower case and assigned a block based on their alphanu-

meric order. Two different overlapping factors have been evaluated: 10% and 50%.

Q-Grams: All person names have been indexing using bigram indexing. As bigram indexing recombines the bigrams of each value into sub-lists, the minimum length of the generated bigrams needs to be specified by setting the minimum length threshold t_q . Three different values for the threshold t_q have been tested: 0.8, 0.9, and 0.95.

StringMap: The StringMap metric embedding algorithm. As explained in detail in Section 5.4.7, StringMap requires a separate step between the mapping and the join phase that is used to determine the threshold in the mapped space that corresponds to the original threshold. The reference implementation determines the threshold by sampling a user defined percentage of the data sets and estimating the threshold based on the mapped distances of the sampled entity pairs. The reference implementation uses a default sample rate 10% if no percentage is specified by the user. However, the reference implementation runs all tests on the included data sets using a sample rate of 50%. We evaluated StringMap with two different sample rates: 10% and 50%. The dimensionality of the mapped space was set to 20 dimensions, which is the same value as has been used in the reference implementation.

MultiBlock: The MultiBlock indexing method that is proposed in this thesis.

Full: Baseline with no indexing method. The linkage rule is evaluated for the full Cartesian product of both data sets.

All evaluated indexing methods have been implemented in the Silk Link Discovery Framework. For StringMap, we used the reference implementation in C++ [Li and Jin, 2010], which is made available by some of the original authors of StringMap in FLAMINGO [Behm et al., 2010].

Table 5.3 lists the evaluation results. Although the *Blocking* configurations achieved the lowest execution times, they also generated the highest number of false dismissals. Soundex is the only phonetic encoding to achieve a pair completeness of over 50%, i.e., NYISS and Metaphone identified less than 50% of all duplicates. On the upside, about 95% of all comparison pairs that have been generated by NYISS and Metaphone denote actual duplicates. *Sorted Blocks* achieves a pair completeness of almost 95% and reduces the number of comparisons by a factor of about 10.

As also confirmed by similar studies [Christen, 2011], *Q-grams* indexing achieves a poor performance for low values of the minimum length threshold.

Method	Comparisons	Completeness	Quality	Runtime
Blocking (Soundex)	5,766	58.07%	4.49%	0.3s
Blocking (NYSIIS)	97	20.85%	95.88%	0.3s
Blocking (Metaphone)	123	26.01%	94.31%	0.3s
Sorted Blocks (10%)	317,339	94.39%	0.13%	0.9s
Sorted Blocks (50%)	446,013	94.62%	0.09%	1.1s
Q-Grams (0.8)	664	77.58%	52.11%	70.3s
Q-Grams (0.9)	225	47.76%	94.67%	2.8s
Q-Grams (0.95)	65	14.57%	100%	1.2s
StringMap (0.1)	335	61.88%	82.39%	3.2s
StringMap (0.5)	34,298	98.78%	1.27%	4.2s
MultiBlock	438,784	100%	0.10%	0.7s
Full	4,004,001	100%	0.01%	4.6s

Table 5.3: Comparison of different indexing methods.

The reason for this is that the number of generated sub-lists explodes for values that consist of many bigrams [Christen, 2012]. Because of that, q-grams indexing only outperformed the full method in runtime for thresholds above 0.8. However, both configurations that outperform the full method achieve a pair completeness below 50%, i.e., dismiss more than every second true match. The reason for the poor pair completeness can very likely be found in the rather noisy data set, which contains many pairs of matching entities that differ by an edit distance of two.

With a pair completeness of 98.75%, StringMap achieves roughly the same recall as has been reported by the original authors for this data set [Li and Jin, 2010]. Also *StringMap* generated 10 times fewer comparison pairs than MultiBlock. On the downside, StringMap is the slowest indexing method that has been compared and almost needs as much time as the full comparison. In the next experiment, we will analyze the performance of StringMap in more detail.

MultiBlock outperformed all indexing methods except Blocking, but still achieved a pair completeness of 100%. MultiBlock reduced the number of comparison by a factor of about 10 compared to the full evaluation.

5.6.3 Scalability

In this experiment we evaluated, whether MultiBlock is able to scale to large data sets. For this purpose, it is essential that MultiBlock reduces the number of comparisons drastically without dismissing correct pairs. We evaluated the scalability of MultiBlock by applying it to interlink two geographic data sets. We used a data set consisting of 182,660 settlements from DBpedia¹² and 560,122 settlements from LinkedGeoData¹³. LinkedGeoData collects information from the OpenStreetMap project and makes it available as Linked Data. This experiment has been executed on a virtual machine that has been assigned two cores of a four core Intel Xeon E5-2609 CPU and 8GB of RAM.

We employed a linkage rule that compares settlements by their labels as well as by their geographic coordinates. Figure 5.10 depicts the used linkage rule.

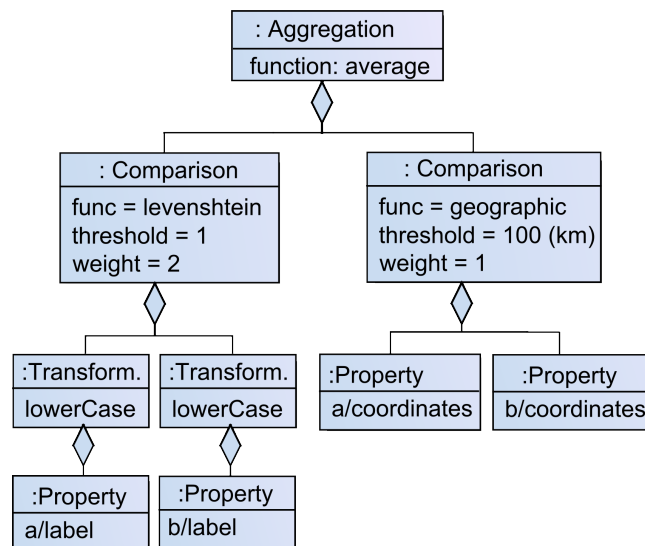


Figure 5.10: Scalability experiment: Linkage rule for linking settlements

First, we interlinked both data sets by evaluating the complete Cartesian product without the use of any blocking method. As this results in over 100 billion comparisons, it is a clear case when matching the complete Cartesian product is not reasonable anymore. The link generation took over 5 days hours and generated 104,365 links. The generated links have been spot-checked for correctness. Two configurations have been compared to the full method:

¹²<http://dbpedia.org>

¹³<http://linkedgeo.org>

Method	Comparisons	Completeness	Quality	Runtime
StringMap (1%)	610,495	99.68%	16.79%	30,731s
StringMap (10%)	610,324	99.97%	16.75%	41,403s
StringMap (50%)	610,494	100%	16.74%	191,308s
MultiBlock	4,023,290	100%	2.59%	595.7s
Full	102,311,884,520	100%	0.0001%	474,023s

Table 5.4: Results of scalability experiment

Method	Map	Determine Threshold	Join
StringMap (1%)	5,792s	62s	24,877s
StringMap (10%)	5,778s	5,936s	29,689s
StringMap (50%)	5,781s	147,603s	37,924s

Table 5.5: Different phases of StringMap: Runtimes.

StringMap: The StringMap metric embedding algorithm. As this data set is considerably larger than the previous data set, the default sampling rate of 10% for determining the threshold needs to sample $18266 \cdot 56012$ pairs. For this reason, we evaluated a rate of 1% in addition. Although Jin et al. [2003] propose a method how StringMap can be extended to include multiple properties, the reference implementation that has been used in our the experiments only considers the name property for generating the comparison pairs.

MultiBlock: The MultiBlock indexing method.

Table 5.4 summarizes the evaluation results. The evaluation results show that MultiBlock and StringMap with a sampling rate of 50% achieved the full recall.

The most apparent difference between MultiBlock and StringMap is the big difference in runtime. In order to identify the reason why StringMap performs much worse than MultiBlock, we analyzed the runtimes of the different phases: A detailed description of each phase has been provided in Section 5.4.7. The join time combines the join time of the StringMap algorithm and the time needed to evaluate the linkage rule for all found comparison pairs.

The runtime of the mapping phase depends on the number of dimensions as well as the number of entities and is independent of the sampling rate used to determine the threshold. When the sampling rate is set to 50%, $91330 \cdot 280061$ pairs of entities need to be sampled and thus one-fourth of

Phase	Time
Build index	14 %
Generate comparison pairs	41 %
Similarity comparison	45 %

Table 5.6: The runtimes of the different phases

the entire data set. The required time of 147,603s constitutes as expected about one-fourth of the runtime of the full method 474,023s. Choosing a lower sampling rate significantly reduces the time needed to determine the threshold. The join time only changes slightly as the number of generated comparisons did not change considerable.

An apparent question is whether the runtime of StringMap could be improved by extending it to use both the labels and the geographic coordinates for mapping. In this case, the number of comparisons could be reduced by a factor of about six at best, considering that StringMap generates about six times more comparison pairs than necessary. However, this would not affect the mapping time positively as it depends on the number of dimensions as well as the number of entities. For this reason, StringMap would still be significantly slower than MultiBlock.

On the other hand, MultiBlock was able to reduce the number of comparisons to 4,023,290, by indexing in three dimensions. It generated the identical 104,365 links as in the full evaluation, but ran in only 596 seconds. MultiBlock reduced the number of comparisons by a factor of over 25,000 and is almost 800 times faster than evaluating the complete Cartesian product.

In order to determine which phase of MultiBlock is responsible for the complete runtime, we evaluated the runtimes of the different phases of the matching process. As we can see in Table 5.6, a big part is spent for creating the comparison pairs. For this reason future improvements will focus on using a more efficient algorithm for the comparison pair generation.

5.6.4 Effectiveness

In order to be applicable to discover links between arbitrary data sources, MultiBlock must be flexible enough to be applied even to complex data sets without losing recall. For this purpose we employed Silk with MultiBlock to interlink drugs in DBpedia and DrugBank. DrugBank is a large repository of almost 5,000 FDA-approved drugs and has been published as Linked Data on <http://wifo5-03.informatik.uni-mannheim.de/drugbank/>. Here it is not sufficient to compare the drug names alone, but also necessary to take the

Method	Comparisons	Completeness	Quality	Runtime
MultiBlock	122,630	100%	1.14%	6s
Full	22,242,292	100%	0.0063%	430s

Table 5.7: Results of experiment 2

various unique bio-chemical identifiers, e.g., CAS number, into consideration. Therefore, the corresponding linkage rule compares the drug names and their synonyms as well as a list of well-known and used identifiers of which not all have to be set on the entity. A slightly simplified version of this rule, as already used for the evaluation of the GenLink algorithm, is shown in Figure 3.20 on page 93. This experiment has been run on a 3GHz Intel(R) Core i7 CPU with four cores and 8GB of RAM.

The employed linkage rule results in 1,403 links. Table 5.7 shows the results using MultiBlock compared to the full evaluation. Using Silk with MultiBlock, we achieved a speedup factor of 71 with full recall. The gain in this example is smaller than in the previous one, because the data set here is much smaller and the linkage rule is more complicated.

5.6.5 MapReduce

The MapReduce data flow for executing linkage rules on a distributed architecture as proposed in Section 5.5 has been implemented in Silk. In the following, we present evaluation results that have been achieved through running Silk as part of the Linked Data Integration Framework (LDIF) [Schultz et al., 2011, 2012b]. LDIF supports a complete data integration data flow consisting of a vocabulary translation component that translates different data sets into a consistent schema, an identity resolution component that uses the Silk Link Discovery Framework in order to identify duplicates and modules for quality assessment and data fusion. All presented evaluation results are from [Schultz et al., 2012a].

The evaluation has been carried out by integrating two life science data sources: KEGG GENES [Ogata et al., 1999; Kanehisa et al., 2010], a set of sequenced genomes and Uniprot [Bairoch et al., 2005], a set of protein products derived from genes. After both data sources have been translated to a single schema, an entity matching tasks was carried out that identifies pairs of entities that describe that same gene. In order to identify genes, a number of identifiers, such as the KEGG identifiers [Ogata et al., 1999], are commonly used. For each pair of entities, the employed linkage rule compared five different identifiers for equality. Two entities are considered

Cluster Size	Loading Time	Matching Time
1 master, 8 slaves	646s	1546s
1 master, 16 slaves	421s	932s
1 master, 32 slaves	324s	580s

Table 5.8: MapReduce runtimes for different cluster sizes.

to relate to the same gene if one of the compared identifiers matches. For the experiments, a sample of 300,000,000 RDF statements from both data sources was used, from which the linkage rule accessed 32,211,677 RDF statements.

All experiments have been executed on Amazon Elastic MapReduce¹⁴. The experiments have been run on Amazon EC2 *c1.medium instances*. Each *c1.medium* instance provides five EC2 compute units, while the performance of each compute unit is comparable to an 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor¹⁵. Each instance offers 1.7 GB of RAM.

Table 5.8 compares the runtimes of the entity matching task for different number of machines. The purpose of the master machine is limited to distributing the map and reduce jobs and to managing the distributed file system. The map and reduce jobs are executed exclusively on the slave machines. Two runtimes are shown in the table: The loading time indicates the time that LDIF needed to join all statements with the same subject into a single entity. The matching time indicates the time needed by Silk to evaluate the linkage rule according to the distributed data flow that has been described in Section 5.5.

When using 16 instead of 8 machines, the matching time is reduced to 60%. By using 32 machines, the matching time is reduced to 62% compared to 16 machines, and to 37% of the runtime with 8 machines. In each case, the entity matching task generated 1,084,808 links.

5.7 Summary

In this chapter, we covered methods to efficiently execute linkage rules. We observed that efforts to increase the efficiency of executing a linkage rule are usually aimed at reducing the number of linkage rule evaluations by dismissing definitive non-matches prior to comparison. In the course of that, we reviewed existing indexing methods, which have been proposed to reduce the number of required entity comparisons.

¹⁴<http://aws.amazon.com/elasticmapreduce/>

¹⁵<http://aws.amazon.com/ec2/instance-types/>

We proposed a novel indexing method called MultiBlock to efficiently execute linkage rules. The proposed method uses a multidimensional index in which similar objects are located near each other in order to improve the efficiency beyond existing methods. We also proposed a distributed data flow that employs MultiBlock to scale out the execution of a linkage rule by using the MapReduce paradigm.

We evaluated both the efficiency and the effectiveness of the proposed MultiBlock approach. Efficiency was demonstrated by showing that MultiBlock outperforms four previously proposed indexing methods. Effectiveness was demonstrated by showing that MultiBlock reduces the number of comparisons significantly without any false dismissals even when used with complex linkage rules. Finally, we demonstrated that MultiBlock can be run using the MapReduce paradigm to scale efficiently to a cluster of machines.

While the previous chapters introduced methods to learn and execute linkage rules, the next chapter presents the Silk Link Discovery Framework, which implements the proposed techniques.

Chapter 6

The Silk Link Discovery Framework

The Silk Link Discovery Framework supports users in discovering relationships between entities within different data sources that are represented as RDF. The Silk Link Discovery Framework is implemented in Scala¹ and can be downloaded from the project homepage² under the terms of the Apache License.

Silk provides the declarative Silk Link Specification Language for specifying entity matching tasks, which include the data sources that should be interlinked, the linkage rules used for matching, as well as the type of RDF link that should be generated for all pairs of matching entities. The Silk Link Specification Language supports the full expressivity of the linkage rule representation that has been introduced in this thesis. An overview of the Silk Link Specification Language is given in Section 6.1.

In order to address different use cases, Silk provides three applications to execute link specifications:

Silk Single Machine supports the execution of link specifications on a single machine. Silk Single Machine implements the execution workflow described in Section 5.2 to execute linkage rules efficiently using multiple threads.

Silk MapReduce supports the execution of link specifications on a cluster consisting of multiple machines. Silk MapReduce implements the MapReduce data flow described in Section 5.5 for executing linkage rules on Hadoop clusters.

¹<http://www.scala-lang.org/>

²<http://silk.wbbsg.de/>

Silk Server matches instances from an incoming stream of entities based on a set of link specifications. It can thus be used as an identity resolution component within Linked Data applications that consume a stream of entities to populate a duplicate-free cache. This stream of entities may for instance originate from a Linked Data crawler. For an overview of architectures of Linked Data applications that might employ Silk please refer to Chapter 6 in [Heath and Bizer, 2011].

6.1 Silk Link Specification Language

The *Silk Link Specification Language* (Silk-LSL) is a declarative language for representing link specifications. A link specification encapsulates all information needed to interlink two data sets. This includes the type of link that is to be generated, information on how to access the data sets that are to be interlinked, the linkage rule including a link filter and finally the outputs to which the link should be written to.

Listing 6.1 shows an example of a link specification that specifies how links between movies in DBpedia and LinkedMDB are generated. A link specification consists of four parts:

Prefixes are key-value pairs that define abbreviations for common URL prefixes. The general pattern for specifying a prefix is:

```
1 <Prefix id="prefix id" namespace="namespace URI"/>
```

For instance, a prefix for URLs from the FOAF vocabulary [Brickley and Miller, 2005] can be defined with:

```
1 <Prefix id="foaf" namespace="http://xmlns.com/foaf/0.1/">
```

Data Sources define how the input data is accessed. Two kind of data sources are supported: SPARQL endpoints and RDF files. Additional kind of data sources can be integrated into the Silk Link Discovery Framework as plugins.

Linkage Tasks (identified by the `<Interlink>` directive) specify how specific types of entities in two data sources are to be interlinked. For instance, a linking task may specify how to interlink movies in DBpedia and LinkedMDB. A detailed description of the components of a linkage task will be given later .

Listing 6.1: Example of a links specification.

```

1 <Silk>
2   <Prefixes>(…)</Prefixes>
3   <DataSources>
4     <DataSource id="dbpedia" type="sparqlEndpoint">
5       <Param name="endpointURI" value="http://dbpedia.org/sparql" />
6     </DataSource>
7     <DataSource id="linkedmdb" type="sparqlEndpoint">
8       <Param name="endpointURI" value="http://data.linkedmdb.org/sparql"/>
9     </DataSource>
10  </DataSources>
11  <Interlinks>
12    <Interlink id="movies">
13      <LinkType>owl:sameAs</LinkType>
14      <SourceDataset dataSource="dbpedia" var="a">
15        <RestrictTo> ?a rdf:type dbpedia-owl:Film </RestrictTo>
16      </SourceDataset>
17      <TargetDataset dataSource="linkedmdb" var="b">
18        <RestrictTo> ?b rdf:type movie:film </RestrictTo>
19      </TargetDataset>
20      <LinkageRule>
21        <Aggregate type="min">
22          <Compare threshold="0.0" metric="levenshteinDistance">
23            <TransformInput function="lowerCase">
24              <Input path="?a/rdfs:label[@lang='en']" />
25            </TransformInput>
26            <TransformInput function="lowerCase">
27              <Input path="?b/rdfs:label" />
28            </TransformInput>
29          </Compare>
30          <Compare threshold="400.0" metric="date">
31            <Input path="?a/dbpedia-owl:releaseDate" />
32            <Input path="?b/movie:initial_release_date" />
33          </Compare>
34        </Aggregate>
35      </LinkageRule>
36      <Filter limit="1" />
37    </Interlink>
38  </Interlinks>
39  <Outputs>
40    <Output type="file">
41      <Param name="file" value="links.nt"/>
42    </Output>
43  </Outputs>
44 </Silk>

```

Outputs define the sinks to which generated links are written. Silk supports different types of outputs including writing all links to a file or writing them directly to a triple store by using SPARQL/Update [Gearon et al., 2012]. When writing to a file, the OAIE alignment format is supported [Ontology Alignment Evaluation Initiative, 2011]. An acceptance window can be defined for each output. An acceptance window consists of a minimal confidence as well as a maximum confidence. Only links that are inside this window will be written to the specific output. The acceptance window can be used for writing links with a high confidence, which are to be immediately accepted, and links with a lower confidence, which need to be verified manually, to separate outputs.

A linkage task consists of the following parts:

Link Type: The type of RDF link that should be generated between each pair of matching entities. For instance, the link type can be set to `owl:sameAs` in order to follow the common practice of generating `owl:sameAs` links between entities that are assumed to represent the same real-world object. Silk can also be used to detect other kind of relations. For instance, a property `movie:director` could be specified that relates all entities in a data source about movies with their directors in a data source about persons.

Data Sets: A pair of data sets that are to be interlinked. Data sets can either be loaded from a remote SPARQL [Prud'hommeaux and Seaborne, 2008] endpoint or from a local file. In addition, the data set can be restricted in order to only match a sub set of all entities of the complete data set. For instance, the restriction can be used to only match entities of a specific type, such as persons. Silk-LSL allows arbitrary SPARQL patterns to be used as restrictions.

Linkage Rule: The linkage rule defines the condition that must hold true for two entities in order to generate a link between both entities. The Silk LSL allows the specification of linkage rules using the representation that has been introduced in Section 2.5.

Filter: The link filter allows for filtering of the generated links. It can be used to restrict the number of links that are generated for a single entity. For instance, by setting the limit to one, Silk will only generate at most one link for each entity. If a higher number of links have been generated than the limit allows, only the links with the highest similarity are retained.

A link specification can be executed by the Silk Link Discovery Engine, which is used by all three Silk applications.

6.2 Silk Workbench

The Silk Workbench is a web application that guides the user through the process of interlinking different data sources. It provides the following components:

Workspace Browser: The Workspace Browser allows the user to manage different data sources, corresponding linking tasks and outputs.

Linkage Rule Learner: Linkage Rules can be learned either from existing reference links using the GenLink algorithm or by using the ActiveGenLink algorithm, which asks the user to confirm or decline candidate links in order to generate reference links.

Linkage Rule Editor: Linkage Rules can be viewed and edited using the Linkage Rule Editor.

Reference Links Manager: Reference links can be loaded and edited using the Reference Links Manager.

Execution of Linkage Rules: The current linkage rule can be executed and the generated links can be viewed.

6.2.1 Workspace Browser

The Workspace Browser allows the user to create and edit link specifications. For each link specification it allows the user to manage their prefixes, data sources, linking tasks and outputs in a graphical user interface. The individual parts that make up a link specification have already been described in detail in the previous section.

Figure 6.1 shows an example workspace. Two link specifications are shown in this screenshot, while the second one, `DBpediaDrugBank`, is expanded. `DBpediaDrugBank` contains two data sources: `DBpedia` and `DrugBank`. The `DBpedia` data source has been expanded by the user, in order to view its parameters: As the first parameter, the official `DBpedia` SPARQL endpoint is specified. As this endpoint is usually under heavy load, the maximum number of retries, after a query failed, is set to 100, as specified by the `retryCount` parameter. Whenever a query fails a minimum pause time

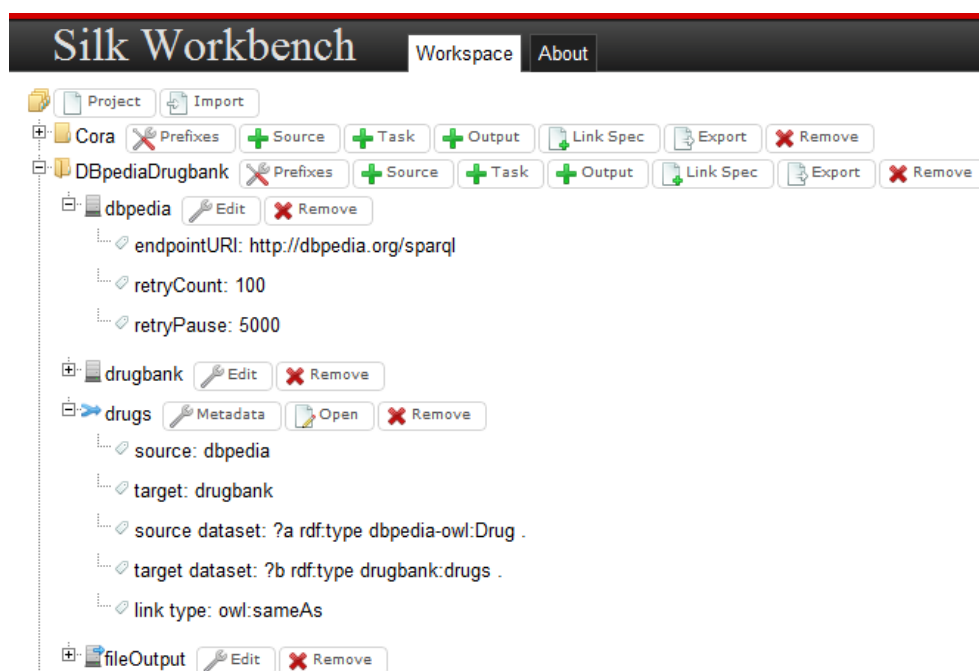


Figure 6.1: Workspace

of 5000 ms is required between two retries, as specified by the `retryPause` parameter.

In addition to the data sources, the shown link specification also includes a linking task for interlinking drugs in both data sets. The linking task is defined by specifying the following parameters:

source/target: The names of the data sources that are to be interlinked.

source/target dataset: Restrictions that define which entities are to be matched. In this example, only entities that are marked as drugs are selected.

link type: The type of link which should be generated for each matching pair.

A newly created linking task starts with an empty linkage rule and an empty set of reference links. The remaining components of the Workbench guide the user through generating a suitable linkage rule and a set of reference links. The reference links can be used to evaluate the quality of the linkage rule as well as for learning a linkage rule using the algorithms which are presented in this thesis.

Finally, all links are to be written to the specified file output. The parameters are not shown in this example.

6.2.2 Linkage Rule Learner

Silk Workbench supports both supervised learning of linkage rules and active learning of linkage rules. Supervised learning from existing reference links is supported through the GenLink algorithm. In cases when no reference links are available, the ActiveGenLink algorithm is used for learning a linkage rule interactively.

We now illustrate active learning of a linkage rule below. After a new linking task has been created in the Workspace Browser, the user can directly start the active learning workflow. In each iteration of the ActiveGenLink algorithm, the GUI shows the five most uncertain links to the user for confirmation. Figure 6.2 shows a simple example of a set of links that are to be verified by the user that have been selected from two data sets that contain movies. The first link candidate has been expanded and displays the

Source: DBpedia	Target: linkedmdb	Score	Correct?
▼ Topaz_%281969_film%29	mdb.org/resource/film/230	-4.1%	<input checked="" type="checkbox"/> ? ✖
<div style="border: 1px solid #ccc; padding: 5px;"> <p> <input checked="" type="checkbox"/> http://dbpedia.org/resource/Topaz_%281969_film%29 ...?a/<http://xmlns.com/foaf/0.1/name> Topaz ...?a/<http://dbpedia.org/ontology/releaseDate> 1969-12-19 ...?a/<http://dbpedia.org/property/name> Topaz </p> <p> <input checked="" type="checkbox"/> http://data.linkedmdb.org/resource/film/230 ...?b/<http://purl.org/dc/terms/title> Topaz ...?b/<http://data.linkedmdb.org/resource/movie/initial_release_date> 1945 ...?b/<http://www.w3.org/2000/01/rdf-schema#label> Topaz </p> </div>			
▶ erbolt_%281910_film%29	ndb.org/resource/film/350	-14.6%	<input checked="" type="checkbox"/> ? ✖
▶ _from_Brazil_%28film%29	ndb.org/resource/film/353	14.6%	<input checked="" type="checkbox"/> ? ✖
▶ kness_%282002_film%29	db.org/resource/film/2320	14.6%	<input checked="" type="checkbox"/> ? ✖
▶ a:Castaway_%28film%29	db.org/resource/film/2051	-15.8%	<input checked="" type="checkbox"/> ? ✖

Figure 6.2: Selected Uncertain Links

properties of the two entities that are referred to by the link candidate. For illustration purposes, we use two simplified data sets in this example, which only contain three properties, while other properties, such as the movie directors, have been removed. The score of link candidate represents the average similarity score according to all linkage rules in the population. The task of the user is to compare the property values of both entities and to decide whether both entries refer to the same movie. The shown link candidate is to be rejected by the user, as both entries refer to distinct movies that share the same title, but have been released at different dates.

After the user confirmed or declined a set of links, ActiveGenLink evolves

the current population of linkage rules. Figure 6.3 shows an excerpt of an evolved population. The four linkage rules with highest score are shown,

Description	Score	MCC	F-Measure	Actions
▼ 2 C. and 2 T.	100.0%	100.0%	100.0%	
<div style="border: 1px solid #ccc; padding: 5px;"> ▢ Aggregation: min <ul style="list-style-type: none"> ▢ Comparison: date (388.86344538560115) <ul style="list-style-type: none"> ...Input ?a/<http://dbpedia.org/ontology/releaseDate> ...Input ?b/<http://purl.org/dc/terms/date> ▢ Comparison: levenshteinDistance (0.6433162607260297) <ul style="list-style-type: none"> ▢ Transformation: lowerCase <ul style="list-style-type: none"> ...Input ?a/<http://dbpedia.org/property/name> ▢ Transformation: lowerCase <ul style="list-style-type: none"> ...Input ?b/<http://www.w3.org/2000/01/rdf-schema#label> </div>				
▶ 2 C. and 0 T.	98.1%	98.1%	99.0%	
▶ 1 C. and 2 T.	94.4%	94.4%	97.2%	
▶ 1 C. and 2 T.	94.4%	94.4%	97.2%	

Figure 6.3: Evolved Population (Top 4)

while the first one has been expanded by the user, in order to view its structure. Three performance measures are shown for each linkage rule in the population: The MCC, the F-measure as well as the score according to the configured fitness measure. The expanded linkage rule compares both movies by their title as well as their release date. The letter case of both titles are normalized by using a transformation that converts all titles to lower case. The normalized titles are compared using the Levenshtein distance, allowing a maximum distance of zero³. The release dates are compared using the date distance measure, allowing a maximum distance of 388 days. The learned linkage rule is the result of reference links that represent different corner cases in the data sets. For instance, by declining the link candidate in Figure 6.2, a negative reference link is added that forces the learning algorithm to generate a linkage rule that does not solely compare movies by their title. The reason for this is that according to the declined link candidate, two movies with the same title may still represent distinct movies.

The learning process can be ended at any time, usually after executing a couple of iterations. The user now may choose to either view and further edit the linkage rule in the linkage rule editor or to directly execute it.

³The actual threshold is 0.64, but the Levenshtein distance only yields integral distances and 0 is the biggest whole number smaller than the threshold.

6.2.3 Linkage Rule Editor

Linkage rules can be viewed and edited in a graphical editor depicted in Figure 6.4. The editor enables the user to experiment and see how changes

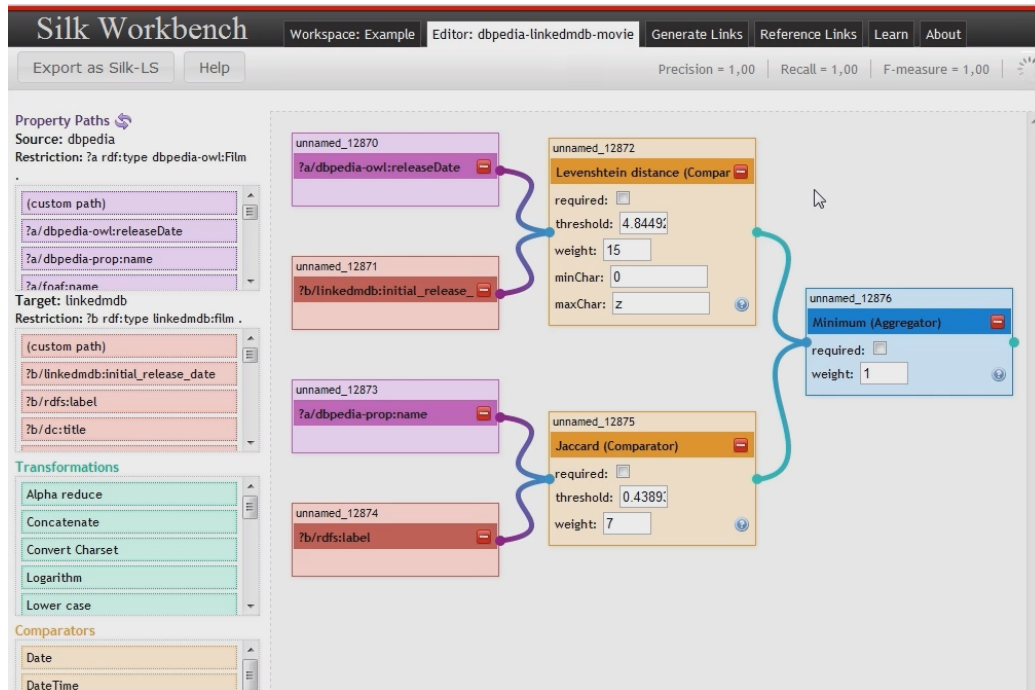


Figure 6.4: Linkage Rule Editor

of the linkage rule affect the accuracy of the generated links. The editor is divided in two parts: The right part shows the linkage rule and enables the user to modify it. The left pane contains the most frequent used properties for the given data sets as well as a list of linkage rule operators, which can be used to modify to the current linkage rule. In the top right corner of the editor precision, recall and F-measure based on the given reference links are shown.

6.2.4 Reference Links Manager

Reference Links can server two purposes: Firstly, the quality of a linkage rule can be assessed by testing how well the rule performs on a set of reference links. Secondly, reference links can be used as a basis for learning a new linkage rule or for improving an existing linkage rule.

The Reference Links Manager allows the user to import an existing set of reference links. The user may browse through positive and negative ref-

erence links and also remove wrong ones. Concerning the task of evaluating the current linkage rule, the Reference Links Manager enables the user to drill down on how the current linkage rule performs on a specific reference link. Via the drill-down feature the individual similarity score of each comparison for a specific link can be viewed. This information allows the user to spot parts of the linkage rule that do not behave as expected or reference links that are labeled incorrectly. Figure 6.5 shows the drill-down feature. In the depicted screenshot, the positive reference links are listed. For each

Source	Target	Score	Status	Correct?
Phantom of the Opera_%281925_film%29	http://data.linkedmdb.org/resource/film/732	80.3%	correct	✖
dbpedia: Demon_Knight	http://data.linkedmdb.org/resource/film/1310	100.0%	correct	✖
Aggregation: min (unnamed_7) 100.0% Comparison: levenshteinDistance (unnamed_6) 100.0% Transform: lowerCase (unnamed_8) tales from the crypt presents: demon knight Input: ?a/<http://xmlns.com/foaf/0.1/name> (unnamed_1) Tales from the Crypt Presents: Demon Knight Transform: lowerCase (unnamed_9) tales from the crypt presents: demon knight Input: ?b/<http://www.w3.org/2000/01/rdf-schema#label> (unnamed_2) Tales from the Crypt presents: Demon Knight Comparison: date (unnamed_5) 100.0% Input: ?a/<http://dbpedia.org/ontology/releaseDate> (unnamed_4) 1995-01-13 Input: ?b/<http://data.linkedmdb.org/resource/movie/initial_release_date> (unnamed_3) 1995-01-13				
ia: Sherman%27s_March_%281986_film%29	http://data.linkedmdb.org/resource/film/217	100.0%	correct	✖
ia: Sherman%27s_March_%281986_film%29	http://data.linkedmdb.org/resource/film/217	100.0%	correct	✖

Figure 6.5: Reference Links

reference link, the URIs of the two referred entities are shown in addition to the similarity score that the current linkage rule assigns to both entities. All positive reference links that are shown in this example have been correctly identified as matches by the linkage rule, as indicated by the **status** column. When a reference link is selected by the user, an extended view is shown that annotates the current linkage rule with the output of each operator in the process of computing the similarity of the two entities that are referred to by the selected link. In the screenshot, the learned linkage rule, which already had been shown in Figure 6.3, is annotated. Taking a closer look at the titles of both movies, which are shown next to the first two input operators, reveals that although both titles match, their letter case is inconsistent. After applying the lower case transformation operators, the normalized titles are identical. For this reason, the first comparison according to the Levenshtein

distance yields a similarity of 100% on the title properties. As the release dates are identical as well, the second comparison also yields full similarity. Finally, the aggregation selects the minimum similarity score yielding a overall similarity score of 100%.

Chapter 7

Conclusion

Identifying entities in data sources that describe the same real-world object is a central problem in data integration. This thesis proposed approaches for learning and executing linkage rules for entity matching. In this chapter, we summarize the key contributions of this thesis, discuss known limitations of the proposed approaches, and suggest directions for future work.

7.1 Summary

This thesis focuses on entity matching approaches that employ domain-specific linkage rules in order to generate links with a high accuracy. The contributions in this thesis are grounded on an expressive linkage rule representation that subsumes linear and threshold-based boolean classifiers and includes data transformations. Based on this foundation, we proposed genetic programming algorithms that are capable of learning linkage rules covering the full expressivity of the proposed representation. We also showed that learned linkage rules can be executed efficiently by presenting a data flow that employs a novel indexing technique to reduce the runtime and that can be executed on distributed system architectures. In the following, we discuss three central challenges that occur in entity matching, which we tackled in this thesis.

7.1.1 Lowering the Effort

As pointed out in the introduction, writing a linkage rule by hand is time consuming and requires a high level of expertise together with detailed knowledge of the data sets. The ActiveGenLink active learning algorithm reduces the expertise required and the manual effort of interlinking data sources by

automating the generation of linkage rules. The user is only required to perform the much simpler task of confirming or declining a set of link candidates which are actively selected by the learning algorithm to include link candidates that yield a high information gain. The user does not need any knowledge of the characteristics of the data set or any particular similarity computation techniques. Each time the user labeled a link candidate, ActiveGenLink uses the supervised GenLink algorithm in order to learn an update linkage rule.

We have discussed previous active learning algorithms for learning linkage rules and highlighted that all previous algorithms known to us use a query strategy that is based on the disagreement of a committee of candidate solutions. We propose an improved query strategy that aims at selecting link candidates for labeling that convey different information than the already labeled link candidates. We show in the experimental evaluation that the proposed query strategy required the user to label fewer links than a query strategy that is solely based on the disagreement committee members.

We have shown that ActiveGenLink outperforms previous unsupervised entity matching systems after labeling at most 30 links on data sets proposed by the Ontology Alignment Evaluation Initiative. On six evaluation data sets, labeling at most 50 link candidates was necessary in order to match the performance that is achieved by the supervised GenLink algorithm on a much bigger set of reference links.

7.1.2 Increasing the Accuracy

GenLink employs genetic programming in order to learn linkage rules from existing reference links by using a set of specialized crossover operators. GenLink is capable of learning linkage rules covering the full expressivity of the linkage rule representation. While many previous approaches for learning linkage rules focus on learning linear or threshold-based boolean classifiers, we present learning algorithms that generate more expressive linkage rules. In addition to enable the representation of non-linear classifiers beyond pure boolean classifiers, we allow linkage rules to include data transformations, which normalize the values prior to comparison.

In a discussion of previous work on supervised learning algorithms, we pointed out that a number of previously proposed approaches use the Cora citation data set and the Restaurant data set for evaluation. In the evaluation of GenLink, we have shown that GenLink outperforms four previous entity matching approaches:

- An entity matching system that uses support vector machines for learn-

ing linear classifiers [Bilenko and Mooney, 2003].

- The state-of-the-art genetic programming approach for entity matching by de Carvalho et al. [2012].
- Two collective entity matching approaches [Domingos, 2004; Dong et al., 2005].

In addition to the Cora and the Restaurant data set, we evaluated our approach on two data sets from the Ontology Alignment Evaluation Initiative and two data sets that have been interlinked by the LATC EU project.

Compared to previous approaches, GenLink uses a more expressive linkage rule representation. In particular, many previously proposed approaches only support linear or threshold-based boolean classifiers. Moreover, none of the previously proposed approaches is capable of learning a linkage rule that includes chains of data transformations. We evaluated the effect of different linkage rule representations to the performance of GenLink on all six evaluation data sets. On two data sets, the capability to represent non-linear classifiers improved the performance over linear and threshold-based boolean classifiers. What’s more, the introduction of transformations improved the performance on all data sets. In particular, we have shown that on the Cora data set, on which GenLink achieves a validation F-measure of 96.6% and outperforms the genetic programming algorithm by de Carvalho et al. by over 5%, the inclusion of data transformations in our proposed linkage rules representation has a big effect: when running GenLink on the Cora data set with a limited linkage rule representation that does not include data transformations the performance drops to an F-measure that is similar to the one that is reported for the genetic programming algorithm by de Carvalho et al.

7.1.3 Increasing the Efficiency

We propose an execution workflow that is both efficient and can be run on cluster of multiple machines. The execution workflow employs a novel indexing method called MultiBlock to efficiently execute linkage rules. MultiBlock uses a multidimensional index in which similar objects are located near each other in order to improve the efficiency beyond existing methods. We evaluated MultiBlock on different data sets and showed data it substantially reduces the number of required entity comparisons, while it guarantees that no false dismissals occur.

On a set of person names, we compared the performance of MultiBlock to four previously proposed indexing methods. Each indexing method has been run with different configurations so that overall ten configurations have

been compared with the MultiBlock approach. MultiBlock was the only method that achieved a recall of 100% on the evaluation data set. Amongst the configurations that achieve a recall of at least 50%, MultiBlock also demonstrated the lowest runtime.

On a large geographic data set, we compared the performance of MultiBlock with the StringMap indexing approach, which also generates an multi-dimensional index. While StringMap accomplishes a bigger reduction in the number of comparisons, the time required for the mapping all entities into the multidimensional index space took about 10 times longer than runtime of the entire entity matching task with MultiBlock. MultiBlock reduced the number of comparisons by a factor of 25,000 and was about 800 times faster than evaluating the complete Cartesian product.

Finally, we evaluated the scalability of MultiBlock on a large life science data set that contains over one million duplicates. The evaluation has been performed on clusters that consist of 8, 16 and 32 machines. Doubling the number of machines reduced the runtime to about 60% of the runtime with half as many machines. On the cluster with 32 machines, all one million links have been generated in about 15 minutes including the time to load the data set.

7.2 Limitations and Future Work

In what follows, we discuss current limitations of the presented approaches and promising directions for future work.

7.2.1 Collective Matching

The learning algorithms that have been proposed in this thesis generate linkage rules that match pairs of entities independently of each other. Thus, matching data sets using the generated linkage rules does not exploit relationships between different entities. For instance, if a pair of citations in a bibliographic data set is identified as match, this does not influence the matching decision concerning other citations. The promise of collective approaches is to exploit relationships between different types of entities in order to improve the overall matching result. For example, if the first author of two citations is found to be matching together with the citations themselves, this information can be used to infer that other pairs of citations that refer to the same author are more likely to match as well. A number of previously proposed collective approaches have been discussed in Section 3.4.4.

On the Cora citation data set, collective approaches outperformed previous approaches that are based on learning linkage rules, but achieved worse results than GenLink. On this data set, the superior performance of GenLink, compared to other supervised approaches, mostly depends on the inclusion of transformations into the linkage rule, as discussed in Section 3.5.4. However, the discussed previously proposed collective approaches do not learn data transformations. As the Cora data set is rather noisy and contains many typographical errors¹, combining the learning of linkage rules that include transformations with collective matching is promising.

In the following, we discuss how GenLink could be extended for collective matching. Currently when matching citations, a linkage rule is learned that includes comparisons for a set of properties. For instance, a linkage rule for matching citations may include a comparison of the author names that includes data transformations for normalizing author names, a distance measure that is used for comparing two author names and a distance threshold. When multiple comparisons are aggregated using a weighted average aggregation, each comparison may also specify a weight that specifies the influence of a specific comparison to the overall similarity score.

Collective approaches typically assume that the data sets that are to be matched provide multiple types of entities [Christen, 2012]. For instance, a citation data set includes entity types such as citations, authors, and venues. On data sets that provide multiple types of entities, GenLink could be used to learn multiple linkage rules concurrently, one linkage rule for each type of entity. In order to support collective approaches in GenLink, the linkage rule representation could be extended to include relationships to linkage rules that match related types of entities. Instead of including a comparison of author names into a linkage rule for matching citations, the linkage rule could include a pointer to the corresponding linkage rule for matching authors, at the same place. The pointer would fulfil the role of specifying that matching decisions of two types of entities are interdependent, e.g., matching two authors influences the matching of citations that refer to these authors. The weight would fulfill the task of specifying the influence of a given relation to the overall score. After linkage rules have been learned for each type of entity that is present in the data set, a collective entity matching approach could be used to execute all rules concurrently.

As collective approaches do not conduct matching decisions independently, but take a global view of the data set, performance can be an issue, although recent efforts are targeted at improving the matching performance on large data sets [Rastogi et al., 2011].

¹Table 3.7 lists some common errors.

7.2.2 Distance Measures and Transformations

In our experiments, we have only used a limited number of distance measures and data transformations. As the evaluation on the Cora data set in Section 3.5.4 showed, the performance could be further improved by implementing more distance measures and data transformations. In particular, transformations that remove frequent stop words or normalize person names could be improve the matching performance. In addition, the use of hybrid distance measures that combine the advantages of character-based measures and token-based measures as described in Section 2.3.3 could further improve the accuracy.

7.2.3 Learning of Functions Parameters

GenLink generates a linkage rule by building an operator tree from a set of predefined operators. For each operator it also learns a number of parameters. For instance, for each comparison operator it learns a suitable distance measure, an appropriate distance threshold as well as the operator weight. In the same way, it finds suitable aggregation functions for aggregation operators and transformation functions for transformation operators. Currently, distance measures, aggregation functions and transformation functions are selected from a user-defined list of functions. If a function needs to be configured (e.g., a transformation function may be configured to filter specific stop-words), the respective configuration needs to be supplied by the user.

In order to avoid this, GenLink could be extended to learn function parameters. As GenLink already achieves a high accuracy on all employed evaluation data sets, this idea has not been explored further so far.

7.2.4 Population Seeding

Instead of starting with a population of random linkage rules, GenLink generates an initial population that only includes linkage rules that compare properties with similar values. For our experiments, two values have been considered similar if the Levenshtein distance between both values after they have been normalized was one or less. The complete heuristic that we used to seed the population has been described in Section 3.3.1.

While the employed seeding heuristic worked well on all evaluation data sets, it could fail if matching entities in different data sets use values that are not detected as similar. For instance, if values in different data sets use different units of measurements the current strategy is likely to fail on detecting both values as similar. In order to cover cases that are now missed,

the heuristic for identifying properties with similar values could be extended in two ways: The normalization of the values could be improved by applying more advanced transformations. In addition, instead of using solely the Levenshtein distance for comparing values for similarity, all distance measures that have been configured to be used by GenLink for learning linkage rules could be used for this purpose.

7.2.5 Query Strategy

ActiveGenLink uses a query strategy that reduces user involvement by selecting link candidates which yield a high information gain. We proposed a query strategy that outperforms the query-by-vote-entropy strategy by selecting a link that is as different as possible from any already labeled reference link. By this, it aims to distribute the labeled links uniformly across the similarity space. Future work may focus on further improving the distribution of the link candidates, which are chosen by the query strategy for manual labeling. For this purpose, clustering algorithms could be employed in order to select link candidates from the unlabeled pool that are in the center of diverse cluster of similar unlabeled link candidates. The query strategy could favor cluster with many link candidates over smaller clusters in order to increase the number of links that are covered by the selected link candidate.

List of Figures

1.1	Entity Matching Workflow.	3
1.2	Execution Data Flow.	4
1.3	Woody Allen in Freebase and DBpedia. Retrieved June 18, 2012.	7
1.4	Screenshot of Google Scholar.	9
1.5	Excerpt of an RDF graph.	12
1.6	Interconnected RDF data sources.	13
1.7	LOD cloud from 2011-09-19.	14
2.1	Case normalization.	26
2.2	Concatenation and splitting of properties as examples of a structural transformations.	28
2.3	Segmentation of a product offer.	28
2.4	Extract a literal value from an URI.	29
2.5	Structure of a linkage rule.	42
2.6	Example of a linkage rule that compares locations.	42
3.1	Example of an entity pair in a geographical data set.	50
3.2	Control flow for genetic programming.	51
3.3	Example of subtree crossover.	54
3.4	Finding compatible properties	58
3.5	Example application of the transformation crossover operator.	62
3.6	Example application of the distance measure crossover operator.	63
3.7	Example application of the threshold crossover operator.	64
3.8	Example application of the combine operators crossover operator.	65
3.9	Example application of the aggregation hierarchy crossover operator.	68
3.10	Example of a support vector machine. Green circles indicate positive reference links, while red circles indicate negative links.	70

3.11	Decision tree that has been learned by the supervised algorithm proposed by Cochinwala et al.	72
3.12	Example of a decision tree that has been learned by Active Atlas.	73
3.13	Relationship graph between two publications (adapted from: [Dong et al., 2005]).	75
3.14	Comparison of linkage rules that have been learned for the Cora data set by two different genetic programming learning algorithms.	80
3.15	Typical precision-recall diagram.	86
3.16	Cora: Learned linkage rule after 10 generations	88
3.17	Cora: Learned linkage rule after 40 generations	88
3.18	Cora: Learned linkage rule without transformations	89
3.19	Learned linkage rule for the LinkedMDB data set	92
3.20	Original linkage rule for the DBpediaDrugBank data set	93
3.21	Average number of comparisons and transformations	94
3.22	Learning performane	99
3.23	Number of comparisons	99
3.24	Number of transformations	100
3.25	Number of comparisons	100
3.26	Number of transformations	101
4.1	Example of an entity pair in a geographical data set.	103
4.2	Active learning workflow.	105
4.3	Kullback-Leibler divergence and Jensen-Shannon divergence.	109
4.4	Overview of the ActiveGenLink workflow.	110
4.5	Distribution of movies in the similarity space	112
4.6	Sampling of entities by label	115
4.7	Example of a decision tree that has been learned by ALIAS.	117
4.8	Example of a learned linkage rule after labeling 10 links.	125
4.9	Example of a learned linkage rule after labeling 20 links.	126
4.10	Example of a learned linkage rule after labeling 30 links.	127
5.1	Data Flow for executing linkage rules.	138
5.2	Example cache.	139
5.3	Example cache	140
5.4	Aggregating a geographic and a string similarity	142
5.5	Index of 1,000 cities in DBpedia	142
5.6	Projection of an entity onto an axis in StringMap.	154
5.7	Overall MapReduce Data Flow	160
5.8	Indexing MapReduce Task	160

5.9	Matching MapReduce Task	161
5.10	Scalability experiment: Linkage rule for linking settlements . .	167
6.1	Workspace	178
6.2	Selected Uncertain Links	179
6.3	Evolved Population (Top 4)	180
6.4	Linkage Rule Editor	181
6.5	Reference Links	182

List of Tables

1.1	Excerpt of a movie database.	1
1.2	Examples of census records.	8
1.3	Excerpt from the Cora data set.	10
2.1	Transformations used in the experiments.	44
2.2	Distance functions used in the experiments.	45
2.3	Aggregation functions used in the experiments.	46
3.1	Experiments in the OAEI instance matching challenges. The third column states the F-measure of the best system. For experiments that consist of multiple data sets the harmonic mean is stated.	78
3.2	Performance comparison of different approaches.	79
3.3	The number of entities in each data set as well as the number of reference links.	83
3.4	The total number of properties in each data set as well as the percentage of properties that are actually set on an entity.	83
3.5	Parameters	84
3.6	Results for the Cora data set. The last row contains the best results of de Carvalho et al. for comparison.	87
3.7	Two examples of false positives. '-' indicates missing values.	89
3.8	Results for the Restaurant data set. The last row contains the best results of de Carvalho et al. for comparison.	90
3.9	Results for the SiderDrugBank data set.	91
3.10	Results for the NewYorkTimes data set.	91
3.11	Results for the LinkedMDB data set.	92
3.12	Results for the DBpediaDrugBank data set	94
3.13	Representations: F-measure in round 25	95
3.14	Seeding: Initial F-measure	96
3.15	Seeding: F-measure in round 10	96
3.16	Crossover experiment: F-measure after 10 and 25 iterations.	97

3.17	F-measure after 10 iterations for different bloating control strategies.	98
3.18	Number of comparisons and transformations in a learned linkage rule size after 10 iterations for different bloating control strategies.	98
4.1	ActiveGenLink Parameters	123
4.2	Results for the SiderDrugBank data set.	124
4.3	Results for the NewYorkTimes data set.	128
4.4	Results for the Cora data set. The last row contains the results of the supervised algorithm.	129
4.5	Results for the Restaurant data set. The last row contains the results of the supervised algorithm.	129
4.6	Results for the LinkedMDB data set. The last row contains the results of the supervised algorithm	130
4.7	Results for the DBpediaDrugBank data set. The last row contains the results of the supervised algorithm	130
4.8	Passive learning	131
4.9	Active learning	131
4.10	Query Strategy: F-measure after 10 iterations	132
4.11	Query Strategy: F-measure after 20 iterations	132
5.1	Steps involved in computing the similarity of two entities compared to computing the index of a single entity.	143
5.2	Comparison of different blocking methods	157
5.3	Comparison of different indexing methods.	166
5.4	Results of scalability experiment	168
5.5	Different phases of StringMap: Runtimes.	168
5.6	The runtimes of the different phases	169
5.7	Results of experiment 2	170
5.8	MapReduce runtimes for different cluster sizes.	171

Listings

2.1	Example of a linkage rule in AJAX (from [Galhardas et al., 2001]).	39
2.2	Two example constraints in Dedupalog (from [Arasu et al., 2009]).	40
3.1	Pseudocode of the GenLink algorithm. The specific parameter values used in our experiments are listed in Section 3.5.2 . . .	56
3.2	Find compatible properties given a set of reference links R^+ , a list of distance measures F^d and a distance threshold θ . . .	58
3.3	Pseudocode of the transformation crossover operator.	61
3.4	Pseudocode of the distance measure crossover operator.	63
3.5	Pseudocode of the threshold crossover operator.	64
3.6	Pseudocode of the combine operators crossover operator.	65
3.7	Pseudocode of the aggregation function crossover operator. . .	66
3.8	Pseudocode of the weight crossover operator.	67
3.9	Pseudocode of the aggregation hierarchy crossover operator. .	67
4.1	Pseudocode of the ActiveGenLink algorithm.	110
5.1	Index map function	161
5.2	Index reduce function	162
5.3	Match map function	162
5.4	Match reduce function	163
6.1	Example of a links specification.	175

Bibliography

- José Luis Aguirre, Bernardo Cuenca Grau, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Robert Willem van Hague, Laura Hollink, Ernesto Jimenez-Ruiz, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Ondrej Sváb-Zamazal, Cássia Trojahn, and Benjamin Zepilko. Results of the Ontology Alignment Evaluation Initiative 2012. In *Proceedings of the Seventh International Workshop on Ontology Matching (OM)*, pages 73–115, 2012.
- Rohit Ananthkrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 586–597, 2002.
- Peter J. Angeline. Genetic programming and emergent intelligence. In *Advances in Genetic Programming*, pages 75–97. MIT Press, 1994.
- Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 952–963, 2009.
- Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 783–794, 2010.
- Samur Araujo, Arjen de Vries, and Daniel Schwabe. SERIMI results for OAEI 2011. In *Proceedings of the Sixth International Workshop on Ontology Matching (OM)*, pages 212–219, 2011.
- Joe Armstrong. A history of erlang. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, pages 1–26, 2007.
- Askarunisa A. Askarunisa A., Prameela P. Prameela P., and Ramraj N. Ramraj N. DBGEN–database (test) generator-an automated framework for

- database application testing. *International Journal of Database Theory and Application*, 2(3):27–54, 2009.
- Amos Bairoch, Rolf Apweiler, Cathy H. Wu, Winona C. Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, and Michele Magrane. The universal protein resource (UniProt). *Nucleic Acids Research*, 33(1):154–159, 2005.
- Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27, 2003.
- Edmon Begoli. A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data. In *Proceedings of the WICSA/ECSA Companion Volume*, pages 177–183, 2012.
- Alexander Behm, Rares Vernica, Sattam Alsubaiee, Shengyue Ji, Jiaheng Lu, Liang Jin, Yiming Lu, and Chen Li. UCI Flamingo Package 4.1, 2010. URL <http://flamingo.ics.uci.edu/releases/4.1/>.
- Christoph Brandt Benjamin Braatz. Graph transformations for the Resource Description Framework. In *Proceedings of the Seventh International Workshop on Graph Transformation and Visual Modeling Techniques*, pages 90–105, 2008.
- Tim Berners-Lee. Linked Data - design issues. <http://www.w3.org/DesignIssues/LinkedData.html>, 2006.
- Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifiers (URI): Generic syntax. <http://www.ietf.org/rfc/rfc2396.txt>, 1998.
- Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1):5, 2007.
- M. Bilenko and R.J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2003.

- Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data – the story so far. *International Journal on Semantic Web and Information Systems*, 4(2):1–22, 2009a.
- Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009b.
- David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In *Proceedings of the Workshop on Genetic Algorithms within the Framework of Evolutionary Computation*, pages 33–38, 1994.
- Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, 1995.
- Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.
- Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1247–1250, 2008.
- Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 175–186, 2001.
- Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. Chapman & Hall/CRC, 1984.
- Dan Brickley and Libby Miller. FOAF vocabulary specification. <http://xmlns.com/foaf/0.1/>, 2005.
- Wentao Cai, Shengrui Wang, and Qingshan Jiang. Address extraction: extraction of location-based information from the web. In *Proceedings of the Seventh Asia-Pacific Web Conference on Web Technologies Research and Development*, pages 925–937, 2005.

- José Ramón Cano, Francisco Herrera, and Manuel Lozano. Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. *Data & Knowledge Engineering*, 60(1):90–108, 2007.
- Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.
- Moisés G. Carvalho, Alberto H. F. Laender, Marcos A. Gonçalves, and Altigran S. da Silva. Replica identification using genetic programming. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1801–1806, 2008.
- Peter Christen. A comparison of personal name matching: Techniques and practical issues. In *Proceedings of the Second International Workshop on Mining Complex Data*, pages 290–294, 2006.
- Peter Christen. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the Second Australasian Workshop on Health Data and Knowledge Management*, pages 17–25, 2008.
- Peter Christen. Development and user experiences of an open source data cleaning, deduplication and record linkage system. *ACM SIGKDD Explorations*, 11(1):39–48, 2009.
- Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2011.
- Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.
- Kenneth L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 226–232, 1983.
- Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137(1):1–15, 2001.
- William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 201–212, 1998.

- William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the Workshop on Information Integration on the Web*, pages 73–78, 2003.
- Ronan Collobert and Samy Bengio. SVM Torch: Support vector machines for large-scale regression problems. *The Journal of Machine Learning Research*, 1:143–160, 2001.
- Lewis Conn and Glenys R. Bishop. *Exploring methods for creating a longitudinal census dataset*. Australian Bureau of Statistics, 2005.
- B. Jack Copeland. *The Essential Turing*. Clarendon Press, 2004.
- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- Shirley A. Cousins. Duplicate detection and record consolidation in large bibliographic databases: the COPAC database experience. *Journal of Information Science*, 24(4):231–240, 1998.
- Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 183–187, 1985.
- Isabel F Cruz, Cosmin Stroe, Federico Caimi, Alessio Fabiani, Catia Pesquita, Francisco M Couto, and Matteo Palmonari. Using Agreement-Maker to align ontologies for OAEI 2011. In *Proceedings of the Sixth International Workshop on Ontology Matching (OM)*, pages 114–121, 2011.
- Ido Dagan and Sean Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the 12th International Conference on Machine Learning*, pages 150–157, 1995.
- Moisés G. de Carvalho, Marcos A. Gonçalves, Alberto H. F. Laender, and Altigran S. da Silva. Learning to deduplicate. In *Proceedings of the Sixth ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 41–50, 2006.
- Moisés G de Carvalho, Alberto HF Laender, Marcos André Gonçalves, and Altigran S da Silva. A genetic programming approach to record deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(3):399–412, 2012.

- Junio de Freitas, Gisele L. Pappa, Altigran S. da Silva, Marcos A. Gonçalves, Edleno Moura, Adriano Veloso, Alberto H. F. Laender, and Moisés G. de Carvalho. Active learning genetic programming for record deduplication. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.
- Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation*, 2004.
- Peter J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.
- Debabrata Dey, Sumit Sarkar, and Prabuddha De. Entity matching in heterogeneous databases: A distance based decision model. In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences*, pages 305–313, 1998.
- Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.
- AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83, 2005.
- AnHai Doan, Ying Lu, Yoonkyong Lee, and Jiawei Han. Profile-based object matching for information integration. *IEEE Intelligent Systems*, 18(5): 54–59, 2003.
- Pedro Domingos. Multi-relational record linkage. In *In Proceedings of the KDD Workshop on Multi-Relational Data Mining*, pages 31–48, 2004.
- Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 85–96, 2005.
- Troy B. Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc., 1998.
- Uwe Draisbach and Felix Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the Seventh International Workshop on Quality in Databases*, pages 51–56, 2009.
- Mohamed G. Elfeky, Vassilios S. Verykios, and Ahmed K. Elmagarmid. TAILOR: A record linkage toolbox. In *Proceedings of 18th International Conference on Data Engineering*, pages 17–28, 2002.

- Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
- Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007.
- Jérôme Euzenat, Alfio Ferrara, Christian Meilicke, Juan Pane, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Šváb-Zamazal, Vojtech Svátek, and Cássia Trojahn. Results of the Ontology Alignment Evaluation Initiative 2010. In *Proceedings of the Fifth International Workshop on Ontology Matching (OM)*, pages 85–117, 2010.
- Jérôme Euzenat, Alfio Ferrara, Willem Robert van Hage, Laura Hollink, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej vb Zamazal, and Cssia Trojahn. Results of the ontology alignment evaluation initiative 2011. In *Proceedings of the Sixth International Workshop on Ontology Matching*, pages 85–113, 2011a.
- Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. Ontology alignment evaluation initiative: six years of experience. *Journal on Data Semantics XV*, pages 158–192, 2011b.
- Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.
- Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):407–418, 2009.
- Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328), 1969.
- Alfio Ferrara, Stefano Montanelli, Jan Noessner, and Heiner Stuckenschmidt. Benchmarking matching applications on the semantic web. In *Proceedings of the Eighth Extended Semantic Web Conference (ESWC)*, pages 108–122, 2011.
- Roy Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. <http://tools.ietf.org/html/rfc2616>, 1999.

- Lyle Ford and Lisa Hanson O'Hara. It's all academic: Google scholar, scirus, and windows live academic search. *Journal of Library Administration*, 46 (3-4):43–52, 2008.
- Carol Friedman and Robert Sideli. Tolerating spelling errors during patient validation. *Computers and Biomedical Research*, 25(5):486–509, 1992.
- Hristescu Gabriela and Farach Martin. Cluster-preserving embedding of proteins. Technical report, Rutgers University, 1999.
- Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 371–380, 2001.
- Paul Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 Update. W3C proposed recommendation. <http://www.w3.org/TR/sparql11-update/>, 2012.
- Rayid Ghani, Katharina Probst, Yan Liu, Marko Krema, and Andrew Fano. Text mining for product attribute extraction. *ACM SIGKDD Explorations Newsletter*, 8(1):41–48, 2006.
- C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: an automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, pages 89–98, 1998.
- David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.
- David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.
- Luis Gravano, Panagiotis G. Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugaelayut Muthukrishnan, Lauri Pietarinen, and Divesh Srivastava. Using q-grams in a DBMS for approximate string processing. *IEEE Data Engineering Bulletin*, 24(4):28–34, 2001a.
- Luis Gravano, Panagiotis G. Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugaelayut Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 491–500, 2001b.

- Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford. Record linkage: Current practice and future directions. Technical report, CSIRO Mathematical and Information Sciences, 2003.
- Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2. W3C recommendation. <http://www.w3.org/TR/soap12-part1/>, 2007.
- Honglei Guo, Huijia Zhu, Zhili Guo, XiaoXun Zhang, and Zhong Su. Address standardization with latent semantic association. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1155–1164, 2009.
- Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
- Okkie Hassanzadeh and Mariano Consens. Linked movie data base. In *Proceedings of the Second Workshop on Linked Data on the Web*, 2009.
- Tom Heath and Christian Bizer. Linked Data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.
- Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 127–138, 1995.
- Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2, 1998. Introduced the multi-pass approach for blocking.
- Gisli Hjaltason and Hanan Samet. Contractive embedding methods for similarity searching in metric spaces. Technical report, University of Maryland, 2000.
- Gísli R. Hjaltason and Hanan Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003a.
- Gísli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems*, 28(4): 517–580, 2003b.

- John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The University of Michigan Press, 1975.
- Wei Hu, Jianfeng Chen, Gong Cheng, and Yuzhong Qu. ObjectCoref & Falcon-AO: Results for OAEI 2010. In *Proceedings of the Fifth International Workshop on Ontology Matching (OM)*, pages 158–165, 2010.
- Wei Hu, Jianfeng Chen, and Yuzhong Qu. A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th International Conference on World Wide Web*, pages 87–96, 2011.
- Jeremy A. Hylton. *Identifying and merging related bibliographic records*. PhD thesis, Massachusetts Institute of Technology, 1996.
- Robert Isele and Christian Bizer. Learning linkage rules using genetic programming. In *Proceedings of the Sixth International Workshop on Ontology Matching*, pages 13–24, 2011.
- Robert Isele and Christian Bizer. Learning expressive linkage rules using genetic programming. *Proceedings of the VLDB Endowment (PVLDB)*, 5(11):1638–1649, 2012.
- Robert Isele and Christian Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2013.
- Robert Isele, Anja Jentzsch, and Christian Bizer. Silk Server – adding missing links while consuming Linked Data. In *Proceedings of the First International Workshop on Consuming Linked Data*, pages 85–97, 2010.
- Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of the 14th International Workshop on the Web and Databases (WebDB)*, 2011.
- Robert Isele, Anja Jentzsch, and Christian Bizer. Active learning of expressive linkage rules for the web of data. In *Proceedings of the 12th International Conference on Web Engineering (ICWE)*, pages 411–418, 2012.
- Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
- Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 2006.

- Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.
- Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Ian Horrocks. LogMap and LogMapLt results for OAEI 2012. In *Proceedings of the Seventh International Workshop on Ontology Matching (OM)*, pages 152–159, 2012.
- Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, pages 137–146, 2003.
- Terry Jones. Crossover, macromutation, and population-based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80, 1995.
- Dmitri V. Kalashnikov and Sharad Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems*, 31(2):716–767, 2006.
- Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. Exploiting relationships for domain-independent data cleaning. In *Proceedings of the SIAM International Conference on Data Mining*, pages 262–273, 2005.
- Minoru Kanehisa, Susumu Goto, Miho Furumichi, Mao Tanabe, and Mika Hirakawa. KEGG for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Research*, 38(1):355–360, 2010.
- Anitha Kannan, Inmar E. Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 404–412, 2011.
- Ron Kohavi, George John, Richard Long, David Manley, and Karl Pfleger. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 740–743, 1994.
- Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.
- Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

- John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1993.
- John R. Koza. *Genetic programming II: Automatic discovery of reusable programs*. Massachusetts Institute of Technology, 1994.
- John R. Koza. *Genetic programming III: Darwinian invention and problem solving*. Morgan Kaufmann, 1999.
- John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*. Springer, 2005.
- Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.
- John Lafferty, Andrew McCallum, and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.
- William B. Langdon and Riccardo Poli. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*, pages 13–22, 1997.
- Dongwon Lee, Jaewoo Kang, Prasenjit Mitra, C Lee Giles, and Byung-Won On. Are your citations clean? *Communications of the ACM*, 50(12):33–38, 2007.
- Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
- Michael Ley. The dblp computer science bibliography: Evolution, research issues, perspectives. In *String Processing and Information Retrieval, Lecture Notes in Computer Science*, pages 481–486. Springer, 2002.
- Michael Ley. Dblp: some lessons learned. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2):1493–1500, 2009.
- Chen Li and Liang Jin. Documentation of the stringmap implementation in the UCI Flamingo Package 4.1, 2010. URL <http://flamingo.ics.uci.edu/releases/4.1/docs/StringMapDoc.html>.
- Huajing Li, Isaac Councill, Wang-Chien Lee, and C. Lee Giles. CiteSeerX: an architecture and web service design for an academic document search engine. In *Proceedings of the 15th International Conference on World Wide Web*, pages 883–884, 2006.

- Huifeng Li, Rohini K. Srihari, Cheng Niu, and Wei Li. Location normalization for information extraction. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7, 2002.
- Ee-Peng Lim, Jaideep Srivastava, Satya Prabhakar, and James Richardson. Entity identification in database integration. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 294–301, 1993.
- Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.
- Frank Manola and Eric Miller. RDF primer. W3C recommendation. <http://www.w3.org/TR/rdf-primer/>, 2004.
- Brian W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442 – 451, 1975.
- Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000a. First use of the Cora data set in the version it is used today.
- Andrew K. McCallum and Kamal Nigam. Employing em in pool-based active learning for text classification. In *Proceedings of 15th International Conference on Machine Learning*, pages 350–358, 1998.
- Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000b. First mention of the Cora dataset.
- Maged Michael, Jose E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, 2007.
- Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock. Exploiting secondary sources for unsupervised record linkage. In *Proceedings of the VLDB Workshop on Information Integration on the Web*, pages 34–39, 2004.
- Alvaro Monge and Charles Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

- David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
- Ion Muslea, Steven Minton, and Craig A. Knoblock. Active learning with multiple views. *Journal of Artificial Intelligence Research*, 27(1):203–233, 2006.
- Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool, 2010.
- Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computer Survey*, 33, 2001.
- Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES – a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, volume 3, pages 2312–2317, 2011.
- Axel-Cyrille Ngonga Ngomo and Klaus Lyko. EAGLE: Efficient active learning of link specifications using genetic programming. In *Proceedings of the Ninth Extended Semantic Web Conference (ESWC)*, pages 149–163, 2012.
- Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. RAVEN – active learning of link specifications. In *Proceedings of the Sixth International Workshop on Ontology Matching (OM)*, pages 25–37, 2011.
- Nikolay Nikolaev and Vanio Slavov. Inductive genetic programming with decision trees. In *Proceedings of the Ninth European Conference on Machine Learning*, pages 183–190, 1997.
- Xing Niu, Shu Rong, Yunlong Zhang, and Haofen Wang. Zhishi.links results for OAEI 2011. In *Proceedings of the Sixth International Workshop on Ontology Matching (OM)*, pages 220–227, 2011.
- Jan Noessner and Mathias Niepert. Codi: Combinatorial optimization for data integration: Results for OAEI 2010. In *Proceedings of the Fifth International Workshop on Ontology Matching (OM)*, pages 142–149, 2010.
- Object Management Group. Common Object Request Broker architecture specification. <http://www.omg.org/spec/CORBA/>, 2012.
- Hiroyuki Ogata, Susumu Goto, Kazushige Sato, Wataru Fujibuchi, Hidemasa Bono, and Minoru Kanehisa. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 27(1):29–34, 1999.

- Ontology Alignment Evaluation Initiative. Alignment format and API. <http://oaei.ontologymatching.org/2011/align.html>, 2011.
- Lawrence Philips. Hanging on the metaphone. *Computer Language*, 7(12), 1990.
- Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A field guide to genetic programming*. Lulu Enterprises Uk Limited, 2008.
- David M.W. Powers. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.
- Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C recommendation. <http://www.w3.org/TR/rdf-sparql-query/>, 2008.
- Dorian Pyle. *Data preparation for data mining*, volume 1. Morgan Kaufmann, 1999.
- J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- John Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers, 1993.
- Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.
- Vibhor Rastogi, Nilesh Dalvi, and Minos Garofalakis. Large-scale collective entity matching. *Proceedings of the VLDB Endowment (PVLDB)*, 4(4):208–218, 2011.
- Lior Rokach and Oded Z. Maimon. *Data mining with decision trees: theory and applications*. World Scientific Publishing Company Incorporated, 2008.
- R.C. Russell. Index, United States patent 1261167, April 1918.
- R.C. Russell. Index, United States patent 1435663, November 1922.
- Fatiha Sais, Noyal Niraula, Nathalie Pernelle, and Marie-Christine Rousset. LN2R—a knowledge based reference reconciliation system: OAEI 2010 results. In *Proceedings of the Fifth International Workshop on Ontology Matching (OM)*, pages 172–180, 2010.

- Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.
- Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 269–278, 2002.
- Andreas Schultz, Andrea Matteini, Robert Isele, Christian Bizer, and Christian Becker. LDIF–linked data integration framework. In *Proceedings of the Second International Workshop on Consuming Linked Data*, 2011.
- Andreas Schultz, Andrea Matteini, Robert Isele, Christian Bizer, and Christian Becker. LDIF–benchmark results. <http://ldif.wbsg.de/benchmark.html>, 2012a.
- Andreas Schultz, Andrea Matteini, Robert Isele, Pablo N Mendes, Christian Bizer, and Christian Becker. LDIF—a framework for large-scale linked data integration. In *21st International World Wide Web Conference, Developers Track*, 2012b.
- Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.
- H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 287–294, 1992.
- Anestis Sitas and Sarantos Kapidakis. Duplicate detection algorithms of bibliographic descriptions. *Library Hi Tech*, 26(2):287–301, 2008.
- Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Don-
garra. *MPI: The Complete Reference*. MIT Press, 1995.
- Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. Linked-GeoData: A core for a web of spatial open data. *Semantic Web Journal*, 2011.
- Robert L. Taft. Name search techniques. Technical report, Bureau of Systems Development, New York State Identification and Intelligence System, 1970.
- Aynaz Taheri and Mehrnoush Shamsfard. SBUEI: Results for OAEI 2012. In *Proceedings of the Seventh International Workshop on Ontology Matching (OM)*, pages 189–196, 2012.

- Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. *Information Systems*, 26(8): 607–633, 2001.
- David W. Walker and Jack J. Dongarra. MPI: A standard message passing interface. *Supercomputer*, 12:56–68, 1996.
- Jason Tsong-Li Wang, Xiong Wang, King-Ip Lin, Dennis Shasha, Bruce A. Shapiro, and Kaizhong Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 307–311, 1999.
- Zhichun Wang, Xiao Zhang, Lei Hou, Yue Zhao, Juanzi Li, Yu Qi, and Jie Tang. RiMOM results for OAEI 2010. In *Proceedings of the Fifth International Workshop on Ontology Matching (OM)*, pages 194–201, 2010.
- Melanie Weis and Felix Naumann. Detecting duplicate objects in xml documents. In *Proceedings of the International Workshop on Information Quality in Information Systems*, pages 10–19, 2004.
- William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, pages 354–359, 1990.
- William E. Winkler. Matching and record linkage. In *Business Survey Methods*, pages 355–384, 1995.
- William E. Winkler. Methods for record linkage and bayesian networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002.
- William E Winkler and Yves Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 us decennial census. *Bureau of the Census, Statistical Research Division, Statistical Research Report Series, n. RR91/09*, 1991.
- Su Yan, Dongwon Lee, Min-Yen Kan, et al. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the Seventh ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 185–194, 2007.
- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory

cluster computing. In *Proceedings of the Ninth USENIX Symposium on Networked Systems Design and Implementation*, 2012.

Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Software: Practice and Experience*, 25(3):331–345, 1995.