Record Linkage for Web Data

by

Oktie Hassanzadeh

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Record Linkage for Web Data

Oktie Hassanzadeh

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2012

Record linkage refers to the task of finding and linking records (in a single database or in a set of data sources) that refer to the same entity. Automating the record linkage process is a challenging problem, and has been the topic of extensive research for many years. Several tools and techniques have been developed as part of research prototypes and commercial software systems. However, the changing nature of the linkage process and the growing size of data sources create new challenges for this task.

In this thesis, we study the record linkage problem for Web data sources. We show that traditional approaches to record linkage fail to meet the needs of Web data because 1) they do not permit users to easily tailor string matching algorithms to be useful over the highly heterogeneous and error-riddled string data on the Web and 2) they assume that the attributes required for record linkage are given. We propose novel solutions to address these shortcomings.

First, we present a framework for record linkage over relational data, motivated by the fact that many Web data sources are powered by relational database engines. This framework is based on declarative specification of the linkage requirements by the user and allows linking records in many real-world scenarios. We present algorithms for translation of these requirements to queries that can run over a relational data source, potentially using a semantic knowledge base to enhance the accuracy of link discovery.

Effective specification of requirements for linking records across multiples data sources requires understanding the schema of each source, identifying attributes that can be used for link-

age, and their corresponding attributes in other sources. Existing approaches rely on schema or attribute matching, where the goal is aligning schemas, so attributes are matched if they play semantically related roles in their schemas. In contrast, we seek to find attributes that can be used to link records between data sources, which we refer to as *linkage points*. In this thesis, we define the notion of linkage point and present the first linkage point discovery algorithms.

We then address the novel problem of how to publish Web data in a way that facilitates record linkage. We hypothesize that careful use of existing, curated Web sources (their data and structure) can guide the creation of conceptual models for semistructured Web data that in turn facilitate record linkage with these curated sources. Our solution is an end-to-end framework for data transformation and publication, which includes novel algorithms for identification of entity types (that are linkable) and their relationships out of semistructured Web data. A highlight of this thesis is showcasing the application of the proposed algorithms and frameworks in real applications and publishing the results as high-quality data sources on the Web.

Dedication

To my mother, Zahra Tarmast

and father, Hossein Hassanzadeh.

Acknowledgements

In a thesis on "Record Linkage for Web Data", it would be ironic to have acknowledgments in the traditional format that will only become part of a single unlinked record in the library. So here is a link to the acknowledgments data on the Web, that include links to URIs that describe (and link) those who are acknowledged. I hope one day, all the contents of theses and scientific publications become properly linked on the Web.

Persistent URL: http://purl.org/oktie/phdack

Backup Link: http://www.oktie.com/phdack

To follow the requirements of a traditional thesis document, I will only list the name of those whom I have collaborated with as a part of the work described in this thesis (sorted by the number of publications we have co-authored, in descending order): Renée J. Miller, Anastasios Kementsietsidis, Lipyeow Lim, Min Wang, Reynold S. Xin, Ken Pu, Soheil Hassas Yeganeh, Mariano Consens, Fei Chiang, Hyun Chul Lee, Christian Fritz, Shirin Sohrabi; and the members of my committee: Chen Li, Sheila McIlraith, and Gerald Penn.

Last but not least, I would like to thank my family for their unconditional love and support during the past five years: my parents, my brother Aidin, sister-in-law Sophie, the Sohrabi family and in particular my beloved Shirin.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Discovering links within and between data sources is a challenging problem and attractive research area. The existence of links adds value to data sources, enhances data and information discovery, and allows or enhances many increasingly important data mining tasks [79]. Without proper links, data sources resemble *islands of data* (or *data silos*), where each island maintains only part of the data that may be necessary to satisfy a user's information needs. In such settings, apart from finding which data are of interest in a given island, users must manually find which other islands might hold relevant data, and what the specific pieces of relevant data are within these islands.

To illustrate, consider a biologist investigating a particular gene. The biologist not only has to find which source contains information for this gene, but also identify which sources might contain information about relevant proteins and genetic disorders. Then, further investigation is necessary to identify the specific proteins and disorders relevant to the particular gene. Similarly, consider a student who wants to study the relationship between the genre of Hollywood movies and their directors' life conditions. For example, she wants to find if the climate of the cities that the movies' directors are from is correlated with the genre of the movies. For this, she not only needs to know where to find information such as a list of Hollywood movies along with their genres and directors, where their directors are born, and information such as aver-

age temperature of the cities; she needs to be able to query the *entities* (or *objects*) and their properties, and then *join* the information about these entities that come from multiple possibly heterogeneous sources.

Recognizing this need, several technologies, data formats and standards have been proposed to realize the vision of *Semantic Web*, a *Web of Data* that extends the existing Web to enable machines to understand the semantics (or meaning) of the information available on the Web. In particular, a set of principles have been proposed to publish structured data using the Resource Description Framework (RDF) data model on the Web, that describe *entities* in a data source with facts (or *triples*) that describe their attributes and link them to other related entities. Each entity is a Hypertext Transfer Protocol (HTTP) resource that can be identified and located on the network by a Uniform Resource Identifier (URI). Data published following these principles is often referred to as *Linked Data* [18]. The Linking Open Data (LOD) community project at W3C [31] started in 2007 to promote publication of open Linked Data on the Web. The number of triples published by the LOD data sources (known as the *LOD cloud*) has grown from around 500 million triples in 2007, to more than 30 billion triples (as of September 2011) that currently describe millions of entities in various domains [32]. Part of the success of the LOD project and its recent growth is due to the development of tools and frameworks that significantly simplify and enhance the process of publishing Linked Data from existing structured (relational) sources.

Despite the significant growth of the LOD cloud, the quality and quantity of the links among entities in different LOD sources is still limited. The number of such links in the LOD cloud is currently only around 500 million (3 million in 2007), and several quality issues have been reported with the existing links such as the existence of incorrect links or links with incorrect types [60, 88]. This is mainly due to the inherent difficulty of the task and the lack of proper tools that assist users and data publishers in establishing an accurate and complete set of links to other sources in a generic and domain-independent way. In what follows, we briefly describe a few problems that make the automation of record linkage over Web data challenging, along with an outline of the contributions of this thesis.

trial	*cond*	*inter*	*loc*	*city*	*pub*
NCT00336362	Beta-Thalassemia	Hydroxyurea	Columbia University	New York	14988152
NCT00579111	Hematologic Diseases	Campath	Texas Children's Hospital	Austin	3058228

(a) Clinical trials (*CT*)

visitid	*diag*	*prescr*	*location*
VID770	Thalassaemia	Hydroxyura	Texas Hospital
VID777	PCV	Hydroxycarbamide	Westchester Med. Ctr

(b) Patient visit (*PV*)

name
Thalassemia
Blood_Disorders

(c) DBpedia Disease (*DBPD*)

name
Alemtuzumab
Hydroxyurea

(d) DBpedia Drug (*DBPG*)

Figure 1.1: Sample relations

1.1 Motivation

1.1.1 Link Discovery over Relational Data

To motivate the problems considered in this thesis, we use an example inspired by our experience in publishing a linked clinical trials data source on the Web and linking this data to existing Web data sources. This application is described in full in Chapter 6. Here we consider the example tables shown in Figure 1.1. One of the sources is a clinical trials data set which includes the sample relation in Figure 1.1(a). For each trial, the *CT* relation stores its identifier *trial*, the *cond*ition considered, the suggested *inter*vention, as well as the *loc*ation, *city*, and related *pub*lication. Another source stores patient electronic medical records (EMR) and includes a patient visit relation *PV* (Figure 1.1(b)) that stores for each patient visit its identifier *visitid*, the *diag*nosis, recommended *prescr*iption, and the *location* of the visit. Finally, we consider a web source extracted from DBpedia [33] (or Wikipedia). This third source stores information about drugs and diseases and includes the *DBPD* and *DBPG* relations (Figure 1.1(c) and (d)) that store the *name*s of diseases and drugs in DBpedia, respectively.

We now describe briefly some of the links that can be found between the records in these tables for two classes of users: 1) a clinician who wants to match patients with clinical trials; and 2) a data publisher trying to publish the clinical trials data online. The data publisher may want to establish links to existing web sources so that users (patients or clinicians) can query the data in both the clinical trials and these linked data sources.

For the *CT* and *PV* relations, we note that the *cond*ition column in the *CT* relation is semantically related to the *diag*nosis column in the *PV* relation, and can be used to establish links between the records in the two relations. Similarly, the *inter*vention column in the *CT* relation and the *prescr*iption column in the *PV* relation are related. Linking records in the two relations based on these columns can be useful to clinicians since they can refer patients to related clinical trials, or can use the result of completed trials to suggest alternative drugs or interventions. In Figure 1.1, patient visit "VID770" with diagnosis "Thalassaemia" in the *PV* relation should be linked to the trial "NCT00579111" with condition "Hematologic Diseases" since "Thalass*a*emia" is a different representation of "Thalassemia" and according to the NCI medical thesaurus "Thalassemia" *is a type of* "Hematologic Diseases". Also, both patient visits in Figure 1.1(b) should link to trial "NCT00336362" (Figure 1.1(a)) based on the fact that "hydroxycarbamide", "Hydroxyura" and "Hydroxyurea" all refer to the same drug. It is clear that for effective linkage, link discovery should be tolerant of errors and differences in the data, such as typos or abbreviation differences. In addition, different types of relationships need to be considered, such as the *same-as* relationship between the drug names, and the *type-of* relationship between the disease names in the above example.

In order to build an online web-accessible view of the clinical trials data, the data publisher can take advantage of an existing class of tools known as RDB2RDF [156] to transform the data into RDF. A Linked Data view over the RDF data can be generated using tools such as Pubby [52], or using an RDB2RDF framework such as D2RServer [29] that publishes the relational data directly on the Web following the Linked Data principles. Using this approach, the *CT* table in Figure 1.1 turns into several triples in the form of subject-predicate-object expressions that describe the two entities of type *trial*, such as the following triple:

```
<http://data.linkedct.org/trial/NCT00336362>          (object)
    <http://www.w3.org/2000/01/rdf-schema#label>      (predicate)
        "NCT00336362"                                 (subject)
```

Note that objects and predicates are HTTP URIs, and subjects can be literals or URIs of other objects. When a user looks up the URI for a trial, e.g., `http://data.linkedct.org/trial/`

NCT00336362, then all the triples related to this object are returned. The subjects could be URIs of internal or external entities. For example, the above trial entity can be linked to http://sws.geonames.org/5128581/ or http://dbpedia.org/resource/New_York_City which are the URIs of *New York City* in the GeoNames [217] and DBpedia [33] Linked Data sources. These latter data sources provide information about the location (such as latitude, longitude, population, etc.). Once we have these links, users can query the clinical trials data using the linked data. For example, a patient looking at Thalassemia trial can ask for trials in large cities (with a population over say 1 million people).

However, the above-mentioned RDB2RDF frameworks do not assist data publishers in discovering and creating links to external sources, or in finding missing links in the internal relational data. In our example, assuming that we have stored DBpedia data in *DBPD* and *DBPG* relations as stated above, we need to find links between the *cond* and *inter* columns of *CT* and the *name* column of the *DBPD* and *DBPG* relations, respectively. The online trials data source can link the condition "Hematologic Diseases" to DBpedia resource (or Wikipedia page) on "Blood_Disorders", and link the intervention "Campath" to DBpedia resource "Alemtuzumab" using the semantic knowledge that "Campath" is a brand name for the chemical name "Alemtuzumab". Here again, it is clear that an effective linkage technique needs to take into account errors and differences in the data (such as typos or abbreviations) in addition to semantic knowledge about the source and target data (such as synonymy or homonymy information).

1.1.2 Discovering Linkage Points

In the above example, we assumed that the correspondence between the attributes is known to the users in advance. For example, we assumed that we know that the *cond* column in *CT* table can be matched with *diag* column in the *PV* table in order to link trials to patient visit records. Similarly, $(CT.inter, PV.prescr)$, $(CT.loc, PV.location)$, $(CT.cond, DBPD.name)$, $(CT.inter, DBPG.name)$, $(PV.diag, DBPD.name)$, and $(PV.prescr, DBPG.name)$ are pairs of columns containing values that can be matched and can be used to link the records.

We refer to these pairs as *linkage points*. In this example, finding the linkage points is easy and straightforward since: 1) the number of attributes (columns) is small; and 2) we have a small set of records with high overlap (or similarity) in values. In practice, the size of both the schema and the instance values maybe very large, rendering it impossible for users to be able to manually find an accurate and complete set of relationships and linkage points.

Table 1.1 shows schema and data statistics for three example entity types and Linked Data sources from the LOD cloud. In the table, *Record#* shows the number of records (or entities), *Fact#* shows the number of triples that have a literal as their subject, and *Attr.#* is the number of unique sequences of predicate URIs (or paths) that associate a record with a literal value. In these real examples, the number of attributes can be as large as $1,738$ (for DBpedia Company records) and the number of facts can be up to 49 million (for IMDb Movie records). Clearly, manual and accurate discovery of *all* the related attributes that can be used in link discovery is not feasible at this scale. Another issue that makes the problem more challenging is the heterogeneity of different sources. Even in our small example in Figure 1.1, some of the values in column $CT.pub$ are in fact identifiers of the trial's related publication in PubMed (a highly popular online source of biomedical literature), and so these records can easily be linked. However, even an expert may easily miss this correspondence due to the vague column name and simple integer identifiers. $CT.pub$ may also contain other types of identifiers, further obfuscating the identification of links. Our experience (which we describe in more detail in Chapter 4) shows that there are plenty of such linkage points over Web data sources.

1.1.3 Linking Semistructured Data

In the the example in Figure 1.1, the data is in a clean, structured, relational format. In reality, many existing Web sources are published using a standard semistructured format such as JSON [204], XML [1] or XML-based formats, or other industry-standard or domain-specific text-based formats such as BibTeX (and other bibliographic data formats), ID3 [203] (audio file data tagging format) or DrugCard [183] (drug information) to name a few. Transforming such data

Table 1.1: Schema and data statistics for three sample entity types and Linked Data sources

Entity	Source	Records#	Fact#	Attr.#
Company	Freebase	74,971	1.92M	167
	SEC	1,981	4.54M	72
	DBpedia	24,367	1.91M	1,738
Drug	Freebase	3,882	92K	56
	DrugBank	4,774	1.52M	145
	DBpedia	3,662	216K	337
Movie	Freebase	42,265	899K	57
	IMDb	1.2M	49.0M	67
	DBpedia	15,165	1.57M	1,021

formats into relational format or high-quality Linked Data could be a challenging task. Consider for example the XML tree shown in Figure 1.2 from ClinicalTrials.gov, a repository of clinical studies conducted all around the world, published online in HTML and XML formats by the U.S. National Institutes of Health. Transforming this data into a structured format with the goal of creating Linked Data on the Web requires building a database schema or *conceptual model* that facilitates proper linking. For high-quality Linked Data, a good conceptual model is one that allows or enables better linking to existing sources. Hence, the model should be created to use entity types and attributes that facilitate linking to known sources. Our goal is to ensure that any instance values that can be linked to external repositories are represented as entities (and not as literals) so that they can be identified and linked.

In the example of Figure 1.2, a simple heuristic might consider non-leaf elements as entity types (e.g., `<clinical_study/>`), while leaf elements containing only text (e.g., `<agency_class/>`) are translated into attributes of the parent entity type. However, simple heuristics like this will not always lead to *linkable* and *duplicate-free* data. For instance, in our example, there is a non-leaf, container element (`<id_info/>`) which does not represent a common real-world entity type (that is, its role is structural rather than semantic). There is also a leaf element (`<country/>`) which represents a real-world entity type. It is important to ensure that country values are represented as entities and can be linked to appropriate entities in the LOD

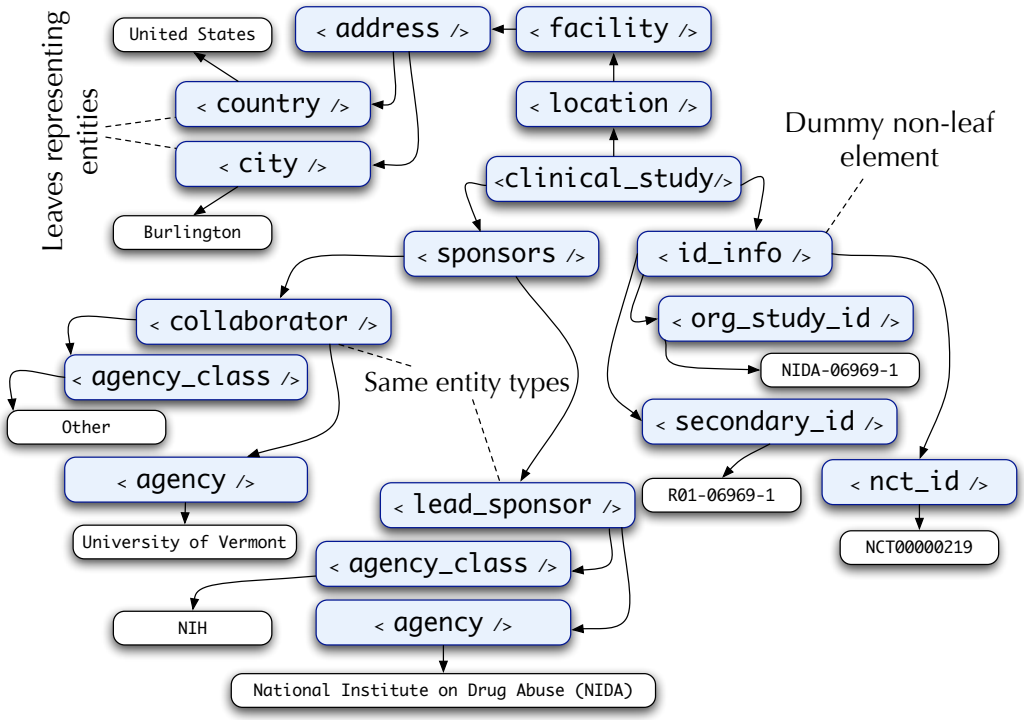


Figure 1.2: Sample XML tree from ClinicalTrials.gov

cloud. Moreover, non-leaf elements such as <id_info/> may exist only for human-readability purposes, and may not represent a commonly-used concept. Another challenge associated with automatic entity type identification is detecting *duplicate types*. In Figure 1.2, there are two identical subtrees in the structure (<lead_sponsor/> and <collaborator/>). These two subtrees are different instances of the same entity type, which may not be obvious at first glance. Our goal is to detect such subtrees as representing identical types in order to facilitate linking.

1.2 Objectives

The challenges described above could make the process of publishing high-quality Linked Data on the Web a daunting task, and pose a serious obstacle for the growth in the amount and quality of Linked Data on the Web. On the other hand, there has been a lot of research on these or similar problems, in several communities such as statistics, data management, information retrieval, data mining, machine learning, and bioinformatics, to name a few. Research in different communities often address different aspects of the problem (e.g., scalability, accuracy, or

domain-dependence). In addition, different terminologies are often used to refer to the same problem. This has led to limited reuse or sharing of work between these communities. Therefore, one of the main goals of this thesis is not only to provide a new perspective on the state-of-the-art in research on these problems in several fields, but also to take advantage of years of research particularly in data management and information retrieval (IR) communities.

Traditionally, the term *record linkage* is used to refer to the process of finding records in databases that refer to the same real-world entity. In this thesis, we consider a more generic form of this classic definition where the goal is not only to link records that refer to the same entity, but also link records that are semantically related (e.g., records of locations that are close to each other, medical treatment records related with clinical trial records, etc.). Our approach is based on the existence of a function that computes the *similarity* (or *distance*) between the records. Two records r_s and r_t are linked if they match based on a Boolean *record matching* function $f(r_s, r_t)$, which returns true if the two records match according to a matching criteria. The matching criteria can be defined, for example, using an *attribute matching* function $f_{(a_s,a_t)}(r_s, r_t)$, that returns true if the values associated with the two attributes (or fields) a_s of r_s and a_t of r_s are (semantically) *related*. There are several ways to define the notion of semantic relatedness. For example, two values v_s and v_t can be considered related if their similarity score according to a value similarity function $sim()$ is above a threshold θ, i.e., $sim(v_s, v_t) \geq \theta$. Using this approach, our proposed solutions can be classified as *distance-based techniques* [68]. Such techniques have the advantage of being unsupervised, and therefore do not require tuning through training data or human input.

Another major goal of this thesis is simplicity and ease of use of the proposed solutions to the above-mentioned problems. Our goal is to help non-expert data publishers, who only have a basic knowledge of data management, to be able to take advantage of these solutions to publish or increase the quality of their data on the Web. To this end, we propose frameworks that assist users in all the steps of data transformation and linkage. This includes understanding the data and building a basic but accurate conceptual model, the discovery of attributes that are related

and can be used for linkage, and the discovery of links to external sources. In addition, our goal is to provide generic and domain-independent solutions that could assist users in a variety of domains and applications. Achieving this goal is not possible without extensive evaluation over real-world data and scenarios. Therefore, a by-product of our research is a set of high-quality Linked Data sources that extend the LOD cloud. These sources include a source of clinical trials, a bibliographic data source, and a source of movies and movie-related information.

Thesis Statement
A generic and extensible set of linkage algorithms combined within an easy-to-use framework that integrates and allows tailoring and combining of these algorithms can be used to effectively link large collections of Web data from different domains.

1.3 Contributions

First, we provide an overview of the state-of-the-art in record linkage. We present a brief overview of the work on record linkage in the statistics community that addresses the problem of linking those records in two files that represent the same individual. We then focus on a few extensions of the classic record linkage problem and those aspects of the problem that have received less attention in the literature, in particular by the data management community.

The first technical contribution of this thesis is a declarative framework for link discovery over relational data. We use the term *link discovery* to refer to the more generic form of the classic record linkage problem, where the goal is to link records that refer to the same or semantically related entities. We introduce a generic, extensible, and easy to use framework for finding links between relational sources. One of the main users of our framework are data publishers who want to link their own data with that in other sources. These publishers, although non-experts in computers, are usually knowledgeable of basic declarative languages like SQL. With this in mind, we propose an SQL-like declarative language for finding links, called LinQL. A query written in LinQL identifies: (a) a subset of the *source* data to be linked; (b) *target* data that are to be linked with the source data from (a); and (c) the algorithm(s) to be used to find links between

the local and external data. We note that this is the first attempt to bridge the gap between query and linking languages. This, not only adds expressiveness to both languages, but also allows for a more *online* and *interactive* specification and experimentation with finding links. In contrast, existing techniques find links in a rather *offline* batch mode.

Irrespectively of the language used, a declarative specification must be translated into some form of a program that finds the links between the local and external data. Often, such programs are implemented using programming languages like Java or C by third-party developers, and are automatically invoked with arguments defined through the declarative specification. As such, for the data publishers these programs act as *black-boxes* that sit outside the data publishing framework and whose modification requires the help of these developers. Our solution addresses these shortcomings by providing a vanilla SQL implementation for link discovery. Our approach has several advantages including the ability to: (a) easily implement this framework on existing relational data sources with minimum effort and without any need for externally written program code; (b) take advantage of the underlying DBMS optimizations in the query engine while evaluating the SQL implementations of the link finding algorithms; and (c) use several efficiency and functionality enhancements to improve the effectiveness of these algorithms. We have implemented our framework and several algorithms on a commercial database engine and validated the effectiveness of our approach.

The second contribution of this thesis is a framework for discovering pairs of attributes in two data sources that are semantically related and can be used to establish links between records in the two data sources. This is traditionally performed using a *match* operator, that associates the schema elements of one database to another. However, we will show that the massive growth in the amount of unstructured and semistructured data on the Web has created new challenges for this task. Such data sources often do not have a fixed pre-defined *schema* and contain large numbers of diverse attributes. Furthermore, the end goal is not schema alignment as these schemas may be too heterogeneous (and dynamic) to meaningfully align. Rather, the goal is to link the records using any overlapping data shared by these sources. We will show that even

attributes with different meanings (that would not qualify as schema matches) can sometimes be useful in linking records. We refer to such attribute pairs as *linkage points*. The solution we propose in this thesis replaces the basic schema-matching step with a more complex instance-based schema analysis and linkage discovery. We present a framework consisting of a library of efficient lexical analyzers and similarity functions, and a set of search algorithms for effective and efficient identification of linkage points over Web data. We experimentally evaluate the effectiveness of our proposed algorithms in real-world integration scenarios in several domains.

Our third contribution is an end-to-end framework for transforming a possibly large set of semistructured data instances into rich high-quality Linked Data. The input to the system can be instances of static semistructured data sources, or dynamic user-generated content such as BibTeX entries, or RSS feeds. The output is a high-quality Linked Data source that is linked to external sources, includes provenance information (information on how each entity or link in the source is created), and allows users to provide feedback on the quality of the data by providing a custom data browsing interface and metadata repository. As a part of this framework, we present a set of algorithms for effective identification of entity types and their relationships out of semistructured Web data, which is tailored to enable the generation of high-quality Linked Data. First, by taking advantage of the given hierarchy in the data, we derive an initial set of entity types and relationships. We then use techniques to enhance the quality of these entity types. Our approach includes methods to identify and remove duplicate entity types, enhance precision (*i*.e., identify those non-leaf nodes in the structure graph that should not be mapped to entity types), and improve recall (*i*.e., identify those leaf nodes that need to be mapped to entity types). In order to evaluate the effectiveness of our proposed entity type identification algorithms, we create a novel set of benchmark data sets for this task using real-world data sets from several domains.

A highlight of the contributions in this thesis is extensive evaluation over real-world data and applications. We used our experience in managing and publishing three data sets to both identify the problems studied in this thesis and to evaluate our solutions. We also ex-

tended the Linked Open Data cloud with these three sources together with the links we discovered. These sources include: 1) BibBase (http://data.bibbase.org/), a web service for publishing and managing bibliographic data from BibTeX files. BibBase uses a powerful yet lightweight approach to instantly transform users' BibTeX files into rich Linked Data as well as custom HTML code and RSS feed that can readily be integrated within a user's website. 2) LinkedCT (http://linkedct.org/), a Linked Data source of clinical trials, with links to several external sources, meta-data about the links, and data refreshed daily. 3) LinkedMDB (http://linkedmdb.org/) a source of movie data integrated from several online sources on the Web.

1.4 Organization of the Dissertation

This dissertation is organized as follows.

- Chapter 2 presents an overview of the state-of-the-art in record linkage. The focus of this chapter is on providing a new perspective on the literature on record linkage by focusing on those aspects of the problem that have received less attention in the literature, in particular the data management literature. We also discuss related work in other communities that are related to or have application in record linkage. The topics we cover include string and semantic matching methods, graph clustering algorithms, and entity identification. We also present an overview of a few existing record linkage libraries and systems with a brief overview of their features.

- In Chapter 3, we present the Linkage Query Writer (LinQuer), a declarative framework for link discovery over relational data. We introduce LinQL, an extension of SQL that integrates querying with linkage methods. We present a set of linkage methods natively supported by LinQL that can be translated into standard SQL queries. These methods allow syntactic and semantic matching of records. LinQL also supports indexing techniques

that can significantly improve scalability. We present a set of algorithms for transformation of LinQL into SQL. We evaluate the effectiveness of the proposed linkage methods over real data sets.

- Chapter 4 presents a novel class of algorithms for discovering linkage points between multiple sources. We first formally define the notion of a *linkage point*. We then show how our proposed algorithms take advantage of a library of lexical analyzers and similarity functions to effectively and efficiently discover and rank linkage points. We evaluate and compare the algorithms using several real-world scenarios.

- In Chapter 5, we present xCurator, a framework for transforming semistructured Web data into high-quality Linked Data automatically with *optional* human intervention. We discuss several research challenges involved in the design and implementation of this framework. We present algorithms for the extraction of entities and their types along with the extraction of relationships out of a semistructured source. We present the results of benchmarking these algorithms using a set of real data sets.

- Several applications of the proposed techniques are described in Chapter 6. These applications include: 1) BibBase, a web service for publishing and managing bibliographic data from BibTeX files; 2) LinkedCT, a Linked Data source of clinical trials; and 3) LinkedMDB, a source of movies and movie-related information integrated from several online sources on the Web. We present an overview of the unique challenges involved in data transformation and record linkage in each of the data sources. We also discuss the impact of these data sources on research and applications, and our future plans for extending them.

- Finally, Chapter 7 concludes the thesis and presents some directions for future work. These include extensions of the proposed frameworks and algorithms. We also present our vision for a lightweight, generic, easy-to-use framework for online enterprise data analytics using the power of (Linked) Web data.

Chapter 2

State-of-the-art in Record Linkage

In an early article published in December 1946 in the American Journal of Public Health, H. L. Dunn defines the term *record linkage* as the name given to the process of assembling the pages of the *Book of Life* into a single volume [64]. This Book of Life for each person is a book that starts with birth, ends with death, and its pages are made up of records of the principle events in a person's life. The article points out several aspects of the importance of proper record linkage to the society and governments, including the effect it can have on health, welfare and public services. Dunn notes that at the time, Canada was among the few countries in the world that enforced linkage of vital records of individuals. This was done due to legislation on government-provided family allowances, and by maintaining a *Life Records Index* at The Dominion Bureau of Statistics for each person under twenty years of age. As the population grew, the need for record linkage for all citizens became apparent, as well as the need for automated accurate methods for linking the records of individuals. This was soon found to be an incredibly challenging task.

The first proposal for an automated computer-oriented solution to the record linkage problem appears in *Science* in 1959 [142]. The authors describe the challenges they faced in linking $34,138$ birth records from the Canadian province of British Columbia all gathered in one year, with $114,471$ marriage records gathered over the 10-year period before the births, in an attempt to "look for possible differentials of family fertility in relation to the presence or absence

of hereditary disease". The challenges include the existence of differences in spellings of the names, the ambiguity in linkage as a result of common family names in the region of study, and cases such as marriage of two brothers with two sisters. Later on in 1969, Fellegi and Sunter from the Dominion Bureau of Statistics (currently Statistics Canada) formalized the record linkage problem and presented a mathematical model which formed the foundation behind many record linkage tools and techniques [72]. This model, which is still being widely used, is based on assigning record pairs (one record from each source) to two classes M and U. The class M, called *matched*, consists of record pairs that refer to the same entity (or *match*), whereas the class U, called *unmatched*, contains record pairs that represent different entities (are *non-match*). The record linkage task between source A and B involves making one of three decisions for each record pair $< \alpha, \beta > (\alpha \in A, \beta \in B)$: A_1 (*link*) to assign them to M, A_3 (*non-link*) to assign them to U, and A_2 (*possible link*) where there is insufficient evidence to justify either A_1 or A_3. The A_1 and A_3 decisions are called positive dispositions.

In the Fellegi-Sunter model, assuming that sources A and B contain n comparable fields (or attributes), each record pair $< \alpha, \beta >$ is represented as a vector γ (called *comparison vector*) with n components where the i^{th} component corresponds to the level of agreement between the i^{th} field in α and β. Then, a *linkage rule* is defined as a rule that assigns probabilities $P(A_1|\gamma)$, $P(A_2|\gamma)$, and $P(A_3|\gamma)$ to each possible realization of $\gamma \in \Gamma$ where Γ is the set of all possible realizations of γ, referred to as the *comparison space*. Given a fixed error level for decisions A_1 and A_3, an *optimal* linkage rule is defined as a rule that minimizes the probability of failing to make positive dispositions at those fixed error levels. In practice, failure of the system to make a positive disposition means requiring an expert to review the results and manually make the decision.

There has been considerable research that builds upon and extends the Fellegi-Sunter model [14, 68, 85, 178, 179, 181, 182]. A general assumption in these probabilistic linkage models is that the comparison vector γ is a random vector whose density function is different for each of the M and U classes. Then, if the density function for each class is known, the record linkage

problem becomes a Bayesian inference problem. The density functions are often derived in a supervised manner using manually classified records, and often based on conditional independence assumption between the distribution of different components of γ. Unsupervised and semi-supervised methods have also been proposed, using an expectation maximization (EM) algorithm for example [111, 179, 180].

Today, the initial record linkage problem as defined by Dunn [64] which inspired the work of Fellegi and Sunter and many others, has mostly been solved after years of research followed by solid government regulations that enforced assigning unique identifiers to individuals (such as birth certificate, passport, driving license, social security and health card identifiers) and properly linking all the identifiers (e.g., laws that prohibit being able to get a driving license or health card without a valid birth certificate). However, with the huge amount of data that resides nowadays in data warehouses and on the Web, the general problem of identifying records in databases that refer to the same real-world entity is more widespread than ever, and its *scale* and *impact* have grown significantly. A white paper from The Data Warehousing Institute in 2002 estimates that data quality problems cost U.S. businesses more than 600 billion a year [65]. This report identifies the existence of duplicate records as one of the main data quality problems in warehouses. The same report states that according to a firm that specializes in auditing data (quality), the duplication rate in their customers records ranges from 5 to 20 percent.

With the growing size of data and duplication in data warehouses and other data repositories, there has been a growing interest in the data management community to address the scalability of the record linkage problem. The majority of the proposed solutions in the data management literature are unsupervised, and rely on so-called *distance-based* techniques [68], where a similarity (or distance) measure is used together with a (learned or manually assigned) threshold to determine members of record pairs in the matched and unmatched classes. Two records *match* if their similarity is above a matching threshold (or the distance is below a distance threshold) and are a *non-match* otherwise. In addition, several indexing, hashing and blocking techniques have been proposed to significantly improve the efficiency of the linkage by avoiding a possibly

expensive similarity computation for all record pairs and computing the similarity for only a subset (or a *block*) of candidate pairs [47]. This subset needs to be carefully selected to avoid missing matches and minimize the effect on the accuracy.

In this thesis, we only consider distance-based techniques due to their unsupervised nature and the scalability of their implementation. In the remainder of this chapter, we first present an overview of several similarity measures that can be used in distance-based record linkage techniques. We focus on string data, and a subset of the measures proposed in the literature that are most relevant to the work presented in this thesis. Although in certain linkage scenarios it is possible to treat records as string values and compare the strings, in some cases this approach may ignore important information that can be used to enhance record linkage. We discuss a few proposals that take into account co-occurrence information in addition to the similarity values. We also present an overview of clustering algorithms that can be used to avoid inconsistent links, e.g., cases where $< \alpha, \beta >$ and $< \alpha, \gamma >$ are returned as matches but $< \beta, \gamma >$ is returned as a non-match. We then present a brief summary of the research in the literature on *entity identification* and its relationship to record linkage. Finally, we present a brief discussion of a few existing record linkage libraries and systems, and end the chapter with a brief summary and concluding remarks.

2.1 String Matching for Record Linkage

String data is abundant in large databases and on the Web. A large body of work in the literature addresses the problem of matching strings for record linkage and many other applications. The simplest similarity measure that can be used to match string records is *string equality*, which returns 0 if the strings are not identical and 1 if they are identical. Clearly, string equality works well only if the records are from the same source and follow a strict standard to uniquely represent entities, which is rarely the case.

2.1.1 String Normalization

A typical way to overcome the problem of different string representations of the entities in data records is *normalization* (or *standardization*) where a set of transformations are performed over the strings before comparing them using string equality. Some common normalization functions include the following [71].

- **Case normalization** converting the strings into all lowercase (or alternatively into all uppercase).
- **Diacritics suppression** replacing characters with diacritic signs with their common non-diacritical replacement, e.g., transforming `Renée` to `Renee`.
- **Blank normalization** removing any whitespace such as blank characters, tabs and carriage returns or replacing them with a single blank character.
- **Link stripping** removing links between words such as apostrophes and underline characters or replacing them with blanks.
- **Digit suppression** removing all the digits.
- **Punctuation elimination** removing punctuation signs.

It is clear that the effectiveness of normalization relies heavily on the data characteristics and good domain knowledge. Careful investigation of the results is needed in order to find the right normalization technique for each data source.

2.1.2 String Similarity Measures

There are a large number of similarity functions for string data. Such similarity measures take as input two strings (and in some cases some additional knowledge about the full data set such as the frequency of occurrence of the strings or parts of the strings) and return a score (usually between 0 and 1) that indicates the similarity between the two strings or the probability that the

two strings refer to the same real-world entity. The choice of the similarity function also highly depends on data characteristics. In what follows, we provide an overview of three classes of string similarity measures. The first class, called *character-based*, consists of measures that calculate a similarity score based on comparing characters and computing the cost of transforming one string to another. The second class are based on transforming the strings into a set of tokens, which we refer to as *token-based* measures. Tokens can be word tokens (splitting the strings by whitespace) or q-grams (sequences of q consecutive characters of a string). The third class, called *hybrid* measures, are those that combine character-based and token-based measures. The measures that we discuss in more detail share one or both of the following properties.

- High accuracy: Previous work has shown that these measures perform better or equally well in terms of accuracy when compared with other string similarity measures. Specifically, these measures have shown good accuracy in name-matching tasks [49] (where the goal is to perform record linkage by matching names of objects) or in approximate selection [89] (where the goal is to find a set of strings most similar to a given query string) over company names and scientific publications data.

- High scalability: There are various indexing and hashing techniques proposed in the literature for enhancing the performance of the similarity computation (similarity join operation). Not all similarity measures have such scalable implementations.

Edit Similarity

The most widely-used character-based similarity measure is edit similarity. Several proposals have been made to improve the efficiency of calculating this measure for large data sets. Specifically, previous work has shown how to use q-grams for an efficient implementation of this measure in a declarative framework [84]. Recent work on enhancing the performance of similarity joins has also proposed techniques for scalable implementation of this measure [8, 128].

Edit distance between two strings r_1 and r_2 is defined as the transformation cost of r_1 to r_2, $tc(r_1, r_2)$, which is equal to the minimum cost of edit operations applied to r_1 to transform it to r_2. Common edit operations include character *insert* (inserting a new character in r_1), *delete* (deleting a character from r_1) and *substitute* (substitute a character in r_1 with a new character) [87]. The edit similarity is defined as:

$$sim_{edit}(r_1, r_2) = 1 - \frac{tc(r_1, r_2)}{\max\{|r_1|, |r_2|\}} \tag{2.1}$$

There is a cost associated with each edit operation. There are several cost models proposed for edit operations. The most commonly used measure is called Levenshtein distance [127], which we will refer to as edit distance throughout this thesis. It uses equal cost for all operations and only the three edit operations *insert*, *delete*, and *substitute*.

Other Character-Based Similarity Measures

Several other character-based measures exist in the literature. The *affine gap distance* [173] introduces two extra edit operations: *open gap* and *extend gap*. This allows decreasing the cost for gap mismatches to better match strings such as "John R. Smith" and "Jonathan Richard Smith". Bilenko and Mooney [25] present a method for learning the cost of edit operations from a training set. Smith and Waterman [162] propose an extension to edit and affine gap distance in which mismatches at the beginning and the end of the strings are assigned a lower cost. This will enhance matching of strings that only share a substring. Another popular character-based string similarity measure is the Jaro-Winkler measure [178], which has proven to work well on short strings such as English first name and last names, and is an extension of the Jaro measure [110]. The Jaro measure is defined as:

$$Jaro(r_1, r_2) = \frac{1}{3}\left(\frac{m}{|r_1|} + \frac{m}{|r_2|} + \frac{m-t}{m}\right) \tag{2.2}$$

where m is the number of characters that are not farther than $\lfloor \frac{max(|r_1|,|r_2|)}{2} \rfloor - 1$ from each other (called *matching* characters), where t is half the number of transpositions. The number of transpositions is computed by comparing the i^{th} matching character in r_1 with the i^{th} matching character in r_2. The i^{th} matching characters that do not match are considered to be a transposition. The Jaro-Winkler measure extends the Jaro measure by giving a higher weight to prefix matches which are generally more important especially in name matching.

Set Similarity Measures

By treating strings as sets of word tokens or q-grams, it is possible to use set similarity measures. Let **r** denote the set of tokens in string r. The simplest way to measure set similarity is to use the size of the intersection of the token sets, i.e.:

$$sim_{Intersect}(r_1, r_2) = |\mathbf{r}_1 \cap \mathbf{r}_2| \tag{2.3}$$

The normalized version of the above measure, along with q-gram tokens, is often referred to as the *q-gram* (or *n-gram*) similarity between the two strings [71]:

$$sim_{q-gram}(r_1, r_2) = \frac{|\mathbf{r}_1 \cap \mathbf{r}_2|}{min(|\mathbf{r}_1, \mathbf{r}_2|)} \tag{2.4}$$

The above measure can also be seen as the maximum inclusion degree of one set in the other.

Jaccard similarity (or Jaccard coefficient) is the fraction of the tokens in r_1 and r_2 that are present in both, i.e.:

$$sim_{Jaccard}(r_1, r_2) = \frac{|\mathbf{r}_1 \cap \mathbf{r}_2|}{|\mathbf{r}_1 \cup \mathbf{r}_2|} \tag{2.5}$$

Weighted Jaccard similarity is the weighted version of Jaccard similarity:

$$sim_{WJaccard}(r_1, r_2) = \frac{\sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w_R(t)}{\sum_{t \in \mathbf{r}_1 \cup \mathbf{r}_2} w_R(t)} \tag{2.6}$$

where $w_R(t)$ is a weight function that reflects the commonality of the token t in the relation R.

This weight can be the Inverse Document Frequency (IDF) or a modified form that has shown to be more effective than simple IDF in previous work on string matching [89]:

$$w_R(t) \quad = \quad log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \tag{2.7}$$

where N is the number of tuples in the base relation R and n_t is the number of tuples in R containing the token t.

Measures from IR

A well-studied problem in information retrieval is the problem of given a query and a collection of documents, return the most *relevant* documents to the query. If we view a document and query as records and the words in a document and query as the tokens or q-grams of the records, then document relevance measures can be applied to record matching. Here, we present three such measures that have been shown to have higher performance for the approximate selection problem [89].

Cosine w/tf-idf The *tf-idf cosine* similarity is a well-established measure in the IR community [157]. This measure determines the closeness of the input strings r_1 and r_2 by first transforming the strings into unit vectors and then measuring the angle between their corresponding vectors. The *cosine* similarity with tf-idf weights is given by:

$$sim_{Cosine}(r_1, r_2) = \sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w_{r_1}(t) \cdot w_{r_2}(t) \tag{2.8}$$

where $w_{r_1}(t)$ and $w_{r_2}(t)$ are the normalized *tf-idf* weights for each common token in r_1 and r_2 respectively. The normalized *tf-idf* weight of token t in a given string record r is defined as follows:

$$w_r(t) = \frac{w'_r(t)}{\sqrt{\sum_{t' \in \mathbf{r}} w'_r(t')^2}} \quad , \quad w'_r(t) = tf_r(t) \cdot idf(t)$$

where $tf_r(t)$ is the term frequency of token t within string r and $idf(t)$ is the inverse document

frequency with respect to the entire relation R.

BM25 The BM25 similarity score for strings r_1 and r_2 is defined as follows [155, 89]:

$$sim_{BM25}(r_1, r_2) = \sum_{t \in \mathbf{r_1} \cap \mathbf{r_2}} \hat{w}_{r_1}(t) \cdot w_{r_2}(t) \tag{2.9}$$

where:

$$\begin{aligned} \hat{w}_{r_1}(t) &= \frac{(k_3+1) \cdot tf_{r_1}(t)}{k_3 + tf_{r_1}(t)} \\ w_{r_2}(t) &= w_R(t) \frac{(k_1+1) \cdot tf_{r_2}(t)}{K(r_2) + tf_{r_2}(t)} \\ w_R(t) &= \log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \\ K(r) &= k_1\left((1-b) + b\frac{|r|}{avg_{rl}}\right) \end{aligned}$$

and $tf_r(t)$ is the frequency of the token t in string record r, $|r|$ is the number of tokens in r, avg_{rl} is the average number of tokens per record, N is the number of records in the relation R, and n_t is the number of records containing the token t. Also, k_1, k_3 and b are set of independent parameters. The BM25 measure is based on the probabilistic relevance model [177] and it has been shown to outperform the tf-idf cosine similarity in document retrieval [155]. Note that BM25 is an asymmetric measure. When applied to documents, r_1 is the query.

Hidden Markov Model The string matching could be modeled by a discrete Hidden Markov process which has been shown to have better performance than Cosine w/tf-idf in the IR literature on standard IR benchmarks [135]. A similarity function based on Hidden Markov Model (HMM) has high accuracy and low running time for approximate selection on a large number of benchmark data sets [89]. This Markov model consists of only two states where the first state models the tokens that are specific to one particular "String" and the second state models the tokens in "General English", i.e., tokens that are common in many records. A complete description of the model and possible extensions are presented elsewhere [89, 135].

The HMM similarity function accepts two string records r_1 and r_2 from a set of records R

and returns the probability of generating r_1 given r_2 is a similar record:

$$sim_{HMM}(r_1, r_2) = \prod_{t \in \mathbf{r}_1} (a_0 P(t|GE) + a_1 P(t|r_2)) \tag{2.10}$$

where a_0 and $a_1 = 1 - a_0$ are the transition states probabilities of the Markov model and $P(t|GE)$ and $P(t|r_2)$ is given by:

$$P(t|r_2) = \frac{\text{number of times } t \text{ appears in } r_2}{|\mathbf{r}_2|}$$
$$P(t|GE) = \frac{\sum_{r \in R} \text{number of times } t \text{ appears in } r}{\sum_{r \in R} |\mathbf{r}|}$$

Hybrid Measures

Hybrid measures use two similarity functions, one similarity function which is more suitable for short strings (used for comparison of the word tokens), and another similarity measure that compares the full strings using the similarity scores of the word tokens. We consider two hybrid measures, GES and Soft TF/IDF.

GES The generalized edit similarity (GES) which is a modified version of *fuzzy match similarity* [44], takes two strings r_1 and r_2, tokenizes the strings into a set of words and assigns a weight $w(t)$ to each token. GES defines the similarity between the two given strings as a minimum transformation cost required to convert string r_1 to r_2 and is given by:

$$sim_{GES}(r_1, r_2) = 1 - \min\left(\frac{tc(r_1, r_2)}{wt(r_1)}, 1.0\right) \tag{2.11}$$

where $wt(r_1)$ is the sum of weights of all tokens in r_1 and $tc(r_1, r_2)$ is the minimum cost of a sequence of the following transformation operations:

- *token insertion:* inserting a token t in r_1 with cost $w(t) \cdot c_{ins}$ where c_{ins} is an insertion factor constant and is in the range between 0 and 1.
- *token deletion:* deleting a token t from r_1 with cost $w(t)$.

- *token replacement:* replacing a token t_1 by t_2 in r_1 with cost $(1 - sim_{edit}(t_1, t_2)) \cdot w(t)$ where sim_{edit} is the edit-distance between t_1 and t_2.

SoftTFIDF SoftTFIDF is another hybrid measure proposed by Cohen et al. [49], which relies on the normalized *tf-idf* weight of word tokens and can work with any arbitrary similarity function to find the similarity between word tokens. In this measure, the similarity score is defined as follows:

$$sim_{SoftTFIDF}(r_1, r_2) = \sum_{t_1 \in C(\theta, r_1, r_2)} w(t_1, r_1) \cdot w(\arg\max_{t_2 \in r_2}(sim(t_1, t_2)), r_2) \cdot \max_{t_2 \in r_2}(sim(t_1, t_2)) \tag{2.12}$$

where $w(t, r)$ is the normalized *tf-idf* weight of word token t in record r and $C(\theta, r_1, r_2)$ returns a set of tokens $t_1 \in r_1$ such that for $t_2 \in r_2$ we have $sim(t_1, t_2) > \theta$ for some similarity function $sim()$ suitable for comparing word strings, such as the Jaro-Winkler similarity.

2.1.3 Semantic (Language-Based) Similarity Measures

Very often, strings that are syntactically very different (i.e., their sequence of characters or sets of tokens are not similar) have the same meaning or represent the same real-world concept or entity. Several methods have been proposed in the literature to measure the *semantic* similarity of two strings. Such methods, often referred to as *language-based* methods, rely on Natural Language Processing (NLP) techniques to extract and compare the meaningful terms from the strings. Euzenat and Shvaiko [71] classify such methods into *intrinsic* and *extrinsic*. Intrinsic methods perform *linguistic normalization*, which aims at reducing each form of a term to an standardized form. Some such normalization operations are *lemmatization* (e.g., transforming `reviewed` into `review`, and `better` into `good`), *term extraction* (e.g., to recognize `theory paper` is the same as `paper on theory`), and *stopword elimination* (removing words such as `of`, `as`, and `the`). Extrinsic methods on the other hand use external sources such as dictionaries or thesauri to capture semantic similarity between two strings (for example, to recognize that `flammable` and `combustible` are similar in meaning).

There are a wide range of semantic similarity measures in the literature [114, 147, 152, 153, 158, and others]. Their full description is clearly beyond the scope of this chapter and thesis. Here, we only discuss a few examples of (extrinsic) measures. A common lexical resource used in extrinsic methods is WordNet [136]. WordNet consists of a group of *synsets* or sets of synonyms. Each synset represents a concept or a sense of a set of terms, and has a unique identifier. WordNet also includes other relations such as hyponymy/hypernymy (sub-concept/superconcept) and meronymy (*part of* relations). It also provides textual descriptions of the concepts (referred to as *gloss*) that provide definitions and examples. There are other (domain-specific) lexicons that follow the style of Wordnet. For example, the NCI thesaurus is an excellent source of biomedical concepts and terminologies connected with different types of relationships such as synonymy and hyponymy. Another example thesaurus is PhraseNet [129] which extends WordNet by enhancing a synset with its contextual information and refining its relational structure by maintaining only those relations that respect contextual constraints. PhraseNet is thus organized in *Consets*, which capture the underlying lexical concept, and are connected with several semantic relations that respect contextually sensitive lexical information.

A basic measure using WordNet's synset is the *synonymy similarity* [71]:

$$sim_{Synonymy}(r_1, r_2) = \begin{cases} 1 & \text{if } \Sigma(r_1) \cap \Sigma(r_2) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

where $\Sigma(t)$ denotes the set of synset identifiers associated with the term t. This measure basically indicates if the two strings can be used as synonyms (score=1) or not (score=0). A similar measure can be defined for other relationships. For example, the *hyponymy similarity* can be defined as:

$$sim_{Hyponymy}(r_1, r_2) = \begin{cases} 1 & \text{if } \mathcal{H}(r_1) \cap \mathcal{H} \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

where $\mathcal{H}(t)$ denotes the set of hypernyms of the term t. Note that hypernymy is a transitive relationship, therefore finding $\mathcal{H}(t)$ requires traversing the lexicon (WordNet) hierarchy. This

measure basically indicates if the two strings are hyponyms of the same concept (score=1) or not (score=0). An extension of the synonymy similarity is the *cosynonymy similarity* [71]:

$$sim_{Cosynonymy}(r_1, r_2) = \frac{\Sigma(r_1) \cap \Sigma(r_2)}{\Sigma(r_1) \cup \Sigma(r_2)} \tag{2.15}$$

which gives a lower score to strings that have multiple synonyms but only a subset of their set of synsets overlap.

The above measures do not indicate how far the two terms are from each other in the lexicon's hierarchy. Several measures have been proposed to take into account the distance between the two terms or their synsets in the hyponym/hypernym hierarchy. One such measure, known as edge-count (or the *structural topological disimilarity* [71]), counts the number of edges separating the two synsets. Other measures have been proposed based on information theoretic measures. One such measure proposed by Resnik [153] associates each concept (synset) c with a probability of occurrence $p(c)$ of an instance of the concept in a data set. The *information content* of a concept can then be quantified as negative the log likelihood, $-\log p(c)$. A concept c is said to *subsume* concept c' if c' is a hyponym of c. The Resnik similarity is defined as:

$$sim_{Resnik}(r_1, r_2) = \max_{c \in S(r_1, r_2)} [-\log p(c)] \tag{2.16}$$

where $S(r_1, r_2)$ denotes the set of all concepts that subsume both r_1 and r_2. Several alternatives have been proposed, including but not limited to Jiang and Conrath's measure [114] that also uses information content, but takes into account the information content of the two concepts in addition to their most specific subsumer in the taxonomy. Budanitsky and Hirst [37] found that Jiang and Conrath's measure is superior in detection and correction of word spelling errors compared to several others measures. See Navigli [139] for a description of these and other measures. To the best of our knowledge, these measures have not been evaluated for record linkage. The main application of semantic similarity measures however will be on non-classic record linkage (or link discovery). We will show in Chapter 3 how synonymy and hyponymy measures can be used in link discovery.

2.2 Beyond String Matching

In many applications, record linkage requires going beyond just matching string representations of the records. The most common case is when records contain several fields each with different properties (such as type, length, amount of noise, etc.). A common approach is to use different similarity measures for different fields, with weights assigned to (or learned for) each field that correspond to the importance of the field or the accuracy of the measure used for that field. Using the Fellegi-Sunter model, this means finding a set of measures to build the comparison vectors that will minimize the error in matching decisions. One possible solution to this problem relies on using a training set to find good measures and their weights. For example, Bilenko and Mooney [25] present a machine learning approach that is based on learning distance measures for each field, and employing a classifier that uses several diverse measures for each field. More recent proposals include the learning framework of the Active Atlas system [164], and the example-driven approach of Chaudhuri et al. [43] that is based on designing of an *operator tree* that is obtained by composing a few primitive matching operators.

In addition to field comparison techniques (that only use information from the records themselves), it is often possible to use the structure of the records and their relationships to enhance the accuracy of record linkage. For example, if records a_1 and a_2 in one relation co-occur frequently in another relation and similarly records b_1 and b_2 from the same relation occur frequently together in the other relation, then finding that pair $< a_1, b_1 >$ is a match may imply that $< a_2, b_2 >$ is also a match. In some other cases, finding $< a_1, b_1 >$ as a match may imply that $< a_1, b_2 >$ can not be a match. Such techniques are often referred to as *Collective* record linkage (entity resolution) [23] or *Context matchers* [124]. In what follows, we discuss two classes of such techniques. The first class relies on co-occurrence information while the second class is based on graph clustering algorithms to remove inconsistent matches or find missing ones.

2.2.1 Leveraging Structure and Co-occurrence

Bhattacharya and Getoor [22, 23] propose a method called relational clustering for entity resolution (RC-ER), which iteratively creates clusters of matched records. In each iteration, clusters that are deemed most similar are merged. The similarity between two clusters is measured based on matching (strings) of the cluster labels, in addition to matching attributes of their *related* records:

$$sim_{RC-ER}(c_1, c_2) = (1 - \alpha) \times sim_A(c_1, c_2) + \alpha \times sim_R(c_1, c_2) \tag{2.17}$$

where sim_A is the string similarity, sim_R is the *relational* similarity between the related records of the two clusters, and α is a combination weight valued between 0 and 1. What makes this measure different from the previously discussed measures is that its value can change after each iteration, since the cluster labels and their set of related records can change. As an example, records can represent author entities, with their co-authors as their related records. Two author records that are considered non-matches due to dissimilarity of their names as a result of abbreviation in one of the names or spelling errors, can become a match if several of their co-authors are matched in previous iterations of the algorithm, and therefore have a high relational similarity. Extensions of this algorithm are proposed to improve its accuracy and efficiency [23, 24].

Another similar approach is presented by Dong et al. [63]. This approach relies on existence of a dependency graph whose nodes represent similarities between record pairs, and edges represent dependencies between matching decisions. For example, the dependency graph can indicate that the similarity between two author names will affect the similarity (and matching decision) of their co-authors and their publications, and the matching of their publications will affect the matching decision of their corresponding venues (and/or vice versa). In general, the records and their relationships themselves can be represented as a graph. The graph indicates that, for example, if two publications match (maybe based on the fact that all their authors match) then their venues must match as well. Naumann and Herschel [105, 138] refer to the record linkage algorithms that take advantage of such graphs as *graph algorithms*, and present

efficient algorithms to minimize re-comparison of the records by updating the order of comparison each time a matching decision is made. This is based on the observation that, for example, if the publications are compared first, then their venues (that do not match solely based on string similarity) can be matched in the first comparison, otherwise they need to be compared again after their publications match.

2.2.2 Clustering for Record Linkage[1]

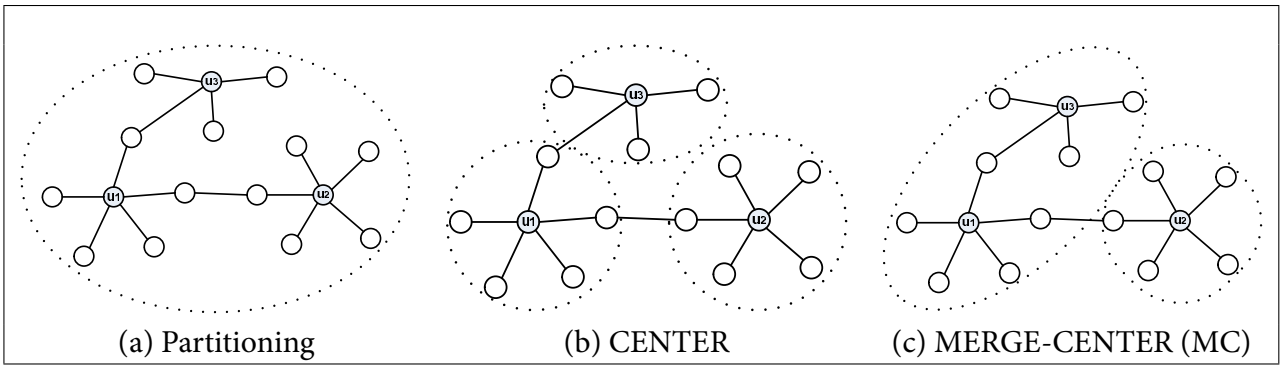
Another known problem is the existence of inconsistent matching results, e.g., a record a matches with two records b and c but b and c do not match. To resolve such inconsistencies we can partition the graph of records. Consider the graph $G(U, V)$ where each node $u \in U$ represents a record and each edge $(u, v) \in V$ connects two nodes u and v only if their corresponding records match. We call this graph the *linkage graph*. The goal of the clustering algorithm is then to cluster the linkage graph in a way that the number of intra-cluster edges are maximized while minimizing edges between clusters. This problem is proven to be NP-complete [12]. Several heuristics and approximation algorithms have been proposed. In what follows, we present an overview of all such clustering algorithms in the literature. Appendix B presents the results of our thorough evaluation of these algorithms in record linkage .

Single-pass Algorithms

In this class of algorithms, we do not materialize the linkage graph. In fact, all the algorithms can be efficiently implemented by a single scan of the list of similar record pairs that match, although some require the list to be sorted by similarity score. We only use the graph G to illustrate these techniques. Figure 2.1 illustrates the result of applying these algorithms to a sample linkage graph.

Partitioning (Transitive Closure). The Partitioning algorithm returns each connected component in the graph as a cluster. The algorithm performs clustering by first assigning each

[1] Part of this section has appeared in Proceedings of the VLDB Endowment (PVLDB) [90].



(a) Partitioning (b) CENTER (c) MERGE-CENTER (MC)

Figure 2.1: Illustration of single-pass clustering algorithms

node to its own cluster. Then, the list of matched pairs is scanned once and if two connected nodes are not in the same cluster, their clusters are merged. Figure 2.1(a) shows the result of this algorithm on a sample graph. As shown in this figure, the algorithm may produce big clusters, and the records that are not similar may be put in the same cluster. Partitioning is the common approach used in early record linkage work [104].

CENTER. The CENTER algorithm [100] performs clustering by partitioning the linkage graph into clusters that have a *center*, and all records in each cluster are similar to the center of the cluster. This algorithm requires the list of the matched record pairs to be sorted by decreasing order of similarity scores. The algorithm then performs clustering by a single scan of the sorted list. The first time a node u is in a scanned pair, it is assigned as the center of the cluster. All the subsequent nodes v that are similar to u (i.e., appear in a pair (u, v) in the list) are assigned to the cluster of u and are not considered again (when they appear in another pair in the list). Figure 2.1(b) illustrates how this algorithm clusters a sample graph of records. In this figure, node u_1 is in the first pair in the sorted list of similar records and node u_2 appears in a pair right after all the nodes similar to u_1 are visited, and node u_3 appears after all the nodes similar to u_2 are scanned. As the figure shows, this algorithm could result in more clusters than Partitioning since it assigns to a cluster only those records that are similar to the center of the cluster.

MERGE-CENTER. The MERGE-CENTER algorithm [95] is a simple extension of the CENTER algorithm. It is similar to CENTER, but merges two clusters c_i and c_j whenever a record matched with the *center* node of c_j is in the cluster c_i, i.e., a record that is similar to the center of

the cluster c_i is similar to the center of c_j. This is done by a single scan of the list of the similar records, but keeping track of the records that are already in a cluster. Again, the first time a node u appears in a pair, it is assigned as the center of the cluster. All the subsequent nodes v that appear in a pair (u, v) in the scan and are not assigned to any cluster, are assigned to the cluster of u, and are not assigned as the center of any other cluster. Whenever a pair (u, v') is encountered such that v' is already in another cluster, the cluster of u is merged with the cluster of v'. Figure 2.1(c) shows the clustering of the sample linkage graph by this algorithm, assumimg that the nodes u_1, u_2 and u_3 appear first in the sorted list of similar records that are assigned as the center of a cluster. As this figure shows, MERGE-CENTER creates fewer clusters for the sample graph than the CENTER algorithm, but more than the Partitioning algorithm.

Star Clustering Algorithm

This algorithm is motivated by the fact that high-quality clusters can be obtained from a weighted linkage graph by: (1) removing edges with weight less than a threshold θ, and (2) finding a *minimum clique cover* with maximal cliques on the resulting graph. This approach ensures that all the nodes in one cluster have the desired degree of similarity (i.e., similarity score above the threshold θ). Furthermore, minimal clique covers with maximal cliques allow vertices to belong to several clusters, which is a desirable feature in many applications. Unfortunately this approach is computationally intractable. The clique cover problem is NP-complete and does not even admit polynomial-time approximation algorithms [166]. The Star clustering algorithm [10] is proposed as a way to cover the graph by *dense star-shaped subgraphs* instead. Aslam et al. [10] prove several interesting accuracy and efficiency properties, and evaluate the algorithm for document clustering in information retrieval. The Star algorithm performs clustering on a weighted linkage graph $G(U, V)$ as follows:

a. Let each vertex in G be initially unmarked.

b. Calculate the degree of each vertex $u \in U$.

c. Let the highest degree unmarked vertex be a star center, and construct a cluster from the center and its associated vertices. Mark each node in the newly constructed star.

d. Repeat step c until all the nodes are marked.

Note that this algorithm is similar to the single-pass algorithm CENTER, but may produce overlapping (non-disjoint) clusters. Implementing this algorithms requires another scan of the input list of similar records to calculate the degree of each vertex and sort the vertices based on their degrees.

Ricochet family of algorithms

Wijaya and Bressan [176] recently proposed a family of parameter-free graph clustering algorithms called 'Ricochet' due to their analogy that the steps of the algorithm resemble the rippling of stones thrown in a pond. These algorithms perform clustering by alternating between two phases. In the first phase, the seeds of the clusters are specified, which is similar to selecting star centers in the Star algorithm. In the second phase, vertices are assigned to clusters associated with seeds. This phase is similar to the re-assignment phase in the K-means algorithm [109]. Wijaya and Bressan propose four versions of the algorithm. In two of the algorithms, seeds are chosen sequentially one by one, while in the two other algorithms seeds are chosen concurrently. The sequential algorithms produce disjoint clusters, whereas concurrent algorithms may produce overlapping clusters (similar to the Star algorithm). In all four algorithms, a weight is associated with each vertex which is equal to the average weight of their adjacent edges. We briefly describe the four algorithms below.

Sequential Rippling (SR) performs clustering by first sorting the nodes in descending order of their weight (average weight of their adjacent edges). New seeds are chosen one by one from this sorted list. When a new seed is added, vertices are re-assigned to a new cluster if they are closer to the new seed than they were to the seed of their current cluster. If there are no re-assignments, then no new cluster is created. If a cluster is reduced to singleton, it is reassigned to its nearest cluster. The algorithm stops when all nodes are considered.

Balanced Sequential Rippling (BSR) is similar to the sequential rippling in selecting the first seed, and has a similar second phase. However, its first phase differs whereby it chooses the next seed to maximize the ratio of its weight to the sum of its similarity to the seeds of existing clusters. This strategy is employed to select a node with a high weight that is far enough from the other seeds.

Concurrent Rippling (CR) initially marks every vertex as a seed. In each iteration, the algorithm picks for each seed the edge with highest weight. If the edge connects the seed to a vertex that is not a seed, the vertex is assigned to the cluster of the seed. If the vertex is a seed, it is assigned to the cluster of the other seed only if its weight is smaller than the weight of the seed. This iteration (propagation of a ripple) is performed at equal speed for all seeds. That is, the minimum value of the weight of the edges picked in each iteration of the algorithm is found, and all the edges that have a weight above the minimum weight value are processed.

Ordered Concurrent Rippling (OCR) performs clustering similar to concurrent rippling but removes the requirement that the rippling propagates at equal speeds. Therefore this algorithm is relatively more efficient and also could possibly create higher quality clusters by favoring heavy seeds.

Correlation Clustering

Suppose we have a graph G on n nodes, where each edge (u, v) is labeled either $+$ or $-$ depending on whether u and v have been deemed to be similar or different. Correlation clustering, originally defined by Bansal et al. [12], refers to the problem of producing a partition (a clustering) of G that agrees as much as possible with the edge labels. More precisely, correlation clustering solves a maximization problem where the goal is to find a partition that maximizes the number of $+$ edges within clusters and the number of $-$ edges between clusters. Similarly, correlation clustering can also be formulated as a minimization problem where the goal is to minimize the number of $-$ edges inside clusters and the number of $+$ edges between clusters.

Correlation clustering is an *NP-hard* problem [12]. Thus, several attempts have been made

to approximate both the maximization and minimization formulations [12, 42, 57, 163]. Most of them are different ways of approximating its linear programming formulation. For the minimization formulation, Bansal et al. give a constant factor approximation algorithm, called *Cautious*. They also present a result which states that any constant factor approximation for the minimization problem in $\{+,-\}$-graphs can be extended as a constant factor approximation in general weighted graphs. Using a notion of "δ-goodness", the algorithm Cautious expands a cluster associated with an arbitrary node by adding its neighbors that are δ-good into the cluster while removing its neighbors that are δ-bad from the given cluster. Ailon et al. [4] proposed a better approximation scheme for the minimization formulation of correlation clustering. The proposed algorithm *CC-Pivot* is similar to the *CENTER* algorithm except that the edges are processed randomly (and not from a sorted list).

Markov Clustering (MCL)

The Markov Cluster Algorithm (MCL), proposed by Stijn van Dongen [168], is an algorithm based on simulation of (stochastic) flow in graphs. MCL clusters the graph by performing random walks on a graph using a combination of simple algebraic operations on its associated stochastic matrix. Similar to other algorithms described above, it does not require any priori knowledge about an underlying cluster structure. The algorithm is based on a simple intuition that a region with many edges inside forms a cluster and therefore the amount of flow within a cluster is strong. On the other hand, there exist a few edges between such produced regions (clusters) and therefore the amount of flow between such regions (clusters) is weak. Random walks (or flow) within the whole graph are used to strengthen flow where it is already strong (e.g., inside a cluster), and weaken it where it is weak (e.g., between clusters). By continuing with such random walks an underlying cluster structure will eventually become visible. Therefore, such random walks are finally ended when we find regions (clusters) with strong internal flow that are separated by boundaries with hardly any flow.

The flow simulation in the MCL algorithm is as an alternate application of two simple al-

gebraic operations on stochastic matrix associated with the given graph. The first algebraic operation is called *expansion*, which coincides with normal matrix multiplication of a random walk matrix. Expansion models the spreading out of flow as it becomes more homogeneous. The second algebraic operation is called *inflation*, which is a Hadamard (entrywise) power followed by a diagonal scaling of another random walk matrix. Inflation models the contraction of flow, becoming thicker in regions of higher current and thinner in regions of lower current. The sequential application of expansion and inflation causes flow to spread out within natural clusters and evaporate in between different clusters. By varying the inflation parameter of the algorithm, clusterings on different scales of granularity can be found. Therefore, the number of clusters cannot and need not be specified in advance, and the algorithm can be adapted to different contexts.

Cut Clustering

Given a directed graph $G = (U, V)$ with edge capacities $c(u, v) \in \mathbf{Z}^+$, and two vertices s, t, the $s - t$ maximum flow problem is to find a maximum flow path from the source s to the sink t that respects the capacity constraints.[2] Intuitively, if the edges are roads, the max flow problem determines the maximum flow rate of cars between two points. The *max flow-min cut* theorem proven by Ford and Fulkerson [74] states that finding the maximum flow of a network is equivalent to finding the minimum cut that separates s and t. Specifically, this involves finding a non-trivial partition of the vertices into two sets, where s and t are in different sets, such that the cut weight (the sum of edge weights in the cut) is minimal. There are several algorithms proposed for finding the minimum cut of G, including finding augmenting paths by Ford and Fulkerson [74], Edmonds and Karp [66], and Dinic [61], and other variations suited for dense graphs or sparse graphs.

The *Cut-clustering* algorithm proposed by Flake, Tarjan, and Tsioutsiouliklis [73] is based on finding the minimum cut tree. The minimum cut tree of a graph has the property that we can

[2] Undirected graphs are modeled with bi-directional edges.

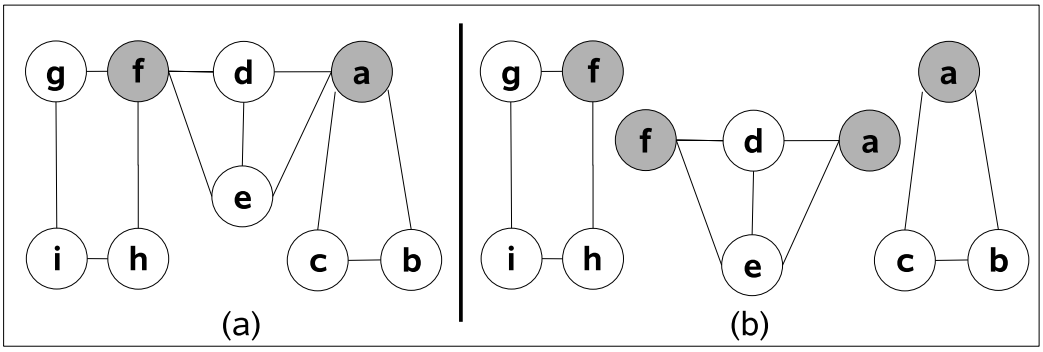


Figure 2.2: (a) Articulation points are shaded, (b) Biconnected components.

find the minimum cut between any two nodes of the graph by inspecting the path that connects the two nodes in the tree. The Cut-clustering algorithm is based on inserting an artificial sink t into the graph and finding the minimum cut tree. Removing the edges in the minimum cut tree results in a set of connected components which are returned as the output clusters of the algorithm.

Articulation Point Clustering

This algorithm is based on a scalable technique for finding articulation points and biconnected components in a graph. Given a graph G, the algorithm identifies all articulation points in G and returns all vertices in each biconnected component as a cluster. An articulation point is a vertex whose removal (together with its incident edges) makes the graph disconnected. A graph is biconnected if it contains no articulation points. A biconnected component of a graph is a maximal biconnected graph. Finding biconnected components of a graph is a well-studied problem that can be performed in linear time [51]. The 'removal'[3] of all articulation points separates the graph into biconnected components. These components are returned as clusters of G. Note that overlapping clusters are produced. Figure 2.2 shows an example. A depth first search traversal is used to find all articulation points and biconnected components in G.

[3]Descriptive terminology. The articulation points are not actually removed. These vertices serve as links between the biconnected components and participate in each incident biconnected component vertex set.

2.3 Entity Identification and Record Linkage

We use the term *entity identification* to refer to the general task of identifying entities (or objects) in a given data source or across multiple sources. Therefore, the entity identification task *may* involve record linkage to identify records that refer to the same entity and avoid duplicates in the identified set of entities, but it also requires resolving schema heterogeneity of the data sources. Some previous work on entity identification over relational databases has proposed solutions that only perform record linkage, based on the assumption that schema heterogeneity of data sources are resolved and a single duplicated database is created that requires identification of duplicate record pairs for entity identification [130]. Subsequently, some work has used the term entity (or object) identification to refer to the classic record linkage task (e.g., [78, 164]).

On the other hand, solving the schema heterogeneity problem while merging records from different sources and performing entity identification is a challenging task, and has received little attention in the record linkage work. The majority of existing record linkage techniques assume that the data resides in a data source with a known given schema. For example, bibliographic information is stored in two relational tables `PaperRefs(id,title,venue,year)` and `Wrote(id,pos,author)` (example from [9]). Each of these tables may contain *dirty* data due to various reasons such as misspellings or different naming conventions. The goal is then to find records that represent the same entity. But it is known in advance (or is easy to discover) that the `id` columns link the two tables, and that if two papers match, their venue must match, etc. Very often in real record linkage scenarios, this is not the case. Part of the difficulty arises from large, undocumented structured sources with schema labels that are not descriptive or schema elements whose usage has changed over time. Many additional challenges arise in matching semistructured or unstructured data sources. In such cases the *records* (and their schema) need to be identified prior to performing record linkage, and the number of schema elements is often large and very heterogeneous. In what follows, we discuss a few of these challenges in more detail along with some of the existing solutions.

2.3.1 Schema Matching and Record Linkage

The *schema matching* problem is the problem of finding schema elements of two or more databases that are semantically related. Schema matching is a very well-studied problem in the database literature with several systems and tools, as well as several (recent) survey articles [62, 150, 159] and books [17, 75]. Based on a classification proposed by Rahm and Bernstein [150], the existing techniques can be classified into two categories: instance-based, where attribute values and their properties are used to find attribute correspondences, and schema-based matching, where only schema elements (labels, data types, etc.) are used for matching. More recent work has also considered a combination of the two approaches (e.g., [165]). However, there has been little work in the past that considers the relationship between schema matching and record linkage, and how they can affect each other.

When schema matching is needed as a part of the entity identification process and prior to record linkage, an all-to-all matching of all the elements may not be required since some schema elements may not play an important role in the record linkage process. This is particularly true for matching techniques that aim at matching attributes (or fields) in the schema even if their instances do not match, based on data properties and distribution of the values for example (cf. [53, 117]). Another important problem in using traditional schema matching approaches occurs when one attribute in a source schema can match multiple attributes in a target schema. Many traditional matching techniques do not allow such matchings. However, in entity identification and for record linkage, multiple matches can in fact be helpful, and dropping one or some of the matches could negatively affect the ability to perform accurate record linkage.

One of the few existing approaches that considers the relationship between schema matching and record linkage is the work of Bilke and Naumann [28]. The authors present a novel schema matching approach based on first performing record linkage without any given schema alignments and based only on field values. Records are treated as flat string values and Cosine similarity with tf-idf (cf. Section 2.1) is used to find matches. Then, the average similarity between the field values in the matched record pairs are calculated using the SoftTFIDF similarity

(cf. Section 2.1) and are used as the matching score for field pairs. A field similarity matrix is built using the matching scores and field matches are found using a bipartite weighted matching algorithm. The algorithm can then use the certain field matches to go back to the record linkage and identify more (or higher-quality) matches, and repeat the algorithm until all or most fields can be matched. Note that this work also assumes 1-to-1 matching between the fields.

Other work in this area addresses the problem of schema matching for record linkage over XML data. In the DogmatiX system [174, 175], this problem is referred to as *description selection*. Schema-based and instance-based description selection methods that use several *heuristics* and *conditions* have been proposed. The schema-based heuristics include for example using data type or the proximity of the nodes in the XML tree to prune candidate descriptions. Examples of instance-based heuristics are pruning attributes that have only one distinct value, or pruning attributes whose values have on average a low inverse-document frequency (IDF) value (occur too frequently). Related to this problem is the work on Approximate XML Joins by Guha et al. [86]. The goal of approximate XML join operation is to identify XML documents that are similar (and therefore likely to represent the same entity or set of entities). The solution proposed by Guha et al. is based on finding the *tree edit distance* of two XML trees, which is based on generalization of the string edit distance (cf. 2.1). Since computing tree edit distance is an expensive operation, efficient algorithms based on filtering and sampling are proposed and evaluated using real and sythetic data.

2.3.2 Entity Identification from Text

Entity extraction or recognition is the problem of identifying entities from unstructured text. A wide variety of entity extraction techniques have been proposed. Recently, there has been an increasing interest in techniques that exploit the existence of a structured database (or dictionary) of entities to improve the accuracy and performance of the entity recognition [2, 3, 40, 45, 50, 171]. Such approaches are most useful for record linkage when an unstructured source (or unstructured portion of a semistructured source) needs to be linked to an structured

source. In addition, record linkage techniques can be used to improve the quality and performance of the extraction.

Another closely related problem is that of providing keyword search over a structured or semistructured database [187]. In this work, an (unstructured) keyword query is "matched" to entities in a structured database, or used to identify tuples in multiple tables that may be relevant to the query. Some approaches assume clean, unambiguous queries that refer to a single database entity. Closer to entity extraction work are approaches that segment the query (keywords) to provide better entity matching to multiple database entities [131]. However, dirty queries containing spelling errors or syntactic differences are generally not handled well by such approaches. To address this, recent work has considered the problem of keyword segmentation in the presence of spelling errors and semantic mis-matches and the problem of incremental segmentation of the (unstructured) query [149].

Recently, we proposed a framework and algorithm for annotating unbounded text streams with entities of a structured database [148]. The algorithm allows one to correlate unstructured and dirty text streams from sources such as emails, chats and blogs, to entities stored in structured databases. In contrast to previous work on entity extraction, our framework performs entity annotation in a completely online fashion. The algorithm continuously extracts important phrases and assigns to them top-k relevant entities. Our framework allows the online annotation algorithm to adapt to changing stream rate by self-adjusting multiple run-time parameters to reduce or improve the quality of annotation for fast or slow streams, respectively. The framework also allows the annotation algorithm to incorporate query feedback to learn preferences and personalize the annotation depending on the application domain.

Work in web search and information retrieval has also addressed several problems related to segmenting queries or longer pieces of text. Examples include Query by Document (QBD) [185], and Yahoo! Term Extraction API which is part of Yahoo! Search Web Services[4]. Yahoo! Term Extraction extracts (segments) key phrases from text. Our online annotation framework

[4] `http://developer.yahoo.com/search/`

augments this capability by considering the relevance of a segmented phrase to the entities in a structured database and use this information to achieve a better segmentation. The QBD approach also does segmentation independent of the database, but then ranks the segments by querying the target data (where the end goal is quite different from ours - to rank related documents).

Another related and active area of research is that of semantic annotations in knowledge management (KM). Uren et al. [167] present a survey of existing manual and (semi-)automatic annotation and entity extraction techniques. As an example, KnowItAll [70] performs unsupervised named-entity extraction from the Web, based on specification of patterns by the user, e.g., extracting city names by finding occurrences of the phrase "cities such as …" in web documents. AktiveDoc[5] performs annotation on-the-fly while reading or editing documents, but, to the best of our knowledge, does not allow any errors.

2.4 Record Linkage Tools and Systems

Research in data management, machine learning, and statistics has resulted in a large number of research prototypes and commercial record linkage systems. This large number is in part due to the difficulty of the record linkage problem, and that existing systems are often designed for specific domains and applications. They may fail when it comes to dealing with new applications or the need to scale to very large and heterogeneous data sources. A common observation is that there is no single system capable of handling linkage problems across different domains and applications. For some applications, there are highly specialized systems that perform extremely well in certain scenarios. For example, there are several systems and even companies specialized in linking people names, or finding duplicate address records. Such systems often rely on domain knowledge, for example the existence of a large repository of domain information that is manually maintained and proprietary.

[5] http://nlp.shef.ac.uk/wig/aktivedoc.htm

Table 2.1: String matching libraries

Library	Language	Supported Measures
SimMetrics [212]	Java	Levenshtein, Jaro, Jaro-Winkler, Smith-Waterman, Needleman-Wunch, Monge-Elkan Gotoh, Matching coefficient, Jaccard, Dice coefficient, Cityblocks, Euclidean, Cosine, Overlap, Soundex
SecondString [211]	Java	n-grams, Levenshtein, Jaro, Jaro-Winkler, Needleman-Wunch, Monge-Elkan, Jaccard, Cosine w/tf-idf
AlignAPI [190]	Java	n-grams, Levenshtein, Jaro, Jaro-Winkler, Needleman-Wunch, Smoa
SimPack [213]	Java	Levenshtein, Jaccard, Dice coefficient, Cosine w/tf-idf, Cityblocks, Euclidean, Cosine, Overlap, Resnik, Lin, Leacok-Chodorow, Edge count, Wu-Palmer
WordNet::Similarity [221]	Perl	Resnik, Jiang-Conrath, Lin, Leacok-Chodorow, Hirst-St.Onge, Edge count, Wu-Palmer, Extended Gloss Overlap, Vector on gloss
FLAMINGO Package [200]	C++	Levenshtein, Jaccard, Cosine w/tf-idf, Dice
SimString [145]	C++, Python, Ruby	Cosine w/tf-idf, Jaccard, Dice, Overlap
pylevenshtein [208]	C, Python	Levenshtein
difflib [198]	Python	Ratio of matches according to an extension of the Gestalt pattern matching [151]

Table 2.1 shows some of the most widely-used record linkage and matching libraries that implement a wide range of string matching-based techniques, some of which are discussed in Section 2.1. There is clearly a lack of efficient and stable implementations of similarity measures in most languages. Very often, programmers resort to using other similar capabilities in existing libraries and implement a custom string similarity measures to address their needs. Using an extension of the Gestalt pattern matching [151] in Python's difflib library [198] is an example of such an approach. The Java libraries provide the most complete set of similarity measures. However, there is only one library that supports both string similarity and semantic similarity measures, and this implements only a limited set of each type of measure.

Table 2.2 shows a list of some record linkage systems and research prototypes, along with

their supported input format, linkage techniques, the domain they are designed for (or evaluated in), and highlights of their features. Given the focus of this thesis on generic domain-independent techniques, we have only included such systems, although as mentioned earlier, there is a large number of domain-specific record linkage systems and libraries. The TAILOR system [67] includes an implementation of most of the early work on probabilistic record linkage. Cohen's WHIRL is the first to take advantage of efficient IR-based indexing methods along with the Cosine similarity with tf-idf to improve the efficiency. The focus of most of the recent work is mainly on the efficiency of the record linkage operation. The Febrl system [46], initially designed for record linkage in the biomedical domain, includes a wide range of fast blocking and indexing techniques. The system has been used for thorough evaluation of indexing techniques [47] and has resulted in new blocking algorithms [55].

Two very recent efforts have developed systems for linking Web data. The Silk framework [36] and LIMES [143] both take as input RDF data from Web sources and a description of linkage rules (or *linkage specifications*), and return links along with score or confidence level as output. Silk implements an efficient multi-dimensional blocking technique to speed up matching when several dimensions (or attributes) are involved in the linkage. LIMES provides a technique for speeding up metric similarity measures by taking advantage of the triangle inequality between the similarity values (i.e., assuming that $sim(x, z) \leq sim(x, y) + sim(y, z)$, which is not the case for many string similarity measures described in Section 2.1.2). Both systems are under active development, and include graphical user interfaces (UI) and learning strategies to learn linkage rules based on a set of manually labeled reference links. These systems perform the linkage process over the Web, and therefore are suitable for applications that require discovery of links on the fly and over the Web. They will not directly assist data publishers that have data residing in relational backends, which is the case for a large number of Web sources. They also have only a limited set of string and semantic similarity measures discussed earlier in this chapter. Having limited input type, limited set of similarity measures, and not having the ability to easily extend the features of the systems are common problems in all existing systems.

Table 2.2: Some generic record linkage systems

System	Input Type	Availability	Supported Techniques	Highlight of Features
WHIRL [48]	Text data in DBMS	Free for academic and research use	Word-based Cosine with tf-idf measure, along with A* search for query processing	Using IR indexing methods to improve efficiency
TAILOR [67]	DBMS	Not available	Probabilistic Record Linkage (e.g., EM-based Model), Blocking algorithms, 5 string similarity measures	Several Linkage Models, Machine learning approaches
D-Dupe [26]	Custom text format representing a graph	Free for noncommercial use	10 string similarity measures, 4 relational similarity measures (using co-occurrence), blocking methods to improve efficiency	Relational measures using co-occurrence, GUI
Febrl [46]	Comma Separated Values (CSV) and other text-based formats	Free, open source	More than 20 string similarity measures, and 10 indexing (blocking) techniques	GUI, A wide range of blocking techniques to boost performance
Silk [36]	RDF	Free, open-source (Apache Software License)	Several string normalization functions, 5 string similarity measures, similarity measures for numbers, date & time, and geographical coordinates, several aggregation operators, multi-dimensional blocking for efficiency	Free and open source, Several extensions including a MapReduce-based implementation, GUI and workbench, and support for learning linkage rules. Under active development
LIMES [143]	RDF, CSV	Free, open source (License unclear)	All the string similarity measures in SimMetrics (cf. Table 2.1) that are a *metric* (i.e., *triangle inequality* holds for them)	Efficient implementation taking advantage of the triangle inequality in metric spaces

2.5 Summary and Concluding Remarks

In this chapter, we have only discussed a subset of the work on record linkage, which has been and still is a very active research area in several communities. Elmagarmid et al. [68] provide an excellent overview of the early work on record linkage and a more complete overview of similarity measures and record linkage algorithms. Naumann and Herschel [138] also provide an overview of record linkage measures and algorithms with focus on the work in the data management community and some more recent work not covered by Elmagarmid et al. including graph-based and clustering-based algorithms, and evaluation methodologies. Köpcke and Rahm [123] provide a comparative analysis of more recent work on record linkage, by classifying methods based on their 1) input entity types 2) indexing strategy 3) type of matching (string or semantic) 4) strategy to combine several matchers, and 5) if and how they use training data. Wölger et al. [184] present an overview of the existing tools and frameworks for record matching with focus on Web data linking approaches.

Euzenat and Shivako [71] present a comprehensive and detailed description of several matching techniques including many string matching techniques discussed earlier in this chapter, in addition to many graph-based matching algorithms but in the context of *ontology matching* (or alignment). The goal of the ontology matching process is to identify correspondences between concepts in ontologies, which is different from the record linkage problem, although the same or similar techniques can be used for both problems. Similarly, as mentioned in Section 2.3, record linkage methods have been proposed as a part of an entity identification algorithm. The term *identity resolution* has also been used often interchangeably with record linkage, although it is mostly used to refer to the task of resolving identities of people and organizations as a part of an intelligence process [115].

Despite all the work in this area, many problems still remain, and new problems arise as the scale of data increases in data warehouses and on the Web. As new algorithms and techniques are developed, a major problem is the evaluation of the systems in order to verify superiority of the new proposals and their performance in different domains. As our evaluations in this thesis

also reveal, there is still a gap between the work on accuracy of record linkage, and the work on efficiency. In addition, due to the difficult nature of the problem, there is a need for approximate and probabilistic query processing techniques that accommodate existence of uncertain links or duplicated data. Our recent work [95] presents an overview of related work in this area, and a framework and algorithms for creating probabilistic databases out of dirty duplicated data.

In this thesis, we focus on creating new frameworks that enhance the usability of string similarity measures in publishing Web data and linking it to existing web sources. Our solutions consider both scalability and accuracy as important requirements.

Chapter 3

Link Discovery over Relational Data[1]

3.1 Introduction

From small research groups to large organizations, there has been tremendous effort in the last few years in publishing data online so that it is accessible to users. These efforts have been widely successful across a number of domains and have resulted in a proliferation of online sources. In the field of biology, there were 1,330 major online molecular databases at the beginning of 2011, which is 96 more than a year earlier [77]. In the field of medicine almost every major hospital now has its own database of patient visits and clinical trials. In the Linking Open Data (LOD) community project at W3C, the number of published RDF triples has grown from 500 million in May 2007 to over 30 billion triples in September 2011 [32].

While publishing data is now easier than ever, attempts to establish semantic relationships between different published data sources has been less successful. In this chapter, we provide solutions to help turn silos of data into rich linked open data. Consider a biologist interested in a specific gene. It is not enough for the biologist to search for the gene, even using a robust search that accommodates for aliases and errors in the representation of data which are very

[1]Part of this work has appeared in Proceedings of the 18th ACM Conference on Information and Knowledge Management [94]. ©2009 Association for Computing Machinery, Inc. The system has been demonstrated at the VLDB 2009 conference [97].

common in web repositories. Both errors and aliases are domain specific so the biologist may have to try several *approximate search* methods to find one best for her domain. Furthermore, she may also want to find information about proteins or genetic disorders that are known to be related to this gene. Again, the search for this semantically related information must be tolerant of aliasing and errors, and yet must be tailored to the specific semantic relationships the user wishes to find.

What users need is automated support for creating referential links between data that reside in different sources and that are semantically related. Such links would provide a biologist with the ability to start from a gene and directly navigate to its protein and related genetic diseases, even through sources with no direct connection and which may use different naming conventions and different representations for information. Of course, discovering such links requires the use of both approximate matching (to overcome syntactic representational differences and errors) and semantic matching (to find specific semantic relationships). These two types of matching must be used in concert to accommodate for the tremendous heterogeneity found in web repositories.

In spite of their importance, research in discovering such semantic links has mainly focused on the classic record linkage problem as described in Chapter 2, i.e., the identification of records that represent the same *real-world* entity. Yet, the general problem investigated here considers links between entities that are not necessarily identical, although they are semantically related (e.g., genes related with their corresponding proteins, medical treatments related with their corresponding clinical trials[1]). The importance of discovering such links is also highlighted by the recent efforts of the LOD project, where a number of tools and frameworks have been developed that allow the generation and publication of linked data from relational databases. Examples of such frameworks include D2RQ [35], Triplify [11] and OpenLink's Virtuoso [69]. Although these frameworks simplify the process of publishing linked data, the number of links between

[1]Note that as mentioned in Section 1.2, we rely on the existence of a notion of *similarity* (or *distance*) between the records that the user can use to define semantic relatedness. Studying how different types of semantic relatedness between the records can be modeled using record matching functions is beyond the scope of this chapter and thesis.

existing LOD data sources is two orders of magnitude less than the number of base data triples. The majority of the links are either a result of existing links in the relational data (e.g., links between two entities that are both derived from a Wikipedia page, both having the same URL), or a laborious implementation of a semi-automatic and domain-specific linkage algorithm.

For typical users, this means they must experiment with a myriad of different link discovery methods to find one that suits their needs. Our goal in this chapter is to develop a generic and extensible framework for integrating link discovery methods. Our aim is to facilitate experimentation and help users find and combine the link discovery methods that will work best for their application domain, and work over their exisiting relational sources. To ease experimentation, we use a declarative framework that permits the interleaving of standard data manipulation operators with link discovery.

Our framework permits the discovery of links within and between relational sources. We introduce LinQL, an extension of SQL that integrates querying with link discovery methods. Our implementation includes a variety of native link discovery methods and is extensible to additional methods, written in SQL or as user-defined functions (UDF). This permits users to interleave declarative queries with interesting combinations of link discovery requests. The link discovery methods may be syntactic (approximate match or similarity functions), semantic (using ontologies or dictionaries to find specific semantic relationships), or a combination of both.

In this chapter, we show that by integrating *ad hoc* querying and a rich collection of link discovery methods, our framework supports rapid prototyping, testing and comparison of link discovery methods. A common way to use our framework would be to declaratively specify a portion of the data of interest (over which accuracy can be assessed) and to invoke one or more link discovery methods. The results can be evaluated by a user or automated technique, and the specification of the link method interactively refined to produce better results. Here, we assume that the data resides in a well-designed relational database, and the users are able to identify the attributes that can be used to specify the linkage requirements. Automatic discovery of such attributes and linkage over non-relational sources are discussed in the next two chapters.

Often, link discovery algorithms are implemented using programming languages like Java or C by third-party developers, and are automatically invoked with arguments defined through the declarative specification. For data publishers these programs act as *black-boxes* that sit outside the data publishing framework and whose modification requires the help of these developers. We address these shortcomings by leveraging native SQL implementations for a number of link discovery algorithms [84, 89]. Our approach has several advantages including the ability to: (a) easily implement this framework on existing relational data sources with minimum effort and without any need for externally written program code; (b) take advantage of the underlying DBMS optimizations in the query engine while evaluating the SQL implementations of the link finding algorithms; and (c) use specific efficiency and functionality enhancements to improve the efficiency of these algorithms.

We show how our declarative invocation of linkage methods permits users to tune link methods and their performance. The native support for methods permits customization where domain knowledge is available. We give examples where domain knowledge can be specified in the database and used to greatly enhance the performance of the discovery process.

Finally, we discuss the implementation of our framework along with several link discovery algorithms on a commercial database engine. We describe a case study of how our framework can be used to discover links over real clinical trial data draw from a number of disparate web sources.

The rest of this chapter is organized as follows. Section 3.2 introduces our running example, while Section 3.3 describes how links between data sources can be specified declaratively. Section 3.4 presents the algorithms for translating the link specifications into SQL queries. Our experimental study is described in Section 3.5. Section 3.6 highlights related work and we conclude in Section 3.7.

trial	*cond*	*inter*	*loc*	*city*	*pub*
NCT00336362	Beta-Thalassemia	Hydroxyurea	Columbia University	New York	14988152
NCT00579111	Hematologic Diseases	Campath	Texas Children's Hospital	Austin	3058228

(a) Clinical trials (*CT*)

visitid	*diag*	*prescr*	*location*
VID770	Thalassaemia	Hydroxyura	Texas Hospital
VID777	PCV	Hydroxycarbamide	Westchester Med. Ctr

(b) Patient visit (*PV*)

name
Thalassemia
Blood_Disorders

(c) DBpedia Disease (*DBPD*)

name
Alemtuzumab
Hydroxyurea

(d) DBpedia Drug (*DBPG*)

Figure 3.1: Sample relations

3.2 Motivating Example

Through out this chapter (and in our case study in Section 3.5), we use an example from the health care domain drawn from a set of real-world data sources. One of our sources is a clinical trials database which includes the sample relation in Figure 3.1(a). For each trial, the *CT* relation stores its identifier *trial*, the *cond*ition considered, the suggested *inter*vention, as well as the *loc*ation, *city*, and related *pub*lication. Another source stores patient electronic medical records (EMR) and includes a patient visit relation *PV* (Figure 3.1(b)), which stores for each patient visit its identifier *visitid*, the *diag*nosis, recommended *prescr*iption, and the *location* of the visit. Finally, we consider a web source extracted from DBpedia (or Wikipedia) which stores information about drugs and diseases and includes the *DBPD* and *DBPG* relations (Figure 3.1(c) and (d)) that store the *name* of diseases and drugs in DBpedia, respectively.

We now describe briefly some types of links data publishers may like to discover between these sources. For the *CT* and *PV* relations, we note that the *cond*ition column in the *CT* relation is semantically related and can be linked to the *diag*nosis column in the *PV* relation. Such links may be useful to clinicians since they associate a patient's condition with related clinical trials, and might be used to suggest alternative drugs or interventions. In Figure 3.1, patient visit "VID770" with diagnosis "Thalassaemia" in the *PV* relation should be linked to the trial "NCT00579111" with condition "Hematologic Diseases" since "Thalass*a*emia" is a different representation of "Thalassemia" and according to the NCI medical thesaurus "Thalassemia" *is a type of* "Hematologic Diseases". As this example illustrates, a clinician may be interested

in not only *same-as* relationships, but also hyponym relationships such as *type-of*. Similarly, note that the *inter*vention column in the *CT* relation can be linked to the *prescr*iption column in the *PV* relation. Such links can provide evidence for the relevance and effectiveness of a drug for a particular condition. For example, both patient visits in Figure 3.1(b) should link to trial "NCT00336362" (Figure 3.1(a)) based on the fact that "hydroxycarbamide", "Hydroxyura" and "Hydroxyurea" all refer to the same drug.

Additional links are possible if one considers the existence of links between the locations of patients and the presence of clinical trials in these locations. As an example, "Westchester Med. Ctr" from visit "VID777" could link to "Columbia University" based on the geographical information that both locations are in the New York state. Another interesting link discovery scenario arises when a user who is interested in a particular trial, wants to find other related trials based on certain criteria, e.g., the similarity of the title and authors of the trials' corresponding publications. Obviously, to be effective, links should be tolerant of errors and differences in the data, such as typos or abbreviation differences.

Data publishers often build online web-accessible views of their data. In such settings, they often want to provide links between their data and those in other online web sources. As an example, a web source of clinical trials requires links to other web sources related to the trials like, say, the DBpedia or YAGO sources. In our sample relations, the above example translates into finding links between the *cond* and *inter* columns of *CT* and the *name* column of the *DBPD* and *DBPG* relations, respectively. The online trials data source can link the condition "Hematologic Diseases" to DBpedia resource (or Wikipedia page) on "Blood_Disorders", and link the intervention "Campath" to DBpedia resource "Alemtuzumab" using the semantic knowledge that "Campath" is the brand name for the chemical name "Alemtuzumab".

3.3 The LinQL Language

In this section, we introduce the LinQL declarative link specification language. We discuss the characteristics of the link finding primitives required by a flexible framework that is capable of effective link discovery in many real world scenarios, and show how LinQL supports such primitives. Although the linkage specification elements we discuss below are expressible in a number of different languages and notations (e.g., RDF/XML, N3, NTriples), in our framework (and implementation) we chose an SQL-like syntax.

A *link specification*, or *linkspec* for short, defines the conditions that two given values must satisfy before a link can be established between them. In more detail, a linkspec is defined using the grammar of Figure 3.2 (only a subset of the grammar shown). As shown in the figure, a CREATE LINKSPEC statement defines a new linkspec and accepts as parameters the name of the linkspec and the names of the relation columns whose values need to be linked. To create the links, our framework provides several *native* (or *built-in*) methods including synonym, hyponym, and a variety of string similarity functions (see more details on the native methods in the following sections). Such native methods can be used as such or they can be customized by setting their parameters.

3.3.1 Examples

Example 1 *A common string similarity measure that has been shown to have good accuracy and efficiency is the weighted-Jaccard measure (described in Section 2.1.2). A user can create a link specification using this measure by setting the parameters used in the similarity computation. For example, she may set the threshold parameter to* 0.5*, the length of the q-gram to* 2*, and the maximum string length to* 50*, creating the following link specification*

```
CREATE LINKSPEC myJaccard1
AS weightedJaccard (0.5, 2, 50).
```

```
linkspec_stmt:= CREATE LINKSPEC linkspec_name
                AS link_method opt_limit;

linkindex_stmt:= CREATE LINKINDEX opt_idx_args
                 linkindex_name ON table(col)
                 USING native_method;

link_method:= native_method | link_clause_expr | UDF;

native_method:= ( synonym | hyponym | string_method ) opt_args;

string_method:= jaccard | weightedJaccard | cosine | bm25 | hmm | ··· ;

link_clause_expr:= link_clause AND link_clause_expr
                 | link_clause OR link_clause_expr
                 | link_clause;

link_clause:= LINK source WITH target
              USING link_terminal opt_limit;

link_terminal:= native_method | UDF opt_args | linkspec_name;

opt_limit:= LINKLIMIT number;
```

Figure 3.2: The LinQL grammar (Version 1.1)

Now this link specification can be used as a join predicate in queries by any user. Notice that this specification does not indicate processing constraints.

A link specification can also be defined in terms of *link clause expressions*. Link clause expressions are boolean combinations of link clauses, where each link clause is semantically a boolean condition on two columns and is specified using either (a) a native method; (b) a user-defined function (UDF); or (c) a previously defined linkspec.

Example 2 *Consider a setting in which a link between two values is established if a semantic relationship (e.g., synonym or hyponym) exists between these values in an ontology. This scenario commonly occurs in a number of domains, including healthcare, where sources are free to use their own local vocabularies (e.g., diagnosis and drug names) as long as these vocabularies can eventually be matched through a commonly accepted ontology (e.g., the NCI thesaurus). Assume that the*

ontology is stored in table `ont` *with concept IDs in column* `cid` *and the terms in column* `term`.

The following linkspec illustrates the power of link clauses by creating a link between two values if their corresponding values in an ontology are used to match individual values to corresponding values in the ontology, while the synonym native linkspec is used to test for the synonymy of the ontology terms.

```
CREATE LINKSPEC mixmatch
AS LINK src WITH tgt
   USING synonym(ont,cid,term) LINKLIMIT 10
   AND
   LINK src WITH ont.term
   USING myJaccard1 LINKLIMIT 10
   AND
   LINK ont.term WITH tgt
   USING myJaccard2 LINKLIMIT 10;
```

Clearly, the links between values are not necessarily one-to-one and, in general, a value from one relation can be linked to more than one values from a second relation. For example, it is common for drugs to have more than one name. Therefore, while a drug appears as *"aspirin"* in one relation it might appear as *"acetylsalicylic acid"* or *"ASA"* in another. When multiple such links are possible, users often want to limit the number of such links and only consider k results, or the *top-k* where ordering is possible. The LINKLIMIT essentially specifies the value of this *k* parameter.

Example 3 *In the previous example, while defining the mixmatch linkspec,* LINKLIMIT *is set equal to 10, for all three link clauses. Hence, only the top 10 links are considered in each method.*

The previous examples consider the *local* version of the LINKLIMIT construct which is associated with a particular clause. The LinQL grammar also includes a *global* LINKLIMIT construct

which is associated with the whole `CREATE LINKSPEC` statement which can be thought of as a post-processing filter of the links returned by the linkspec methods used in the statement.

We conclude the presentation of LinQL by introducing Boolean valued user-defined functions (UDFs). The primary difference between using a UDF versus a native linkspec method is that the UDF allows only simple substitution-based rewriting of the query, whereas the native linkspec uses a non-trivial rewriting of the query into SQL (see next section on the translation of LinQL to SQL). The ability to use UDFs in link specification is provided for extensibility to non-SQL-based linking functions. Of course, native implementations have numerous advantages. Using native implementations, the query optimizer can optimize the processing of link specifications yielding very efficient execution times. Furthermore, declarative implementations of the methods within a relational engine permit great extensibility as described in the following section.

Example 4 *Suppose a user writes a UDF that implements his own similarity function. This UDF,* `myLinkUDF(thr, delC, insC, subC)` *returns true only if the edit similarity the values on which it is applied is above the threshold value* `thr` *(where* `delC`*,* `insC` *and* `subC` *are the costs of delete, insert and substitute operations, respectively). Then, the following linkspec can use that UDF as follows*

```
CREATE LINKSPEC myLink
AS myLinkUDF(0.5, 1, 1, 1).
```

In the previous paragraphs, we looked at how linkspecs are defined. We now show how linkspecs are used inside queries.

Example 5 *Suppose we want to find the tuples in the PV relation for which there is a link with the condition in the CT relation, using the native weightedJaccard linkspec with default parameter values. Then, the following query can be used.*

```
SELECT PV.*, CT.*
FROM   visit PV, trial CT
```

```
WHERE  PV.visitid = 1234 AND
       CT.city='New York' AND
       PV.diag = CT.cond
       LINK PV.diag WITH CT.cond
       USING weightedJaccard LINKLIMIT 10
```

Notice that the linkspec here is essentially defined inline. For more complex linkspecs, or for situations where the same linkspec is used multiple times by one or more users, the query can refer to a previously defined linkspec in a similar fashion. This is a way for a DBA to provide a set of specifications for methods using the best parameter settings for different domains, making these methods more accessible to less expert users who may not know how to set the parameters. For example, in the query above we can use the `mixmatch` *linkspec instead of the weightedJaccard, as follows:*

```
SELECT PV.*, CT.*
FROM   visit PV, trial CT
WHERE  PV.visitid = 1234 AND
       CT.city='New York' AND
       PV.diag = CT.cond
       LINK PV.diag WITH CT.cond
       USING mixmatch LINKLIMIT 10
```

3.3.2 Native Link Methods

In what follows, we present the currently supported native methods, including our reasons for including them in the initial implementation.

Approximate String Matching Specification

String data is prone to several types of inconsistencies and errors including typos, spelling mistakes, use of abbreviations or different conventions. Therefore, finding similar strings, or ap-

proximate string matching (or apprioxmate join), is an important feature of an (online) link discovery framework. Approximate string matching is performed based on a similarity function $sim()$ that quantifies the amount of *closeness* (as opposed to *distance*) between two strings. A similarity threshold θ is set by the user to specify that there is a link from the base record to the target record if their similarity score, returned by function $sim()$, is above θ. The right value of the threshold depends on the characteristics of the data set, the similarity function, and the application. The user can find the optimal value of the threshold for each application by trying different thresholds and manually evaluating the results.

There exists a variety of similarity functions for string data in the literature. The performance of a similarity function usually depends on the characteristics of data, such as length of the strings, and the type of errors and inconsistencies present in the data. As stated earlier, in our framework we are interested in algorithms that are fully expressible in SQL (the benefits of which are well-known [84]). There are additional benefits of this choice for our application. Specifically, the use of SQL as an implementation language for our methods permits on-the-fly calculations of the similarity scores that can be enhanced dynamically to increase the functionality of the matching algorithm by relying on characteristics of the domain of the source and target relations.

A popular class of string similarity functions is based on tokenization of the strings into q-grams, i.e., substrings of length q of the strings. By using q-gram tokens, we can treat strings as sets of tokens and use a set similarity measure as the measure of similarity between the two strings. Furthermore, q-gram generation, storage and set similarity computation can all be done in SQL. This makes the following class of functions suitable for our framework.

• Size of the *intersection* of the two sets of q-grams, i.e., the number of common q-grams in the two strings.

• *Jaccard similarity* of the two sets of q-grams which is the size of the intersection set over the size of the union, i.e., the percentage of common q-gram tokens between the two strings.

• Weighted version of the above measures. The weight of each q-gram is associated with its

commonality in the base (or target or both) data sources. The higher the weight of a q-gram, the more important the q-gram is. For example, when matching diagnosis across medical sources, q-grams for commonly occurring strings like "Disorder" or "Disease" should have low weights so that the value of the similarity function for the strings "Coagulation Disorder" and "Phonation Disorder" is small, compared to that for the strings "Coagulation Disorder" and "Coagulation Disease".

As described in Section 2.1.2, there are several other string similarity measures including but not limited to Edit-Similarity, Jaro, Jaro-Winkler, SoftTFIDF, Generalized Edit similarity, and methods derived from relevance score functions for documents in information retrieval, namely Cosine with tf-idf, Okapi-BM25, Language Modeling and Hidden Markov Models. Some of these functions can be implemented in SQL and some others can only be implemented using a UDF. However, we focus only on the above q-gram based measures based on their better accuracy and efficiency and also their flexibility for functionality and efficiency enhancements, as discussed below.

Token Weight Assignment: To assign weights to q-gram tokens, we use an approach inspired by the Inverse Document Frequency (IDF) metric in information retrieval. IDF weights reflect the commonality of tokens in documents with tokens that occur more frequently in the documents having less weight. So, by analyzing (offline) the q-gram token frequency, we assign less weight to common tokens like "Disorder" or "Cancer" in a medical source. As a functionality enhancement, we also let the user manually specify in a user-defined table the weights of some tokens. These weights override the automatically assigned (IDF-based) weights for these tokens. Manual weight-assignment is useful in applications where the user has prior knowledge about the importance of some tokens. For example, when matching diagnosis across sources, the user knows that often the use of numbers plays a more important role in the diagnosis than the name of the disease itself. So, by assigning a very low (or negative) weight to numbers, wrong matches between highly similar strings like "Type 1 Diabetes" and "Type 2 Diabetes" can be avoided. Similarly, when matching conditions (e.g., "Diabetes", "Cancer") from an on-

line source such as WebMD to their corresponding entries in, say, Wikipedia, the conditions in Wikipedia might include the term "(disease)" to disambiguate the disease from other terms with the same name (e.g., "Cancer" has close to seven entries in Wikipedia, in addition to the one for the disease, including one for astrology and one for the constellation). Knowing this, the user can adjust the weight of the otherwise common token "disease" to increase the likelihood of a correct link.

Scalability Although, we do not address explicitly the efficiency of the string matching implementation, our similarity predicates can be used along with several existing scalable indexing and hashing techniques. Examples of such techniques include the indexing algorithms of [15], the *Weighted Enumeration* (WtEnum) signature generation algorithm [8] and *Locality Sensitive Hashing* [106].

Semantic Matching Specification

Link discovery between values often requires the use of domain knowledge. In a number of domains, there are existing, commonly accepted, semantic knowledge bases that can be used to this end. In domains where such semantic knowledge is not available, users often manually define and maintain their own knowledge bases.

A common type of such semantic knowledge is an ontology. In the health care domain, well-known ontologies such as the NCI thesaurus are widely used and encapsulate a number of diverse relationship types between their recorded medical terms, including, synonymy, hyponymy/hypernymy, etc. Such relationship types can be conveniently represented in the relational model and (recursive) SQL queries can be used to test whether two values are associated with a relationship of a certain type [119]. Therefore, semantic knowledge in the form of ontologies can be seamlessly incorporated in our framework and used for the discovery of links. So, while considering links between two sources, semantic knowledge can be used to link a diagnosis on "Pineoblastoma" to one on "PNET of the Pineal Gland", since the two terms are synonyms of each other. Similarly, a diagnosis on "Brain Neoplasm" can be potentially linked

Algorithm 1: The LinQL2SQL Algorithm

input : LinQL query L
output: SQL query Q

1 $lce \leftarrow$ extract link clause expr. from L;
2 $Q_{base} \leftarrow L - lce$;
3 $Q \leftarrow$ LinkClauseExpr2SQL(Q_{base}, lce);
4 **return** Q;

with both of the previous diagnoses, since the latter term is a hypernym of the former terms. No level of sophistication in string matching can result in links such as the ones described earlier and therefore semantic matching complements the string matching techniques described in the previous section.

3.4 From LinQL to SQL

In what follows, we describe the algorithm for translating a LinQL query to an SQL query, and then describe the algorithm for implementing each native link specification. Finally, we outline a number of efficient strategies to combine some of the specifications.

Algorithm LinQL2SQL translates a LinQL query to a SQL query by first splitting the former query into the base query (which is in SQL) and the link clause expression. The latter is translated into SQL by LinkClauseExpr2SQL by iterating through the boolean combination of link clauses and generating a separate SQL query for each clause. Then, the boolean combination of link clauses is translated into a set of intersections/unions between the generated SQL queries.

LinkClause2SQL parses a link clause to determine what type of link terminal is used. If the link terminal is a UDF, we simply add an invocation of the UDF in the where clause of the SQL base query. If the link terminal is a native link, we rewrite the SQL base query using the rewrite rules associated with that particular native link. If the link terminal is a reference to a named linkspec, we retrieve the associated linkspec statement and parse the associated link method. The link method can be a UDF, native link or link clause expression. UDFs and native links are translated as described previously. Link clause expressions are translated by a recursive call to

Algorithm 2: The LinkClauseExpr2SQL Algorithm

input : An SQL base query Q_{base}, a link clause expression lce
output: SQL query Q

```
if lce ≠ ∅ then
    l ← next link clause in lce;
    Q ← LinkClause2SQL(Q_base, l);
    if operator = AND then
        Q ← LinkClauseExpr2SQL(Q, lce − l );
    else if operator = OR then
        Q ← Q + 'UNION' + LinkClauseExpr2SQL(Q_base, lce − l );

return Q;
```

Algorithm 3: The LinkClause2SQL Algorithm

input : A SQL base query Q_{base}, a link clause l
output: SQL query Q

```
Q ← Q_base ;
if link_terminal(l) = UDF then
    add UDF invocation to where clause in Q;
else if link_terminal(l) = native_link then
    rewrite Q using native_link's rewriting rules;
else if link_terminal(l) = link_spec_name then
    get link_method from associated link_spec_stmt ;
    if link_method = UDF then
        add UDF invocation to where clause in Q;
    else if link_method = native_link then
        rewrite Q using native_link's rewriting rules;
    else if link_method = link_clause_expr then
        lce ← get associated link_clause_expr;
        Q ← LinkClauseExpr2SQL(Q_base, lce);

return Q;
```

the LinkClauseExpr2SQL sub-routine. The recursion stops when either a UDF or a native link is encountered.

The main translation logic is in the rewriting rules associated with the native links. A native link's rewriting rules are specified in two parts: view definitions and link conditions. For example, the rewriting rules for the *weightedJaccard* native link on *tab1.col1* and *tab2.col2* consists of the view definitions for *tab1col1weights, tab1col1sumweights, tab1col1tokenweights,*

tab2col2tokens, scores, scores2, and the link conditions *scores.tid1=col1 AND scores.tid2=col2 AND s.tid1=s2.tid1 AND scores2.mx=s.score*.

The use of the view definition syntax is purely for readability. In practice, the SQL queries associated with the view definitions are inlined into the actual query itself (resulting in a possibly hard to read query). The WITH statement supported by some DBMS (e.g. IBM DB2) is another means for inlining some of the view definitions. Depending on the application, one may choose to materialize all or part of these views using *link index* statements in order to speed up the query time. We present the SQL queries based on view definitions in this section and discuss briefly the cost of materializing these views in the experimental results. The rest of this section describes the rewriting rules used to implement some of our native link methods.

3.4.1 Approximate Matching Implementation

The rewriting of the approximate string matching native link specification into SQL consists of three steps, namely, (a) the creation of tables containing the tokenization of the strings into q-grams or word tokens; (b) the gathering of statistics and calculation of token weights from the token tables; and (c) the calculation of link scores based on the weights. In more detail:

Step 1: This step can be done fully in SQL using standard string functions present in almost every DBMS. Assume a table `integers` exists that stores integers 1 to N (maximum allowable length of a string). The main idea is to use basic string functions `SUBSTR` and `LENGTH` along with the sequence of integers in table `integers` to create substrings of length q from the string column `col1` in table `table1`. The following SQL code shows this idea for $q = 3$:

```
SELECT tid, SUBSTR(col1,integers.i,3) as token
FROM   integers INNER JOIN table1
       ON integers.i <= LENGTH(col1) + 2
```

In practice, the string `col1` is used along with `UPPER()` (or `LOWER()`) functions to make the search case insensitive. Also, the string is padded with $q - 1$ occurrences of a special character not in

any word (e.g. '$') at the beginning and end using the `CONCAT()` function. Similarly the spaces in the string are replaced by $q - 1$ special characters. In case of tokenization using word tokens a similar SQL-based approach can be used. At the end of this process, the token generation queries are declared as views, or are materialized in tables like `table1_col1_tokens` and `table2_col2_tokens`.

Steps 2 and 3: These steps are partly native-link specific and are more easily presentable through an example. In what follows, we use the weighted Jaccard specification as an example.

Example 6 (weightedJaccard native linkspec) *Consider the following* LinQL *specification:*

```
SELECT PV.visitid, CT.trial
FROM   visit AS PV, trial AS CT
WHERE  PV.visitid = 1234 AND CT.city='NEW YORK' AND
       PV.diag = CT.cond
       LINK PV.diag WITH CT.cond
       USING weightedJaccard
```

This specification is translated into the following SQL queries. Initially, two queries calculate the IDF weights for the tokens and the auxiliary views/tables needed for the final score calculation:

```
CREATE VIEW visit_diagnosis_weights AS
SELECT token, LOG(size - df + 0.5) - LOG(df+0.5) as weight
FROM   ( SELECT token, count(*) as df
       FROM     (SELECT * FROM visit_diagnosis_tokens
                GROUP BY tid, token) f
       GROUP BY token ) D,
       ( SELECT count(*) as size
       FROM visit_diagnosis_tokens ) S

CREATE VIEW visit_diagnosis_sumweights AS
SELECT tid, B.token, weight
```

```
FROM   visit_diagnosis_tokenweights idf,
       (SELECT DISTINCT tid, token
       FROM visit_diagnosis_tokens B) B
WHERE  B.token = idf.token

CREATE VIEW visit_diagnosis_tokenweights AS
SELECT      tid, sum(weight) as sumw
FROM        visit_diagnosis_weights
GROUP BY    tid
```

Then, the next query returns the links along with their final scores:

```
WITH   scores(tid1, tid2, score) AS (
       SELECT tid1, tid2,
              (SI.sinter/(BSUMW.sumw+QSUMW.sumw-SI.sinter))
              AS score
       FROM   (SELECT  BTW.tid AS tid1,QT.tid AS tid2,
                       SUM(BTW.weight) AS sinter
              FROM     (SELECT * FROM visit_diagnosis_weights
                       WHERE id = 1234) AS BTW,
                       trials_condition_tokens AS QT
              WHERE    BTW.token = QT.token
              GROUP BY BTW.tid, QT.tid) AS SI,
              (SELECT  *
              FROM     visit_diagnosis_sumweights
              WHERE    id = 1234 ) AS BSUMW,
              (SELECT  Q.tid, SUM(BTW.weight) AS sumw
              FROM     trials_condition_tokens Q,
                       visit_diagnosis_tokenweights AS BTW
              WHERE    Q.token = BTW.token
```

```
            GROUP BY Q.tid ) AS QSUMW
      WHERE  BSUMW.tid=SI.tid1 and SI.tid2 = QSUMW.tid )
SELECT PV.visitid, CT.trial, s.score
FROM   visit AS PV, trials AS CT, scores AS S
WHERE  PV.visitid = 1234 AND CT.city='NEW YORK' AND
       S.tid1=PV.visitid AND S.tid2=CT.trial AND S.score>0.5
```

3.4.2 Semantic Matching Implementation

Assume that the synonym and hyponym data are stored in two tables `synonym` and `hyponym` with columns `src` and `tgt`. The column `src` contains *concept IDs* of the terms, and the column `tgt` contains the terms. This is a common approach in storing semantic knowledge, used in NCI thesaurus and Wordnet's synsets for example. Alternatively, this data could be stored in a table `thesaurus` with an additional column `rel` that stores the type of the relationship, or it could even be stored in XML. In the case of XML, `synonym` and `hyponym` can be views defined in a hybrid XML relational DBMS such as DB2. For brevity, we limit our discussion in this chapter to semantic knowledge stored as relational data, although our framework is easily extensible to other formats. We show the details of the SQL implementation of the synonym and hyponym native link specifications in the following two examples.

Example 7 (Synonym Native Linkspec) *Consider the following query written using* LinQL.

```
SELECT PV.visitid, CT.trial
FROM   visit AS PV, trial AS CT
WHERE  PV.visitid = 1234 AND CT.city='NEW YORK' AND
       PV.diag = CT.cond
       LINK PV.diag WITH CT.cond
       USING synonym
```

This query is rewritten to:

```
SELECT DISTINCT PV.visitid, CT.trial
FROM   trials AS CT, visit AS PV, synonym AS syn
WHERE  PV.visitid = 1234 AND CT.city='NEW YORK' AND
       (src in (SELECT src
               FROM synonym s
               WHERE s.tgt = CT.cond))
       AND PV.diag = syn.tgt
UNION
SELECT PV.visitid, CT.trial
FROM   trials AS CT, visit AS PV
WHERE  PV.visitid = 1234 AND CT.city='NEW YORK' AND
       CT.cond = PV.diag
```

Example 8 (Hyponym Native Linkspec) *Consider the following query written using* LinQL.

```
SELECT PV.visitid, CT.trial
FROM   visit AS PV, trial AS CT
WHERE  PV.visitid = 1234 AND CT.city ='NEW YORK' AND
       PV.diag = CT.cond
       LINK PV.diag WITH CT.cond
       USING hyponym
```

This query is rewritten to:

```
WITH   traversed(src, tgt, depth) AS (
       (SELECT src,tgt,1
       FROM    hyponym AS ths
       UNION ALL
       (SELECT ch.src, pr.tgt, pr.depth+1
       FROM    hyponym AS ch, traversed AS pr
       WHERE   pr.src=ch.tgt AND
```

```
                pr.depth<2 AND ch.src!=`root_node'))
SELECT distinct PV.visitid, CT.trial
FROM   trials AS CT, visit AS PV, hyponym AS ths
WHERE  PV.id = 1234 AND CT.city ='NEW YORK' AND
       (src in (SELECT distinct src
                FROM traversed tr
                WHERE tr.tgt = CT.cond)) AND
       PV.diag = ths.tgt
```

Note that the hyponym depth is by default set to 2, which could be customized to any other value.

3.5 Experiments

In this section, we illustrate the flexibility of the LinQuer framework by applying it in a variety of linkage scenarios. We use these scenarios to justify the choices we have made in developing LinQuer. These scenarios are built around an online database of clinical trials published on ClinicalTrials.gov. This database is a registry of federally and privately supported clinical trials conducted in research centers all around the world. It contains detailed information about the trials, including information about the medical conditions associated with the trials, their eligibility criteria, and locations.

3.5.1 Data Sets

The clinical trials database used in our experiments contains 111,227 trials. We retrieved the data in XML format, and transformed the data into relational (and RDF) using the xCurator framework (Chapter 5), as a part of the LinkedCT project (Chapter 6). Other data sets that we used in our experiments include a database of patient visits or Electronic Medical Records (EMR), and a database of DBpedia [33] objects of type *disease* and *drug*. We also used the

Table 3.1: Data set statistics

Data set	Entity (*table.attribute*)	Count
Clinical Trials	Trial (*trial.trialid*)	111,227
	Condition (*trial.condition*)	25,935
	Intervention (*trial.intervention*)	149,958
	Drug (*trial.drug_intervention*)	87,231
	Reference (*trial.reference*)	86,214
Patient Visits	Visit (*visit.visitid*)	10,000
	Diagnosis (*visit.diagnosis*)	9,319
	Prescription (*visit.prescription*)	8,289
DBpedia	Disease (*dbpedia_disease.label*)	5,154
	Drug (*dbpedia_drug.label*)	4,658
NCI Thesaurus	Concept (*ontology.conceptid*)	89,129
	Term (*ontology.term*)	246,511
	Synonym Relationship (*synonym.conceptid, synonym.term*)	244,192
	Hyponym Relationship (*hyponym.conceptid, hyponym.term*)	99,553

National Cancer Institute (NCI)'s thesaurus [160] as a source of semantic information about medical terms. Detailed statistics on these data sets is shown in Table 3.1[2].

Due to privacy issues associated with EMR data, our patient visits database is synthetic, generated using an extension of the UIS database generator [104] that picks diagnosis and prescription values from the NCI terms to fill the patient visits table. Our data generator also randomly injects a small amount of string error in the diagnosis field to resemble real EMR records. The error injected in the string resembles real errors and typos occurring in string databases, e.g., replacing a character with an adjacent character on a keyboard, or swapping two characters or word tokens.

[2]To make our example queries simple, we assume that the databases are denormalized and we have a single table for clinical trials (*trial*), a table storing patient visits (*visit*), and tables storing DBpedia disease (*dbpedia_disease*) and drug (*dbpedia_drug*) data. In reality, the database is normalized and these relations are decomposed into multiple relations.

3.5.2 Effectiveness and Accuracy Results

In what follows, we describe several link discovery scenarios involving clinical trials. While the first scenario is described in more detail (including its intermediate steps and corresponding linkage specifications), for the other scenarios we only show the final results and only mention changes to preceding LinQL statements.

Case 1 (Linking patient visits to trial conditions) The objective here is to discover links to clinical trials that are related to the conditions of certain patients. For this study, we consider 1,000 random patients from table *visit*, where column *diagnosis* stores the condition associated with a patient's visit. The *trial* table stores the trial condition in its column *condition*. The records linked by a simple exact matching are obtained by the SQL query:

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM   visit PV, trial CT
WHERE  PV.diagnosis = CT.condition
```

The query links only 46 out of the 1,000 patient visits records, each to an average of 90 trial records, resulting in 4,127 visit-trial links. The reason why only 5% of the patient visits are linked is that very different (string) representations of the same entities (conditions) are used in the two sources, while string errors in *diagnosis* values make the situation worse. As a next step, we try a string matching linkage method starting with a low similarity threshold (expecting this to give a high recall) using the following linkspec and query:

```
CREATE LINKSPEC weightedJaccard04
AS weightedJaccard (0.4, 2, 50).

SELECT PV.*, CT.*
FROM   visit PV, trial CT
WHERE  LINK PV.diagnosis WITH CT.condition
       USING weightedJaccard04
```

The user can now go through a small subset of the results obtained from the above query to estimate the precision of the returned links with respect to the similarity score. The accuracy of the links is subjective and depends on the application. In our scenario, for example, links from a visit record with diagnosis "Alpa Thalassemia" (misspelled record of "Alpha Thalassemia") to trials with conditions "Alpha Thalassemia", "α-Thalassemia" and "Thalassemia" should be considered correct and we would like to find them. However, linking to a trial with condition "Beta Thalassemia" is not a correct link for "Alpha Thalassemia" diagnosis. We obtained the following accuracy results for this scenario by investigating the results of the above query for 100 random distinct *diagnosis* values matched with *condition* values:

Threshold	Number of Links	Accuracy (Precision)
0.70	22	91%
0.65	36	86%
0.60	63	84%
0.55	104	77%
0.50	182	66%
0.45	303	54%
0.40	579	40%

By choosing a high threshold of 0.70, 22 links are returned out of which 20 (91%) are correct. However, by choosing a threshold of 0.4, 579 links are returned (more than 5 links per diagnosis value in the 100 results), but only 231 (40%) of them are correct. Given these observations, a user can choose the appropriate threshold that works best for the specific linkage needs. For our example, we choose a threshold of 0.55 that returns on average almost one link per diagnosis value, and has a reasonable accuracy. Overall, we get 32,919 links for 421 (out of 1,000) distinct patient visits. Based on the above evaluation, we can estimate that around 324 (77%) of the visits will link correctly to a trial's condition.

The next step is to use the semantic information in NCI to improve the linkage using the LinQL query below:

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM   visit PV, trial CT
WHERE  LINK PV.diagnosis WITH CT.condition
       USING synonym
```

Linking using semantic matching based only on synonyms results in 12,343 links from 130 distinct visits. Note that these links are all correct links since they are derived using accurate synonyms provided by the NCI thesaurus, and no further approximate string matching is performed at this stage. From these, 2,453 links from 15 distinct visits could not be found using exact or string matching.

Repeating the above query with `hyponym` linkage method for semantic matching based on hyponyms of depth 2 from NCI, results in 15,464 links from 60 visit records, out of which 14,571 links from 14 visit records could not be found in previous steps. Note that the number of discovered links per visit is much higher for the `hyponym` method when compared to the `synonym` or string matching methods. This not only shows the importance of the `hyponym` method but also illustrates its usefulness in situations like the current setup where trials refer to broad disease categories instead of specific disease names (e.g., "Blood Disorder" instead of "Beta-Thalassemia").

One reason for the relatively low total number of discovered links by the `synonym` and `hyponym` methods (linking only 173 out of the 1,000 visit records) is the errors present in the *diagnosis* values of the *visit* table. This calls for using a linkspec that combines semantic matching (e.g., `synonym` method) with string matching (e.g., `weightedJaccard` method). The LinQL code to do this is:

```
CREATE LINKSPEC mixmatch
AS LINK source WITH target
   USING synonym(synonym,conceptid,term)
   AND
   LINK source WITH synonym.term
   USING weightedJaccard (0.7, 2, 50)
```

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM   visit PV, trial CT
WHERE  LINK PV.diagnosis WITH CT.condition
       USING mixmatch
```

Using a combined string and semantic matching results in 49,931 links from 363 distinct visits, 28,444 more links from 33 more visit records when compared with matching based on synonyms or string matching. In the above linkspec, we have chosen a higher similarity threshold for string matching which results in highly accurate links based on our manual verification of the results. Depending on the results of the above steps, the user can write a single query for the linkage needs specific to the application. Here we choose to combine exact matching, string matching, semantic matching based on synonyms and hyponyms, and mixed semantic matching allowing string errors. This can all be expressed using the query below:

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM   visit PV, trial CT
WHERE  LINK PV.diagnosis WITH CT.condition
      USING weightedJaccard
      OR
      LINK PV.diagnosis WITH CT.condition
      USING synonym
      OR
      LINK PV.diagnosis WITH CT.condition
      USING hyponym
      OR
      LINK PV.diagnosis WITH CT.condition
      USING mixmatch
```

The combined approach results in 73,387 links from 482 visit records to the related clinical trials. Overall, we have:

Method	Links #	Visits #
1.Exact Match	4,127	46
2.String Match	32,919	421
3.Synonym Match	12,343	130
4.Hyponym Match	15,464	60
5.Mixed Match	49,931	363
Total (Combined)	73,387	482

The above clearly illustrate that the LinQuer framework facilitates the fast development and testing of linkage methods. Since LinQuer is fully integrated with declarative querying, the accuracy (precision and recall) of linkage methods can easily be tested over small portions of the data, or over portions where a user knows what links she desires.

Case 2 (Linking prescriptions to trial interventions) Now consider a user who wishes to link patients who were prescribed a drug with *all* clinical trials that use that drug. To collect all these trials, the user will again need the results produced from a variety of algorithms. As in the previous case, we start with exact matching, try string matching and tune its threshold, and then use synonym, hyponym and the `mixmatch` linkspec defined above. For string matching using `weightedJaccard` method, we choose threshold 0.6 based on a similar manual inspection of a subset of the results. The table below summarizes the results obtained for linking 1,000 visit records to trials by matching prescription values of the visits to drug intervention values of trial records:

Method	Links #	Visits #
1.Exact Match	1,323	93
2.String Match	7,200	270
3.Synonym Match	16,804	349
4.Hyponym Match	145	12
5.Mixed Match	20,730	399
Total (Combined)	23,528	470

Notice that the `synonym` method is much more effective here (when compared to Case 1), both in terms of the number of records linked and the number of links per record. This is mainly due to the fact that the same drug can be listed under a variety of different names (brand names and scientific names), depending on the institution that performs the trial. Also notice that (like in Case 1) multiple methods do find the same links (the total number of links is less than the sum of the links returned by each method). However, each of the five methods is effective in finding links that cannot be found using any of the other methods. This is highly desirable since the goal is to link as many potential visit records as possible.

Case 3 (Linking trial conditions to DBpedia diseases) and

Case 4 (Linking trial interventions to DBpedia drugs) In these scenarios, we are seeking links from the clinical trials' condition and interventions fields to the DBpedia disease and drug entities, respectively. Unlike the previous cases, assume here that the user only needs to link to a single DBpedia entity per each condition and drug. This makes sense since in most cases there should be a single record in DBpedia for a single disease (condition) or drug intervention in the trials data. Therefore the user uses the `LINKLIMIT 1` option in the LinQL query to limit the number of links. Then, when a link based on exact matching is found for a record, there is no need to look for approximate string or semantic matches for that record. This could potentially lead to a significant performance improvement.

The final linkage specification query for linking trial conditions to DBpedia diseases (table *dbpedia_disease*) is as follows. The query for trial interventions to DBpedia drugs is similar.

```
SELECT DISTINCT CT.condition, DBPD.uri
FROM   trial CT, dbpedia_disease DBPD
WHERE  LINK CT.condition WITH DBPD.label
       USING weightedJaccard
       OR
       LINK CT.condition WITH DBPD.label
       USING synonym
       OR
       LINK CT.condition WITH DBPD.label
       USING hyponym
       OR
       LINK CT.condition WITH DBPD.label
       USING mixmatch
       LINKLIMIT 1
```

Again we choose threshold 0.6 for `weightedJaccard` method based on investigation of the accuracy of the links for a random subset of the conditions. The table below summarizes the results for different steps of the linkage from 1,000 condition and drug interventions:

Method	Disease Links#	Drugs Links#
2.Exact Match	22	239
3.String Match	239	393
4.Synonym Match	26	333
5.Hyponym Match	46	0
5.Mixed Match	101	347
Total	292	422

Notice the obvious need for allowing mixed string and semantic matching in these two cases. The trials source, NCI thesaurus and DBpedia/Wikipedia names all use different conventions

and therefore there are cases where strings do not exactly match. For example, "Adenocarcinoma of Esophagus" in trials matches with "Carcinoma of Esophagus", synonym of "Esophageal Cancer" in the thesaurus which matches with "Esophageal_cancer" in DBpedia. Notice that though we are combining many methods, the LinQL `LINKLIMIT 1` option allows us to (efficiently) return the best match for each record.

Case 5 (Finding related trials) To show the flexibility of the LinQuer framework, we investigate its effectiveness in a rather different scenario. In this case, the goal is linking trials that are related to each other. Different attributes and measures can be used to identify trials that are related. In this experiment, we use the *reference* attribute of the trials and consider two trials related if the title and authors of their associated references (publications) are similar. Our trials database stores references in a single long text record that includes the names of the authors, title of the paper, the conference or journal and the date of the publication. Therefore, in order to find similarity we do not have any relevant semantic information to exploit. Furthermore, we are not interested in typos and different representations of the same string here. Instead the similarity function should measure the amount of co-occurrence of (important) words in the two strings. The following weightedJaccard linkspec performs linkage based on word tokens:

```
CREATE LINKSPEC wordTokenJaccard
AS weightedJaccard (0.7, 0, 100)
```

Using the linkspec over 10,000 random trials results in 667 links between these trials, whereas exact matching results in only 7 links. Note that here we assume that similarity score above 0.7 between the reference strings indicates correlation between the trials, and we have verified that this assumption is reasonably accurate by manually going through a subset of the links.

3.5.3 Effectiveness of Weight Tables

In what follows, we briefly show the effectiveness of the functionality enhancement we proposed based on manual definition of a weight-adjustment table by the user. Assume that the user

defines the following simple weight table:

String	Weight
'Syndrome'	0.4
'The'	0.1
'Disorder'	0.2
'Disease'	0.2

We repeat the experiment for Case 1 (linking to trial conditions from a database of patient visits) with updated weight tables based on the above input weight adjustment table. This results in lower weights for the tokens of the above strings in the weight tables (views) of *visit.diagnosis* and *trial.condition* columns. String matching with the same settings, i.e., using `weightedJaccard` similarity function with threshold 0.55 on 1,000 random base records, results in 121 additional links out of which 91 are correct (accuracy 75%) and drops 63 of the links found with no weight adjustments out of which 15 were wrong links. This means that overall, the linkage has resulted in 58 more links with roughly the same accuracy as the case with no adjustments.

Note that we obtained these results by choosing the weight adjustment values in the above simple table based only on our domain knowledge, and we have not varied the values to obtain the best results. What is more important is that we can use this extensible technique and leverage weight values to improve the accuracy of the link discovery, as a result of the SQL-based implementation method. The implementation of the weighted Jaccard similarity function and the above customization of weights using a UDF, rather than a native method, could be quite complex and inefficient.

3.5.4 Performance Results

Here, we report running times of the above examples to show the performance of the system. Note that since our focus in this chapter has been on the functionality of the framework, we have not applied specific hashing techniques (some of which we discussed briefly in Section 3.3.2)

to improve efficiency. We also do not compare the benefits of possible indexing and hashing strategies. We ran the experiments on a Dell PowerEdge R310 Server with 24GB memory and Intel X3470 Xeon Processor 2.93GHz (8MB Cache, Turbo HT), running Ubuntu 10.04.3 LTS (64-bit) and IBM DB2 9.7 Data Server Trial Edition. To obtain statistical significance, we report the average time from several runs of each experiment.

We first report the running times for `LINKINDEX` statements over the attributes and link methods used in all the cases described in this section. For Case 1 (linking visits to trial conditions) for example, we define `LINKINDEX` over *visit.diagnosis*, *trial.condition* and *synonym.term* columns using `weightedJaccard` method as follows:

```
CREATE LINKINDEX FULL FOR visit.diagnosis USING weightedJaccard;
CREATE LINKINDEX FULL FOR trial.condition USING weightedJaccard;
CREATE LINKINDEX FULL FOR synonym.term USING weightedJaccard;
```

As stated earlier, the above statements translate into a set of pre-processing queries that materialize the views for tokenization and token weight calculation required for the final linkage queries. With the `FULL` argument, database indices are also built over the result tables. It is also possible to use the `HALF` option that will result only in materialization of the tokenization views. This is useful for more dynamic tables, and when the attribute is used only as target (and not source) in the LinQL queries, where token weights are not needed. Figure 3.3 shows the running time for tokenization queries (that materialize token views), weight table queries (that materialize token weight views), and the time for building database indices for token and weight tables, for all the attributes and link methods used in our study.

Notice that the overall pre-processing times are less than a minute for small tables, and a few minutes for larger ones. Word tokenization for references (case 5) takes the longest time due to the more expensive tokenization query and longer average string size (the average *reference* string length is 226, whereas as the average length of ontology terms is 25 characters).

Other linkage methods can also take advantage of `LINKINDEX` definitions. For the `hyponym` method and a given depth for example, the view for traversing the ontology tree can be ma-

	Tokenization Queries	Tokenization Indexing	Weight Table Queries	Weight Table Indexing	Total
trial.condition	3.4	66.1	9.0	69.5	148.0
trial.drug_intervention	9.7	146.1	113.1	119.0	387.9
visit.diagnosis	1.2	21.1	3.6	21.8	47.7
visit.prescription	0.7	12.8	4.5	12.2	30.2
dbpedia_drug.label	0.4	8.4	3.0	7.8	19.6
dbpedia_disease.label	3.6	21.5	6.2	15.6	46.9
trial.reference	58.7	547.7	43.6	696.7	1346.7
synonym.term	84.6	171.3	64.6	227.0	547.5

Figure 3.3: Pre-processing (indexing) time (in seconds)

terialized in advance. For our ontology table and for our cases (using depth 2), this can be done using the statement below, which takes just below 53 minutes to run. For the custom link method `mixmatch` over *visit.diagnosis* for example, defining `LINKINDEX FULL` as shown below will result in materializing the scores view between the ontology terms and diagnosis values using the `weightedJaccard` method with the given parameters, which takes around 42 minutes in this case. Note that these are expensive pre-processing queries but can speed up running time significantly when the query needs to be repeated many times, and when the indexed relations do not change frequently. In our scenario, NCI thesaurus is updated monthly, and the clinical trials are updated once every night, which will allow enough time to repeat the pre-processing. Of course, better index update strategies can be used, and depending on the DBMS, the view materialization can be automated, using Materialized query tables (MQTs) in DB2 for example.

```
CREATE LINKINDEX USING hyponym(hyponym,term,2);
CREATE LINKINDEX FULL FOR visit.diagnosis USING mixmatch;
```

The table below shows the running time (in seconds) for all the cases and link methods described in this section. Note that these are running times for batch processing of 1,000 base records in each case. The average running time of each method for linking a single record is under a second (a few milliseconds in most cases) which shows the suitability of the methods for online link discovery scenarios. Also note that due to the above `LINKINDEX` statements, `hyponym`

and `mixmatch` methods take only slightly longer than the `synonym` method. The string matching however is the most expensive step. This calls for more advanced hashing and indexing strategies for larger databases (see Section 3.3.2 for a brief discussion).

	Case 1	Case 2	Case 3	Case 4	Case 5
Exact Match	0.01	0.02	<0.01	<0.01	0.27
String Match	187.10	240.20	38.32	16.97	672.23
Synonym Match	0.12	0.22	0.13	0.11	N/A
Hyponym Match	0.21	0.22	0.19	0.19	N/A
Mixed Match	0.21	0.31	0.11	0.11	N/A

3.6 Related Work

In terms of the overall framework, the work described in this chapter is closely related to the work on declarative data quality and cleaning [27, 76, 89]. When compared with all the existing techniques in this area, a distinctive feature of the LinQuer framework is its focus on enabling the discovery of any type of semantic link, not just the *same-as* link. Another closely related framework is the work of Das et al. [54], in which a set of SQL operators are proposed to support ontology-based semantic linking over relational data. A key advantage of the LinQuer framework is allowing string matching along with semantic matching which is crucial in many real world scenarios. Moreover, the LinQL specification language allows the definition of new operators, which could be a mix of several semantic and string matching operators. Overall, LinQuer complements and extends the above-mentioned frameworks by providing a framework for fast prototyping and testing of semantic and syntactic matching for link discovery. LinQuer can clearly be used in combination with a non-relational framework for link discovery such as SILK [170]. Whereas LinQuer discovers links over relational data *before* these are published as RDF, SILK focuses on the discovery of semantic links *after* the RDF publication occurs. Another key difference between the two systems is that while LinQuer is an extensible framework that employs several semantic and syntactic methods particularly suitable for string

data, SILK offers a limited set of attribute-similarity measures that can be combined in various ways for link discovery.

As stated earlier, one of the main applications of the LinQuer framework is to enhance Web data publishing. There are many methodologies for generating RDF views over relational data. These methodologies are often based on declarative specification of the mapping between relational tables and RDF triples. These frameworks include, but are not limited to, D2RQ and D2R Server [35], Openlink Virtuoso [69] and Triplify [11]. The success of these tools motivates a similarly declarative framework for link discovery such as LinQuer so that the published data sources can be enriched internally and linked to other data sources to enhance data sharing.

3.7 Conclusion

In this chapter, we presented a declarative extensible framework for link discovery from relational data. We proposed a simple specification language, LinQL, along with the details of its implementation. We adopted and extended existing string matching and semantic matching techniques, and proposed functionality enhancements specifically designed for our framework. We showed the effectiveness of our approach in several link discovery scenarios in a real world health care application. Our focus has been on developing efficient techniques that can handle large data sets, but also on usability. We showed how a user can interactively experiment with and customize different link methods to better understand what are the most effective methods for her domain. We believe that our framework can significantly enhance the process of publishing a high-quality data source with links to other data sources on the web. A user/data publisher can use our framework to easily find the appropriate linkage algorithm for the specific application, as well as the optimal value of the required parameters. Our framework combined with an existing popular declarative approach for generating linked data on the web such as D2RQ [35], can lead to a quick and simple way of publishing an online data source with high-quality links. This could significantly enhance the value of the data in the next generation of web.

Chapter 4

Discovering Linkage Points for Record Linkage over Web Data

4.1 Introduction

In Chapter 3, we made the assumption that users are able to accurately identify the attributes that can be used to specify the linkage requirements. This assumption is reasonable when data sources are well-designed relational databases with a limited number of columns that have descriptive names and/or a sample of records that can be linked is available. In practice and on the Web in particular, the number of attributes can be very large, and the schema of different sources (if available) may not follow the same attribute naming standards. Therefore the first step in linking such sources is identification of attributes that are shared or related, and can be used for record linkage.

Traditionally, this step is performed by *schema matching*, where the goal is to identify the schema elements of the input data sources that are semantically related. However, the massive growth in the amount of unstructured and semistructured data in data warehouses and on the Web has created new challenges for this task. In what follows, we first describe an example real-world integration scenario and then describe the unique challenges not addressed in previous work.

Table 4.1: Data set statistics for the example scenario

Data Source	Records#	Fact#	Distinct Paths#
DBpedia	24,367	1.9 Million	1,738
Freebase	74,971	1.9 Million	167
SEC	1,981	4.5 Million	72

Consider the scenario of gathering data on public companies from the Web. We have collected three data sets from online sources: Freebase[1], DBpedia[2], and the U.S. Securities and Exchange Commission[3] (SEC). The data sets are respectively extracted using Freebase's Metaweb Query Language, DBpedia's SPARQL endpoint, and IBM SystemT [125] applied on the online SEC forms. Each of the three data sets is converted into a collection of JSON encoded records. Each JSON record is a tree representing various facts about a public company (an *entity*). Table 4.1 shows some statistics over these data sets. One can see that the three data sets are very different in structure. JSON trees in DBpedia have over 1700 different paths, and describe 1.9 million *fact*s (or entity-attribute-value triples), while JSON trees in SEC have only 72 distinct paths, but describe 4.5 million facts. This shows that DBpedia uses several naming conventions and contains very heterogeneous records, while records in the SEC data set have very consistent structure. Another observation is that Freebase has over 74,000 JSON records describing 1.9 million facts, while SEC has only approximately 2,000 records, but describes 4.5 million facts. It shows that SEC contains much more complex records than Freebase does[4].

The three data sets should have significant overlap due to the common topic (i.e. public companies). Despite the fact that the SEC data set has the greatest structural regularity, we are motivated to integrate all three data sets to take advantage of DBpedia and Freebase data sets. Toward record linkage, we first need to identify *linkage points*, i.e., the paths in the JSON trees that are shared or related among the data sets (formal definition in the next section). One

[1] http://www.freebase.com
[2] http://www.dbpedia.org
[3] http://www.sec.gov
[4] The data we have used was retrieved in January 2010. DBpedia data and schema have considerably changed since then, so the statistics and examples here may no longer be accurate, but can be verified using the data dumps available on our project page [218].

possible approach is to apply schema matching algorithms [150] based on the schema information of the data sets. A purely schema-based matching algorithm would fail in many cases. For instance, DBpedia contains labels of `dbpedia:stockSymbol`, `dbpedia:stockTicker` and `dbpedia:stockTicket` (human error in data entry[4]). Further investigation of the instances reveals that all three candidate attributes in DBpedia actually contain only a single value `NYSE` which shows that DBpedia extraction algorithm has been unable to extract the true stock symbols and only contain the name of the stock exchange. Moreover, matching Freebase and SEC based on `stockSymbol` attributes results in ambiguous links (one company matched with more than one company on the other side) which reveals the fact that some (subsidiary) companies in Freebase share stock symbol with their parent companies. This shows that these stock symbol attributes are not very *strong* linkage points as one would expect.

Even if the schema labels match, there could be differences in style and representation that make matching the records difficult. In fact, our experience in this and other similar scenarios (as described in Section 4.5) show that for the most interesting and useful linkage points, both schema labels and values do not match using simple string comparison. For example, there are different attribute labels used for URLs (e.g. `url` in Freebase, `foaf:page` and `foaf:homepage` in DBpedia) and there are different ways of writing URLs (e.g., `http://ibm.com` vs. `http://www.ibm.com/`). Another example is different representation of identifiers, e.g., the unique identifiers in SEC, called CIKs, are fixed-length numbers such as `#0000012345` stored in attribute labeled `cik` but Freebase represents them as `/business/cik/12345` stored in identifier attribute `id`. Similarly, Freebase uses a unique identifier `/m/03sc8` in an attribute labeled `mid` and DBpedia provides links to Freebase in an attribute labeled `owl:sameAs` that contains value `http://rdf.freebase.com/ns/m/03sc8`. There are also cases where only a part of the values can be used to link the records. For example, URI `http://dbpedia.org/resource/Citigroup` in DBpedia matches with ID `/en/topic/citigroup` in Freebase, or URL `http://www.audi.com/` matches with name `Audi`, and since `Citigroup` and `Audi` are unique names, these attributes can effectively be used

to link the records. Such linkage points can easily be missed unless the user has a thorough knowledge of both data sets possibly by carefully examining a large representative portion of the data which if possible, will be a tedious task.

In traditional data integration systems, identification of linkage points is performed either manually (possibly using a user-interface designed for matching schema elements) or by an automatic or semi-automatic *schema matching* algorithm. However, the size and heterogeneity of the schema, along with the schema errors present in many sources that use automated information extraction, make many existing schema-based approaches impractical. In addition, the size and heterogeneity of Web data makes many existing instance-based approaches (e.g., [117, 188]) ineffective in helping to align data.

In this chapter, we present a framework for identification of linkage points for multi-source record linkage. This framework includes 1) a library of lexical analyzers 2) fast record-level and token-level inverted indices 3) a library of similarity functions 4) a set of search algorithms to discover linkage points using the similarity functions, and 5) a set of filtering strategies to filter false-positive results of the search. We have implemented and experimentally evaluated the framework in several real world Web data integration scenarios such as the one described above.

In the following section, we present a brief discussion of background and our problem definition. Section 4.3 presents our proposed framework, and Section 4.4 presents the details of the search algorithms for attribute selection and identification of linkage points. We present a thorough experimental evaluation of the search algorithms in Section 4.5. Section 4.6 presents a brief overview of the related work, and Section 4.7 concludes the chapter with a summary of the results and a few interesting directions for future work.

4.2 Preliminaries

Let LABELS be the set of distinct constants for labels, and VALUES be the set of atomic (scalar) values. The data model we work with is defined mutually recursively by the sets DM, DICTIONARY, and LIST.

- If $x \in$ VALUES, then $x \in$ DM
- If $l_1, l_2, \ldots, l_n \in$ LABELS are distinct labels, and $v_1, v_2, \ldots, v_n \in$ DM, then $\{l_1 : v_1, l_2 : v_2, \ldots, l_n : v_n\} \in$ DM. We denote these as DICTIONARY.
- If $v_1, v_2, \ldots, v_n \in$ DM, then $[v_1, v_2, \ldots, v_n] \in$ DM. We denote these as LIST.
- Nothing else is in DM.

We refer to elements $x \in$ DM as *documents*.

We define FLAT to be an operator that maps nested lists to flat sets defined as:

$$\text{FLAT}(x) = \begin{cases} \{x\} & \text{if } x \notin \text{LIST} \\ \bigcup_i \text{FLAT}(v_i) & \text{if } x = [v_1, v_2, \ldots, v_n] \end{cases}$$

If $x = \{l_1 : v_1, l_2 : v2, \ldots, l_n : v_n\}$, then we define $x[l_i] = v_i$, and KEYS$(x) = \{l_1, l_2, \ldots, l_n\}$.

Definition 1 (Path and evaluation) *A path is a sequence of labels, written* $p = \langle \texttt{root}, l_1, l_2, \ldots, l_n \rangle$ *where* $l_i \in$ LABELS. *We write* $p \cdot l_{n+1}$ *to mean* $\langle \texttt{root}, l_1, l_2, \ldots, l_n, l_{n+1} \rangle$. *Given a document* x*, we define the evaluation of a path* p*, denoted by* $[\![p|x]\!]$ *as:*

$$\begin{aligned} [\![\texttt{root}|x]\!] &= \mathit{FLAT}(x) \\ [\![p \cdot l|x]\!] &= \bigcup \{\mathit{FLAT}(y[l]) : y \in [\![p|x]\!] \wedge l \in \mathit{KEYS}(y)\} \end{aligned}$$

Definition 2 (Data sets, attributes and instances) *A data set,* D*, is defined as a collection of documents:* $D = \{r_1, r_2, \ldots, r_n\}$ *where* $r_i \in$ *DM. We refer to* r *as the* records *of* D.

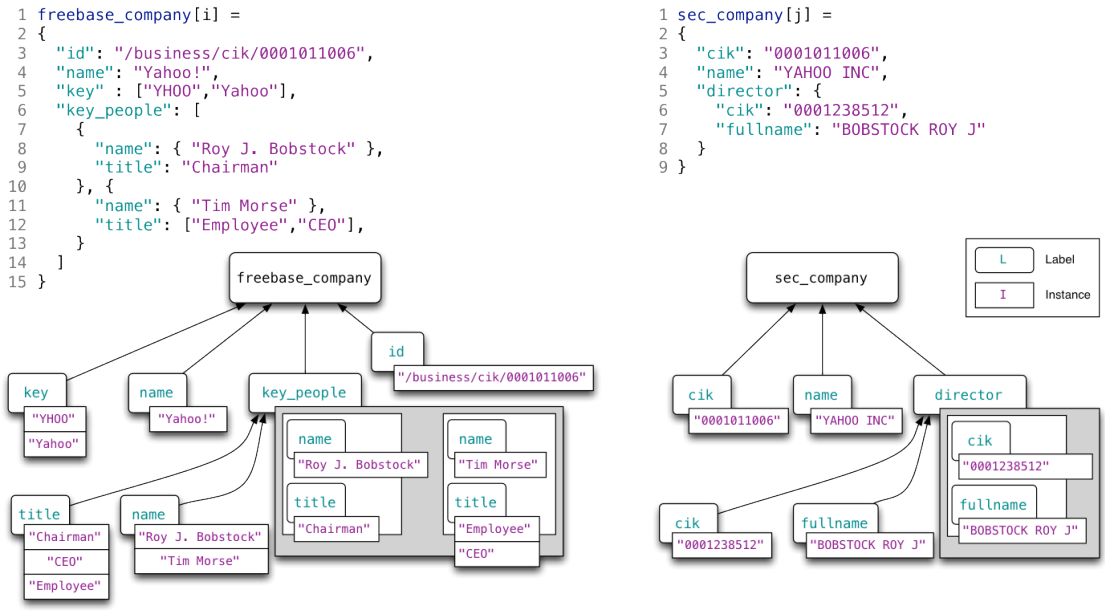


Figure 4.1: Example documents, paths, and their evaluations

An attribute *of* $r \in D$ *is a path,* p*, such that:*

$$[\![p|r]\!] \cap \textsc{Values} \neq \emptyset$$

The set of all attributes of r *in the data set is denoted as* $\text{Attr}_r(D)$*. Similarly,* $\text{Attr}(D) = \bigcup_{r \in D} \text{Attr}_r(D)$.

The instances *of attribute* p *in a record* r *are defined as* $\text{Instance}_r(p) = [\![p|r]\!] \cap \textsc{Values}$*. The instances of a data set* D *are* $\text{Instance}_D(p) = \bigcup_{r \in D} \text{Instance}_r(p)$.

We sometimes refer to an attribute $\langle \texttt{root}, l_1, \ldots, l_k \rangle$ in data set D as D `->` l_1 `->` $\ldots$ `->` l_n. For example, in Figure 4.1, `freebase_company -> key_people -> name` refers to the attribute $\langle \texttt{root}, \texttt{key\_people}, \texttt{name} \rangle$ in the first record. Also, we sometimes refer to the set of instances of a data set as the set of *facts* in the data set.

A *lexical analyzer* (or tokenizer) l is defined as a function that takes an atomic value v and converts it into a set of *tokens* **v**. Some analyzers only split the string value into a set of tokens such as word tokens or q-grams (substrings of length q of the string). Other analyzers perform string transformations (or *normalization*) by for example removing specific characters or

changing letter case in addition to tokenization (or without any tokenization). We refer to the set of tokens of all the instance values of attribute p in record r as $Instances_r^l(p)$. Similarly, $Instances_D^l(p)$ represents the set of all the tokens of all the instance values of attribute p in data set D.

A *record matching* function is defined as a Boolean function $f(r_s, r_t)$ that returns true if the two records r_s and r_t match, according to a matching criteria. The matching criteria can be defined using an *attribute matching* function $f_{(p_s,p_t)}(r_s, r_t)$, that returns true if the instance values of the two attributes $Instances_{r_s}(p_s)$ and $Instances_{r_t}(p_t)$ are *relevant*. Using a Boolean *relevance function* $r(V_s, V_t)$, we say that two sets of values V_s and V_t are relevant if $r(V_s, V_t)$ is true. There are several ways to define the notion of relevance. For example, two sets V_s and V_t can be considered relevant if there exists $v_s \in V_s$ and $v_t \in V_t$ such that v_s and v_t are *similar*. Two atomic values v_s and v_t are considered similar if their similarity score according to a value similarity function $sim()$ is above a threshold θ, i.e., $sim(v_s, v_t) \geq \theta$.

Definition 3 (Linkage Point) *We define a linkage point between two data sets D_s and D_t as a pair of attributes (p_s, p_t) such that for some attribute matching function f, the following set is non-empty:*

$$M_{(p_s,p_t)} = \{(r_s, r_t) | r_s \in D_s \wedge r_t \in D_t \wedge f_{(p_s,p_t)}(r_s, r_t)\} \tag{4.1}$$

*M is referred to as the **linkage set**.*

Definition 4 (Strength & Coverage) *We define the strength of a linkage set as the percentage of distinct records that appear in the set, i.e.:*

$$strength(M_{(p_s,p_t)}) = \frac{|S_{(p_s,p_t)}| + |T_{(p_s,p_t)}|}{|M_{(p_s,p_t)}| \times 2} \tag{4.2}$$

where $S_{(p_s,p_t)} = \{s | (s, t) \in M_{(p_s,p_t)}\}$ and $T_{(p_s,p_t)} = \{t | (s, t) \in M_{(p_s,p_t)}\}$.

We define the coverage of a linkage set as the percentage of source and target records that appear

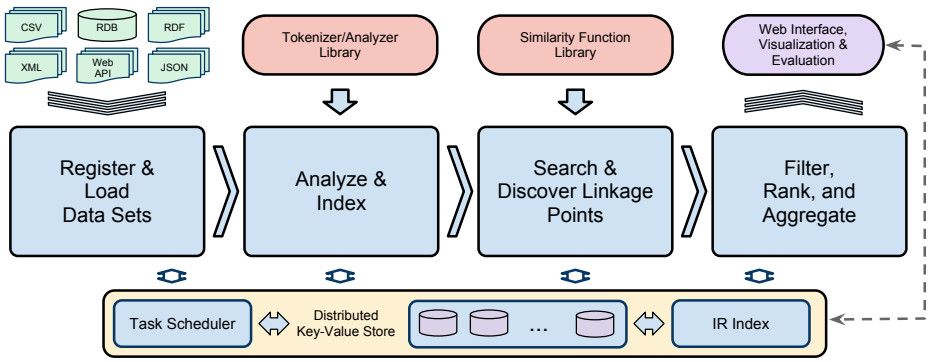


Figure 4.2: SMaSh framework

in the set:

$$coverage(M_{(p_s,p_t)}) = \frac{|S_{(p_s,p_t)}| + |T_{(p_s,p_t)}|}{|D_s| + |D_t|} \tag{4.3}$$

While integrating two data sets D_s and D_t, we need to identify attributes in D_s and D_t that are semantically related (or are used to represent the same concept) and therefore can be used (along with a value similarity function) to link source and target records that refer to the same entity. Based on the above definitions, this requires identification of linkage points between D_s and D_t whose corresponding linkage sets are of the highest *quality*, i.e., 1) they link a large number of records in the two data sets (measured by their cardinality and coverage), and 2) they only link records that refer to the same entity (measured by their strength, since each source record must be linked to at most one record in the target).

4.3 Framework

In this section, we present our framework for online discovery of linkage points out of semistructured Web data. This framework takes as input data sets (as defined in Section 4.2) and returns as output a (ranked) set of linkage points. The discovery is performed in a scalable, online fashion, suitable for large Web data sources. The framework is shown in Figure 4.2 and consists of the following components.

Task scheduler, storage and indexing backend. The backend includes a task scheduler that

manages (Web) data retrieval, indexing, and discovery tasks, in addition to fast memory and disk-based key-value store, and indexing engines. All the tasks are performed in a way that at any point, partial results can be made available to the users of the system. The task scheduler prioritizes the tasks, and keeps track of their status. Depending on the size of the data and index and the type of the discovery process, the memory-based key-value store can be used or can be replaced by the disk-based index.

Register and Load Data Sets. This component allows users to *register* a wide variety of data sources. The input could be in XML, RDF (XML, N3 or NTriples), JSON, relational or CSV format. The data can also come directly from a set of popular Web APIs including those that support SPARQL or the Freebase API. Users can then *load* the data, which will transform the data into our custom JSON format for representing data sets, store it locally, and create basic statistics to help optimize linkage point discovery. Note that our techniques will apply to Web sources that publish data sets of records representing entities of a single type (that is, one data set could be a set of companies, another could be a set of clinical trials, etc.) Furthermore, we will make the assumption that each data set represents a set of distinct entities (with very few or no duplicates). This assumption is commonly met by most Web sources which are generally curated. Of course if it is not, we could apply a pre-processing deduplication on each source, but we do not consider such a step in this work.

Analyze and Index. This component goes through the input data set, constructs the set of attributes (as defined in Section 4.2) and indexes the atomic values using one of the following analyzers available in the *lexical analyzer* library of the system.

- *Exact analyzer* does not make any change to the string.
- *Lower analyzer* turns the string value into lowercase.
- *Split analyzer* breaks the string into a set of tokens by splitting them by whitespace after using the lower analyzer. E.g., “IBM Corp.” turns into two terms “ibm” and “corp.”.
- *Word token analyzer* first replaces all the non-alphanumeric characters with whitespace

and then uses the split tokenizer to tokenize the string. E.g., "http://ibm.com" is tokenized into terms "http", "ibm" and "com".

- The *q-gram analyzer* tokenizes the string into the set of all lowercase substrings of length q. It also replaces all whitespaces with $q-1$ occurences of a special character such as $. E.g., a 3-gram tokenizer will tokenize "IBM Corp." into "$$i", "$ib","ibm","bm$","m$$","$$c","$co", "cor","orp","rp.","p.$","p.$" and ".$$".

Note that the analyzer library can be extended with other domain-specific analyzers based on the application.

Search and Discover Linkage Points. This component performs the search process for discovery of linkage points. The search is performed based on a similarity function over the values indexed in the previous component using one of the available analyzers. The framework includes a similarity function library including the following set similarity functions which are commonly used to compare tokenized string values.

- *Intersection* similarity function returns the intersection size of the two value sets. Given two value sets V_1 and V_2:

$$intersect(V_1, V_2) = |V_1 \cap V_2| \tag{4.4}$$

- *Jaccard* returns the Jaccard coefficient of the two value sets:

$$jaccard(V_1, V_2) = \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \tag{4.5}$$

- *Dice* returns the Dice coefficient of the two value sets:

$$dice(V_1, V_2) = \frac{2|V_1 \cap V_2|}{|V_1| + |V_2|} \tag{4.6}$$

- *MaxInc* returns the maximum inclusion degree of one value set in another, i.e.,:

$$maxinc(V_1, V_2) = \frac{|V_1 \cap V_2|}{min(|V_1|, |V_2|)} \tag{4.7}$$

The similarity function library also includes similarity functions that quantify the similarity of a given value string to the values in a set of instance values. These functions are based on well-established relevance functions (with highly efficient implementations) in information retrieval, such as Cosine similarity with tf-idf [157] and the Okapi BM25 [155]. These relevance functions are used to quantify the relevance of a query to a document. We use the same approach to determine the closeness of a value string $v_1 \in V_1$ to a value string $v_2 \in V_2$. For brevity, we describe only one such relevance function, the BM25 measure which is considered the state-of-the-art in document retrieval systems, though our system includes of a library of these functions. A description of the other similarity measures can be found in Section 2.1.2.

- *BM25* similarity score is defined as follows:

$$bm25(v_1, v_2) = \sum_{t \in \mathbf{v}_1 \cap \mathbf{v}_2} \hat{w}_{v_1}(t) \cdot w_{v_2}(t) \tag{4.8}$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the set of tokens in v_1 and v_2 (tokenized using one of the above-mentioned analyzers), and:

$$\begin{aligned} \hat{w}_{v_1}(t) &= \frac{(k_3+1)\cdot tf_{v_1}(t)}{k_3+tf_{v_1}(t)} \\ w_{v_2}(t) &= \log\left(\frac{N-n_t+0.5}{n_t+0.5}\right)\frac{(k_1+1)\cdot tf_{v_2}(t)}{K(v_2)+tf_{v_2}(t)} \\ K(v) &= k_1\left((1-b)+b\frac{|v|}{avg_{vl}}\right) \end{aligned}$$

and $tf_v(t)$ is the frequency of the token t in string value v, $|v|$ is the number of tokens in v, avg_{vl} is the average number of tokens per value, N is the total number of values in the set of values V_2, n_t is the number of records containing the token t and k_1, k_3 and b are a

set of independent parameters.

Details of how these functions are used during search will be discussed in the following section.

Filter, Rank and Aggregate This component takes the output of the previous component with matched attributes and their similarity scores, and filters, ranks and aggregates the results. Filtering can be performed using several measures, for example:

- *Cardinality filter* removes a linkage point (p_s, p_t) if the cardinality of its corresponding linkage set $M_{(p_s,p_t)}$ is small, i.e., $|M_{(p_s,p_t)}| < \theta_l$ where θ_l is a user-defined cardinality threshold.

- *Coverage filter* uses the coverage measure from Definition 4 to remove a linkage point (p_s, p_t) if the coverage of its linkage set is less than a user-defined threshold θ_c, i.e., $coverage(M_{(p_s,p_t)}) < \theta_c$.

- *Strength filter* uses the strength measure from Definition 4 to remove a linkage point (p_s, p_t) if $strength(M_{(p_s,p_t)}) < \theta_s$ where θ_s is a user-defined strength threshold.

The framework also includes a Web interface for task management and monitoring, visualization of the results and evaluation based on user feedback. Such results can be used, for example, to define linkage rules in the user's preferred language such as LinQL (cf. Chapter 3), or Dedupalog [9].

4.4 Search Algorithms

In this section, we present the details of our proposed search algorithms for discovery of linkage points. As mentioned in the previous section, the search process uses a set of similarity functions to search for similar instance values that are tokenized using one or more lexical analyzers, and indexed. The search process starts from a source data set, and goes through all its attributes. For each attribute, the set of instances (or a sample set) is retrieved from the index. Then the target data set is searched for these values using a similarity function.

Algorithm 4: SMaSh-S

Input : Data sets D_s and D_t,
A set similarity function f,
A lexical analyzer l,
A matching threshold θ
Output: (Ranked) list of pairs of attributes (p_s, p_t)

```
foreach attribute p_s in D_s do
    foreach attribute p_t in D_t do
        score(p_s, p_t) ← f(Instances^l_{D_s}(p_s), Instances^l_{D_t}(p_t))
    end
end
return pairs of (p_s, p_t) with score(p_s, p_t) ≥ θ (in descending order of score(p_s, p_t))
```

4.4.1 SMaSh-S: Search by Set Similarity

Algorithm 4 shows our simplest realization of the search strategy, which we refer to as the SMaSh-S algorithm. This algorithm goes through the set of attributes in the source data, and for each attribute p_s in the source data set D_s retrieves the set of instance values $Instances_{D_s}(p_s)$ and calculates the similarity of this set with all the instance value sets in D_t using the given set similarity function. The result is all pairs of attributes (p_s, p_t) that have similarity score above the given threshold θ, along with the value of the similarity score.

We can create a linkage set $M_{(p_s,p_t)}$ for all the returned pairs by Algorithm 4 using an attribute matching function that returns true if the intersection of $Instances^l_{r_s}(p_s)$ and $Instances^l_{r_t}(p_t)$ is non-empty. Therefore the following proposition is true.

Proposition 4.4.1 *All the pairs returned by Algorithm 4 are linkage points (as defined in Definition 3).*

4.4.2 SMaSh-R: Record-based Search

The SMaSh-S algorithm can effectively find linkage points for data sets whose instance values have a sufficient overlap, and very few of the overlapping sets of values are accidental, i.e., only a few instance values are shared between unrelated attributes. Therefore, the performance of the algorithm is highly affected by the choice of lexical analyzer. Using simpler analyzers (such as

exact or *lower*)) can result in high-quality linkage points only if the data sets follow the same style and formats (not very common on the Web), and more complex analyzers (such as q-gram based tokenizers) can result in a large number of accidental matches.

Algorithm 5 shows an alternative search strategy, which we refer to as the SMaSh-R algorithm. This algorithm first picks a sample of values (analyzed using analyzer l) for each source attribute. For each value in the sample set, it then searches the target data set for the top k similar values (i.e., values with value similarity score above a user-defined threshold τ using a string similarity function $sim()$). This search (Lines 4-6 in Algorithm 5) can be performed very efficiently using proper indices and state-of-the-art keyword search algorithms as described in Section 4.3. The algorithm then takes the average of the similarity score for each attribute in the result among all the record-attribute-value triples returned. If this score is above a user-defined threshold θ, the attribute pair will be returned as a linkage point.

For each attribute pair in the output of Algorithm 5, one can derive a non-empty linkage set using an attribute matching function that returns true if at least one value v_s in $Instances_{r_s}(p_s)$ and one value v_t in $Instances_{r_t}(p_t)$ have similarity score $sim(v_s, v_t) \geq \tau$. Therefore, the following proposition holds.

Proposition 4.4.2 *All the pairs returned by Algorithm 5 are linkage points (as defined in Definition 3).*

4.4.3 SMaSh-X: Improving Search by Filtering

The strategy used in the SMaSh-R algorithm can improve the search performance especially for more complex analyzers that are an important part of an effective search strategy. However, low-quality results (irrelevant pairs returned as linkage points) are unavoidable when dealing with highly heterogeneous (Web) data. Therefore, we propose an effective filtering strategy to improve the the accuracy of the linkage points returned by the SMaSh-S and SMaSh-R algorithms. We refer to the resulting algorithm as the SMaSh-X algorithm. The filtering portion of

Algorithm 5: SMaSh-R

Input : Data sets D_s and D_t,
A lexical analyzer l,
A value similarity function $sim()$,
Value similarity threshold τ,
Value of k for top-k search,
Sample value set size σ_v,
A matching threshold θ

Output: (Ranked) list of pairs of attributes (p_s, p_t)

foreach *attribute* p_s *in* D_s **do**
  $Query\_Set \leftarrow$ { Up to σ_v random values in $Instances^l_{D_s}(p_s)$}
  foreach *value* q *in* $Query\_Set$ **do**
    $M \leftarrow \{(r_t, p_t, v) | p_t \in Attr_{r_t}(D_t) \wedge$
      $v \in Instances^l_{r_t}(p_t) \wedge sim(v, q) \geq \tau\}$
    $M_{topk} \leftarrow \{(r_t, p_t, v) \in M$ with top k highest $sim(v, q)\}$
    $MS \leftarrow$ Multiset $\{(p_t, sim(v, q)) | \exists r_t : (r_t, p_t, v) \in M_{topk}\}$
  end
  $score(p_s, p_t) \leftarrow$ Average sim value for all $(p_t, sim) \in MS$
end
return pairs of (p_s, p_t) with $score(p_s, p_t) \geq \theta$ (in descending order of $score(p_s, p_t)$)

the algorithm is shown in Algorithm 6. This part of the algorithm takes as input a set of linkage points along with their scores from the previous algorithms, creates a carefully selected subset of linkage sets for each linkage point, and filters out the linkage points whose linkage (sub-)sets do not have a user-defined *strength*, *coverage*, or *cardinality*.

A key issue in Algorithm 6 is effective and efficient creation of the subset of the linkage set that can effectively be used to prune irrelevant pairs from the results. For each pair (p_s, p_t), the algorithm first picks a sample set of size σ_s out of all the instances $Instances^l_{D_s}(p_s)$. Then, for each value in the sample set, the set of records containing the value (in attribute p_s) are looked up. This lookup can efficiently be performed using a reverse index built during the indexing phase. Then the set of matching records in the target data set are retrieved using the attribute matching function. For attribute pairs found using the SMaSh-S algorithm, the attribute matching function is a Boolean function that returns true if the intersection of $Instances^l_{r_s}(p_s)$ and $Instances^l_{r_t}(p_t)$ is non-empty. For attribute pairs found using the SMaSh-R algorithm, the matching function is a Boolean function that returns true if at least one value v_s

Algorithm 6: SMaSh-X Filtering

Input : Data sets D_s and D_t,
List L of linkage points with their score (p_s, p_t)
with corresponding attribute matching function
$f()$ and lexical analyzer l,
Sample linkage set size σ_s,
Cardinality threshold κ,
Smoothing cutoff limit λ,
Coverage threshold χ,
Strength threshold τ,
Matching threshold θ

Output: (Ranked) list of pairs of attributes (p_s, p_t)

foreach *pair* $(p_s, p_t) \in L$ **do**
  $M_{(p_s,p_t)} \leftarrow \emptyset$
  $V \leftarrow$ Sample of size σ_s of $Instances^{l}_{D_s}(p_s)$
  foreach $v \in V$ **do**
    $R_s \leftarrow \{r \in D_s | v \in Instances^{l}_{r_s}(p_s)\}$
    $R_t \leftarrow \{r_t | r_t \in D_t \wedge f_{(p_s,p_t)}(r_s, r_t)\}$
    if $|R_s| > \lambda$ **then**
      $R_s \leftarrow$ subset of size λ of R_s
    end
    if $|R_t| > \lambda$ **then**
      $R_t \leftarrow$ subset of size λ of R_t
    end
    $M_{(p_s,p_t)} \leftarrow M_{(p_s,p_t)} \cup \{(r_s, r_t) | r_s \in R_s \wedge r_t \in R_t\}$
  end
  if $strength(M_{(p_s,p_t)}) > \tau$ **and** $coverage(M_{(p_s,p_t)}) > \chi$
    and *Cardinality of both* p_s *and* p_t *are above* κ
  then
    $score'(p_s, p_t) \leftarrow score(p_s, p_t) \times strength(M_{(p_s,p_t)})$
  end
end
return pairs of (p_s, p_t) with $score''(p_s, p_t) \geq \theta$ (in descending order of $score(p_s, p_t)$)

in $Instances_{r_s}(p_s)$ and one value v_t in $Instances_{r_t}(p_t)$ have similarity score $sim(v_s, v_t) \geq \tau$. The matching records can be retrieved and saved (cached) while performing the SMaSh-S or SMaSh-R algorithms to speed up the filtering operation.

When the matching source/target record pairs are found, they are added to the linkage set $M_{(p_s,p_t)}$, but we limit the size of each matching records set to a user-defined limit λ. This can be seen as a way to *smooth* the strength measure, in order to decrease the effect of those infrequent values that match with a very large number of records, such as a non-standard null value

representation (for example, `None` or `N/A`) that are very common in Web data.

When the linkage set sample is created, its strength, coverage and cardinality are checked and the pair is omitted from the results if any of the values are below the thresholds given by the user. Finally, the score is adjusted by the strength measure to lower the rank of those linkage points that have lower strength, and potentially remove them if the adjusted score drops below the matching threshold θ.

4.5 Experiments

In this section, we present the results of evaluating the SMaSh algorithms in real-world integration scenarios.

4.5.1 Data Sets

Table 4.2 shows the statistics of the data sets for three real-world integration scenarios. All these scenarios are similar to our motivating example described in Section 4.1, where the goal is to discover linkage points that can be used to perform (classic) record linkage. In other words, we are looking for linkage points whose corresponding linkage sets (or a significant part of the linkage sets) consist of record pairs that refer to the same real-world entity.

Each scenario involves resolution of a single entity type, and includes three data sources. Two data sources Freebase and DBpedia are shared in all the scenarios. As described earlier, the Freebase data is downloaded in JSON format using a query written in Metaweb Query Language (MQL) to fetch all the attributes related to each entity type. DBpedia data is fetched from DBpedia's SPARQL endpoint in RDF/N-Triples format and converted into JSON. Company scenario uses data extracted from the U.S. Securities and Exchange Commission (SEC) online form files using IBM's SystemT [125] with output in JSON format. The drug scenario uses information about drugs extracted from DrugBank (`http://www.drugbank.ca`), which is an online open repository of drug and drug target data. Movie scenario uses movie data from the popular online

Table 4.2: Data set statistics

Entity	Source	Data Set	Record #	Fact#	Attr.#
Company	Freebase	fbComp	74,971	1.92M	167
	SEC	secComp	1,981	4.54M	72
	DBpedia	dbpComp	24,367	1.91M	1,738
Drug	Freebase	fbDrug	3,882	92K	56
	DrugBank	dbankDrug	4,774	1.52M	145
	DBpedia	dbpDrug	3,662	216K	337
Movie	Freebase	fbMovie	42,265	899K	57
	IMDb	imdbMovie	14,405	483K	41
	DBpedia	dbpMovie	15,165	1.57M	1,021

movie database, IMDb (http://www.imdb.com). IMDb contains a huge number of movies, TV shows and episodes, and even video games, from all around the world. For our experiments in this section, we pick a sample of movies that are most likely to appear on DBpedia or Freebase, to make sure that the data sets overlap. Automatically picking a high-quality sample of a data source to guarantee overlap with other sources, is beyond the scope of this section and thesis, and an interesting direction for future work.

4.5.2 Settings & Implementation Details

The framework shown in Figure 4.2 is implemented using a number of state-of-the-art storage, indexing, and task management systems. As our storage engine, we use Redis[5], which is an open-source networked, in-memory key-value store with optional disk-based persistence. We use Redis as our storage backend and as the backend for indexing of all the value sets for efficient set similarity computation. We use separate Redis servers to store the original data sets, meta-data and task information, and reverse indices for each lexical analyzer. We experiment with seven different analyzers, namely *exact*, *lower*, *split*, *word token*, *2-gram*, *3-gram*, and *4-gram* analyzers. This results in nine Redis server instances (two for data and meta-data, seven for analyzers). For our run-time experiments, we ran the Redis instances for each scenario on one of three servers each with an Intel X3470 Xeon Processor, and 24GB of RAM. For a fair comparison, we have

[5] http://redis.io

Table 4.3: Summary of algorithm parameters

Algorithm	Par.	Description	Default
All	θ	Matching threshold	0.5
SMaSh-R	τ_v	Value similarity threshold	0.5
	k	Value of k for top-k search	200
	σ_v	Sample value set size	$20,000$
SMsSh-X	τ_s	Strength threshold	0.6
	κ	Cardinality threshold	30
	χ	Coverage threshold	0.001
	λ	Smoothing cutoff limit	50
	σ_s	Sample value set size to build linkage set	$1,000$

made sure that the Redis instances for each experiment fully fit in memory and no swapping occurs. For the SMaSh-R algorithm and implementation of our search index for the $BM25$ similarity measure, we use Xapian[6] search engine library that includes built-in implementation of the $BM25$ function along with a highly efficient disk-based index.

A summary of the input parameters of the system is shown in Table 4.3. Unless specifically noted, we use the default parameter values in Table 4.3. Note that these default values are ones that one would naturally choose as a reasonable value for an initial experimentation. We have not performed any learning or experimentation with a specific data set to derive these values. In the next section, we also report on the effect of changing these values for each of the algorithms and data sets.

4.5.3 Accuracy Results

For our evaluation in this chapter, given the scope of this thesis, we consider a linkage point to be *relevant* if it consists of attributes that are semantically related *and* it can be used to perform (classic) record linkage, i.e., (a significant portion of) its corresponding linkage set consists of record pairs that refer to the same real-world entity. To find the ground truth, we went through

[6] http://xapian.org

the daunting task of examining hundreds of linkage points for each scenario that were found using several different settings of each of the algorithms (overall 336 set of results for each scenario). We constructed the linkage sets for each of the linkage points and went through a sample to verify that a large number of records in the linkage sets refer to the same real-world entity. We also manually inspected a sample of matching records (e.g., records that represent `IBM` in Freebase, DBpedia and SEC) to make sure there are no linkage points that none of our algorithms can identify.

Measures To evaluate the accuracy of the results, we use well-known measures from IR, namely precision, recall, F measure, reciprocal rank, and fallout. *Precision* is the percentage of the linkage points in the output that are relevant. *Recall* is the percentage of the relevant linkage points that can be found in the output of the algorithm. The F measure is the weighted harmonic mean of precision and recall, i.e.,

$$F_\beta = \frac{1+\beta^2}{\frac{\beta^2}{Re} + \frac{1}{Pr}} \tag{4.9}$$

We report the F_2 scores that weights recall twice as much as precision since in our scenarios, a low recall value negatively affects the ability to perform record linkage whereas a low precision (and a high recall) only results in more linkage points to be examined for relevance. We also report *Reciprocal Rank (RR)* which is defined as the multiplicative inverse of the rank of the first correct answer. RR measures the ability of the algorithm to return a relevant result to the user high in the ranked results. *Fallout* is the percentage of all the possible non-relevant results that are returned in the output. A high fallout score indicates too many irrelevant results in the output and therefore poor performance of the algorithm.

Results We have thoroughly evaluated all the algorithms using all the above mentioned analyzers and similarity functions, and a wide range of parameters. Here, we only report our most interesting observations. First, we report the highest accuracy results achieved by each of the algorithms in each of the scenarios. As baseline, we use the SMaSh-S algorithm with intersection as the similarity function along with exact or lower analyzers. This setting of the SMaSh-S algorithm resembles the case where a user tries to manually examine the intersection of all the

Table 4.4: Accuracy results of the SMaSh algorithms

Scenario			Best Precision				Best Recall				Best F_2				Best Fallout				Best Reciprocal Rank			
ds1	ds2	L.P. #	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X	Baseline	SMaSh-S	SMaSh-R	SMaSh-X
fbComp	secComp	36	0.03	0.13	0.57	1.00	0.19	0.84	0.97	0.97	0.09	0.40	0.68	0.83	0.03	0.02	0.00	0.00	0.33	1.00	1.00	1.00
fbComp	dbpComp	76	0.22	0.22	0.21	1.00	0.57	0.57	0.55	0.55	0.43	0.43	0.42	0.55	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00
dbpComp	secComp	13	0.02	0.04	0.15	0.50	0.33	0.89	0.89	0.89	0.06	0.17	0.43	0.61	0.00	0.00	0.00	0.00	0.01	1.00	0.25	1.00
fbDrug	dbankDrug	17	0.14	0.21	0.36	1.00	0.76	0.94	0.94	0.94	0.40	0.50	0.52	0.71	0.01	0.01	0.00	0.00	1.00	1.00	1.00	1.00
fbDrug	dbpDrug	26	0.07	0.14	0.27	0.60	0.58	0.92	0.88	0.88	0.25	0.36	0.48	0.65	0.01	0.01	0.00	0.00	0.33	1.00	1.00	1.00
dbpDrug	dbankDrug	24	0.01	0.05	0.31	0.60	0.86	0.95	0.91	0.86	0.07	0.20	0.47	0.62	0.03	0.01	0.00	0.00	1.00	1.00	0.50	1.00
fbMovie	imdbMovie	2	0.00	0.01	0.08	0.50	0.50	0.50	0.50	0.50	0.02	0.03	0.25	0.50	0.09	0.08	0.00	0.00	0.25	0.25	0.11	1.00
fbMovie	dbpMovie	22	0.02	0.03	0.05	0.30	0.73	0.95	0.95	0.91	0.09	0.14	0.19	0.56	0.01	0.01	0.00	0.00	1.00	1.00	0.14	1.00
dbpMovie	imdbMovie	8	0.01	0.01	0.50	1.00	0.71	0.71	0.71	0.71	0.04	0.07	0.41	0.69	0.01	0.01	0.00	0.00	0.20	0.50	1.00	1.00

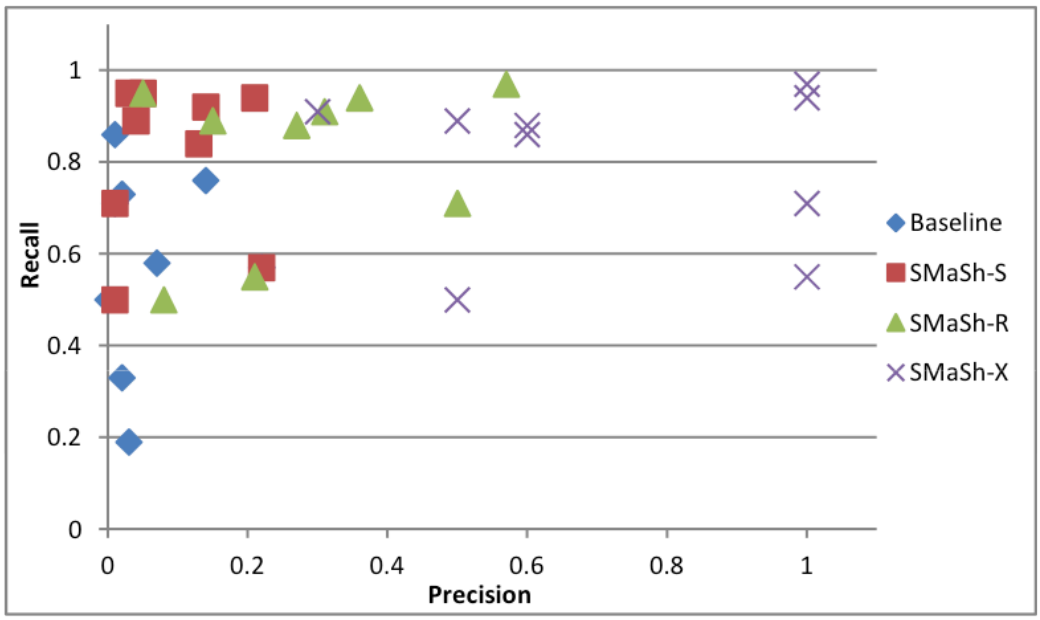


Figure 4.3: Best precision/recall values for all scenarios

values for all the attributes by issuing standard queries over Web APIs or an RDBMS (which is of course a tedious task). Table 4.4 and Figure 4.3 show the best accuracy results achieved for each SMaSh algorithm in each of the scenarios using different analyzers and similarity functions. The second column in Table 4.4 shows the number of linkage points in the ground truth, which varies from 2 (in `fbMovie` vs. `imdbMovie` scenario) to 76 (in `fbComp` vs. `dbpComp` scenario). In the majority of the scenarios, SMaSh-X outperforms the other algorithms, whereas SMaSh-R outperforms SMaSh-S and all the algorithms significantly improve the baseline performance.

Table 4.5 shows the highest F_2 score achieved in each scenario along with the analyzer and

similarity function that achieved this score. The SMaSh-X algorithm has performed best in all the scenarios, although different analyzers and similarity functions result in the highest F_2. This shows that as expected, there is no single analyzer/similarity function that can work well in all scenarios. However, intersection and BM25 similarity functions have performed best along with exact, lower and word token analyzers.

Figure 4.4 shows the effect of sample size value σ_v on the SMaSh-X algorithm for six scenarios, using word token analyzer and BM25 similarity function. The effect is similar for other analyzers and scenarios. As shown in the figure, very low sample size values below 250 will result in poor quality and in some cases reasonable results which only shows the random values can sometimes be used to find high-quality linkage points. On the other hand there is no significant change after $\sigma_v = 1000$ for most cases, which shows the effectiveness of using only a sample of values in the SMaSh-R (and SMaSh-X) algorithm.

Figure 4.5 shows the effect of each of the filtering methods of the SMaSh-X algorithm on the maximum F_2 score achieved using different settings. As shown in the figure, all the filters are effective in all the scenarios (except for one), but each filter behaves differently depending on the scenario. On the other hand, the combination of all the filters performs consistently and relatively comparably to individual filters.

Table 4.5: Best accuracy results for each scenario

Scenario		Best F_2		
ds1	ds2	analyzer	sim	F_2
`fbComp`	`secComp`	wordsplit	BM25	0.83
`fbComp`	`dbpComp`	lower	intersect	0.55
`dbpComp`	`secComp`	wordsplit	BM25	0.61
`fbDrug`	`dbankDrug`	exact	maxinc	0.71
`fbDrug`	`dbpDrug`	wordsplit	BM25	0.65
`dbpDrug`	`dbankDrug`	wordsplit	BM25	0.62
`fbMovie`	`imdbMovie`	exact,lower	intersect	0.5
`fbMovie`	`dbpMovie`	exact	intersect	0.56
`dbpMovie`	`imdbMovie`	split	BM25	0.70

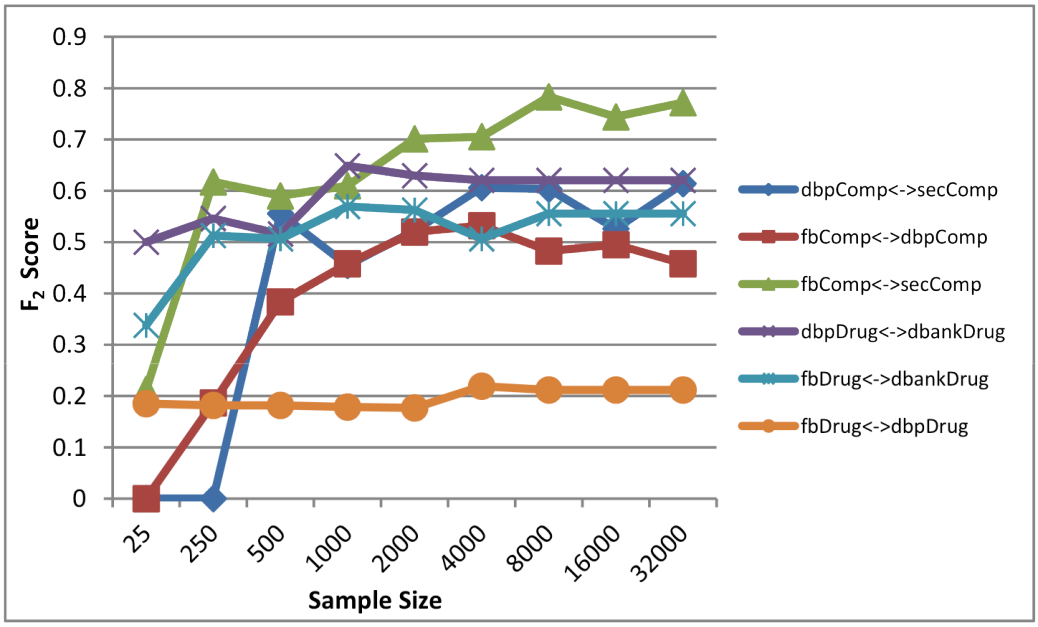


Figure 4.4: Effect of sample size σ_v on accuracy

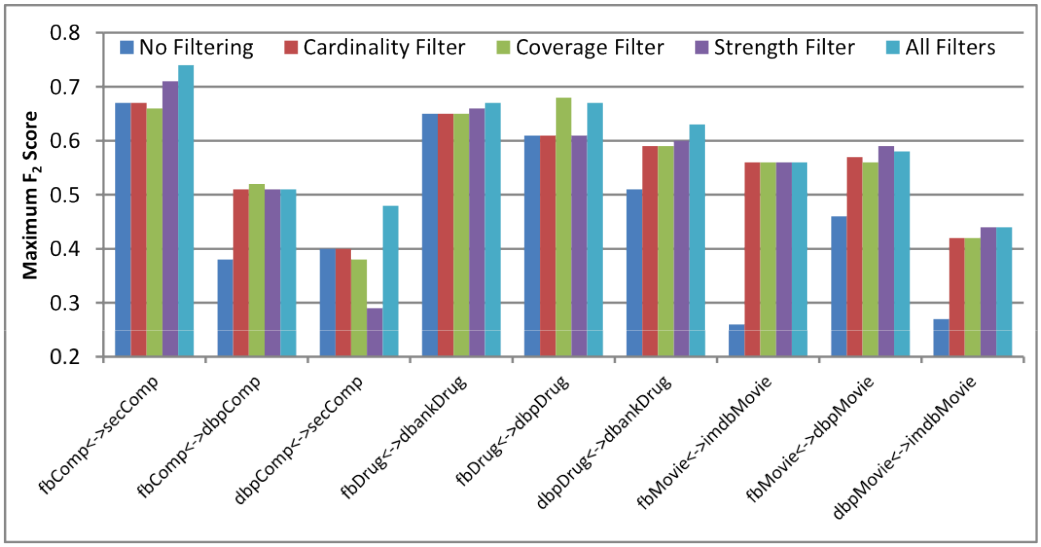


Figure 4.5: Effect of different filters on accuracy

4.5.4 Running Times

Table 4.6 shows the load and indexing times of the nine scenarios. The first column shows the time to load the data from a file, flatten the structure and index the facts. This time is linear in the number of the facts in the data set (see Table 4.2). Note that our system is also able to fetch facts directly from Web APIs. The rest of the columns show the time it takes to index instance

Table 4.6: Load and indexing time (seconds)

data set	load	exact+	lower	split	word	3-gram
fbComp	117.33	1,277.52	514.99	725.13	894.96	4,109.21
dbpComp	113.31	1,081.80	904.93	1,775.87	3,109.40	19,420.09
secComp	289.61	453.49	287.79	326.99	423.54	2,050.25
fbDrug	5.13	30.58	27.29	29.97	41.55	216.81
dbpDrug	12.39	92.25	70.40	113.99	250.28	1,441.49
dbankDrug	108.36	640.72	391.13	853.64	975.80	8,669.88
fbMovie	62.92	347.91	253.97	266.80	388.03	2,123.57
dbpMovie	93.03	685.64	528.76	926.44	1,680.56	9,835.89
imdbMovie	32.01	407.62	255.51	409.96	453.19	2,513.92

value sets both for SMaSh-S (set similarity calculation) and the IR index for BM25 similarity, using different analyzers. The time shown in the second column also includes the time it takes to build the reverse index for values required for building linkage sets for the SMaSh-X algorithm. This time is also linear in the size of the facts, although the complete task is clearly much more expensive than just indexing facts. The only exception is the `secComp` data set which has indexing time comparable to loading time, which is due to the fact that the data is automatically extracted from documents and contains a large number of instance values which are repeated in each record. Since we index value sets, the number of values indexed is much smaller (788K) than the number of facts (4.54M). Overall, these times show the scalability of the framework given that loading and indexing is a one-time process. In addition, our system supports running the indexing in the background and gradually updating the results, as described in Section 4.3.

The running time for the SMaSh-S algorithm across all the scenarios, analyzers, similarity functions, and parameters in our experiments was 22.93 seconds on average, with minimum 1.22 seconds, maximum 119.37, and 23.95 seconds standard deviation. For the SMaSh-R algorithm, the average was 338.08 seconds, with a minimum 14.58, maximum 3648.63, and 451.18 seconds standard deviation. The time for the filtering part of the SMaSh-X algorithm varied between 0.69 and 1062.59 seconds with average 138.00 and 246.54 standard deviation. Table 4.7 shows the running time of the best-performing algorithms in each scenario (maximum F_2

Table 4.7: Running time of the best performing algorithm for each scenario

Scenario		time (seconds)		
ds1	ds2	analyzer	sim	time
fbComp	secComp	wordsplit	BM25	206.13s
fbComp	dbpComp	lower	intersect	9.47s
dbpComp	secComp	wordsplit	BM25	243.54s
fbDrug	dbankDrug	exact	maxinc	2.49s
fbDrug	dbpDrug	wordsplit	BM25	48.56s
dbpDrug	dbankDrug	wordsplit	BM25	47.60s
fbMovie	imdbMovie	exact,lower	intersect	1.29s
fbMovie	dbpMovie	exact	intersect	5.09s
dbpMovie	imdbMovie	split	BM25	68.47s

values shown in Table 4.5). Note that the SMaSh-R algorithm is considerably slower than the SMaSh-S algorithm in our implementation due to memory-based implementation of set similarity measures and disk-based IR index. As the size of the data and index increase beyond the memory size, we expect SMaSh-R to outperform the SMaSh-S algorithm. It is important to note that these are the times to complete running the algorithms, although our framework can return partial results while the data sets are being loaded, indexed, and matched. In our experience, our system can return a reasonable number of linkage points in all the scenarios and settings in just a few seconds.

4.5.5 Finding the Right Settings

In addition to the experiments described above, we have also tested the ability of our framework to find the right analyzer, similarity function, and parameters, without having a ground truth, and solely based on the average score of the linkage points returned by the SMaSh-X algorithm. To verify this, we rank the settings (analyzer, similarity function, and parameters) by the average score of the top 5 linkage points returned by the SMaSh-X algorithm. Figure 4.6 shows the maximum and average F_2 score of the top 5 settings in this ranking for each of the scenarios, along with the maximum possible F_2 score in all settings. The figure shows that in three of the scenarios, the best setting is in fact among the top 5 settings, and except for two cases, the top

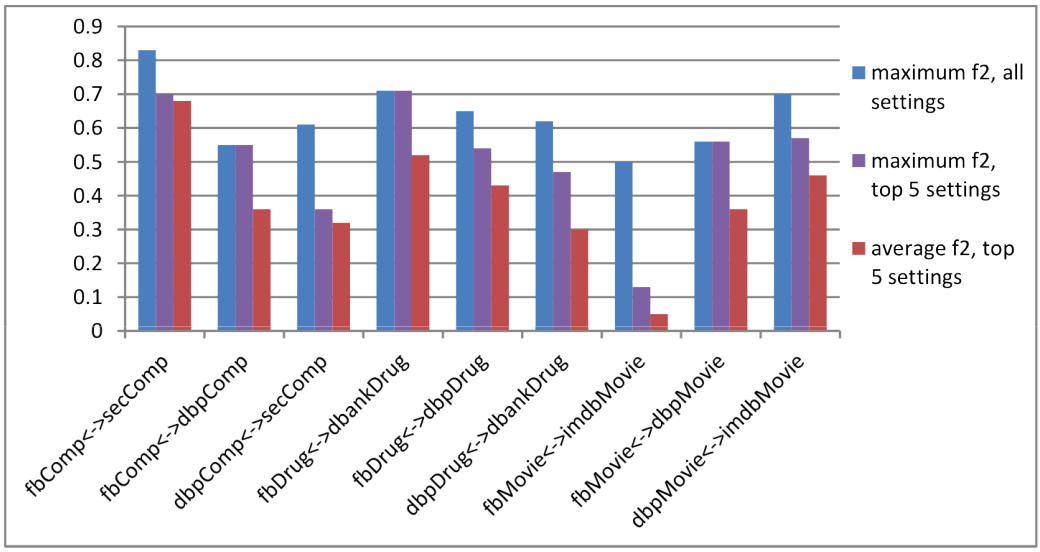


Figure 4.6: Accuracy of top 5 settings ranked by average match score

5 settings have F_2 score comparable to the highest F_2 score. In addition to this, we have also experimented with non-overlapping data sets and in most cases, very few results are returned by the SMaSh-X algorithm, indicating that the data sets cannot be linked.

4.6 Related Work

As mentioned earlier, our work is closely related to the very active research areas of ontology matching, (database) schema matching, and record linkage. Here, we present a brief overview of some recent existing work related to ours. Chapter 2 provides a more detailed overview of the existing work in these areas.

In the schema matching area, our linkage point discovery technique can be seen as an novel instance-based schema matching mechanism for noisy semistructured data. Existing instance-based schema matching techniques often gather statistical knowledge about the instances to extract attribute properties such as type and distribution or to use more complex information theoretical measures to match the attributes [53, 117]. Such approaches are particularly useful for scenarios where matching attributes need to be found even if there is little or no overlap in the instances, which is not the case in record linkage. An example scenario is merging customer

databases from two different companies with very few common customers, different schema designs, but many matching attributes that cannot be identified using simple schema-level matching. Another motivation for such approaches instead of matching based on content is that some unrelated attributes such as product ID and customer ID may match (assuming the they both contain number values and their value sets have a large overlap), i.e., there could be many false positives. As our experiments on real data shows, such cases are less common in Web data, and avoidable using data characteristics such as cardinality, coverage, and strength.

Some other more recent instance-based matching techniques are either less scalable for large data sets (including those in our experiments) due to their use of expensive operators for matching instances, or their use of other domain-specific measures. Such measures are effective only in specific applications, for example, in matching web directories utilizing the URLs [71, 107, 134, 116]. Our method can be seen as an *all-to-all* instance-based matching that has been recognized as extremely useful but prohibitively expensive operation in the literature [150, Section 7, page 343]. Two closely related instance-based matching approaches are the work of Warren and Tompa [172] and iMAP [59]. They also use a search strategy to find schema correspondences, and take advantage of the information gained from the overlapping instances. However, we use a novel and highly efficient search strategy that treats the matching function as a black-box and uses specific measures to reduce the search space and improve the search results. This allows the careful study and evaluation of the search algorithms that we have performed in this work, and application on very large heterogeneous schema-free (Web) data sources as opposed to relational data with a small, well-designed set of attributes.

Our work can also be seen as a first step towards automatic discovery of linkage rules (or linkage specifications). To the best of our knowledge, this problem has only recently been studied for Web data, and as a part of SILK [108] and LIMES [143] RDF link discovery frameworks. These approaches use learning algorithms and rely on a number of manually-labeled link candidates. The SILK framework [108] uses a genetic programming paradigm to automatically learn linkage rules from a set of reference links. LIMES's RAVEN algorithm [144] has the advantage

of performing matching of classes and therefore does not require matching of entities of the same type. On the other hand, it also requires a number of manually-labeled link candidates to perform linking, and a user-specified threshold factor. Our approach is complimentary to such learning approaches as it can provide a number of candidate linkage points (as opposed to a single best-performing rule that uses a fixed set of attributes), and can make use of a larger set of instance values. This is also the reason we have not used the same data sets and linkage rules as previous work as ground truth to evaluate our framework. An important issue here is that even if source and target data describe the same entity types, different portions of the instances may require different linkage rules. For example, a small portion of data may contain identifiers that can be used to effectively discover links (i.e., are strong linkage points). Our experiments over real data confirm that our framework is capable of finding such linkage points.

4.7 Conclusion

In this chapter, we presented a framework for discovering linkage points over large collections of semistructured Web data. Our framework includes a library of lexical analyzers, a library of similarity functions, a set of novel search algorithms along with effective filtering strategies. We experimentally evaluated our framework in nine scenarios involving real Web data sources in three domains. The data sets used in our experiments along with the results are made available on our project's page [218]. We are currently exploring a number of interesting directions for future work. We are planning to further extend our similarity function library to include functions that take into account semantic similarity of the values in addition to syntactic and lexical matching. Our goal is to extend our system to publish the linkage points we discover using existing standards such as the recently proposed R2R mapping language [34]. In addition, we are planning to use the SMaSh algorithms to facilitate online analytics of enterprise data using external Web repositories as a part of the Helix project [93].

Chapter 5

Linking Semistructured Data on the Web[1]

5.1 Introduction

In the previous chapter, we made the assumption that there is a clear notion of an *entity type* in the data. We described application scenarios where the goal was to link entities of a certain type (e.g., company). On the other hand, a significant number of data sources on the Web use generic semistructured data formats such as JSON, XML or XML-based formats, or other industry-standard or domain-specific text-based formats such as BibTeX (and other bibliographic data formats), ID3 [203] (audio file data tagging format) or DrugCard [183] (drug information) to name a few. Although the use of such data formats has made data sharing and exchange significantly easier, they are often used without (an enforced) *schema* and clear *semantics*. As a result, both the data values and their structure may be inconsistent and contain errors, making it difficult for users to understand, query, and link the data source contents.

Consider the XML data shown in Figure 5.1 describing a clinical trial in ClinicalTrials.gov. Transforming this data into high-quality Linked Data involves multiple steps. First, we need to perform *entity type identification*, which involves detecting entities (or *resources*), their types (or *classes*) and attributes (or *properties*). This information can be derived from a reason-

[1] Part of this work has appeared in Proceedings of the Fourteenth International Workshop on the Web and Databases (WebDB 2011) [186].

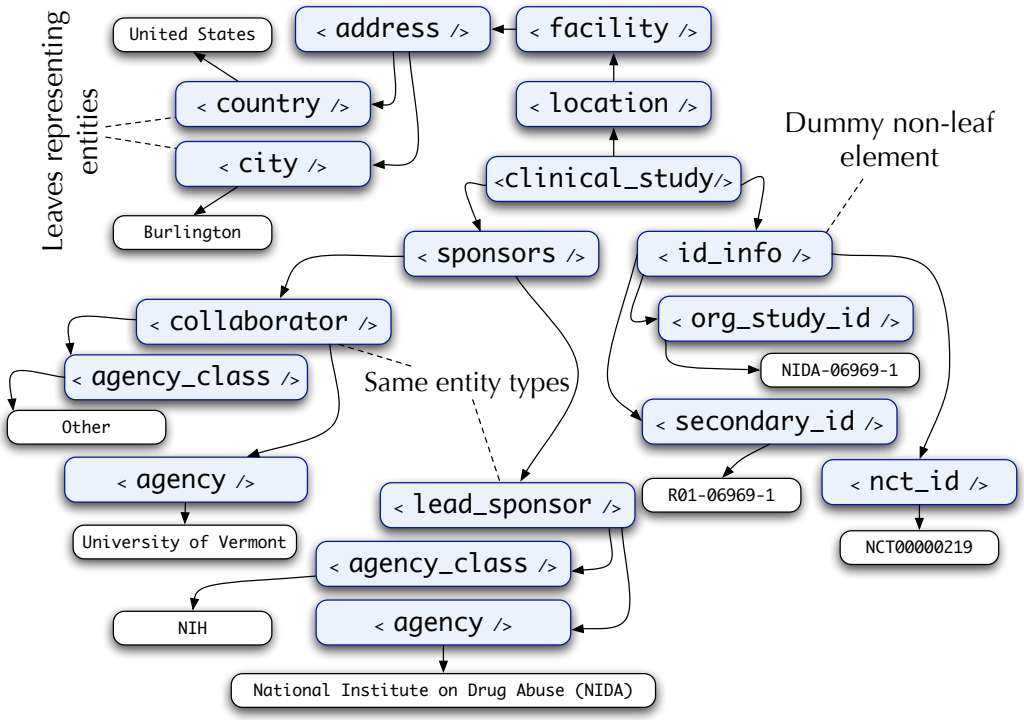


Figure 5.1: Sample XML elements from Clinical Trials

able sample of data. A simple heuristic might consider non-leaf elements as entity types (e.g., `<clinical_study/>`), while leaf elements containing only text (e.g., `<agency_class/>`) are translated into attributes of the parent entity type. However, simple heuristics like this will not always lead to high-quality *linkable* data. For instance, in Figure 5.1, there is a non-leaf, container element (`<id_info/>`) which does not represent an entity type, and there is a leaf element (`<country/>`) which is an entity type. It is important to ensure country values are represented as entities (not literal values) and can be linked to appropriate entities in the LOD cloud. Another challenge associated with automatic entity type identification is detecting *duplicate types*. In Figure 5.1, there are two identical subtrees in the structure (`<lead_sponsor/>` and `<collaborator/>`). These two subtrees are different instances of the same entity type, which may not be obvious at first glance. Such subtrees should be detected as identical types in order to avoid data duplication, and make it possible to properly link related entities.

After detecting entity types and their attributes, the data needs to be transformed into triples that describe the entities, their relationships and attributes. A major challenge in this transfor-

mation is the existence of duplicate instances. Such duplicates may result from (*i*) the existence of multiple occurrences of the same entity in different locations in a data set, (*ii*) different versions of the same data, and (*iii*) actual duplicates in the source. The problem is further complicated by the existence of *fuzzy duplicates*, i.e., entities that have different representations, but refer to the same real-world entity. For example, once we identify `<country/>` as an entity type, we need to detect whether two instances with labels "United States" and "U.S.A" refer to the same country, and therefore merge them into a single instance (entity).

The final step in this process is linking entities and their types to external knowledge repositories and data sources. There could be several ontologies and Web sources that contain information about our source entities and their types. Linking from entity types to external ontologies can enhance entity type identification and enhance the quality of the identified types, in addition to the benefit to entity identification. As an example, by matching leaf element `<country/>` with Freebase's `/location/country` or DBpedia's `dbpedia-owl:Country` entity types, we find additional evidence for our identified type, and we will be able to link the entity with label "U.S.A." with Freebase resource `/en/united_states` or DBpedia resource `dbpedia:United_States`.

Contributions. In this chapter, we present xCurator, a system capable of transforming semistructured data into high-quality Linked Data automatically with *optional* human intervention. Our contributions are threefold:

- We present an end-to-end framework for transforming a possibly large set of semistructured data instances into rich high-quality Linked Data. The input to the system can be instances of static semistructured data sources available on the Web, or dynamic user-generated content such as BibTeX entries, or RSS feeds.

- We present a brief overview of our implementation of each component of the proposed framework. We use or extend existing techniques from the data management literature to address the above-mentioned challenges.

- We present novel algorithms for identification of entity types (that are linkable) and their relationships out of semistructured data. Our proposed algorithms take advantage of the structure of the data to derive an initial set of entity types, and enhance the quality of the entity types in several steps in part using the power of external knowledge repositories.

- We report the results of applying our proposed framework to the transformation of several real-world data sets from different domains into rich Linked Data. The results include a data source of clinical trials generated from thousands of online XML descriptions. We evaluate the effectiveness of the proposed algorithms in identification of (linkable) entity types using these data sets. These results of the data transformation have been made available on the Web as a part of the Linked Open Data cloud.

In the following section, we present the architecture of our framework and a brief overview of related work for each component of the framework. Section 5.3 presents a detailed description of the entity type extraction process. We report the results of using our system in several real-world scenarios in Section 5.4. Section 5.5 concludes the chapter and presents a few interesting future directions.

5.2 Framework

Figure 5.2 shows the xCurator framework. The input to this framework is a semistructured data source, with no restriction on the characteristics of the data. For example, the data could be a single huge XML file stored locally containing all the DBLP publications, or can be URLs of millions of small online BibTeX files, each file containing one or more publication entries. The data could be static or dynamic, i.e., the data and its structure could change at any time. The output of the system is *high-quality* Linked Data, meaning: (*i*) objects that are identified by unique HTTP URIs; (*ii*) when objects are looked up, RDF statements are returned describing the object; (*iii*) the RDF statements link related source objects using predicates from existing or custom vocabularies; (*iv*) duplicate objects, i.e., objects that refer to the same real-world entity,

are identified and merged[1]; and (*v*) source objects are linked to objects in external online repositories that refer to the same or related real-world entities. In addition, the output data can be queried efficiently online using the standard SPARQL query language.

In what follows, we present an overview of different components of our framework. Almost all the problems discussed in this chapter have been studied in the past to some extent. We present a brief overview of related work in the literature as we explain different components of our framework, although a full discussion of all the related work is beyond the scope of this work. In terms of the overall framework, our work is related to systems that perform ontology learning for the Semantic Web (e.g., The OntoEdit ontology engineering workbench [132], Janus [16], and the work of An et al. [7]), where the goal is semi-automatic construction of an ontology from a given set of relational or XML sources and their schemas. In this context, our work can be seen as way of populating several existing ontologies using instances from multiple semistructured sources, in a lightweight approach geared towards creating a high-quality data source following the Linked Data principles. This is similar to what some RDB2RDF systems such as D2R Server [29] perform for relational data (where unlike in semistructured data the entity types are defined and fixed). Also related to our work is the Haystack system [118] that provides a powerful framework for management of semistructured data in the context of personal information management, without our focus on cleaning, deduplicating, and linking data to the LOD cloud.

5.2.1 Entity Type Extractor

This component is responsible for extracting entity types and their attributes and relationships from the input data source(s) in a (semi-)automatic manner. The final output of this component is a mapping definition for each entity type. Most semistructured data formats convey an implicit structure that one can leverage to identify the entities and their types in the data. Apart from the implicit structure, there are also several languages designed specifically to describe

[1] In certain cases, duplicate objects may be created, but linked with `sameAs` predicates.

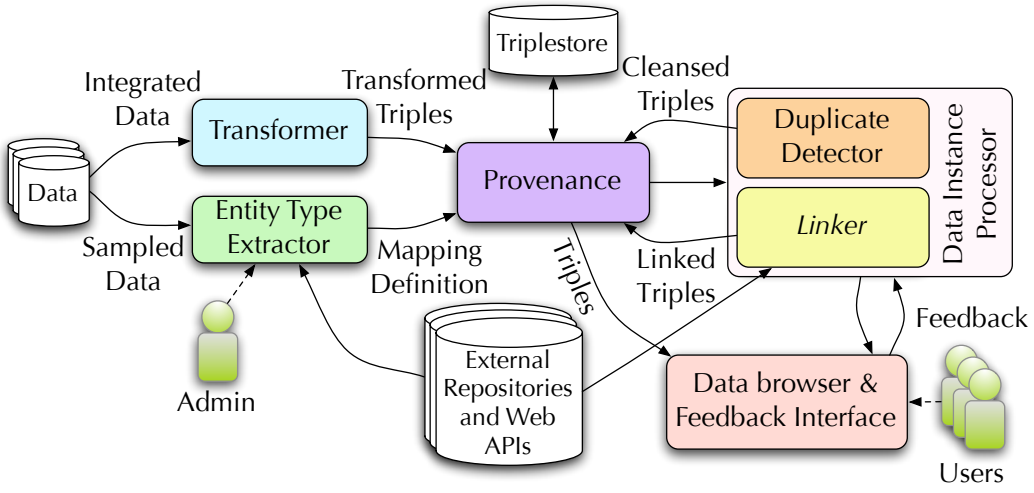


Figure 5.2: The xCurator Framework

the schema of the data, such as the widely used DTD, XSD, or RELAX NG schema languages for XML, or the more recent JSON Schema designed for JSON data. In the absence of such schema descriptions, the problem of understanding and extracting schema for unstructured and semistructured data has been studied extensively in the database literature. Several theoretical aspects of the problem have been studied [39, 140, 141], and some are implemented as a part of a data management tool and experimentally evaluated [19, 102, 118, 137]. Refer to Abiteboul et al. [1] for an overview of early work, and to Bex et al. [19] for a comparison of more recent work in this area.

In our work, unlike the work on schema inference, we are not concerned about deriving a valid schema that can be used to validate new instances and assist querying the data. Instead, we are interested in discovering the entity types, attributes and relationships, which can be used to generate high-quality Linked Data. Even if explicit schema definitions are given, we need to further process the schema as shown in the example in Section 5.1. In Section 5.3, we present the details of the entity type extraction process, which includes an initial structure extraction phase similar to the above-mentioned inference techniques, followed by refinement steps that enhance the quality of derived entity types.

5.2.2 Transformer

This component is responsible for transforming the source data into RDF triples based on the mapping definitions generated by the entity type extractor component. The mapping itself is represented and stored in XML with references to source data in XPath, which requires a simple data format transformation layer for sources with non-XML formats. There are several existing tools that transform XML or other semistructured data formats into RDF [206]. Some of these tools are only data format converters, e.g., they convert BibTeX or XML to RDF/XML without properly discovering entity types and their associations. Recently, the XSPARQL language has been proposed as a more concise and intuitive way of defining the mapping between XML and RDF [5]. There are also systems designed to perform transformation of XML data to Linked Data and RDF in domain-specific frameworks that are only capable of handling a fixed XML schema in a domain such as e-government [6] or bibliographic data [30]. One of the main goals of our work is to reduce the burden of having to manually define the mappings or hard-code the transformation procedure.

5.2.3 Provenance

Since data sources are evolving, the entities and their types, relationships and attributes will evolve over time. The Provenance component is responsible for maintaining information about the origins of the data, mapping definitions, and the data life cycle. In our framework, other components can only access the triple store through this component. It is important to note that for all the entities in our framework, we add provenance information as attributes of entities. The challenges involved in handling dynamic semistructured data such as provenance management and the evolution of structure are very similar to those of curated databases (refer to Buneman et al. [38] for an excellent overview of related work in this area). Information extraction systems (e.g., CIMPLE [58]) also need to handle evolving structure, and often expose provenance information to the users in a similar way.

5.2.4 Data Instance Processor

After generating triples, this component cleans the data and links the entities to the related entities in both internal and external Web repositories. Basically, this component searches for similar entities in source and external repositories. Then it eliminates duplicates by merging source entities of the same type that refer to the same real-world entity, and links similar entities of different types. The problem of finding duplicate records that refer to the same real-world entity has been extensively studied in the literature (cf. Chapter 2). We use and extend the results of Chapter 3 to enhance the data instance processing component of our framework. Briefly, we take advantage of the entity types extracted in the entity type extraction component and their links to external repositories to enhance the accuracy and efficiency of record linkage process by linking only the entities that have the same type (or link to entities that have the same type). Using entity types and their link, we can also take advantage of the linkage point discovery algorithms of Chapter 4 to enhance linkage to external sources.

5.2.5 Data Browse and Feedback Interface

As mentioned earlier, our goal is to publish the output data on the Web following the principles of Linked Data, and allow users to directly query the data by providing a public SPARQL endpoint. In the case of XML input data, we can alternatively (or additionally) embed the RDF data in XML using the standard GRDDL markup format [201].

In addition to existing RDF and Linked Data browsers that can be used to query and explore the data, we provide a custom data browser interface mainly with the goal of receiving feedback from users. User feedback is a critical requirement in our system. Entity type identification, duplicate instance detection and data instance linkage are all error-prone and inherently imperfect tasks. No matter how well they are performed, there will still be inaccuracies and inconsistencies in the data that can only be identified and removed using human intelligence. Our goal is to collect and incorporate feedback from users in an efficient and effective way. The data browser

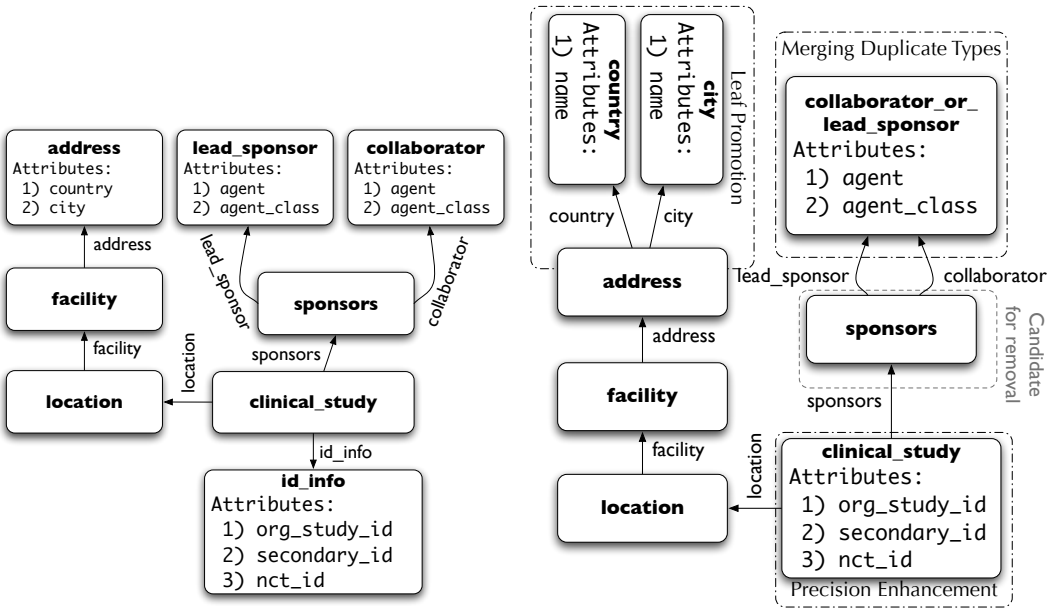
interface allows users to log in to our system using their existing OpenID (an open standard for user authentication), provide feedback on the quality of existing data and links, and report missing or erroneous values, duplicate instances, and new external links.

5.3 Entity Type Extraction

One of our main goals in xCurator is to provide a generic framework capable of transforming almost any type of semistructured data into rich Linked Data. To achieve genericity, we make no assumption on the availability of a predefined structure (or *schema*) for the input data. Even when such information is available, e.g., through XSD (XML Schema Definition) for XML sources, previous work has shown that for a considerable portion of the XML sources on the Web, either the XML documents are not valid based on the given schema definitions, or the schema definitions are not valid with respect to W3C standard definitions [20]. This calls for an automatic way of inferring the structure in the data, after which an initial set of entity types can be derived based on the inferred structure.

5.3.1 Basic Entity Type Extraction

As the first step towards extracting entity types, we can take advantage of the given hierarchy in the data. We do so by first building a *structure graph* from (a sample of) the input data. This graph is similar to the *schema graph* as defined by Abiteboul et al. [1], except that we do not create Schema nodes `Root`, `Any` or type nodes such as `string`. This graph is basically a concise representation of all the possible paths in the input data graph. The structure graph for the example XML data tree in Figure 5.1 can be derived by simply removing all its leaf nodes (literals). We then consider any non-leaf node in the structure graph as an entity type and the leaf nodes under each non-leaf node as the attributes of its corresponding entity type. Figure 5.3(a) portrays the result of applying this strategy to extract entity types for the structure shown in Figure 5.1. As shown in the figure, all the leaf elements (*e*.g., `<country/>` and `<agent/>`), are



(a) Basic Type Extraction (b) Final Entity Types

Figure 5.3: Entity type extraction for the example shown in Figure 5.1

mapped to attributes while non-leaf elements (*e*.g., `<sponsors/>` and `<collaborator/>`) are mapped to entity types. This approach is similar to finding Approximate DataGuides (ADGs) using suffix matching [83] or 1-Representative Objects [141]. Note that this can also be derived from an accurate DTD or XSD for XML data, if available.

In basic entity type extraction, we also find key attributes of the entity types, i.e., the attributes that their values can uniquely identify an entity. We do so using a simple map between attribute values and entities in the sample. Since only a sample is used, the identified keys are *approximate*. We only consider simple keys (consisting of only one attribute), since they will be used to enhance the entity type identification as explained below. Existing techniques can additionally be used to effectively identify composite keys out of the sample [161].

5.3.2 Entity Type Extraction Enhancement

We use several techniques to enhance the quality of the entity types discovered using the basic approach. These include methods to identify and remove duplicate entity types, enhance pre-

cision (*i*.e., identify those non-leaf nodes in the structure graph that should not be mapped to entity types) and improve recall (*i*.e., identify those leaf nodes that need to be mapped to entity types).

Duplicate Type Removal

A common problem with the basic entity type identification is that it could result in duplicate types, i.e., different entity types that represent the same real-world entity type, such as `<lead_sponsor/>` and `<collaborator/>` in Figure 5.3(a). To address this problem, we use a similarity function $f(t_1, t_2)$ for two entity types t_1 and t_2 that returns a similarity value ranging from 0 for no similarity to 1 for highest similarity. We then merge the two entity types t_1 and t_2 if $f(t_1, t_2) \geq \theta$ where θ is a user-defined threshold. This similarity function can be a simple set similarity measure such as the Jaccard coefficient between the sets of attributes and relationships of the two entity types. In Figure 5.3(a), the Jaccard coefficient between entity types `<lead_sponsor/>` and `<collaborator/>` is 1 and therefore they are merged.

Precision Enhancement

Another problem with the basic type extraction is that in reality, there could be non-leaf nodes in the structure graph that do not represent an entity type. Such nodes are usually created for readability or some sort of grouping of entities of the same type. The `<id_info/>` node in Figure 5.1 is an example of such a node, which is created just for grouping the identifiers of a trial. We detect such nodes based on the cardinality of the relationship of their parent node with them. If the relationship is *injective* (or one-to-one), it implies that removing the node and moving its attributes to its parent is possible, and lossless, meaning that we will not lose any semantic information as a result. For example, the `<id_info/>` type can be removed since there is at most one `<id_info/>` for a trial, but removing the `<collaborator/>` type can result in losing the relationship between the `<agency/>` and `<agency_class/>` attributes of a `<collaborator/>` since there could be several collaborators for each trial. Using this approach, it is also possible

to remove the <sponsors/> type since each trial has at most one <sponsors/> associated with it. However, as our experiments on real data (described in Section 5.4) show, users may want to keep such type since, for example, the <sponsors/> node may actually refer to an entity type that represents "sponsor groups" as opposed to a basic set of sponsors. Our system identifies such nodes based on the frequency of occurrence of each entity for each entity type in the data. In the above example, we keep <sponsors/> as an entity type due to the fact that several groups of sponsors frequently occur together.

Recall Enhancement

The other limitation of the basic type extraction method is that many leaf nodes in the structure graph need to be identified as entity types in order to facilitate querying, grouping, deduplication, and linkage. Our approach in xCurator is to identify such types using the power of external knowledge repositories. Our goal is to promote attributes as entity types only if we are able to add additional links from the instances of the entity type to external sources. This general rule has an exception. There are attributes with a very few distinct values (*e*.g., "Yes" and "No", or "Male" and "Female") and there are external entities (*e*.g., "FET intermediate yes or no answer" in the OpenCyc ontology) matching these literal values. For such cases, it makes sense to provide only links between the entity types and not their instances.

As shown in Algorithm 7, to find such attributes, we search a set of knowledge repositories for all distinct values of an attribute. This approach requires using a similarity measure denoted by M to compare the entities. This measure should be asymmetric, and preferably translatable to a SPARQL filter.[2] External types which match at least θ_L (linking threshold) fraction of distinct values are considered appropriate links. If we find such links, and the attribute is a key we add the links to the entity type containing the attribute since a key is a representative of the containing entity type. If the attribute is not a key, we promote it to an entity type if it has many distinct values, otherwise we do not promote the attribute.

[2]These characteristics can significantly improve the performance of recall enhancement in our system.

Algorithm 7: Recall Enhancement Algorithm

Input : a (The attribute), S (Triple repositories), M (The similarity measure), θ_M (The matching threshold), θ_L (The linking threshold), θ_V (The promotion threshold)

/* $V(a)$ is the set of distinct values for attribute a, L is the set of links for an attribute or an entity type, $K(t)$, $A(t)$, and $R(t)$ are the set of keys, attributes, and relationships for type t, respectively. */

1 $T' \leftarrow \varnothing$;
2 **for** $v \in V(a)$ **do**
3 **for** $r \in S$ **do**
4 **for** $(?s\ label\ ?o) \in r$ **do**
5 **if** $M(v, ?o) \geq \theta_M$ **then**
6 $T'[?t|(?s\ type\ ?t)] \leftarrow T'[?t] + 1$;
7 **end**
8 **end**
9 **end**
10 **if** $T' = \varnothing$ *and this is the first time for spell checking* **then**
11 $v \leftarrow spell\_check(v)$;
12 **goto** line 3;
13 **end**
14 **end**
15 $T \leftarrow \{t \in T' | T'[t] \geq \theta_L\}$;
16 **if** $T \neq \varnothing$ **then**
17 $t \leftarrow$ The entity type containing a;
18 **if** $a \in K(t)$ *or* $|V(a)| \leq \theta_V$ **then**
19 $L(t) \leftarrow T$;
20 **else**
21 $t_a \leftarrow$ create a new entity type based on a;
22 $A(t) \leftarrow A(t) - \{a\}$; $R(t) \leftarrow R(t) \cup t_a$; $L(t_a) \leftarrow T$;
23 **end**
24 **end**

Another common problem, especially in user-generated data, is the existence of misspellings and alternative representations in attribute values (*e*.g., misspelled city names in trial locations). To have a better data link quality, if we do not find any type for a value, we check for misspellings and look for alternative representations. This can be done using an internal dictionary or an automatically-generated domain-specific dictionary depending on the domain. In our current implementation, we take advantage of a Web-based spell checker API [189] to find alternative representations of the value on the Web, and then query the knowledge repositories using the returned value(s). This has proven to be more effective in our experiments as our source data also comes from the Web.

A well-known limitation of XML and similar hierarchical structures is in expressing relationships, which is a fundamental reason behind the existence of RDF. Detecting all potential relationships is crucial for xCurator. Therefore, we try to extract even implicitly denoted relationships. There are two styles for expressing relationships in XML and XML-based formats: (*i*) making the data self-contained (*e*.g., copying/repeating the whole proceedings data for each publication) which increases data redundancy, and (*ii*) implicit referencing where implicit identifiers are used for referencing other elements (*e*.g., using the ISBN of the proceedings for each publication). Relationships expressed using the first style are detected by the basic type identification while the latter (implicit referencing) cannot be detected without semantics. However, xCurator is capable of extracting those implicitly denoted relationships using Algorithm 8. This algorithm matches an attribute to all the keys of other entity types in order to find implicit self references. Note that efficient set-similarity join techniques can be used to make this algorithm scalable to very large data sets.

5.4 Experience and Evaluation

In this section, we briefly report our experience using the xCurator framework in transforming several data sets into rich Linked Data.

Algorithm 8: Self Linking Algorithm

Input : a (The attribute), T (The set of entity types), θ_L (The linking threshold)
Output: L (set of entity types which the attribute matches)
`/*` $f(a, b)$ `is a set similarity function such as the Jaccard coefficient */`

```
for t ∈ T do
    for k ∈ K(t) do
        if f(V(a), V(k)) ≥ θ_L then
            L ← L ∪ {t};
        end
    end
end
return L;
```

5.4.1 Clinical Trials Data

ClinicalTrials.gov is a large repository of clinical trials from all around the world, published by the U.S. National Library of Medicine (NLM). Currently, the data consists of more than $100,000$ trials from 174 countries, and is updated regularly. Each trial's data can be retrieved online as a single XML file. In 2008, we manually transformed the data into Linked Data as a part of the Linked Clinical Trials (LinkedCT) project. Manually transforming and updating the data has been extremely tedious and time-consuming, and one of the main motivations behind the xCurator project. We now have transformed and published the data on the Web using xCurator. Here, we report the result of the entity type identification for this data and the effect of sample size on the quality of the generated entity types. The details of the other components of the LinkedCT project are presented in Section 6.2.

The clinical trials data has a very complex structure, which makes it particularly suitable for evaluation of our entity type detection techniques. Similar to our evaluation in the previous chapter, we use well-known measures from IR, namely precision, recall, and F_1 measure to evaluate the accuracy of the set of entity types returned by the algorithms. We first show the effect of sample size on the entity type identification algorithm, and then compare the accuracy of the basic entity entity type extraction individually with each of the enhancements, and the final algorithm including all the enhancements. We have found the ground truth by manually verifying the result of the algorithms with a large sample size of $10,000$ trials, browsing through

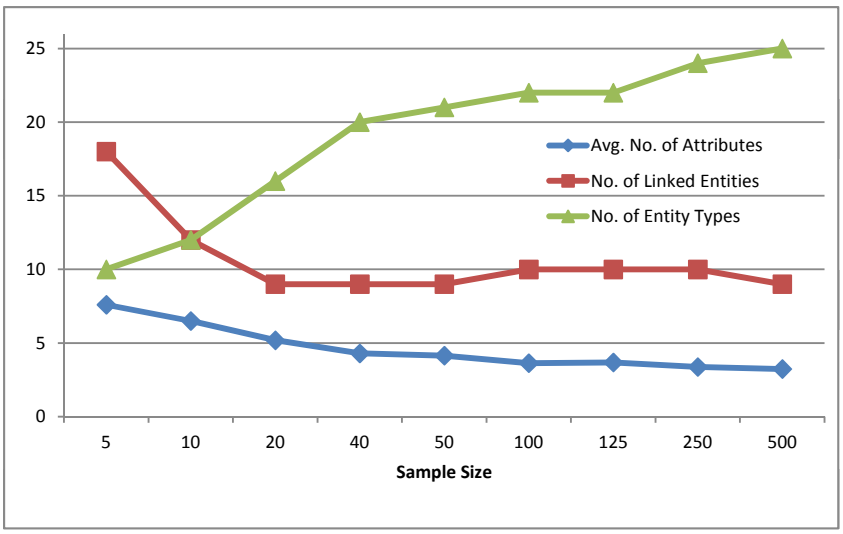


Figure 5.4: Effect of sample size on the number of entity types and average number of attributes per entity type

the data instances, and using our experience from the initial manual transformation and the feedback received from out users since then. The ground truth contains 26 entity types with average 3 attributes per entity type.

Figure 5.4 shows the effect of sample size on the number of detected entity types, average number of attributes, and number of linked entity types. As shown in the figure, xCurator generates almost flat structures (a few entity types having many attributes) when using small sample sizes since many relationships are found to be one-to-one and therefore many entity types are removed. Moreover, a very small sample size introduces many inaccurate external links since the number of distinct values for each entity type is limited. Clearly, the larger the sample, the more quality of the mapping. As we increase the sample size, more entities are extracted with less attributes, as more precise one-to-one relationships are detected. However, we observe that less than one percent of the data can be used to identify entity types very accurately, as shown below.

Figure 5.5 shows the F_1 values for each variant of the algorithm for sample sizes ranging from 10 to 500. It is clear that for sample sizes above 50, each of the enhancements considerably improve the accuracy comparing with the basic extraction. The performance of basic type

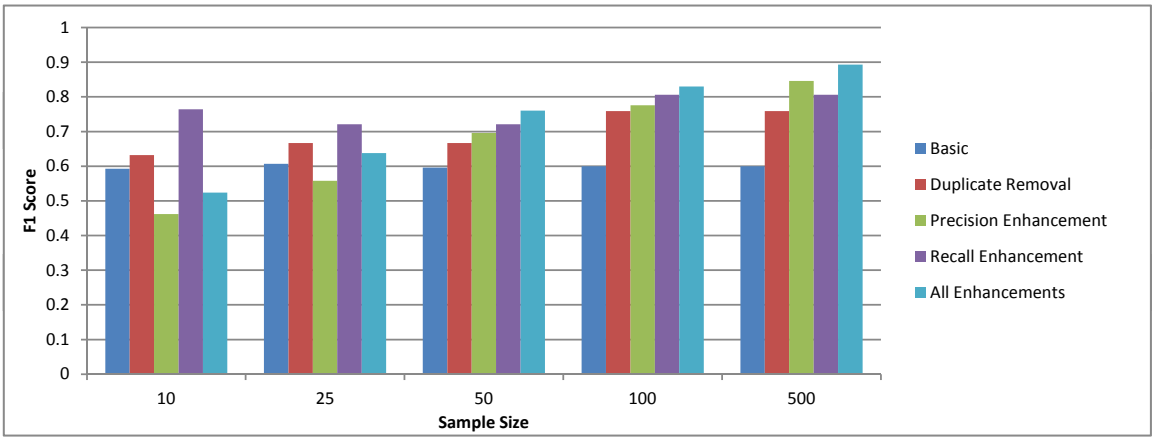


Figure 5.5: Accuracy of entity type identification on clinical trials data

extraction is not affected by the sample size. Duplicate type removal increases the quality of the detected entity types regardless of the sample size. However, precision and recall enhancements perform poorly on very small sample sizes below 25, due to small number of values that result in incorrect estimation of the cardinality of the relationships, and inaccurate links to external sources.

Figure 5.6 shows precision, recall and F_1 values for sample sizes ranging from 10 to 500 for each variant of the algorithm and our initial manual extraction. It can be seen that only the precision of our initial manual extraction outperforms the precision of the basic type extraction, since in our manual transformation, we have carefully picked only accurate entity types. The reason behind imperfect precision of the manual transformation is mainly due to our inability to identify duplicate types based on manually examining a sample of the instances and the schema. It is interesting to note that with even 0.1 percent of the data as sample, xCurator produces a mapping of much higher quality that our tedious manual mapping.

5.4.2 Bibliographic Data

DBLP

DBLP data set has been published as Linked Data by multiple sources. DBLP's XML data has a simple, flat structure and therefore is amenable even for manual extraction of entity types.

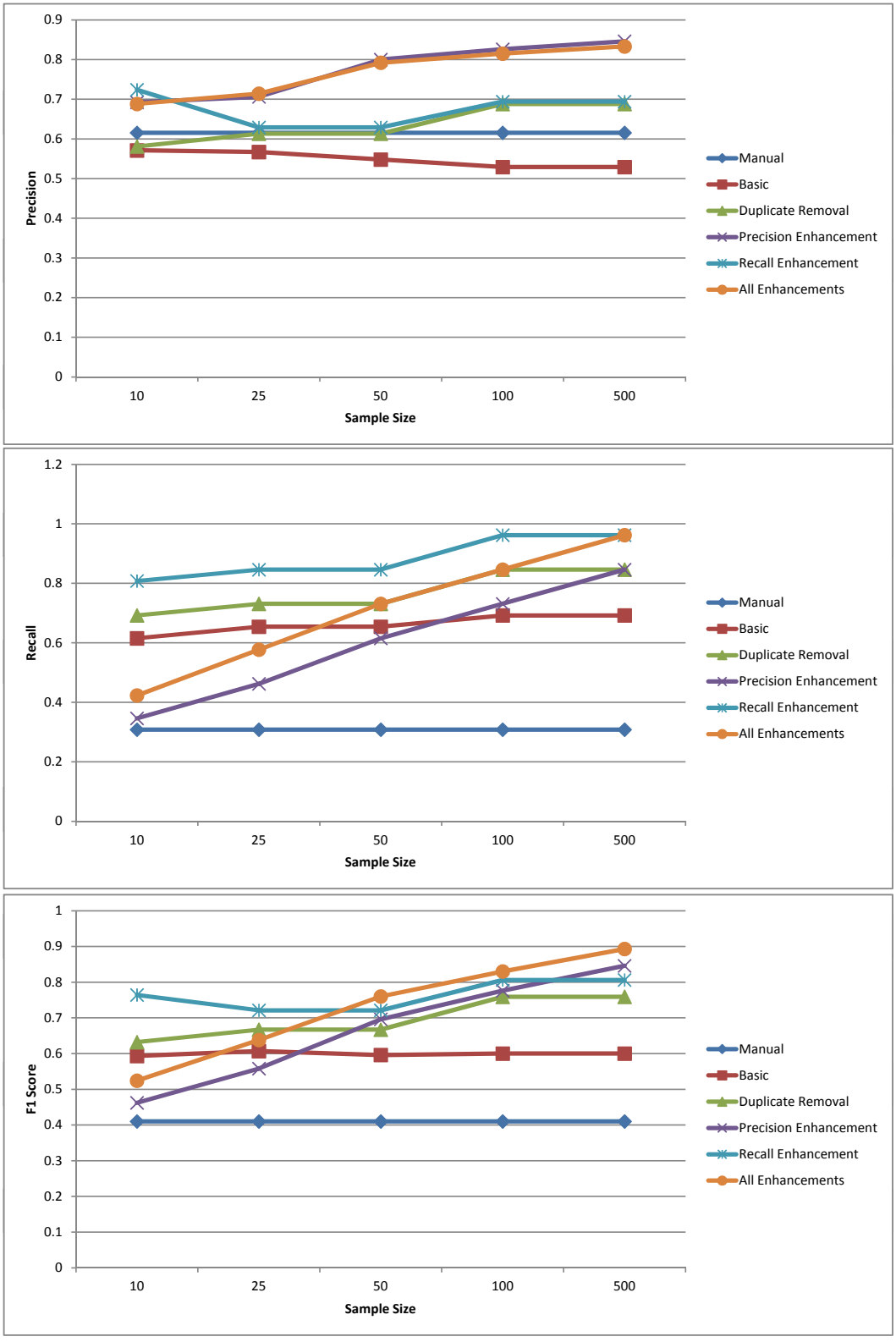


Figure 5.6: Effect of sample size on the accuracy of entity type identification on clinical trials data

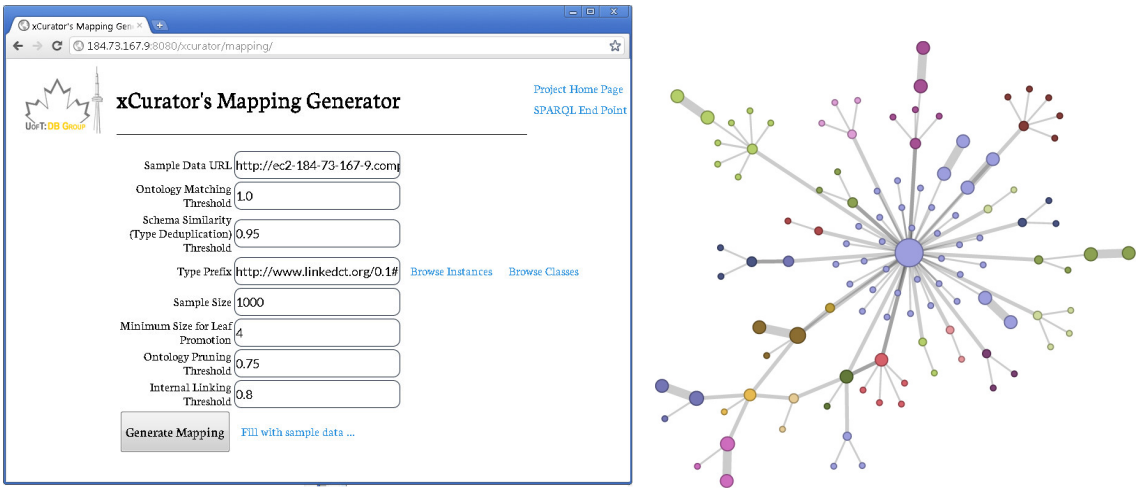
Despite such a straight forward structure, it has two implicit internal links which can be used to evaluate our system: (*i*) cross references to published collections (*e*.g., books and proceedings) and (*ii*) citations to other DBLP-indexed publications. We have performed experiments on identifying entity types which are internally linked together. The results of our experiments show that all publication entity types are properly linked to the entity type for respective collection(s) without manual intervention. For instance, triples about "inproceeding"s are linked with according proceedings. Moreover, our system is able to interlink DBLP data with the related topics in Freebase.

BibBase

BibBase started in 2005 at the University of Toronto as a Web service that transforms scientists and research groups' BibTeX files into good-looking HTML pages with several features such as the ability to use custom style files, group publications based on different attributes, and providing RSS feeds. Recently, we transformed users' BibTeX files into high-quality Linked Data using in part the capabilities of the xCurator framework. The details of this transformation are presented in Section 6.3. The major application of the xCurator framework for this data source has been the data instance processing and the data browse interface. User-generated BibTeX files have various types of quality issues, such as abbreviating author names and using alternative author and conference names. Our Linked Data source currently provides automatic duplicate detection and linkage to external sources. In additional, a group of users are provided with access to the feedback mechanism and are able to report duplicates and external links, and provide feedback on the quality of automatic deduplication and linkage. The results are available online at `http://data.bibbase.org`.

5.4.3 Mapping Transformation Interface

In addition to the above sources, we provide a lightweight Web interface for users to create custom mappings from (a sample of) their XML data, and transform their data into RDF using



(a) Mapping Generator Interface (b) LinkedCT Structure Graph

Figure 5.7: Mapping transformation interface and LinkedCT structure graph

the custom mapping. This web interface publicly exposes many features of xCurator to interested users. Users' RDF data will be stored in the xCurator internal triple repository while being accessible by the users. Figure 5.7(a) shows a snapshot of the mapping generator Web interface, and Figure 5.7(b) portrays the final schema generated by xCurator for LinkedCT data as shown on the interface. The central node in this figure is the `clinical_study` entity type, and bold border edges are external links. This fine structure is generated with no human intervention.

5.5 Conclusion

In this chapter, we presented a modular framework for transforming a semistructured data source into high-quality Linked Data. We described briefly different components of the proposed framework, and the entity type extraction component in more detail. We showed how it can be used for management of several real-world semistructured sources. We are currently investigating several future directions. First, we are planning to further investigate the application of our framework on real-world data sets from various domains. Some applications require additional data format conversion and using more specialized Web repositories for type detection and linkage. We are also in the process of experimentally evaluating the effect of input

parameters on the quality and performance of each component in our framework. Another interesting area we are investigating is effective generation of representative samples from a given data source that result in more efficient mapping generation and could also assist users of the system in updating the given mapping. The results of our evaluation, the data sets used, and the mapping generator interface are available online on our project page [219].

Chapter 6

Creating High-Quality Linked Data: Case Studies

In this chapter, we describe three Linked Data sources that have been used in the design and evaluation of the frameworks described in the last three chapters. Each of these data sources are from a different domain, with different requirements for data transformation, entity identification, record linkage, and data publication. In what follows, we describe an overview of the initial design and features of the data publication frameworks we have developed for each data source, and how they have leveraged the results described in previous chapters. We also provide a brief overview of the applications of these sources, and our future plan in using them to extend and evaluate the work presented in this thesis.

6.1 Linked Movie Data Base[1]

Movies are highly popular on the Web. There are several web resources dedicated to movies and many others containing movie-related information. Creating a single source of information about movies that contains information from existing open web data sources and links to

[1]Part of this section has appeared in Proceedings of the WWW2009 workshop on Linked Data on the Web (LDOW2009) [92].

other related data sources is a challenging task and the goal of the Linked Movie Data Base (LinkedMDB) project. LinkedMDB provides a high quality source of RDF data about movies (http://linkedmdb.org) that appeals to a wide audience, enabling further demonstrations of the linked data capabilities. Furthermore, LinkedMDB demonstrates the value of a novel class of tool to facilitate high volume and dense interlinking of RDF data sets.

Figure 6.1 shows an example of the entities and the interlinking in LinkedMDB. There are several challenges involved in identification of the entities in different data sources that should be interlinked. In some cases, the access to the data in target data source is limited. For example, only the title of the movies with their associated URLs can be obtained from the data source. In such cases, we need to use the string similarity measures described in this thesis to match movie titles. For example, the weighted Jaccard measure along with q-gram tokens could be used to match "The Shining" in LinkedMDB to the movie title "The_Shining_(film)" in DBpedia. Alos, many non-English movie titles have different spellings in English, e.g., the titles "Adu Puli Attam" and "Sacco and Vanzetti" in LinkedMDB are written as "Aadu_Puli_Aattam" and "Sacco_e_Vanzetti" in DBpedia that require using approximate matching using a string similarity measure.

However, exact or approximate matching of movie titles could results in false matches. The movie "Chicago" (1927 movie) would link to the movie "Chicago" (2002 movie) using exact matching. By approximate matching, movie titles "Spiderman 1" and "Spiderman 2" have similar titles but are not the same. There is a similar case for the movies "Face to Face" and "Face to Fate", and some adult movies that have names very similar to popular Hollywood movies. Although using string similarity measures and specific record matching techniques (e.g., using additional structural and co-occurrence information as discussed in Section 2.2) could significantly reduce the amount of false matches, achieving 100% accuracy is not always possible. Also, higher accuracy may result in fewer correct links, as we will show later in this section. Therefore, we propose a framework in which *metadata* about the links and how are they are obtained is published along with the links. This approach has several advantages. The users will be able to

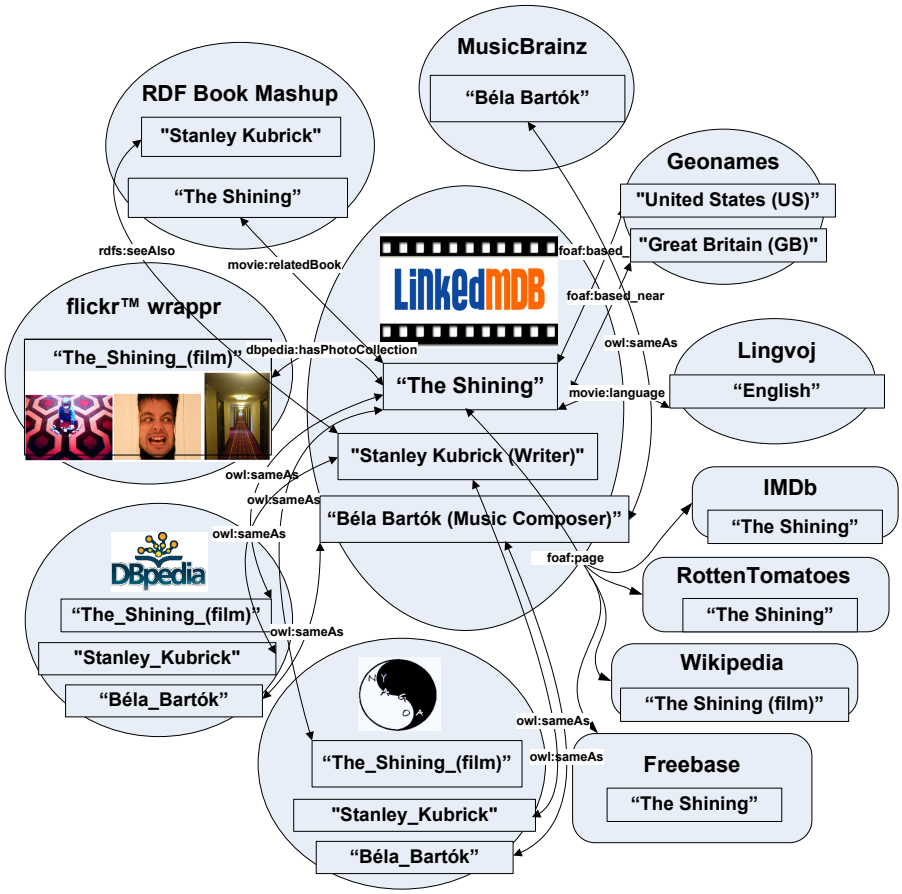


Figure 6.1: Sample LinkedMDB entities

determine the type of the links and level of accuracy depending on the application. Publishing such metadata will help users understand the meaning and quality of these automatically created links and distinguish them from links created manually by users.

In what follows, we present an overview of the data transformation task in LinkedMDB. We then overview the interlinking of the data sources, and provide a brief overview of how LinQuer (cf. Chapter 3) was used for record linkage. We present an evaluation of the performance of some of the linkage methods we used for LinkedMDB. The need for linkage metadata and our approach for providing such data in LinkedMDB is discussed. We conclude the section by a brief discussion of a few future directions.

6.1.1 Data Transformation

Data sources

Currently there are several sources of information on the web about movies:

- IMDb is the biggest database of movies on the Web that provides a huge variety of up-to-date information about movies. Although IMDb data is available for download and personal use, it is strongly protected by copyright laws. We did transform the IMDb data to RDF, however we could not get permission to publish it and therefore our implementation does not include any information from IMDb although we include external links to IMDb pages whenever possible.

- Freebase is an open, shared database of knowledge. The "film" category of Freebase is one of the biggest and most complete domains in this database with more than 38,000 movies (as of May 2009) and thousands of other data items related to movies. Freebase has open data and has recently made its data available for download. Therefore, we use Freebase as the nucleus of our database, although we do not limit our data source to the information available on Freebase.

- OMDB is another open data source of movies. The data set currently contains information about more than 9,000 movies, and its data is available for public use.

- DBpedia contains a large amount of information about more than 36,000 movies (as of May 2009) and thousands of related data items. We provide owl:sameAs links to DBpedia.

- RottenTomatoes.com is another movie website with information about movies. RottenTomatoes data is not available for download and public use, however, we include `foaf:page` links to RottenTomatoes website as well.

- Stanford Movie Database is a free database of movie information initially provided as a real test data for students. This database is relatively old, last updated in November 1999.

Table 6.1: Overall statistics

Total number of triples	3,579,616
Number of interlinks to LOD cloud	162,199
Number of links to movie websites	271,671
Number of entities in LinkedMDB[2]	233,103

Table 6.2: Sample entities

Entity	Count
Film	38,064
Actor	29,361
Director	8,367
Writer	12,990
Producer	9,637
Music Contributor	3,995
Cinematographer	2,169
Interlink	162,199

Therefore it includes only a few data items that are not present in FreeBase. We however plan to extend our database with the additional information that can be obtained from this source.

Entities and Facts

Our database currently contains information about several entities including but not limited to movies, actors, movie characters, directors, producers, editors, writers, music composers and soundtracks, movie ratings and festivals. Table 6.2 shows the statistics for major entities in LinkedMDB.

6.1.2 Record Linkage

LinkedMDB provides links to several Linking Open Data (LOD) cloud data sets. Among these links are links to DBpedia, YAGO, flickr-wrapper, Geonames and lingvoj. Moreover, several data items are linked to external web pages such as pages on Freebase, IMDb, OMDB, Rotten-Tomotoes and Wikipedia.

LinkedMDB is connected to the following LOD data sources:

- DBpedia/YAGO: Apart from the movie titles, person names (such as actors, writers and composers) are linked the related resources in DBpedia and YAGO data sources with `owl:sameAs` links.
- Geonames: We interlink the countries of the movies to Geonames data set by `foaf:based_near` type of links. This is done by matching name of the countries in the two data sets. These links could be extended by matching featured locations of the movies to Geonames items.
- FlickrWrapper: The moviess are linked to their photo collections using FlickrWrapper web service. These links are derived from the corresponding DBpedia URIs of the movies.
- RDF book mashup: Movies are linked to their related stories.
- Musicbrainz: Music composers and soundtracks are linked to muzicbrainz.
- Revyu.com: Movie reviews on Revyu can be linked to movies (and vice versa).

Apart from links to LOD data sets, we also have setup `foaf:page` links to external webpages:

- Freebase.com pages.
- IMDb.com movies and actor profiles.
- RottenTomatoes.com movie information and reviews.

Other potential links include links to external webpages from OMDB, boxoffice and movie show-times website and also homepages of the movies.

Methodology

In LinkedMDB, several links to other data sources are found using string matching native methods in LinQuer. In order to match movie titles between two sources to discover `owl:sameAs` links, we perform the following steps:

Table 6.3: External linkage statistics

Target	Type	Count
DBpedia	`owl:sameAs`	30,354
YAGO	`owl:sameAs`	30,354
flickr wrappr	`DBpedia:hasPhotoCollection`	30,354
RDF Book Mashup (Books)	`movie:relatedBook`	700
RDF Book Mashup (Authors)	`rdfs:seeAlso`	12,990
MusicBrainz	`owl:sameAs`	2,207
GeoNames	foaf:based_near	27,272
GeoNames	`owl:sameAs`	272
lingvoj	`movie:language`	28,253
IMDb, RottenTomatoes, Freebase.com	`foaf:page`	271,671

- We choose one source as the base source and the other as the query source. We pre-process the base and query source using LINKINDEX statements in LinQuer.

- For each string in the query source, we find all those strings in the base source which have similarity score above a threshold θ by a simple LinQL query.

- If there is only one string matched, then we output the query and base strings as certain matches. If there is more than one string or no string with similarity score above θ with the query string, then we do not output a match for the query string.

Evaluation

We provide a summary of the evaluation of the accuracy of the linkage in one of the linkage scenarios. In our experiments we used $q = 2$ for generating q-grams as it showed better performance comparing with other values of q. Here, we present brief accuracy results for matching movie titles from DBpedia to movie titles in our database. We matched 38,064 movie titles in our database with 25,424 movie titles from DBpedia using the similarity predicates described above. We manually inspected different thresholds to find the optimal threshold. Table 6.4 shows the number of matches obtained with different values of the threshold as well as the accuracy obtained. Note that accuracy reported is the precision of the links, i.e., percentage of the

Table 6.4: The number and the accuracy of the owl:sameAs links discovered between LinkedMDB and DBpedia using different linkage methods and similarity thresholds

Measure	Threshold	#Total	#Wrong	Accuracy	Measure	Threshold	#Total	#Wrong	Accuracy
Weighted Jaccard	0.6	12,756	693	94.57%	Cosine w/tf-idf	0.6	24,549	7,189	70.72%
	0.65	9,823	350	96.44%		0.65	21,068	4,546	78.42%
	0.7	7,130	169	97.63%		0.7	17,623	2,541	85.58%
	0.75	4,874	86	98.24%		0.75	14,082	1,243	91.17%
	0.8	3,018	36	98.81%		0.8	10,671	571	94.65%
	0.85	1,913	15	99.22%		0.85	7,169	197	97.25%
	0.9	1,505	6	99.60%		0.9	3,886	61	98.43%
Jaccard	0.6	10,476	785	92.51%	BM25	0.6	7,889	1,258	84.05%
	0.65	7,798	353	95.47%		0.65	5,565	695	87.51%
	0.7	5,545	189	96.59%		0.7	3,760	407	89.18%
	0.75	3,909	104	97.34%		0.75	2,485	254	89.78%
	0.8	2,117	43	97.97%		0.8	1,658	160	90.35%
	0.85	1,531	25	98.37%		0.85	1,092	98	91.03%
	0.9	1,432	7	99.51%		0.9	715	67	90.63%
Edit Similarity	0.6	16,137	3,260	79.80%	HMM	0.6	3,737	374	89.99%
	0.65	12,550	1,219	90.29%		0.65	2,396	226	90.57%
	0.7	9,423	519	94.49%		0.7	1,534	154	89.96%
	0.75	5,848	227	96.12%		0.75	992	92	90.73%
	0.8	2,719	93	96.58%		0.8	646	61	90.56%
	0.85	1,043	7	99.33%		0.85	447	45	89.93%
	0.9	334	4	98.80%		0.9	306	35	88.56%

output links that are correct. The recall is hard to find since the correct number of matches is not known. However, the number of links returned reflects the value of recall. To determine if a link was correct, we used a set of rules for matching in this scenario. For example, all underscores are replaced with whitespaces, and the substring "(film)" is removed from the DBpedia movie titles. These rules themselves are discovered by running the similarity join and manually inspecting thousands of the links returned.

The results in Table 6.4 show that the weighted Jaccard similarity outperforms other predicates in this scenario in terms of the number of correct links found. Based on these results we chose threshold $\theta = 0.7$ with weighted Jaccard similarity for the existing links in our database.

6.1.3 Publishing Linkage Meta-data

As shown in the accuracy evaluation in previous section, although using proper string matching techniques significantly reduces the amount of false matches, achieving 100% accuracy is not always possible or may result in fewer correct links (i.e., lower recall). In our framework, we included *metadata* about the links and how are they are obtained. In LinkedMDB, we provided two entities, namely `interlink` and `linkage_run`, for this purpose. Figures 6.2 and 6.3 show examples of these entities. This approach has several exciting advantages. The users will be able

Property	Value
rdfs:label	1036 (Interlink)
oddlinker:link_source	<http://data.linkedmdb.org/resource/film
oddlinker:link_target	<http://dbpedia.org/resource/The_Shini
oddlinker:link_type	owl:sameAs
oddlinker:linkage_run	<http://data.linkedmdb.org/resource/link
oddlinker:linkage_score	0.567848166224181
movie:linkid	1036 (xsd:int)
rdf:type	oddlinker:interlink

Figure 6.2: Sample `interlink` entity

Property	Value
oddlinker:linkage_date	7-7-2008
oddlinker:linkage_method	WeightedJaccard
is oddlinker:linkage_run of	<http://data.linkedmdb.org/resour
is oddlinker:linkage_run of	<http://data.linkedmdb.org/resour
is oddlinker:linkage_run of	<http://data.linkedmdb.org/resour
is oddlinker:linkage_run of	<http://data.linkedmdb.org/resour

Figure 6.3: Sample `linakge_run` entity

to determine the type of the links and level of accuracy depending on the application. Furthermore, this will facilitate the process of judging the quality of the links by the users and allow users to provide feedback on the quality of the links.

6.1.4 Impact and Future Directions

LinkedMDB started in June 2008, and shortly after received significant attention from the Linked Data community with the first prize at the 2008 "Linking Open Data Triplification challenge" [91], an annual contest that rewards the most promising Linked Data sources and applications each year. Since then, LinkedMDB has been used in various research projects and applications. A search on Google Scholar reveals that as of November 2011, LinkedMDB has been used or mentioned in over 100 research articles. Some example research applications include

experiments on User Experience (UX) [80], and research on exploration and summarization of RDF data [120]. LinkedMDB data has also been used to build a Linked Data Movie Quiz [205], developed in less than 10KB of code to take part in a contest on lightweight, well-optimized Web applications (under 10KB).

In the future, we are planning to 1) apply the results of Chapters 3 and 4 to extend and improve the amount and quality of the links to external sources, and 2) automate the data transformation and publish more meta-data and provenance information using the xCurator framework (Chapter 5). Our plan is to provide an easy-to-use interface to allow users to provide feedback on the quality of the links. In this way, users will only need to report the quality of the links as opposed to manually providing the links, as proposed in *User Contributed Interlinking* framework of Hausenblas et al. [99].

6.2 Linked Clinical Trials

Clinical trials data are used extensively in drug development and clinical research [13]. The data collected in trials are widely used by a variety of different individuals and organizations for different purposes. Creating a single web data source of clinical trials data that is highly interlinked with existing medical data sources and that can be queried with semantically rich and complex queries, is the goal of the Linked Clinical Trials (LinkedCT) project. Such a web data source could significantly enhance the discovery of clinical trials in order to match patients with trials, perform advanced studies and enable *tailored therapeutics* [112].

The LinkedCT data source (`http://linkedct.org`) is published according to the principles of publishing Linked Data. Each entity in LinkedCT is identified by a unique HTTP dereferenceable Uniform Resource Identifier (URI). When the URI is looked up, related RDF statements about the entity is returned in HTML or RDF/XML based on the user□s browser (request method properties). Moreover, a SPARQL endpoint is provided as the standard access method for RDF data.

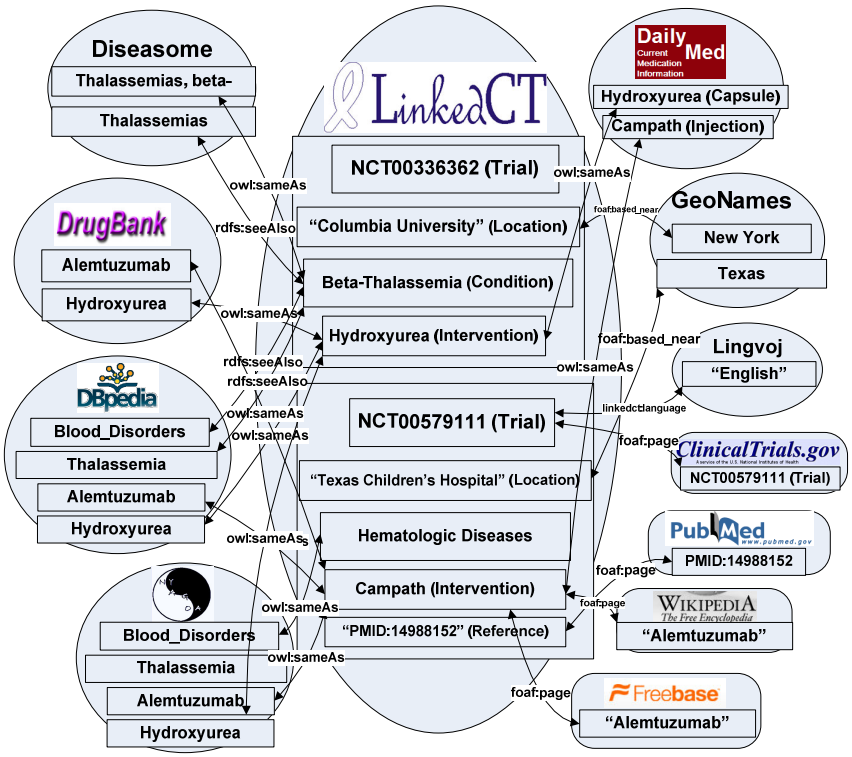


Figure 6.4: Sample LinkedCT entities

To understand the linkage requirements of this application, consider the example entities in Figure 6.4. The entity "Columbia University" of type "Location" is linked with "New York" using a link of type `based_near` and the disease entity "Beta-Thalassemia" is linked with "Blood Disorders" with link type `is_a_type_of`. In practice, the type of the links is usually specified using existing vocabularies, for example, `owl:sameAs` specifies that two entities are the same, `rdfs:seeAlso` specifies that the target resource may contain additional information about the source, and `foaf:based_near` specifies that the source entity is located in the target resource (which is of type location).

6.2.1 Initial Data Transformation

In this section, we outline the linked data generation (or triplification) process of trials data in our initial release, which was done without the xCurator framework (Chapter 5). The core of the LinkedCT data is derived from ClinicalTrials.gov, a registry of clinical trials conducted

Table 6.5: Overall statistics in the initial transformation

Total number of RDF triples	7,011,000
Number of Entities	822,824
Number of links to external linked data sources	308,904

in the United States and around the world. In our initial transformation in November 2008, this registry contained information about more than 60,000 trials conducted in 158 countries. Each trial is associated with a brief description of the trial, related conditions and interventions, eligibility criteria, sponsors, locations, and several other pieces of information. The data on ClinicalTrials.gov is semistructured and is available in HTML and XML formats. Transforming the data into RDF involved several steps. First, the XML collection of the trials was crawled and stored locally. This resulted in a collection of over 60,000 XML files. Second, the XML data was mapped into relational format. The mapping was obtained by manually inspecting the data and the Document Type Definition (DTD) of the data. In addition to all the entities and facts in the data, the mapping also extracted the relationships between the entities and their types. The relational schema was reasonably normalized to ensure performance and avoid update anomalies when the data is refreshed. For this transformation, we used a hybrid relational-XML DBMS (IBM DB2) to create relational views over the XML data, and to materialize the views to enhance the performance.

Finally, the relational data was mapped into RDF format. We used the popular D2RQ tool for transforming the relational data to RDF [35] and the companion D2R server [29] for publishing the relational data as a linked data source. Table 6.5 shows the overall statistics of the data and Table 6.6 shows the statistics of some of the entities in the data space.

6.2.2 Record Linkage

After the triplification of the trials data, we are faced with the challenge of finding useful links between the data in LinkedCT and other data sources. We have interlinked LinkedCT with several linked data sources of medical information: DBpedia and YAGO (linked data sources

Table 6.6: Entities in the initial transformation

Entity	Count
Trials	60,520
Condition	14,243
Intervention	67,271
Location	222,115
Collaborator Agency	7,071
Overall Official	52,660
Primary Outcomes	55,761
Reference	45,110
Criteria	73,688
Total	822,824

derived from Wikipedia articles, containing many useful pieces of information about diseases and drugs), DailyMed (published by the National Library of Medicine, providing high quality information about marketed drugs), Diseasome (containing information about 4,300 disorders and disease genes linked by known disorder-gene associations for exploring known phenotype and disease gene associations and indicating the common genetic origin of many diseases), DrugBank (a repository of roughly 5,000 FDA-approved drugs), Bio2RDF⊠s PubMed (Bio2RDF project⊠s linked data version of PubMed), GeoNames (linked data source of geographical information) and Lingvoj (information about languages). Also, the trials, conditions, interventions, locations and references on LinkedCT have been linked to their related web pages on ClinicalTrials.gov, Wikipedia, Freebase and Pubmed using foaf:page links. Table 3 shows the statistics of the external links in LinkedCT.

Evaluation

We show the effectiveness of semantic link discovery using string and semantic matching techniques described above using five linkage scenarios in LinkedCT. Table 6.7 shows the summary of the results. The columns "Link #" show the number of links discovered using exact matching, string matching, semantic matching and the overall number of links. Matching each entity in LinkedCT with the target data source may result in several links. For example, if there are 10 trials on condition "AIDS", and "AIDS" is matched with its corresponding resource on DBpe-

Table 6.7: Statistics of the links discovered using string and semantic matching

Source < Target	Exact Match		String Matching				Semantic Matching				Overall			
	Link #	Linked Entity #	Link #	Diff %	Linked Entity #	Diff %	Link #	Diff %	Linked Entity #	Diff %	Link #	Diff %	Linked Entity #	Diff %
Intervention <-> DBpedia (drug)	8,442	867	9,716	+15.1%	1,558	+79.7%	10,630	25.9%	1,334	+53.9%	11,527	+36.5%	1,913	+120.6%
Intervention <-> DrugBank (drug)	9,867	926	11,865	+20.2%	1,938	+109.3%	12,127	+22.9%	1,641	+77.2%	23,493	+138.1%	2,477	+167.5%
Intervention <-> DailyMed (drug)	14,257	673	24,461	+71.6%	1,296	+92.6%	27,685	+94.2%	1,099	+63.3%	39,396	+176.3%	1,688	+150.8%
Condition <-> DBpedia (disease)	164	164	333	+103.0%	330	+101.2%	173	+5.5%	173	+5.5%	342	+108.5%	342	+108.5%
Condition <-> Diseasome (disease)	232	192	778	+235.3%	575	+199.5%	301	+29.7%	247	+28.6%	830	+257.8%	615	+220.3%

dia, then 10 links are added to the total number of links. The columns "Linked Entity #" show the number of distinct entities in LinkedCT that are linked. The "Diff %" columns show the improvement over the exact matching of the strings. Note that since the links are automatically discovered, some may be inaccurate. These can best be filtered by a user from the query results. The benefit of using automatically discovered links is that a very large number of links is achieved at the cost of some precision. In our case, manually inspecting a random subset of the links in each of the scenarios showed above 98% precision on average in all the scenarios. Assuming that the precision observed in the small verified sample represents the overall accuracy, the number of the links discovered reflect the recall of the discovered links. Nonetheless, careful verification of this assumption is a difficult task and the topic of future work (cf. Section 7.2.4).

The results in Table 6.7 clearly show the effectiveness of both string and semantic matching in all these linkage scenarios. Using semantic knowledge has resulted in more links between intervention and drugs as compared to conditions and diseases since drugs tend to have more semantic equivalences (brand names and generic names). The use of semantic matching however is useful particularly for matching disease names that are abbreviated. The overall numbers of the links show that although there is some overlap between the links discovered using semantic matching and string matching, there is a significant number of links that are not found using a single matching technique.

6.2.3 LinkedCT Live: Online Data Transformation

The initial data transformation process of LinkedCT was extremely time-consuming and error-prone. The data could be updated at most every 6 months, despite the ClinicalTrials.gov's condition of keeping the data up-to-date at all times for re-publication. Also, our users often asked for changes in in the schema, and pointed out missing or erroneous values, while the source of the errors were almost impossible to find since provenance information was not initially kept. Recently, and using in part the power of the xCurator framework, we re-implemented the LinkedCT transformation framework to perform online transformation of the XML data. The technical details of the framework and the entity type extraction algorithms are discussed in Chapter 5. As of November 2011, LinkedCT contains more than 116,000 trials, almost double the initial number of trials. The overall number of entities and RDF triples have grown even larger due to the higher-quality transformation, with over 2.9 million entities (three times the initial number), and over 27 million RDF triples (almost 4 times the initial number).

6.2.4 Impact and Future Directions

LinkedCT is the largest and highest-quality data source published as a part of the Linking Open Drug Data Task Force of W3C's Semantic Web for Health Care and Life Sciences (HCLS) Interest Group, which won the first prize at the Linking Open Data Triplification Challenge contest in 2009 [113]. Researchers and application developers have shown great interest in the data, and have developed several Linked Data application scenarios such as a competitive intelligence scenario in the pharmaceutical industry developed by Jentzsch et al. [112], and extraction of meaningful networks and patterns in data as a part of the LinkedDataLens project [82, 81]. We are also planning to build a number of interesting applications on top of LinkedCT, including the first location-aware search for clinical trials built on top of Google Maps interface, which becomes possible only as a result of the added structure and semantics in the LinkedCT data that was made possible by the use of xCurator.

6.3 BibBase Triplified: Linking Bibliographic Data on the Web[3]

Management of bibliographic data has received significant attention in the research community. Many online systems have been designed specifically for this purpose, including but certainly not limited to, BibSonomy [194], CiteSeer [195], CiteULike [196], EPrints [199], Mendeley [207], PubZone [202], refbase [220] and RefWorks [209]. The work in the semantic web community in this area has also resulted in several tools (such as BibTeX to RDF conversion tools [103]), ontologies (such as SWRC ontology [214]) and data sources (such as DBLP Berlin [197] and RKBExplorer [210]). These systems, tools, and data sources are widely being used and have considerably simplified and enhanced many bibliographic data management tasks such as data curation, storage, retrieval, and sharing of bibliographic data.

Despite the success of the above-mentioned systems, very few individuals and research groups publish their bibliographic data on their websites in a structured format, particularly following the principles of Linked Data [18]. This is mainly due to the fact that existing systems either are not designed to be used within an external website, or they require expert users to set up complex software systems on machines that meet the requirements of this software. BibBase [193] aims to fill this gap by providing several distinctive features as described in the following sections.

6.3.1 Lightweight Linked Data Publication

BibBase makes it easy for scientists to maintain publication lists on their personal web site. Scientists simply maintain a BibTeX file of their publications, and BibBase does the rest. When a user visits a publication page, BibBase dynamically generates an up-to-date HTML page from the BibTeX file, as well as rich linked data with resolvable URIs that can be queried instantly

[3]Part of this section has appeared in Proceedings of 6th International Conference on Semantic Systems (I-SEMANTICS 2010) [98].

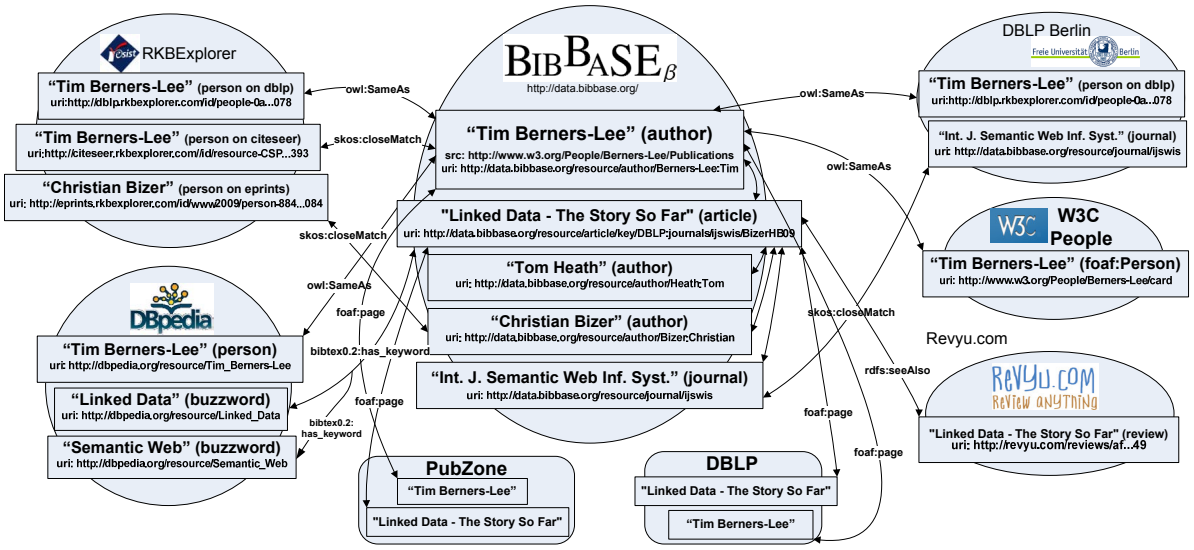


Figure 6.5: Sample entities in BibBase interlinked with several related data sources.

on the system's SPARQL endpoint. Compared to existing linked data publication tools, this approach is notably easy-to-use and lightweight, and allows non-expert users to create a rich linked data source without any specific server requirements, the need to set up a new system, or define complex mapping rules. All they need to know is how to create and maintain a BibTeX file and there are tools to help with that.

It is important to note that this ease of use does not sacrifice the quality of the published data. In fact, although the system is lightweight on the users' side, BibBase performs complex processing of the data in the back-end. When a new or updated BibTeX file arrives, the system transforms the data into several structured formats using a rich ontology, assigns URIs to all the objects (authors, papers, venues, etc.), performs duplicate detection and semantic linkage, and maintains and publishes provenance information as described below.

6.3.2 Internal Record Linkage

BibBase needs to deal with several issues related to the heterogeneity of records in a single BibTeX file, and across multiple BibTeX files. BibBase uses existing record linkage techniques in addition to a novel way of managing duplicated data following the linked data principles. Within a single

BibTeX file, the system uses a set of rules to identify duplicates and fix errors. We refer to this phase as *local* duplicate detection. For example, if a BibTeX file has two occurrences of author names "J. B. Smith" and "John B. Smith", the system matches the two author names and creates only a single author object. In this example, the assumption is that the combination of the first letter of first name, middle name, and last name, "JBSmith", is a unique identifier for a person in a single file. If this assumption does not hold for a specific user (which is unlikely) BibBase allows the user to distinguish two people with the same identifier by adding a number at the end of one of the author names. Note that we do not use a more complex string similarity measure for internal record linkage since we assume that each individual file uses the same naming conventions and format. Also, the user is usually able to quickly identify errors and fix them on his or her local file.

For identification of duplicates across multiple BibTeX files, which we call *global* record linkage, the assumptions made for local record linkage may not hold. Within different publication lists, "JBSmith" may (or may not) refer to the same author. BibBase deals with this type of uncertainty by having a *disambiguation* page on the HTML interface that informs the users looking for author name "J. B. Smith" (by looking up the URI `http://data.bibbase.org/author/j-b-smith`) of the existence of all the entities with the same identifier, and having `skos:closeMatch` and `rdfs:seeAlso` properties that link to related author entities on the RDF interface.

We use some of the LinQuer linkage methods to define local and global linkage rules, for example using string similarity measures or semantic knowledge for matching conference names and paper titles (cf. Chapters 2 and 3). In addition to the definition of rules and online record linkage, we also uses some graph-based record linkage techniques (as described in Section 2.2.1) to identify duplicates in an offline manner. However, in order to avoid loss of user data as a result of imperfect data cleaning, the results of this process are published as additional data on our system that result in disambiguation pages or `skos:closeMatch` predicates.

6.3.3 Linkage to External Data Sources

Figure 6.5 shows a sample of entities in BibBase and several possible links to related linked data sources and web pages. In order to discover such links, similar to our internal record linkage approach, we can leverage online and offline solutions. A main component of the online approach is based on using LinQuer's linkage methods along with a dictionary of terms and strings that can be mapped to external data sets. An important type of links comes from `keywords` in BibTeX entries that can be used to relate publications to entries on DBpedia (and pages on Wikipedia), such as DBpedia entities of type `buzzword` shown in the example figure. LinQuer's synonym method is used to match abbreviated venues, such as "ISWC" to "International Semantic Web Conference". The dictionaries (or ontology tables) are maintained inside BibBase, and derived from sources such as DBpedia, Freebase, Wordnet, and DBLP. We also allow the users to extend the dictionaries by `@string` definitions in their BibTeX files, e.g.,

```
@string{ISWC={Proc. of the Int'l Semantic Web Conference(ISWC)}}
```

An offline link discovery can be performed using more complex graph-based record linkage techniques (described in Section 2.2.1).

6.3.4 Provenance and User Feedback

Another highlight of BibBase s the storage and publication of provenance information, i.e., the source of each entity and each link in the data. This is of utmost importance in a system like BibBase here the data comes from several different users and BibTeX files, and where (imperfect) automatic duplicate detection and linkage is performed over the data. Users will be able to see the source of entities and the facts about the entities. In addition, they will be able to fix their own BibTeX files or provide feedback to the system and to other users who need to fix their files or provide additional information.

User feedback is another important aspect of BibBase Feedback is received in two forms. The first and major part of feedback comes from the BibTeX files. For example, as stated in Section 6.3.2, users can distinguish different authors with the same name by adding a number to the

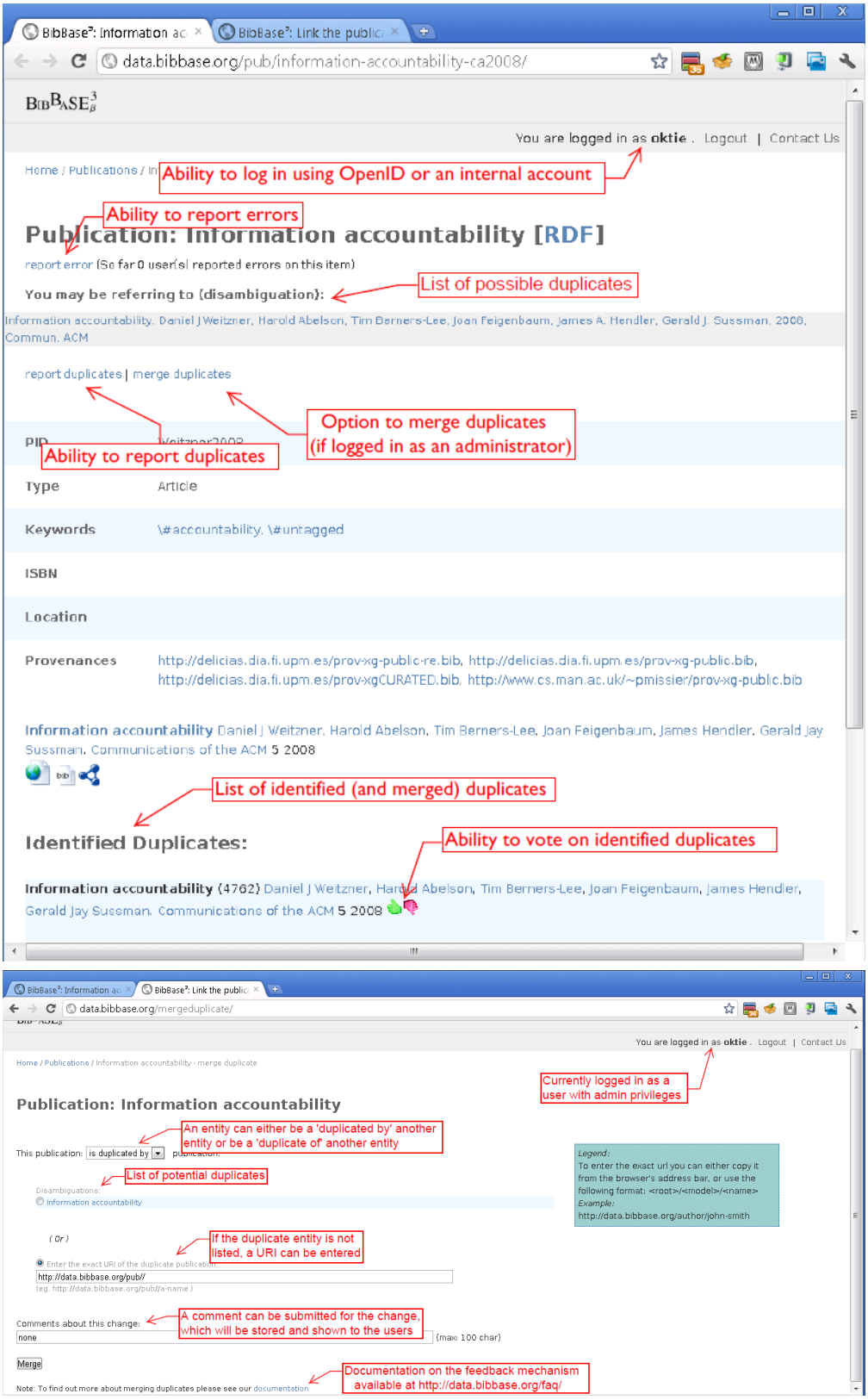


Figure 6.6: Data browse interface (top) and admin interface for merge (bottom).

end of their names. Similarly, as stated in Section 6.3.3, users can provide string equivalences to enhance our internal ontology tables for semantic linkage as well as duplicate detection.

Another form of feedback comes from the web interface, where the users can report errors such as broken links, typos, or wrong duplicate detection outside the scope of their own BibTeX entries. Furthermore, the users will be able to vote for or against existing identified duplicates or discovered links. Figure 6.6 shows a snapshot of our web interface for a publication that has a potential duplicate, and the admin interface for merging the duplicates. Currently, there are three class of users in our system:

- Anonymous users: all users can report errors or vote. The IP address of the users will be recorded, and each IP address will not be able to vote more than once for each link, duplicate, or entity.

- Authenticated Users: users can also log in using their Open ID (an open standard that allows users to be authenticated in a decentralized manner, using their existing Google accounts for example). Feedback from the authenticated users will be marked accordingly and verified with higher priority.

- Administrators: trusted users are assigned as administrators who can log in to verify the provided feedback, and perform specific actions based on the provided feedback. Our admin interface ranks the feedbacks on duplicates and external links based on type of the users (whether they are authenticated), and other criteria, and allows efficient verification of the users' feedback.

By providing feedback, users will not only increase the quality of the data published on their own websites, they will also create a very high-quality data source in the long run that could become a benchmark for the notoriously hard task of evaluating duplicate detection and link discovery systems.

6.3.5 BibTeX Ontology Definition

Using terms from existing vocabularies to publish data in RDF, is recognized as a "best practice" by the Linked Data community [101]. Several different vocabularies exist for bibliographic data. We have chosen to use and build on one that is specifically designed for BibTeX data (as opposed to the more general vocabularies), namely MIT's BibTeX ontology [121]. We have extended the vocabulary in several aspects to meet the requirements of our system, and address some shortcomings of existing ontologies (e.g., those noted by Herman [103]). Some of these changes include:

- Addition of several new classes such as `Keyword`, `Organization`, `Language`, `Journal` and `Author` that were defined as properties with string value ranges in the previous version. This will facilitate grouping and querying of these attributes of publications.

- Addition of a new class `Authorship` with properties `hasPosition` and `hasAuthor` to record the order of authors for a publication.

- Addition of new properties `hasAuthorship` and `firstAuthor` for publication entries in order to facilitate querying based on order of authors or the first author only.

The new namespace (BibTeX 2.0) is available at `http://data.bibbase.org/ontology/`. The OWL-Lite version of the ontology is available at `http://data.bibbase.org/ontology/owl/`. We also publish the relationship between the classes and properties of our ontology with the two other widely-used bibliographic ontologies, namely BIBO [215] and SWRC [214]. Figure 6.7 shows a subset of the relationship between these ontologies. Note that several classes in our extended ontology are also present in these ontologies, whereas some do not exist in these or other ontologies.

BibTeX 2.0	BIBO	SWRC
http://data.bibbase.org/ontology/#Publication	http://purl.org/ontology/bibo/Document	http://swrc.ontoware.org/ontology#Publication
http://data.bibbase.org/ontology/#Article	http://purl.org/ontology/bibo/Article	http://swrc.ontoware.org/ontology#Article
http://data.bibbase.org/ontology/#Book	http://purl.org/ontology/bibo/Book	http://swrc.ontoware.org/ontology#Book
http://data.bibbase.org/ontology/#hasTitle	http://purl.org/dc/terms/title	http://swrc.ontoware.org/ontology#title
http://data.bibbase.org/ontology/#hasPages	http://purl.org/ontology/bibo/pages	http://swrc.ontoware.org/ontology#pages
http://data.bibbase.org/ontology/#hasVolume	http://purl.org/ontology/bibo/volume	http://swrc.ontoware.org/ontology#volume
http://data.bibbase.org/ontology/#hasAddress	Does not exist	http://swrc.ontoware.org/ontology#address"
http://data.bibbase.org/ontology/#hasEdition	http://purl.org/ontology/bibo/edition	http://swrc.ontoware.org/ontology#edition
http://data.bibbase.org/ontology/#hasLocation	Does not exist	http://swrc.ontoware.org/ontology#location
http://data.bibbase.org/ontology/#hasPublisher	http://purl.org/dc/terms/publisher	http://purl.org/dc/elements/1.1/publisher
http://data.bibbase.org/ontology/#hasJournal	Does not exist	http://swrc.ontoware.org/ontology#journal
http://data.bibbase.org/ontology/#hasSchool	Does not exist	http://swrc.ontoware.org/ontology#school
http://data.bibbase.org/ontology/#hasLanguage	http://purl.org/dc/terms/language	http://purl.org/dc/elements/1.1/language
http://data.bibbase.org/ontology/#hasFirstAuthor	Does not exist	Does not exist
http://data.bibbase.org/ontology/#hasAuthorship	Does not exist	Does not exist
http://data.bibbase.org/ontology/#Authorship	Does not exist	Does not exist

Figure 6.7: A subset of the comparison between the BibTeX 2.0 ontology and two other bibliographic ontologies BIBO [215] and SWRC [214].

6.3.6 Additional Features

The success of BibBase as a linked data source depends on scientists using BibBase for their publications pages. To further entice scientists to do so, BibBase sports a number of additional features that make it an attractive solution for this purpose.

- Dynamic, multi-level grouping of publications based on different attributes (e.g., by year or keyword).
- Customizable appearance via CSS style sheets.
- An RSS feed, allowing anyone to receive notifications whenever a specified scientist publishes a new paper.
- A DBLP fetch tool that allows scientists who do not yet have a BibTeX file to obtain their DBLP publications to start using BibBase right away.
- Statistics regarding users, page views, and paper downloads, including a list of most popular papers.

We are currently working on a few additional features such as a generic keyword search interface that accounts for spelling errors, abbreviations, and semantic mismatches, and a visual navigator for the RDF data specifically designed to find correlations between the authors, papers,

and keywords. We will also create an RSS feed for every keyword being used, so that anyone can be notified when new papers in that area are published.

6.3.7 Impact and Future Directions

BibBase received honorary mention in the open track of the Linking Open Data Triplification Challege 2010 [98]. BibBase.org has had a fairly robust and continuous growth in terms of traffic and users. It is already being used in 75 different HTTP domains (which translates into many more users), and has almost 2,000 unique visitors per month. The data browse interface `data.bibbase.org` has had 4,406 unique visitors as of November 2011 (since its launch in June 2010). It indexes more than 7,500 publications, 9,000 authors, and 250 provenances (meaning 250 uploaded BibTeX files).

We are currently working on a number of additional features for BibBase. Most importantly, we are planning to perform thorough evaluation of the record linkage results in BibBase (cf. Section 7.2.4). We are currently working on a *workbench* interface for managing and analyzing the feedback received from the users. Our plan is to extend the current feedback mechanism to a more generic record linkage evaluation and feedback management system and use the power of micro-task markets such as Amazon's Mechanical Turk [191] to create the first large-scale crowd-sourced benchmark data set for record linkage.

Chapter 7

Conclusion and Future Work

In this thesis, we studied several aspects of the record linkage problem for Web data. In what follows, we present a brief overview of the contributions of this thesis. We then outline a number of interesting directions for future work.

7.1 Summary

In Chapter 2, we presented an overview of state-of-the-art in record linkage. Unlike existing survey articles and books on the subject [68, 123, 138, 184], we focused on the unique aspects of the problem that have received less attention in the work on record linkage. Some of the problems discussed have been studied in other contexts such as clustering (of Web documents), ontology and schema matching, and object identification. The lack of attention in previous record linkage work on these problems in part motivates some of the contributions of this thesis, which use and extend (or address shortcomings of) previously proposed approaches for similar problems in other fields.

At the end of Chapter 2, we presented an overview of existing record linkage libraries and systems, which clearly showed the lack of a generic, easy-to-use, and domain-independent system (or library) that non-expert users and data publishers can use to establish links to other data sources or improve the quality of their data and (internal or external) links. In Chapter 3,

we presented LinQuer, to address this problem for relational databases, motivated by the fact that many existing Web data sources rely on relational storage. The LinQuer framework consists of the LinQL language that extends SQL with linkage primitives, and a set of *native* linkage methods that implement a wide range of string and semantic similarity measures that can be translated into standard SQL queries to perform linkage. At the end of the chapter, we presented the results of evaluating the framework using several real-world linkage scenarios. In addition, Appendix A provides the results of our extensive accuracy evaluation of string similarity measures for record linkage.

In Chapter 4 and 5, we studied the challenges faced in record linkage when the data is not provided in relational tables with a fixed predefined schema and known schema correspondences between source and target data. We addressed the problem of unknown schema correspondences for record linkage in Chapter 4. We defined the notion of a *linkage point* between two data sources and provided efficient and effective algorithms for discovery of linkage points over Web data. We thoroughly evaluated the algorithms in several real-world linkage scenarios in different domains.

In Chapter 4, we made the assumption that data sources are carefully designed and have a conceptual model describing the data. In particular, we assumed that *entity types* in each data source are well-defined and the data source can be queried to retrieve records that describe a particular entity type, along with all their attributes. Although this is the case for many Web data sources (e.g., those that use the RDF data model and follow Linked Data principles), there are many sources that do not follow such standards. We presented our solution to this problem in Chapter 5 which includes a framework and a set of algorithms for building a conceptual model out of a given semistructured (e.g., XML or JSON) Web data source that facilitates linking. Again, we evaluated the algorithms in real-world scenarios.

Finally, in Chapter 6, we presented an overview of three high-quality structured data sources that we have published on the Web in part using the frameworks and algorithms presented in Chapters 3, 4, and 5. We described how each of these data sources have unique requirements

for data transformation, entity identification, record linkage, and their user interfaces, and how we have addressed these requirements.

7.2 Future Directions

Our experience in publishing and linking data on the Web in several domains reveals a number of other problems that we have not addressed in this thesis. In what follows, we present an overview of a few of these problems along with a sketch of possible solutions.

7.2.1 Semantic-Aware Linkage Methods

The LinQuer framework, to the best of our knowledge, is the first to combine string (syntactic) similarity and semantic similarity for record linkage. However, our approach relies on the existence of semantic knowledge about the meaning of a set of strings. One of the limitations of this approach is when there is semantic knowledge at the token-level and not about the full strings. Consider the example in Figure 7.1, where the goal is to link patient records in the `PV` table to condition records in the `TC` table. Records r_1, r_2, and r_4 should all be linked to the record s_1 whereas records r_3 and r_4 are not to be linked, although the string representations are highly similar. Using exact matching, only record r_1 will be linked. Using string similarity (possibly combined with semantic similarity) will either fail to link r_5 or will incorrectly link r_3 as well depending on the threshold value used. By only using semantic similarity even if a complete knowledge base is given that contains the fact that r_5's diagnosis is a type of s_1's condition, we may miss r_2 or any records that have misspellings.

What we need in this example and in similar situations which have occurred frequently in our evaluations is the ability to use a semantic knowledge base about words. Such knowledge which can be derived from a dictionary or WordNet [136], could indicate for example that "cancer" is a synonym of "carcinoma", "recurrent" is just a type indicator that can be ignored or should be given a lower weight, and that "large" and "non-small" are antonyms of "small", or

Patient Visits (PV)

id	diagnosis
r1	Small Cell Lung Cancer
r2	Small Cell Lung Carcinoma
r3	Non-Small Cell Lung Carcinoma
r4	Large Cell Lung Carcinoma
r5	Recurrent Small Cell Carcinoma Lung Cancer

Trials Condition (TC)

id	condition
s1	Small Cell Lung Cancer

Figure 7.1: Sample records requiring semantic-aware linkage

that "non-" should be given a very high negative weight in similarity computation. We are currently investigating implementation of a novel set of *semantic-aware* string similarity measures as native linkage methods in LinQuer. One such similarity measure can be derived based on extending the SoftTFIDF measure [49] (Definition 2.12) that takes into account the semantic knowledge in its word token level similarity measure.

7.2.2 Indexing Methods for LinQuer

As noted in our discussion of related work in Chapters 2 and 3, methods to improve the efficiency of string similarity computation for record linkage have been extensively studied. Many such techniques are based on adapting or extending previously proposed methods for near-duplicate detection (of documents) and in general the nearest neighbor search (or similarity search) problem of finding points in a metric space that are nearest to a given query point. Such techniques can be used for alternative implementations of the `LINKINDEX` statements in LinQuer. An example of an efficient hashing technique for similarity search is Charikar's Simhash [41]. Simhash performs hashing in a way that vectors that are similar (based on Cosine similarity) will have similar hash values (small *Hamming distance*). Simhash and other hashing and indexing techniques have been evaluated and compared in near-duplicate detection of Web documents [15, 133, and others]. A similar evaluation for record linkage is the subject of our future work.

7.2.3 Discovering Complex Linkage Points

The SMaSh framework presented in Chapter 4 can be seen as the first step towards a Web-scale complex link discovery system. We have only evaluated the system in record linkage applications and the discovery of 1-to-1 linkage points. Future work includes the following.

- Extension of the analyzer and similarity function libraries to support efficient substring matching and semantic matching.
- Extension of the search algorithms to find *complex linkage points* which are 1-to-n and n-to-m attribute correspondences. Such correspondences will indicate that a set of attributes can be combined to do the linking. For example, the combination of attributes `firstname`, `lastname` and `zipcode` in source data and the combination of `full_name` and `zip_code` in target data can effectively be used to link records in the two data sets.
- Extension of the framework to perform entity type identification prior to the discovery of linkage points, for a truly Web-scale linkage point discovery across sources containing entities of several types. We plan to evaluate this extension by linking the data published by Facebook Pages and the recently-announced Open Graph Protocol [216] to external sources such as DBpedia and Freebase.

7.2.4 Crowd-sourcing Link Discovery and Quality Evaluation

Evaluating the accuracy of record linkage has been recognized as a difficult task in the literature. We adopt the common approach of evaluating the accuracy either on synthetic data sets, or small real data sets (or smaller portions of larger data sets) where ground truth is known. Using synthetic data sets with errors introduced in a way that resembles real-world dirty data, we have been able to evaluate the accuracy of several string similarity measures and clustering algorithms for record linkage, and the effect of several data characteristics such as the amount and distribution of errors on the performance of the algorithms. We are planning to extend our

evaluations using large real data sets with ground truth verified using crowd-sourcing. Using our three Linked Data sources described in Chapter 6, and using the same methodology used in the feedback mechanism implemented in BibBase (cf. Section 6.3) and the xCurator framework (cf. Section 5.2), we are planning to create the first large-scale crowd-sourced benchmark data sets for record linkage.

7.2.5 Collective Linkage Using Semantic Matching

In Appendix B, we present an extensive evaluation of clustering algorithms for record linkage by clustering of the *linkage graph*, in which nodes represent records and edges represent similarity between the two records with their string similarity as the weight of the edge. Extending the algorithms to take into account the semantic similarity of the records is non-trivial. In particular, a link derived using semantic matching based on the synonymy measure (cf. 2.13, Equation 2.1) for example, is a more reliable link than a link derived from a string similarity measure with a low threshold. Therefore, the clustering algorithm needs to take into account two types of edges between the nodes. This clustering problem can be seen as an extension of the correlation clustering problem as defined by Bansal et al. [12] in which a graph can have edges labeled with 1) $+$ indicating certain link, 2) $-$ indicating certain non-link, or 3) a score that represents the confidence in the link or the similarity score. The goal of the clustering algorithm then is to produce a clustering that has zero (or very small) violation of the certain links, but minimizing the edge weights between clusters and maximizing the edge weights inside the clusters.

7.2.6 Parallel Record Linkage

A current trend in large-scale data management in leveraging the power of distributed computing platforms such as the highly popular MapReduce paradigm [56] and its Apache Hadoop [192] implementation. Given the fact that the primary application of such frameworks has been managing large amounts of Web data, and the large amount of text strings available on the Web (that are prone to errors), the need for string matching-based record linkage tech-

niques is apparent. Recent work has proposed efficient MapReduce-based techniques for set similarity computation [169] (with application in record linkage) and record linkage [122]. On the other hand, there have been proposals on declarative scripting languages for MapReduce-based data processing such as Yahoo!/Apache's Pig Latin [146] and IBM's Jaql [21]. Extending existing declarative frameworks with linkage constructs and their efficient MapReduce-based implementation as a part of a declarative link discovery framework similar to LinQuer is an interesting direction for future work.

7.2.7 Online Enterprise Data Analytics

Finally, we are currently investigating the application of our proposed record linkage techniques in managing large amounts of enterprise data in an *online* fashion, as a part of IBM's Helix project[1]. The goal of Helix project is to provide users within the enterprise a platform that allows them to perform online analysis of almost any type and amount of internal data using the power of external knowledge bases available on the Web. Such a platform requires a novel, data-format agnostic indexing mechanism, and lightweight record linkage techniques that could link semantically related records across internal and external data sources of various characteristics.

Figure 7.2 shows the initial architecture of the Helix system. The data registry component keeps a catalog of available internal and external sources and their access methods and parameters. The data processor component performs indexing and data analysis, in addition to a novel instance tagging mechanism, that takes advantage of the online sources (such as Freebase) to identify the *sense* of the internal data attributes. For example, we automatically identify that a column named `NAME` in table `CUSTOMERS` in one of the internal legacy sources stores names of companies (and therefore we tag the column with sense "Company"), but the column `COMP_NAME` from table `COMPLAINTS` refers to software component names (and therefore we tag the column with sense "Software Component"). The query processor component takes in user

[1]The initial result of our work has appeared in the proceedings of the World Wide Web conference (WWW 2011) [93].

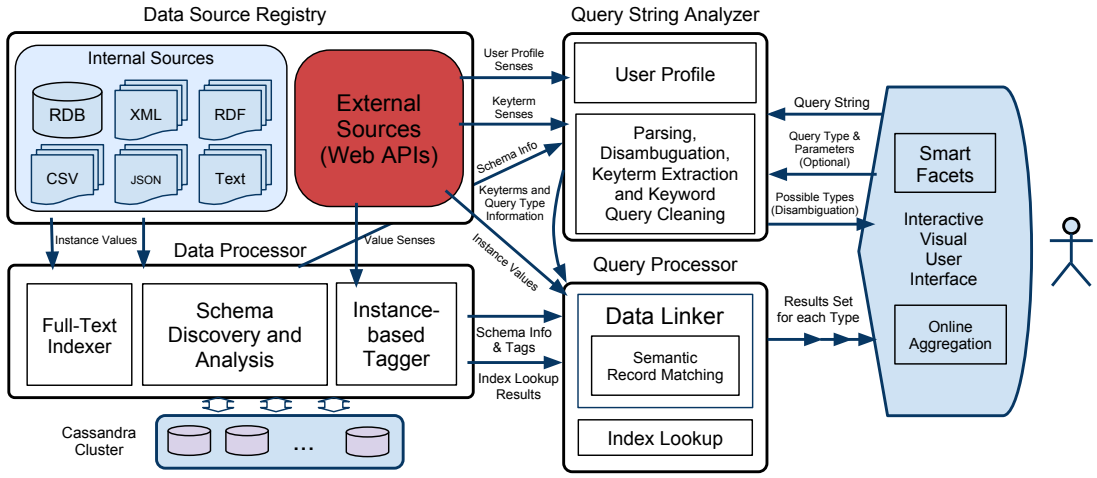


Figure 7.2: Helix framework

profile information and the results of analyzing the user's query, and performs on-the-fly linkage of the internal sources that are relevant to the query (based on the discovered senses) with the related online sources. This way, we can, for example, fetch company information from Freebase when the user looks for a company name found in table `CUSTOMERS`, or the user can filter search results based on company types that are retrieved from Freebase.

Currently, the implemented Helix framework only supports internal sources that are relational. Future work includes adding support for semistructured internal and external sources, and taking advantage of the entity identification and linkage point discovery algorithms of Chapters 4 and 5 to extend the data processor component.

Appendix A

Accuracy Evaluation of String Similarity Measures[1]

In this appendix, we present an evaluation of the the accuracy of different similarity measures described in Section 2.1.2 that fit in the LinQuer framework (Chapter 3), i.e., have efficient SQL-based implementation. We first describe the data sets used in our experiments, and then present an overview of our results.

A.1 Data Sets

To generate data sets for our experiments, we use the same methodology used in our earlier evaluation of approximate selection predicates [89]. Our data generator is an extension of the UIS database generator which has been effectively used in the past to evaluate duplicate detection algorithms and has been made publicly available [104] We follow a relatively standard methodology of using the data generator to inject different types and percentages of errors to a clean database of string attributes. The erroneous records made from each clean record are put in a single cluster (which we use as ground truth) in order to be able to measure quality

[1] Part of this appendix has appeared in Proceedings of the Fifth International Workshop on Quality in Databases (QDB 2007). [96].

(precision and recall) of the similarity join and clustering modules. The generator permits the creation of data sets of varying sizes, error types and distributions, thus is a very flexible tool for our evaluation. The kind of typographical errors injected by the data generator are based on studies on common types of errors present in string data in real databases [126]. Therefore the synthetic data sets resemble real dirty databases, but allow thorough evaluation of the results based on robust quality measures.

Our data generator provides the following parameters to control the error injected in the data:

- the size of the data set to be generated
- the fraction of clean records to be utilized to generate erroneous duplicates
- *distribution of duplicates*: the number of duplicates generated for a clean record can follow a uniform, Zipfian or Poisson distribution.
- *percentage of erroneous duplicates*: the fraction of duplicate records in which errors are injected by the data generator.
- *extent of error in each erroneous record*: the percentage of characters that will be selected for injecting character edit error (character insertion, deletion, replacement or swap) in each record selected for error injection.
- *token swap error*: the percentage of word pairs that will be swapped in each record that is selected for error injection.

We use two different clean sources of data: a data set consisting of *company names* and a data set consisting of titles from *DBLP*. Statistical details for the two data sets are shown in Table A.1. Note that we can generate reasonably large data sets out of these clean sources. For the company names datas et, we also inject domain specific *abbreviation errors*, e.g., replacing `Inc.` with `Incorporated` and vice versa. We describe the characteristics of the specific data sets

Table A.1: Statistics of clean data sets

data set	#rec.	Avg. rec. length	#words/rec.
Company Names	2139	21.03	2.92
DBLP Titles	10425	33.55	4.53

Table A.2: Range of parameters used for erroneous data sets

parameter	range
size of data set	5K - 100K
# clean records	500 - 10000
duplicate distribution	uniform, Zipfian
erroneous duplicates	10% - 90%
extent of error per record	5% - 30%
token swap error	10% - 50%

generated for evaluating each component (parameters used to create data sets) in the related sections.

For both data sets, we generate different erroneous data sets by varying the parameters of the data generator as shown in Table A.2.

For the results presented in this section, the data sets are generated by the data generator out of the clean company names data set described in Table A.1. The errors in the data sets have a uniform distribution. For each data set, on average 5000 dirty records are created out of 500 clean records. We have also run experiments on data sets generated using different parameters. For example, we generated data using a Zipfian distribution, and we also used data from the other clean source in Table A.1 (DBLP titles). We also created larger data sets. For these other data sets, the accuracy trends remain the same. Table A.3 shows the description of all the data sets used for the results in this chapter. We used 8 different data sets with mixed types of errors (edit errors, token swap and abbreviation replacement). Moreover, we used 5 data sets with only a single type of error (3 levels of edit errors, token swap or abbreviation replacement errors) to measure the effect of each type of error individually.

Table A.3: Data sets used for the results in this chapter

Group	Name	Percentage of			
		Erroneous Duplicates	Errors in Duplicates	Token Swap	Abbr. Error
Dirty	D1	90	30	20	50
	D2	50	30	20	50
Medium Error	M1	30	30	20	50
	M2	10	30	20	50
	M3	90	10	20	50
	M4	50	10	20	50
Low Error	L1	30	10	20	50
	L2	10	10	20	50
Single Error	AB	50	0	0	50
	TS	50	0	20	0
	EDL	50	10	0	0
	EDM	50	20	0	0
	EDH	50	30	0	0

A.2 Accuracy Measures

We use well-known measures for accuracy evaluation in information retrieval, namely precision, recall, and F_1, for different values of the threshold to evaluate the accuracy of the similarity join operation. We perform a self-join on the input table using a similarity measure with a fixed threshold θ. *Precision (Pr)* is defined as the percentage of *duplicate* records among the records that have a similarity score above the threshold θ. In our data sets, *duplicate* records are marked with the same cluster ID as described above. *Recall (Re)* is the ratio of the number of *duplicate* records that have similarity score above the threshold θ to the total number of *duplicate* records. Therefore, a join that returns all the pairs of records in the two input tables as output has low (near zero) precision and recall of 1. A join that returns an empty answer has precision 1 and zero recall. The F_1 measure is the harmonic mean of precision and recall, i.e., $F_1 = \dfrac{2 \times Pr \times Re}{Pr + Re}$ We measure precision, recall, and F_1 for different values of the similarity threshold θ. For comparison of different similarity measures, we use the maximum F_1 score across different thresholds.

A.3 Settings

For the measures based on q-grams, we set $q = 2$ since it yields the best accuracy in our experiments for all these measures. We use the same parameters for BM25 and HMM score formula that were suggested elsewhere [135, 154, 89].

A.4 Results

Figures A.1 and A.2 show the precision, recall, and F_1 values for all the measures, over the data sets we have defined with mixed types of errors. For all measures except HMM and BM25, the horizontal axis of the precision/recall graph is the value of the threshold. For HMM and BM25, the horizontal axis is the percentage of maximum value of the threshold, since these measure do not return a score between 0 and 1.

The results show that the "dirtiness" of the input data greatly affects the value of the threshold that results in the most accurate join. For all the measures, a lower value of the threshold is needed as the degree of error in the data increases. For example, Weighted Jaccard achieves the best F_1 score over the dirty group of data sets with threshold 0.3, while it achieves the best F_1 for the low-error data sets at threshold 0.55. BM25 and HMM are less sensitive and the best value of the threshold varies from 0.25 for dirty data sets to 0.3 for low-error data sets. We will discuss later how the degree of error in the data affects the choice of the most accurate measure.

Effect of types of errors

Figure A.3 shows the maximum F_1 score for different values of the threshold for different measures on data sets containing only edit-errors (the `EDL`, `EDM` and `EDH` data sets). These figures show that weighted Jaccard and Cosine have the highest accuracy followed by Jaccard, and edit similarity on the low-error data set `EDL`. By increasing the amount of edit error in each record, HMM performs as well as weighted Jaccard, although Jaccard, edit similarity, and GES perform much worse on high edit error data sets. Considering the fact that edit-similarity is mainly

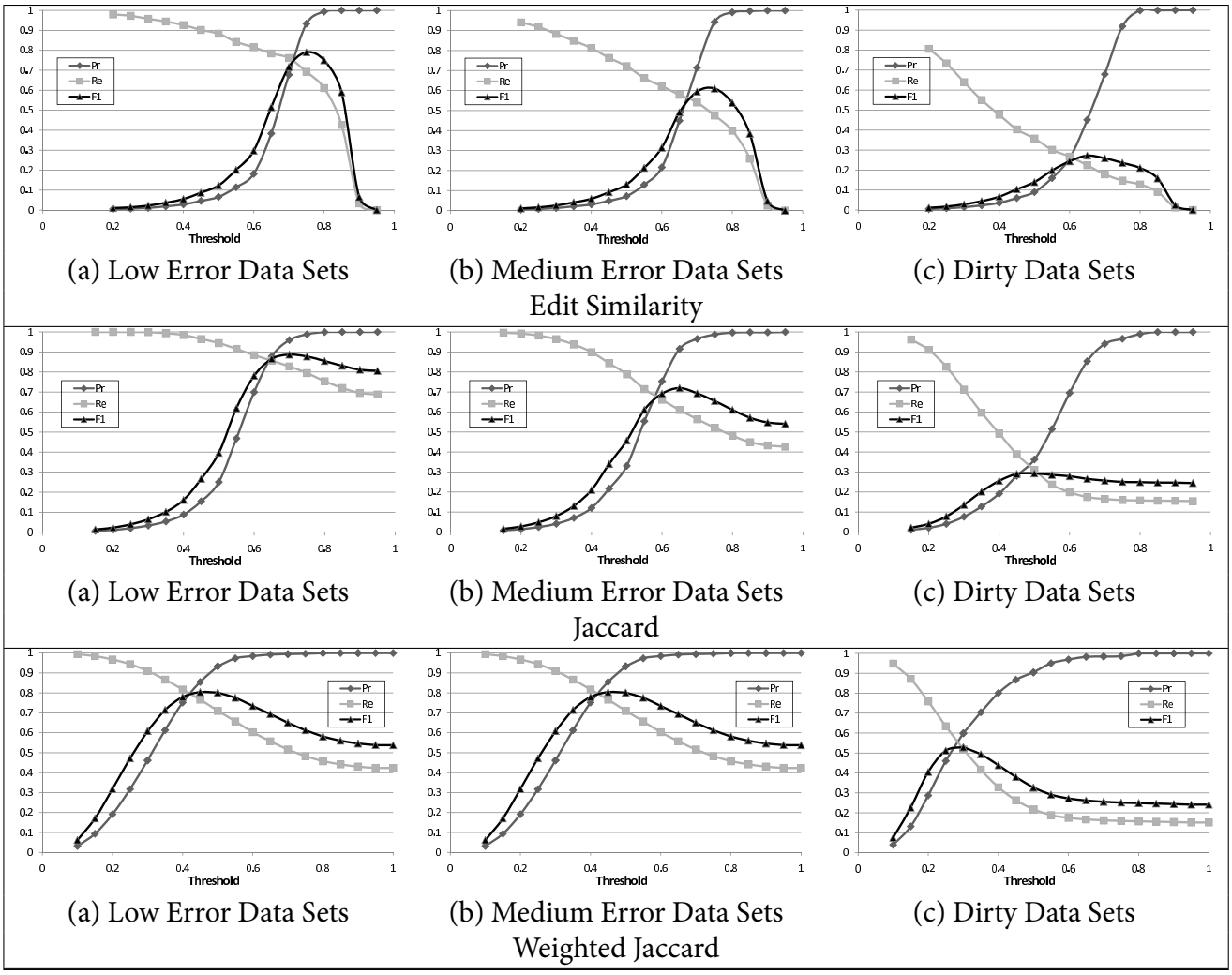


Figure A.1: Accuracy of similarity join using Edit-Similarity, Jaccard and Weighted Jaccard measures relative to the value of the threshold on different data sets

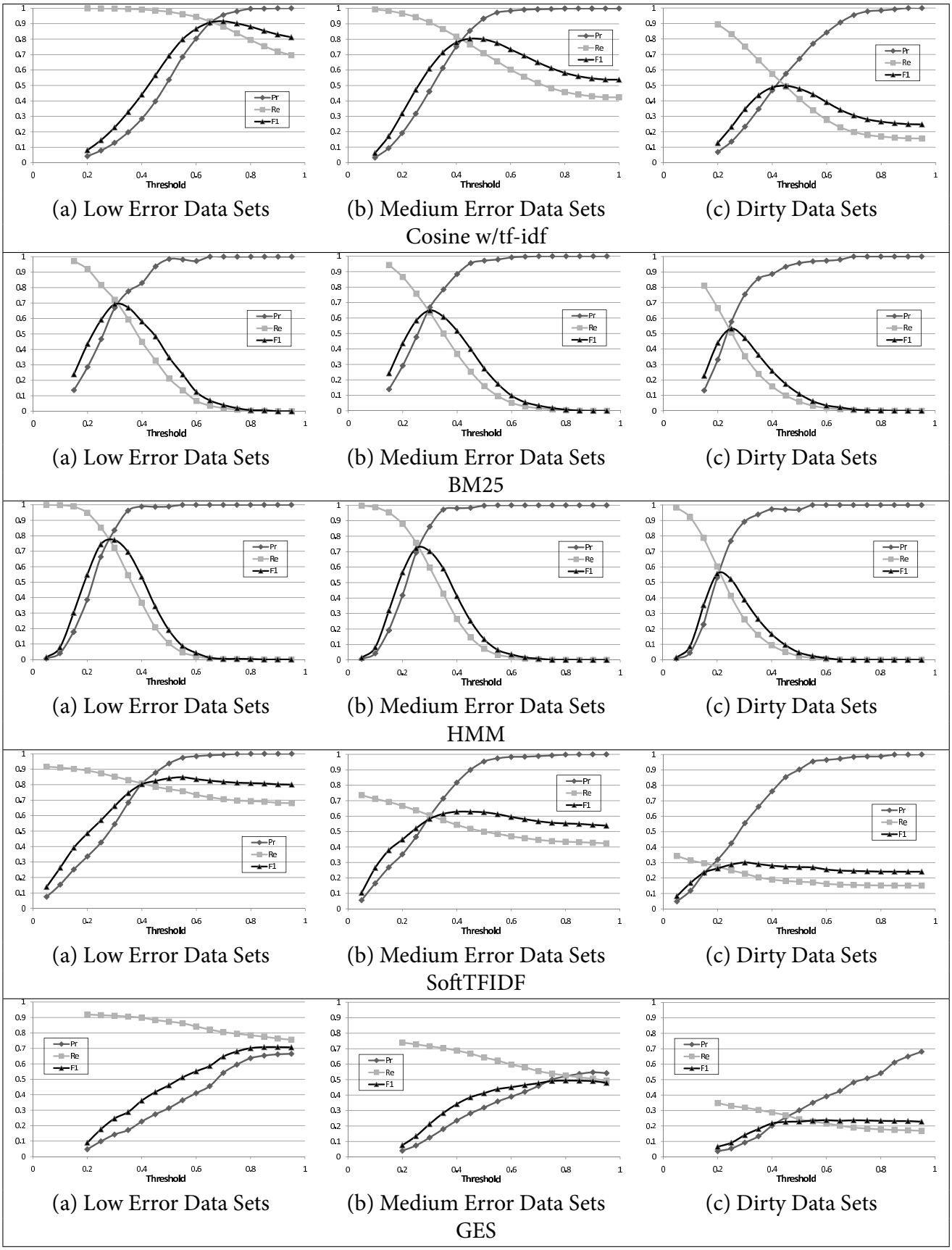


Figure A.2: Accuracy of similarity join using measures from IR and hybrid measures relative to the value of the threshold on different data sets

proposed for capturing edit errors, this shows the effectiveness of weighted Jaccard and its robustness with varying amount of edit errors. Figure A.4 shows the effect of token swap and abbreviation errors on the accuracy of different measures. This experiment indicates that edit similarity is not capable of modeling such errors. HMM, BM25 and Jaccard also are less capable of modeling abbreviation errors than cosine with tf-idf, SoftTFIDF and weighted Jaccard.

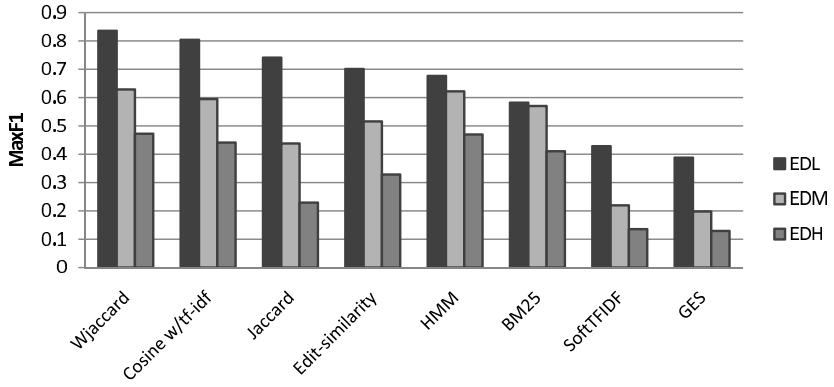


Figure A.3: Maximum F_1 score for different measures on data sets with only edit errors

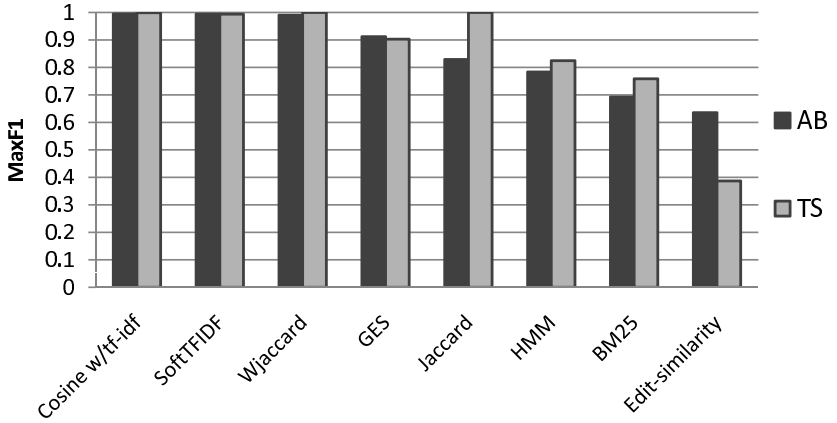


Figure A.4: Maximum F_1 score for different measures on data sets with only token swap and abbr. errors

Comparison of measures

Figures A.5 shows the maximum F_1 score for different values of the threshold for different measures on dirty, medium and low-error data sets. Here, we have aggregated the results for all the

dirty data sets together (respectively, the moderately dirty or medium data sets and the low-error data sets). The results show the effectiveness and robustness of weighted Jaccard and cosine in comparison with other measures. Again, HMM is among the most accurate measures when the data is extremely dirty, and has relatively low accuracy when the percentage of error in the data is low.

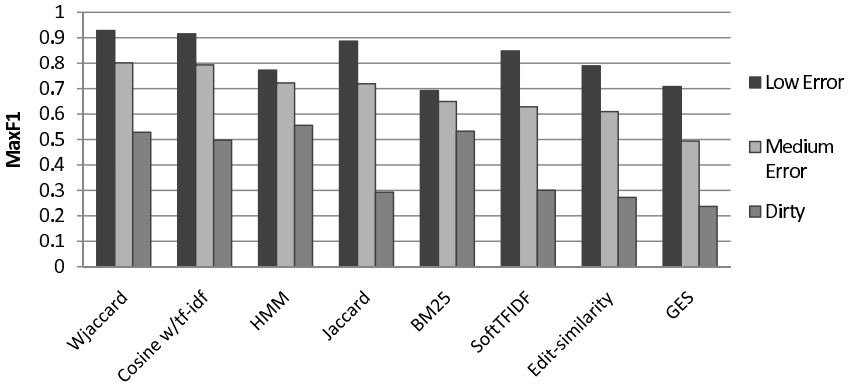


Figure A.5: Maximum F_1 score for different measures on dirty, medium and low-error group of data sets

Appendix B

Evaluation of Clustering Algorithms[1]

In this appendix, we describe our methodology for evaluating clustering algorithms described in Section 2.2.2 for record linkage, and present an overview of the results of our experiments. We first briefly discuss the characteristics of the data sets used in the experiments. We then explain the settings of our experiments including the choice of similarity measure and parameters of the algorithms, and finally present results of our extensive experiments.

B.1 Data Sets

The data sets used in our experiments are generated using the same data generator described in Section A.1. We use two clean sources of data: a data set consisting of *company names* that contains $2,139$ records (name of companies) with average record length of 21.03 characters and 2.92 words in each record on average, and a data set consisting of titles from *DBLP* which contains $10,425$ records with average 33.55 characters record length and average 4.53 words in each record. Note that the data sets created by the data generator can be much larger than the original clean sources. In our experiments, we create data sets of up to $100K$ records.

For the results in this chapter, we used 29 different data sets (tables) with different sizes,

[1]Part of this appendix has appeared in Proceedings of the VLDB Endowment (PVLDB), Volume 2 Issue 1, August 2009 [90].

error types and distributions. Tables B.1 and B.2 show the description of all these data sets along with the percentage of erroneous records in each data set (i.e., the average number of the records in each cluster which are erroneous), the percentage of errors within each duplicate (i.e., the number of errors injected in each erroneous record), the percentage of token swap and abbreviation errors as well as the distribution of the errors (column Dist. in Table B.2), the size of the data sets (the number of records in each table) and the number of the clusters of duplicates (column Cluster# in Table B.2)[1]. Five data sets contain only a single type of error (3 levels of edit errors, token swap or abbreviation replacement errors) to measure the effect of each type of error individually. The data sets with uniform distribution have equal cluster sizes on average (e.g., 10 records in each cluster on average for a data set of $5,000$ records with 500 clusters) whereas the size of the clusters in the Zipfian data sets follow a Zipfian distribution (i.e., most of the clusters have size 1 while a few clusters are very large). Note that we limited the size of the data sets for our experiments on accuracy due to the fact that some of the algorithms do not scale well. However, we run experiments on much larger data sets with the algorithms that do scale and the trends were similar. Following [89], we believe the errors in these data sets are highly representative of common types of errors in databases with string attributes.

B.2 Accuracy Measures

In order to evaluate the quality of the duplicate clusters found by the clustering algorithms, we use several accuracy measures from the clustering literature and also measures that are suitable for the final goal of duplicate detection. Suppose that we have a set of k ground truth clusters $G = \{g_1, \ldots, g_k\}$ of the base relation R. Let $C = \{c_1, \ldots, c_{k'}\}$ denote the set of k' output clusters of a clustering algorithm. We define a mapping f from the elements of G to the elements of C, such that each cluster g_i is mapped to a cluster $c_j = f(g_i)$ that has the highest percentage of

[1] All these data sets along with a small sample of them are available at: http://dblab.cs.toronto.edu/project/stringer/clustering/

Table B.1: Data sets used in the experiments

Group	Name	Percentage of			
		Erroneous Duplicates	Errors in Duplicates	Token Swap	Abbr. Error
High Error	H1	90	30	20	50
	H2	50	30	20	50
Medium Error	M1	30	30	20	50
	M2	10	30	20	50
	M3	90	10	20	50
	M4	50	10	20	50
Low Error	L1	30	10	20	50
	L2	10	10	20	50
Single Error	AB	50	0	0	50
	TS	50	0	20	0
	EDL	50	10	0	0
	EDM	50	20	0	0
	EDH	50	30	0	0
Zipfian High	ZH1	90	30	20	50
	ZH2	50	30	20	50
Zipfian Medium Error	ZM1	30	30	20	50
	ZM2	10	30	20	50
	ZM3	90	10	20	50
	ZM4	50	10	20	50
Zipfian Low	ZL1	30	10	20	50
	ZL2	10	10	20	50
DBLP High	DH1	90	30	20	0
	DH2	50	30	20	0
DBLP Medium Error	DM1	30	30	20	0
	DM2	10	30	20	0
	DM3	90	10	20	0
	DM4	50	10	20	0
DBLP Low	DL1	30	10	20	0
	DL2	10	10	20	0

common elements with g_i. Precision and recall for a cluster g_i, $1 \leq i \leq k$ is defined as follows:

$$Pr_i = \frac{|f(g_i) \cap g_i|}{|f(g_i)|} \quad \text{and} \quad Re_i = \frac{|f(g_i) \cap g_i|}{|g_i|}$$

Intuitively, the value of $Pr_i$is a measure of the accuracy with which cluster $f(g_i)$ reproduces cluster g_i, while the value of Re_i is a measure of the completeness with which $f(g_i)$ reproduces class g_i. Precision, Pr, and recall, Re, of the clustering are defined as the weighted averages of

Table B.2: Size, distribution and source of the data sets

Group	Source	Dist.	Size	Cluster#
High Error, Medium Error, Low Error, Single Error	Company Names	Uniform	5K	500
Zipfian High Zipfian Medium Zipfian Low	Company Names	Zipfian	1.5K	1K
DBLP High DBLP Medium DBLP Low	DBLP Titles	Uniform	5K	500

the precision and recall values over all ground truth clusters. More precisely:

$$Pr = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Pr_i \quad \text{and} \quad Re = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Re_i$$

F_1-measure is defined as the harmonic mean of precision and recall, i.e.,

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re}.$$

We use precision, recall and F_1-measure as indicative values of the ability of an algorithm to reconstruct the indicated clusters in the data set. However, in our framework, the number of clusters created by the clustering algorithms is not fixed and depends on the data sets and the threshold value used in the similarity join. Therefore, we define two other measures specifically suitable for our framework. Let CPr_i be the number of pairs (of records) in each cluster c_i that are in the same ground truth cluster $g_j : c_i = f(g_j)$, i.e.,

$$CPr_i = \frac{|(t,s) \in c_i \times c_i | t \neq s \wedge \exists j \in 1 \ldots k, (t,s) \in g_j \times g_j|}{\binom{|c_i|}{2}} \tag{B.1}$$

We define Clustering Precision, CPr, to be the average of CPr_i for all clusters of size greater than or equal to 2. The value of CPr indicates the ability of the clustering algorithm to assign records that should be in the same cluster to a single cluster, regardless of the number and the

size of the clusters produced. In order to penalizes those algorithms that create greater or fewer clusters than the ground truth, we define Penalized Clustering Precision, $PCPr$, and compute it as CPr multiplied by the percentage of extra or missing clusters in the result, i.e.,

$$PCPr = \begin{cases} \frac{k}{k'} CPr & k < k' \\ \frac{k'}{k} CPr & k \geq k' \end{cases}$$

B.3 Settings

Based on the comparison of several string similarity measures in Appnedix chapter:sjoineval, we use the weighted Jaccard similarity measure along with q-gram tokens of size 2 as the measure of choice due to its relatively high efficiency and accuracy compared with other measures.

To compare the clustering algorithms, we have either implemented or obtained an implementation of the algorithms from their authors. Notably, not all of these algorithms have previously been implemented nor evaluated (even on their own), so we created some new implementations. For other algorithms where the authors provided us with an implementation, their implementation may have been in a different language and used different data structures from our own implementation. As a result, the time taken by different algorithms is not directly comparable. We report running times, but they should be taken as an upper bound on the computation time. All the implementations (our own and those of others) could be optimized further and, more notably for our study, we have not tried to ensure the time optimization is equitable. Rather, we are focusing on comparing the quality of the duplicates detected by each approach.

Some of the clustering algorithms were not originally designed for an input similarity graph and therefore we needed to make decisions on how to transform the similarity graph to suit the algorithm's input format. The original implementation of the Ricochet algorithms obtained from the authors worked only for complete graphs. Therefore, for all pairs of disconnected nodes (their similarity score was below θ), we added an edge with a small constant weight. In

our implementation, for the SR and BSR algorithms we used a constant value of 0.05, and for the CR and OCR algorithms we used a constant value of 0. In Correlation Clustering, we build the input correlation graph by assigning '$+$' to edges between nodes with similarity greater than θ, and assign '$-$' to edges between nodes with similarity less than θ.

For the results in this chapter, we use the term "Correlation Clustering to refer to the approximation algorithm Cautious [12] for general weighted graphs. A more efficient approximation is the CC-PIVOT algorithm [4], which is a randomized expected 3-approximation algorithm for the correlation clustering problem. This algorithm is similar to the CENTER algorithm, but the center nodes are chosen randomly and not from the sorted list of the edges. Based on our experiments, this randomization did not improve the quality of the clusters on average comparing with the CENTER algorithm, and we therefore do not include the results in this chapter.

For the MCL algorithm, we employed the latest C implementation of MCL, provided by the original author of the algorithm.[2] As noted previously, we fix the inflation parameter of the MCL algorithm for all the data sets and treat it as an unconstrained algorithm. We use the default parameter value ($I = 2.0$) as recommended by the author of the algorithm. For the Cut Clustering algorithm, we used a C implementation of the push-relabel algorithm available from the authors.[3] We evaluated different values for the parameter α across a subset of the data sets at varying thresholds to find the α value that would produce a total number of clusters (for each data set) that was closest to the ground truth. We found that $\alpha = 0.2$ worked best and used this value throughout our tests.

B.4 Results

We first report the results of our experiments and observations for each individual algorithm. We use the Partitioning algorithm, which returns the connected components in the similarity graph as clusters, as the baseline for our evaluations. We then present a comparative perfor-

[2] http://micans.org/mcl/src/mcl-06-058/
[3] http://www.avglab.com/andrew/soft.html

mance study among the algorithms, and finally, we report the running times for each algorithm.

Most of the accuracy results reported in this chapter are average results over the medium error class of data sets in Table B.1 since we believe that these results are representative of algorithm behaviour when using data sets with different characteristics. We show the results from other data sets whenever the trends deviate from the medium class data sets. Our extensive experimental results over all the 29 data sets in Table B.1 are publicly available at: http://dblab.cs.toronto.edu/project/stringer/clustering/

B.4.1 Individual Results

Single-pass Algorithms.

Table B.3 below shows the accuracy values for the single-pass algorithms over medium-error data sets and two thresholds that result in the best average F1 measure and the best average PCPr values in these algorithms. Similar trends were observed for the other thresholds and data sets.

Table B.3: Accuracy of single-pass algorithms

	Partitioning		CENTER		MC	
	Best PCPr	Best F_1	Best PCPr	Best F_1	Best PCPr	Best F_1
PCPr	0.554	0.469	0.638	0.298	0.695	0.437
CPr	0.946	0.805	0.818	0.692	0.940	0.795
Pr	0.503	0.934	0.799	0.971	0.658	0.958
Re	0.906	0.891	0.860	0.805	0.950	0.885
F1	0.622	0.910	0.825	0.877	0.776	0.918
Cluster#	354	994	697	1305	459	1030

The results above show that, assuming that the optimal threshold for the similarity join is known for, the CENTER algorithm performs better than the Partitioning algorithm, and that the MC algorithm is more effective than both CENTER and Partitioning over these data sets. This is to be expected since Partitioning puts many unsimilar records in the same cluster resulting in higher recall, but considerably lower precision. CENTER puts many similar records

in different clusters resulting in lower recall, but higher precision. These results show that MC creates clusters with precision lower than CENTER but higher than Partitioning, with a recall that is almost as high as that of Partitioning.

Note that the number of clusters in the ground truth is 500. The last row in the table shows the number of clusters generated by each algorithm. These results show that precision, recall and F_1 measures alone cannot determine the best algorithm since they do not take into account the number of clusters generated, this justifies using the CPr and PCPr measures. Furthermore, we can observe the high degree of sensitivity of all these algorithms to the threshold value used in the similarity join, although MC is less sensitive to this value among the single-pass algorithms.

Star algorithm.

Table B.4 below gives the results for the Star algorithm, revealing that the algorithm has a better performance in terms of accuracy when a lower threshold value is used. With a lower threshold value, the Star algorithm significantly outperforms the Partitioning algorithm, and is less sensitive to the value of the threshold used. However, for higher thresholds, the quality of the clustering considerably decreases. This is because the star centers in this algorithm are chosen based on the degree of the nodes. Using higher threshold values decreases the degree of all the nodes in the graph and makes the choice of a proper cluster center harder, which results in clusters of lower quality. It is also worth mentioning that even with an ideal threshold, the Star algorithm's accuracy is less than the accuracy of the single-pass algorithms.

Table B.4: Accuracy of star algorithm

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	Star	Part.	Star	Part.	Star
PCPr	0.101	0.614	0.554	0.601	0.645	0.445
CPr	0.991	0.726	0.946	0.801	0.879	0.781
Pr	0.104	0.588	0.503	0.778	0.788	0.900
Re	0.953	0.730	0.906	0.842	0.929	0.870
F1	0.177	0.644	0.622	0.805	0.850	0.884
Cluster#	51	521	354	715	704	949

Ricochet family of algorithms.

The accuracy results for SR and BSR, presented in Table B.5 below for two similarity threshold values, show that these algorithms are also more effective at lower thresholds, but are overall more robust (less sensitive) across varying threshold values.

Table B.5: Accuracy of sequential Ricochet algorithms

	$\theta = 0.2$			$\theta = 0.4$		
	Part.	SR	BSR	Part.	SR	BSR
PCPr	0.101	0.628	0.466	0.645	0.590	0.578
CPr	0.991	0.821	0.868	0.879	0.754	0.895
Pr	0.104	0.989	0.675	0.788	0.991	0.828
Re	0.953	0.863	0.932	0.929	0.818	0.930
F1	0.177	0.917	0.779	0.850	0.893	0.873
Cluster#	51	735	268	704	703	323

OCR and CR algorithms, on the other hand, are very sensitive to the threshold value, and are more effective at higher θ values as shown in Table B.6. This is again due to different way of choosing cluster seeds (or centers) used in these algorithms. Marking all the nodes as seeds and gradually merging the clusters, as done in OCR and CR, results in higher quality clusters when the threshold value is high (i.e., the similarity graph is not dense) but does not work well when the threshold value is low (i.e., the similarity graph is very dense). On the other hand, when the seeds are chosen sequentially based on the weight of the nodes, as done in SR and BSR, a lower threshold value (i.e., a dense similarity graph) results in more accurate weight values and therefore better choice of cluster seeds and higher quality clusters.

Table B.6: Accuracy of concurrent Ricochet algorithms

	$\theta = 0.2$			$\theta = 0.5$		
	Part.	CR	OCR	Part.	CR	OC
PCPr	0.101	0.494	0.351	0.469	0.402	0.687
CPr	0.991	0.967	0.981	0.805	0.782	0.817
Pr	0.104	0.434	0.299	0.934	0.958	0.862
Re	0.953	0.869	0.952	0.891	0.869	0.883
F1	0.177	0.567	0.454	0.910	0.910	0.872
Cluster#	51	258	180	994	1079	593

Cut Clustering (MinCut).

Clustering the similarity graph based on minimum cuts improves the quality of the clustering when compared to the Partitioning algorithm, as shown in Table B.7 below. This improvement is significant as we increase the threshold up to 0.4. For $\theta > 0.4$, MinCut Clustering produces the same clusters as the Partitioning algorithm, since as the input graph becomes less dense, only significantly related records remain connected and further cutting the edges does not improve the quality of the clusters.

Table B.7: Accuracy of cut clustering

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	MinCut	Part.	MinCut	Part.	MinCut
PCPr	0.101	0.105	0.554	0.683	0.645	0.689
CPr	0.991	0.509	0.946	0.891	0.879	0.875
Pr	0.104	0.833	0.503	0.672	0.788	0.827
Re	0.953	0.564	0.906	0.908	0.929	0.926
F1	0.177	0.671	0.622	0.771	0.850	0.873
Clstr#	51	2450	354	665	704	735

Articulation Point Clustering (ArtPt).

As the results in Table B.8 show, the Articulation Point clustering slightly enhances the Partitioning algorithm by splitting some of the components in the graph into a few more clusters. This makes the algorithm only slightly less sensitive to the threshold value. The algorithm works best with the optimal threshold for the Partitioning algorithm (the θ value that creates partitions of highest quality in the Partitioning algorithm).

Markov Clustering (MCL).

As shown in Table B.9 below, the MCL algorithm produces clusters of increased quality than those created by the Partitioning algorithm. The MCL algorithm is also most effective when used with the optimal threshold value, although it is much less sensitive overall across varying θ values. This shows the effectiveness of the flow simulation process using random walks on the

Table B.8: Accuracy of articulation point algorithm

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	ArtPt.	Part.	ArtPt.	Part.	ArtPt.
PCPr	0.101	0.169	0.554	0.655	0.645	0.680
CPr	0.991	0.988	0.946	0.941	0.879	0.871
Pr	0.104	0.157	0.503	0.581	0.788	0.825
Re	0.953	0.920	0.906	0.891	0.929	0.925
F1	0.177	0.251	0.622	0.693	0.850	0.871
Cluster#	51	86	354	428	704	754

graph, and that unlike the Star and SR algorithms, pruning the edges with low weights does not affect the quality of the clusters significantly, and unlike Partitioning and CR, a dense similarity graph (i.e., not pruning the low-weight edges) does not result in clusters with low precision.

Table B.9: Accuracy of the MCL algorithm

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	MCL	Part.	MCL	Part.	MCL
PCPr	0.101	0.599	0.554	0.768	0.645	0.644
CPr	0.991	0.934	0.946	0.921	0.879	0.866
Pr	0.104	0.571	0.503	0.754	0.788	0.888
Re	0.953	0.951	0.906	0.952	0.929	0.925
F1	0.177	0.712	0.622	0.841	0.850	0.906
Cluster#	51	323	354	528	704	777

Correlation Clustering (CCL)

This algorithm performs best when using lower threshold values, producing clusters with much higher quality than those created by the Partitioning algorithm. The quality of the produced clusters degrade at higher θ values. This is to be expected since the algorithm performs clustering based on correlation information between the nodes and a higher θ means a loss of this information.

Table B.10: Accuracy of correlation clustering algorithm

	$\theta = 0.2$		$\theta = 0.3$		$\theta = 0.4$	
	Part.	CCL	Part.	CCL	Part.	CCL
PCPr	0.101	0.612	0.554	0.542	0.645	0.406
CPr	0.991	0.711	0.946	0.762	0.879	0.748
Pr	0.104	0.596	0.503	0.803	0.788	0.914
Re	0.953	0.750	0.906	0.822	0.929	0.844
F1	0.177	0.659	0.622	0.808	0.850	0.876
Cluster#	51	538	354	753	704	1000

B.4.2 Overall Comparison

Sensitivity to the value of the threshold.

Table B.11 shows the accuracy results of all the algorithms for different thresholds over the medium-error class of data sets. These results can be used to compare different algorithms when using a fixed threshold, i.e., with the same similarity graph as input. Among all the algorithms, SR and BSR are least sensitive to the threshold value. However their accuracy does not always outperform the other algorithms. In other algorithms, those that use the weight and degree of the edges for the clustering perform relatively better using lower threshold values, when the similarity graph is more dense. Therefore, CENTER, Star, Correlation Clustering and MCL algorithms perform better with low threshold values comparing with the other algorithms. The single-pass algorithms along with Articulation Point and MinCut algorithms are generally more sensitive to the threshold value and are considerably more effective when used with the optimal threshold (when the number of components in the graph is close to the number of ground truth clusters), with MERGE-CENTER being the least sensitive among them.

Table B.11: Average accuracy of all the algorithms for different thresholds over medium-error data sets

θ	Measure	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
0.2	PCPr	0.101	0.593	0.257	0.614	0.628	0.466	0.494	0.351	0.612	0.599	0.105	0.169
	F1	0.177	0.666	0.389	0.644	*0.917*	0.779	0.567	0.454	0.659	0.712	0.671	0.251
	Cluster#	51	472	134	521	735	268	258	180	538	323	2450	86
0.3	PCPr	0.554	0.638	0.695	0.601	0.616	0.564	0.718	0.578	0.542	*0.768*	0.683	0.655
	F1	0.622	0.825	0.776	0.805	0.907	0.863	0.791	0.718	0.808	0.841	0.771	0.693
	Cluster#	354	697	459	715	721	315	527	306	753	528	665	428
0.4	PCPr	0.645	0.445	0.692	0.445	0.590	0.578	0.640	0.629	0.406	0.644	0.689	0.680
	F1	0.850	0.887	0.894	0.884	0.893	0.873	0.887	0.819	0.876	0.906	0.873	0.871
	Cluster#	704	956	750	949	703	323	786	454	1000	777	735	754
0.5	PCPr	0.469	0.298	0.437	0.315	0.560	0.552	0.402	0.687	0.281	0.423	0.465	0.409
	F1	0.910	0.877	0.918	0.892	0.874	0.861	0.910	0.872	0.867	0.921	0.913	0.907
	Cluster#	994	1305	1030	1255	678	324	1079	593	1340	1048	998	1069
0.6	PCPr	0.284	0.225	0.275	0.234	0.546	0.527	0.264	0.546	0.000	0.273	0.284	0.264
	F1	0.897	0.841	0.892	0.861	0.847	0.839	0.886	0.851	0.183	0.892	0.897	0.887
	Cluster#	1345	1650	1378	1593	634	323	1435	746	4979	1382	1345	1438
0.7	PCPr	0.203	0.190	0.201	0.192	0.555	0.496	0.199	0.464	0.000	0.202	0.203	0.198
	F1	0.838	0.798	0.833	0.809	0.806	0.804	0.827	0.792	0.183	0.834	0.838	0.826
	Cluster#	1793	1979	1815	1953	566	317	1853	835	4979	1804	1793	1868
0.8	PCPr	0.175	0.173	0.174	0.173	0.604	0.458	0.174	0.340	0.000	0.175	0.175	0.174
	F1	0.773	0.757	0.768	0.761	0.768	0.730	0.768	0.730	0.183	0.772	0.773	0.765
	Cluster#	2173	2232	2188	2227	*504*	307	2196	1709	4979	2176	2173	2209

Effect of the amount of errors.

The results in Table B.12 show the best accuracy values obtained by the algorithms on data sets with different amounts of error, along with the difference (Diff.) between the value obtained for the high error and low error groups of data sets. Note that the accuracy numbers in this table cannot be used to directly compare the algorithms since they are based on different thresholds and therefore the input similarity graph is different for each algorithm. We use these results to compare the effect of the amount of error on different algorithms. These results suggest that the Ricochet group of algorithms and MCL algorithm are relatively more robust on data sets with different amounts of errors, i.e., they perform equally well on the three groups of data sets with lowest drop in the quality of the clusters on high error data sets comparing with low error groups of data sets.

Sensitivity to the distribution of the errors.

Table B.13 shows the best accuracy values obtained for the algorithms on medium-error data sets with uniform and Zipfian distributions. Note that in Zipfian data sets, there are many records with no duplicates (singleton clusters) and only a few records with many duplicates. Due to the fact that the PCPr measure is the average CPr value for all the clusters and is calculated for clusters of size 2 or more, this accuracy measure is less indicative of the performance of the algorithms on this class of data sets. These results show that all the algorithms are equally robust with respect to the distribution of the errors in the data except SR, BSR and OCR algorithms from the Ricochet family which produce clusters of significantly lower quality. This is mainly due to the inability of these algorithms in finding singleton clusters.

Table B.12: Best accuracy values for all the algorithms over different groups of data sets

Measure	Group	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
Max. PCPr	Low	0.842	0.849	0.904	0.841	0.854	0.661	0.918	0.847	0.818	0.921	0.855	0.900
	Medium	0.645	0.638	0.695	0.614	0.633	0.578	0.718	0.687	0.612	0.768	0.689	0.680
	High	0.399	0.217	0.340	0.197	0.538	0.461	0.632	0.557	0.175	0.476	0.232	0.278
	Diff.	-0.443	-0.632	-0.565	-0.644	-0.316	-0.201	-0.286	-0.290	-0.642	-0.445	-0.623	-0.621
Max. F1	Low	0.959	0.956	0.960	0.953	0.976	0.918	0.957	0.917	0.951	0.960	0.959	0.957
	Medium	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.876	0.921	0.913	0.907
	High	0.685	0.640	0.734	0.660	0.853	0.695	0.733	0.640	0.624	0.760	0.760	0.668
	Diff.	-0.273	-0.316	-0.225	-0.292	-0.123	-0.223	-0.223	-0.277	-0.327	-0.199	-0.198	-0.288
Best Cluster#	Low	428	460	460	471	501	364	468	445	489	471	434	458
	Medium	354	472	459	521	504	386	527	454	538	528	665	428
	High	919	203	200	221	470	356	643	455	236	236	1404	143
	Diff.	+491	-257	-260	-250	-32	-8	+175	+11	-254	-235	+970	-315

Table B.13: Best accuracy values for all the algorithms over medium-error data sets with different distributions

Measure	Group	Part.	CENTER	MergeC	Star	SR	BSR	CR	OCR	CCL	MCL	MinCut	ArtPt.
F1	Uniform	0.910	0.887	0.918	0.892	0.920	0.873	0.910	0.872	0.876	0.921	0.721	0.907
	Zipfian	0.936	0.936	0.938	0.934	0.873	0.463	0.935	0.697	0.929	0.937	0.819	0.934
	Diff.	+0.026	+0.049	+0.020	+0.041	-0.047	-0.411	+0.025	-0.175	+0.054	+0.016	+0.098	+0.027
Cluster#	Uniform	354	472	459	521	504	386	527	454	538	528	665	428
	Zipfian	1018	934	1047.5	933	698.5	158	1061	992	955.25	1021	1038	1067

Effectiveness in predicting the correct number of clusters.

The results of our experiments, partly shown in Tables B.11,B.12 and B.13, show that none of the algorithms are capable of accurately predicting the number of clusters regardless of the characteristics of the data set. For uniform data sets, SR algorithms perform extremely well for finding the correct number of clusters on data sets with different amounts of errors. However, this algorithm fails when it comes to data sets with a Zipfian distribution of errors. Overall, algorithms that find star-shaped clusters, namely CENTER, MERGE-CENTER, Star, CR and OCR algorithms, can effectively find the right number of data sets with an optimal threshold. The graph-theoretic algorithms Correlation clustering and Markov clustering also find a reasonable number of clusters at lower thresholds.

B.4.3 Running Time

As stated previously, in this work we are focusing mainly on comparing the quality of the duplicates detected by each algorithm. However we do report the running times in this section, but the times taken by the different algorithms are not directly comparable, and should be taken as an upper bound on the computation time. All the implementations (our own and those of others) could be optimized further, and more notably for our study, we have not tried to ensure the time optimization is equitable. Different implementations and machines are used to run these experiments. The table below shows the running times for 8 of the algorithms run using a data set of 100K records of DBLP titles described in Section B.1, and with threshold $\theta = 0.4$, giving $386,068$ edges in the similarity graph. PC1 is a Dell 390 Precision desktop with 2.66 GHz Intel Core2 Extreme Quad-Core Processor QX6700, 4GB of RAM running 32-bit Windows Vista. PC2 is an AMD OPTERON 850 2.4GHz, 16 GB of RAM running Red Hat Linux 3.4. PC3 is a Dual Core AMD Opteron Processor 270 (2GHz) with 6GB of memory running Red Hat Linux 2.6. Each experiment is run multiple times to obtain statistical significance.

The implementation for the Ricochet algorithms required building a complete similarity

Table B.14: Running times of the scalable clustering algorithms

Algorithm	Time	Lang./Machine
Partitioning	1.790 sec	Java/PC1
CENTER	3.270 sec	Java/PC1
MERGE-CENTER	3.581 sec	Java/PC1
Star	5.9 min	Java/PC1
CCL	83.5 min	Java/PC2
MCL	8.395 sec	C/PC2
MinCut	52.1 min	C & Perl/PC3
ArtPt.	17.563 sec	Perl/PC3

graph in memory. Therefore, running the algorithm on the data set of 100K records required keeping a graph with 100 billion edges in memory which was not possible. We therefore had to limit the size of the data set for these experiments. We used a data set of 5K records with threshold $\theta = 0.1$ resulting in 391,706 edges. The running times are shown in the table below.

Table B.15: Running times of the Ricochet clustering algorithms

Algorithm	Time	Lang./Machine
SR	19.687 sec	Java/PC1
BSR	54.115 sec	Java/PC1
CR	9.211 sec	Java/PC1
OCR	8.972 sec	Java/PC1

These results support the scalability of single-pass algorithms as well as MCL and Articulation Point clustering algorithms. Note that we used the original implementation of the MCL algorithm which is highly optimized. Such an optimized implementation was not available for the Correlation Clustering algorithm.

B.4.4 Summary and Concluding Remarks

In this chapter, we evaluated and compared several unconstrained clustering algorithms for duplicate detection by extensive experiments over various sets of string data with different characteristics. We made the results of our extensive experiments publicly available and we intend to keep the results up-to-date with state-of-the-art clustering algorithms and various synthetic

and real data sets. We hope these results serve as a guideline for researchers and practitioners interested in using unconstrained clustering algorithms especially for the task of duplicate detection.

A summary of the results is presented in Figure B.1. Our results using partitioning of the similarity graph (finding the transitive closure of the similarity relation) which is the common approach in many early duplicate detection techniques, confirms the common wisdom that this scalable approach results in poor quality of duplicate groups. But more importantly, we show that this quality is poor even when compared to other clustering algorithms that are as efficient. The Ricochet algorithms produce high quality clusterings when used with uniformly distributed duplicates, but failed in other distributions. All other algorithms were robust to the distribution of the duplicates. Our results also show that sophisticated but popular algorithms, like Cut clustering and Correlation clustering, gave lower accuracy than some of the more efficient single-pass algorithms. We were the first to propose the use of Markov clustering as an unconstrained algorithm for duplicate detection and showed that in fact it is among the most accurate algorithms for this task and is also very efficient.

A basic observation here is that none of the clustering algorithms produce perfect clusterings. Therefore a reasonable approach is to not only keep the clustering that results from our algorithms, but to also keep the important quantitative information produced by these algorithms. In [95], we show how this quantitative information can be used to provide an accurate confidence score for each duplicate that can be used in probabilistic query answering.

	Scalability (Current Implementations)	Ability to find the correct number of clusters	Robustness Against		
			Choice of threshold	Amount of Errors	Distribution of errors
Partitioning	High	Low	Low	Low	High
CENTER	High	High	Low	Low	High
MERGE CENTER	High	High	Low	Low	High
Star	Medium	High	Low	Low	High
SR	Low	Medium	High	High	Low
BSR	Low	Low	High	High	Low
CR	Low	High	Medium	High	High
OCR	Low	High	Medium	High	Low
Correlation Clustering	Low	High	Low	Low	High
Markov Clustering	High	High	Medium	Medium	High
Cut Clustering	Low	Low	Low	Low	High
Articulation Point	High	Medium	Low	Low	High

Figure B.1: Summary of the results

Bibliography

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999. ↪ 6, 118, 121

[2] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable Ad-Hoc Entity Extraction From Text Collections. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):945–957, 2008. ↪ 41

[3] S. Agrawal, K. Chakrabarti, S. Chaudhuri, V. Ganti, A. C. König, and D. Xin. Exploiting Web Search Engines To Search Structured Databases. In *Int'l World Wide Web Conference (WWW)*, pages 501–510, 2009. ↪ 41

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. In *ACM Symp. on Theory of Computing (STOC)*, pages 684–693, 2005. ↪ 36, 180

[5] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 432–447, 2008. ↪ 119

[6] F. Amato, A. Mazzeo, A. Penta, and A. Picariello. Building RDF Ontologies from Semi-Structured Legal Documents. In *Proc. of the Int'l Conf. on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 997–1002, 2008. ↪ 119

[7] Y. An, A. Borgida, and J. Mylopoulos. Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies. *J. of Comp. Sci. & Eng.*, 2(1):44–73, 2008. ↪ 117

[8] A. Arasu, V. Ganti, and R. Kaushik. Efficient Exact Set-Similarity Joins. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 918–929, 2006. ↪ 20, 62

[9] A. Arasu, C. Ré, and D. Suciu. Large-Scale Deduplication with Constraints Using Dedupalog. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 952–963, 2009. ↪ 39, 96

[10] J. A. Aslam, E. Pelekhov, and D. Rus. The Star Clustering Algorithm For Static And Dynamic Information Organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004. ↪ 33

[11] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-Weight Linked Data Publication from Relational Databases. In *Int'l World Wide Web Conference (WWW)*, pages 621–630, 2009. ↪ 50, 84

[12] N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004. ↪ 31, 35, 36, 163, 180

[13] J. Barrett. What's Behind Clinical Trial Data? *Applied Clinical Trials (Online)*, January 2009. ↪ 143

[14] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ↪ 16

[15] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Int'l World Wide Web Conference (WWW)*, pages 131–140, Banff, Canada, 2007. ↪ 62, 161

[16] I. Bedini. *Deriving Ontologies Automatically from XML Schemas Applied to the B2B Domain*. PhD thesis, University of Versailles, France., 2010. ↪ 117

[17] Z. Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping (Data-Centric Systems and Applications)*. Springer, 1st edition, 2011. ↪ 40

[18] T. Berners-Lee. Linked Data - Design Issues. http://www.w3.org/DesignIssues/LinkedData.html, 2006. [Online; accessed 31-10-2010]. ↪ 2, 149

[19] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. In *Int'l World Wide Web Conference (WWW)*, pages 825–834, 2008. ↪ 118

[20] G. J. Bex, F. Neven, and J. V. den Bussche. DTDs versus XML Schema: A Practical Study. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, pages 79–84, 2004. ↪ 121

[21] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C.-C. Kanne, F. Özcan, and E. J. Shekita. Jaql: A Scripting Language for Large Scale Semistructured Data Analysis. *Proceedings of the VLDB Endowment (PVLDB)*, 4(11):1272–1283, 2011. ↪ 164

[22] I. Bhattacharya and L. Getoor. Collective Entity Resolution in Relational Data. *IEEE Data Engineering Bulletin*, 29(2):4–12, 2006. ↪ 30

[23] I. Bhattacharya and L. Getoor. Collective Entity Resolution in Relational Data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007. ↪ 29, 30

[24] I. Bhattacharya and L. Getoor. Query-time Entity Resolution. *Journal of Artificial Intelligence Research (JAIR)*, 30:621–657, 2007. ↪ 30

[25] M. Bilenko and R. J. Mooney. Adaptive Duplicate Detection using Learnable String Similarity Measures. In *Proc. of the Int'l Conf. on Knowledge Discovery & Data Mining*, pages 39–48, 2003. ↪ 21, 29

[26] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-Dupe: An Interactive Tool for Entity Resolution in Social Networks. In *IEEE VAST*, pages 43–50, 2006. ↪ 46

[27] A. Bilke, J. Bleiholder, C. Böhm, K. Draba, F. Naumann, and M. Weis. Automatic Data Fusion with HumMer. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 1251–1254, 2005. ↪ 83

[28] A. Bilke and F. Naumann. Schema Matching Using Duplicates. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 69–80, 2005. ↪ 40

[29] C. Bizer and R. Cyganiak. D2R Server - Publishing Relational Databases on the Semantic Web. Poster at the 5th International Semantic Web Conference (ISWC2006), 2006. ↪ 4, 117, 145

[30] C. Bizer, R. Cyganiak, and T. Gauss. The RDF Book Mashup: From Web APIs to a Web of Data. In *Proceedings of the ESWC Workshop on Scripting for the Semantic Web (SFSW)*, 2007. ↪ 119

[31] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data: Principles and State of the Art. In *Int'l World Wide Web Conference (WWW)*, November 2008. ↪ 2

[32] C. Bizer, A. Jentzsch, and R. Cyganiak. State of the LOD Cloud. `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`, September 2011. [Online; accessed 31-10-2011]. ↪ 2, 49

[33] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A Crystallization Point for the Web of Data. *J. Web Sem.*, 7(3):154–165, 2009. ↪ 3, 5, 70

[34] C. Bizer and A. Schultz. The R2R Framework: Publishing and Discovering Mappings on the Web. In *1st International Workshop on Consuming Linked Data (COLD 2010)*, 2010. ↪ 112

[35] C. Bizer and A. Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *Proc. of the Int'l Semantic Web Conference (ISWC)*, November 2004. ↪ 50, 84, 145

[36] C. Bizer, J. Volz, G. Kobilarov, and M. Gaedke. Silk - A Link Discovery Framework for the Web of Data. In *WWW 2009 Workshop on Linked Data on the Web (LDOW2011)*, April 2009. ↪ 45, 46

[37] A. Budanitsky and G. Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1):13–47, 2006. ↪ 28

[38] P. Buneman, J. Cheney, W. C. Tan, and S. Vansummeren. Curated Databases. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 1–12, 2008. ↪ 119

[39] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding Struture to Unstructured Data. In *Proc. of the Int'l Conf. on Database Theory (ICDT)*, pages 336–350, 1997. ↪ 118

[40] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient Batch Top-k Search for Dictionary-based Entity Recognition. In *ICDE*, page 28, 2006. ↪ 41

[41] M. Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symp. on Theory of Computing (STOC)*, pages 380–388, 2002. ↪ 161

[42] M. Charikar, V. Guruswami, and A. Wirth. Clustering with Qualitative Information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. ↪ 36

[43] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-Driven Design of Efficient Record Matching Queries. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 327–338, 2007. ↪ 29

[44] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and Efficient Fuzzy Match for Online Data Cleaning. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 313–324, 2003. ↪ 25

[45] S. Chaudhuri, V. Ganti, and D. Xin. Exploiting Web Search To Generate Synonyms For Entities. In *Int'l World Wide Web Conference (WWW)*, pages 151–160, 2009. ↪ 41

[46] P. Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proc. of the Int'l Conf. on Knowledge Discovery & Data Mining*, pages 1065–1068, 2008. ↪ 45, 46

[47] P. Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2011. ↪ 18, 45

[48] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Transactions on Information Systems*, 18(3):2000, 2000. ↪ 46

[49] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, Acapulco, Mexico, 2003. ↪ 20, 26, 161

[50] W. W. Cohen and S. Sarawagi. Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods. In *Proc. of the Int'l Conf. on Knowledge Discovery & Data Mining*, pages 89–98, 2004. ↪ 41

[51] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw Hill and MIT Press, 1990. ↪ 38

[52] R. Cyganiak and C. Bizer. Pubby - A Linked Data Frontend for SPARQL Endpoints. `http://www4.wiwiss.fu-berlin.de/pubby/`. [Online; accessed 12-July-2011]. ↪ 4

[53] B. T. Dai, N. Koudas, D. Srivastava, A. K. H. Tung, and S. Venkatasubramanian. Validating Multi-column Schema Matchings by Type. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 120–129, 2008. ↪ 40, 110

[54] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting Ontology-Based Semantic matching in RDBMS. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 1054–1065, 2004. ↪ 83

[55] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust Record Linkage Blocking Using Suffix Arrays and Bloom Filters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):9, 2011. ↪ 45

[56] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004. ↪ 163

[57] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation Clustering In General Weighted Graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006. ↪ 36

[58] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. DBLife: A Community Information Management Platform for the Database Research Community (Demo). In *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, pages 169–172, 2007. ↪ 119

[59] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 383–394, 2004. ↪ 111

[60] L. Ding, T. Finin, J. Shinavier, and D. L. McGuinness. owl:sameAs and Linked Data: An Empirical Study. In *Proceedings of the Web Science Conference*, April 2010. ↪ 2

[61] E. A. Dinic. Algorithm for Solution of a Problem Of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl*, 11:1277–1280, 1970. ↪ 37

[62] A. Doan and A. Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1):83–94, 2005. ↪ 40

[63] X. Dong, A. Y. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 85–96, 2005. ↪ 30

[64] H. L. Dunn. Record Linkage. *Am J Public Health Nations Health*, 36(12):1412–1416, 1946. ↪ 15, 17

[65] W. W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Technical report, The Data Warehousing Institute, 2002. ↪ 17

[66] J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972. ↪ 37

[67] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, 2002. ↪ 45, 46

[68] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007. ↪ 9, 16, 17, 47, 158

[69] O. Erling and I. Mikhailov. Virtuoso: RDF Support in a Native RDBMS. In *Semantic Web Information Management*, pages 501–519. Springer, 2009. ↪ 50, 84

[70] O. Etzioni, M. J. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artif. Intell.*, 165(1):91–134, 2005. ↪ 43

[71] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag, 2007. http://book.ontologymatching.org/. ↪ 19, 22, 26, 27, 28, 47, 111

[72] I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. ↪ 16

[73] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004. ↪ 37

[74] L. Ford and D. Fulkerson. Maximal Flow Through a Network. *Canadian J. Math*, 8:399–404, 1956. ↪ 37

[75] A. Gal. *Uncertain Schema Matching*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. ↪ 40

[76] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 371–380, 2001. ↪ 83

[77] M. Y. Galperin and G. Cochrane. The 2011 Nucleic Acids Research Database Issue and the Online Molecular Biology Database Collection. *Nucleic Acids Research*, 39(Database-Issue):1–6, 2011. ↪ 49

[78] M. Ganesh, J. Srivastava, and T. Richardson. Mining Entity-Identification Rules for Database Integration. In *Proc. of the Int'l Conf. on Knowledge Discovery & Data Mining*, page 291. AAAI Press, 1996. ↪ 39

[79] L. Getoor and C. P. Diehl. Link Mining: A Survey. *SIGKDD Explorations*, 7(2):3–12, 2005. ↪ 1

[80] R. Gil, J. M. Brunetti, J. M. Gimeno, and R. Garcia. Can Linked Data Improve the User Experience (UX). In *The 10th International Semantic Web Conference (ISWC2011), Outrageous Ideas Track*, 2011. ↪ 143

[81] Y. Gil and P. Groth. Linked Data for Network Science. In *Proceedings of the First International Workshop on Linked Science 2011 (2011)*, 2011. ↪ 148

[82] Y. Gil and P. Groth. LinkedDataLens: Linked Data as a Network of Networks. In *Proceedings of the Sixth International Conference on Knowledge Capture (K-CAP '11)*, pages 191–192, 2011. ↪ 148

[83] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, 1999. ↪ 122

[84] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 491–500, 2001. ↪ 20, 52, 60

[85] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record Linkage: Current Practice and Future Directions. Technical report, CSIRO Mathematical and Information Sciences, 2003. ↪ 16

[86] S. Guha, N. Koudas, D. Srivastava, and T. Yu. Index-Based Approximate XML Joins. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 708–710, 2003. ↪ 41

[87] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, 1997. ↪ 21

[88] H. Halpin and P. J. Hayes. When owl:sameAs isn't the Same: An Analysis of Identity Links on the Semantic Web. In *Proceedings of the WWW2009 workshop on Linked Data on the Web (LDOW2009)*, April 2010. ↪ 2

[89] O. Hassanzadeh. Benchmarking Declarative Approximate Selection Predicates. Master's thesis, University of Toronto, February 2007. ↪ 20, 23, 24, 52, 83, 166, 170, 176

[90] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1282–1293, 2009. ↪ 31, 175

[91] O. Hassanzadeh and M. P. Consens. Linked Movie Data Base (Triplification Challenge Report). In *Proceedings of the International Conference on Semantic Systems (I-SEMANTICS'08)*, pages 194–196, 2008. ↪ 142

[92] O. Hassanzadeh and M. P. Consens. Linked Movie Data Base. In *Proceedings of the WWW2009 workshop on Linked Data on the Web (LDOW2009)*, 2009. ↪ 134

[93] O. Hassanzadeh, S. Duan, A. Fokoue, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Helix: Online Enterprise Data Analytics. In *Proceedings of the 20th International World Wide Web Conference (WWW2011) - Demo Track*, 2011. ↪ 112, 164

[94] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A Framework for Semantic Link Discovery over Relational Data. In *Proc. of the ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1027–1036, 2009. ↪ 49

[95] O. Hassanzadeh and R. J. Miller. Creating Probabilistic Databases from Duplicated Data. *The Int'l Journal on Very Large Data Bases*, 18(5):1141–1166, 2009. ↪ 32, 48, 192

[96] O. Hassanzadeh, M. Sadoghi, and R. J. Miller. Accuracy of Approximate String Joins Using Grams. In *Proc. of the International Workshop on Quality in Databases (QDB)*, pages 11–18, Vienna, Austria, 2007. ↪ 166

[97] O. Hassanzadeh, R. Xin, R. J. Miller, A. Kementsietsidis, L. Lim, and M. Wang. Linkage Query Writer. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2):1590–1593, 2009. ↪ 49

[98] O. Hassanzadeh, R. S. Xin, C. Fritz, Y. Yang, J. Du, M. Zhao, and R. J. Miller. BibBase Triplified (Triplification Challenge Report). In *Proceedings of the 6th International Conference on Semantic Systems, September 1–3, 2010, Graz, Austria*, 2010. ↪ 149, 157

[99] M. Hausenblas, W. Halb, and Y. Raimond. Scripting User Contributed Interlinking. In *4th Workshop on Scripting for the Semantic Web (SFSW08)*, 2008. ↪ 143

[100] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable Techniques for Clustering the Web. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, pages 129–134, Dallas, Texas, USA, 2000. ↪ 32

[101] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2011. ↪ 155

[102] J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *ICDE Workshops*, page 81, 2006. ↪ 118

[103] I. Herman. BibTeX in RDF. http://ivan-herman.name/2007/01/13/bibtex-in-rdf/, 2007. [Online; accessed 14-06-2010]. ↪ 149, 155

[104] M. A. Hernández and S. J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998. ↪ 32, 71, 166

[105] M. Herschel and F. Naumann. Scaling Up Duplicate Detection in Graph Data. In *Proc. of the ACM Conf. on Information and Knowledge Management (CIKM)*, pages 1325–1326, 2008. ↪ 30

[106] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-Preserving Hashing in Multidimensional Spaces. In *ACM Symp. on Theory of Computing (STOC)*, pages 618–625, 1997. ↪ 62

[107] A. Isaac, L. van der Meij, S. Schlobach, and S. Wang. An Empirical Study of Instance-Based Ontology Matching. In *Proc. of the Int'l Semantic Web Conference (ISWC)*, pages 253–266, 2007. ↪ 111

[108] R. Isele and C. Bizer. Learning Linkage Rules using Genetic Programming. In *ISWC'11 Workshop on Ontology Matching*, 2011. ↪ 111

[109] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988. ↪ 34

[110] M. A. Jaro. Unimatch: A Record Linkage System: User□s Manual. Technical report, US Bureau of the Census, Washington, D.C., 1976. ↪ 21

[111] M. 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. ↪ 17

[112] A. Jentzsch, B. Andersson, O. Hassanzadeh, S. Stephens, and C. Bizer. Enabling Tailored Therapeutics with Linked Data. In *Proceedings of the WWW2009 workshop on Linked Data on the Web (LDOW2009)*, 2009. ↪ 143, 148

[113] A. Jentzsch, J. Zhao, O. Hassanzadeh, K.-H. Cheung, M. Samwal, and B. Andersson. Linking Open Drug Data (Triplification Challenge Report). In *Proceedings of the International Conference on Semantic Systems (I-SEMANTICS'09)*, 2009. ↪ 148

[114] J. J. Jiang and D. W. Conrath. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In *International Conference Research on Computational Linguistics (ROCLING X)*, pages 9008+, 1997. ↪ 27, 28

[115] J. Jonas. Identity Resolution: 23 Years of Practical Experience and Observations at Scale. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, page 718, 2006. ↪ 47

[116] S. C. K. Zai□, T. Schluter. Instance-based ontology matching using different kinds of formalisms. In *World Academy of Science, Engineering and Technology (WASET)*, pages 164–172, 2009. source: http://www.waset.org/journals/waset/v55.php. ↪ 111

[117] J. Kang and J. F. Naughton. On Schema Matching with Opaque Column Names and Data Values. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 205–216, 2003. ↪ 40, 88, 110

[118] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, pages 13–26, 2005. ↪ 117, 118

[119] A. Kementsietsidis, L. Lim, and M. Wang. Supporting Ontology-based Keyword Search over Medical Databases. In *Proceedings of the AMIA 2008 Symposium*, pages 409–13. American Medical Informatics Association, 2008. ↪ 62

[120] S. Khatchadourian and M. P. Consens. Explod: Summary-based exploration of interlinking and rdf usage in the linked open data cloud. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference (ESWC 2010)*, pages 272–287, 2010. ↪ 143

[121] N. Knouf. BibTeX Ontology. `http://purl.org/net/nknouf/ns/bibtex`. [Online; accessed 8-06-2011]. ↪ 155

[122] L. Kolb, A. Thor, and E. Rahm. Load Balancing for MapReduce-based Entity Resolution. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, page to appear, 2012. ↪ 164

[123] H. Köpcke and E. Rahm. Frameworks for Entity Matching: A Comparison. *Data and Knowledge Engineering*, 69(2):197–210, 2010. ↪ 47, 158

[124] H. Köpcke, A. Thor, and E. Rahm. Evaluation of Entity Resolution Approaches on Real-World Match Problems. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):484–493, 2010. ↪ 29

[125] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: A System for Declarative Information Extraction. *SIGMOD Record*, 37(4):7–13, 2008. ↪ 86, 101

[126] K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992. ↪ 167

[127] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. ↪ 21

[128] C. Li, B. Wang, and X. Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 303–314, Vienna, Austria, 2007. ↪ 20

[129] X. Li, D. Roth, and Y. Tu. Phrasenet: towards context sensitive lexical semantics. In *Proc. of the Annual Conference on Computational Natural Language Learning (CoNLL)*, 2003. ↪ 27

[130] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity Identification in Database Integration. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 294 –301, 1993. ↪ 39

[131] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search In Relational Databases. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 563–574, 2006. ↪ 42

[132] A. Maedche and S. Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001. ↪ 117

[133] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In *Int'l World Wide Web Conference (WWW)*, pages 141–150, 2007. ↪ 161

[134] S. Massmann and E. Rahm. Evaluating Instance-based Matching of Web Directories. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, 2008. ↪ 111

[135] D. R. H. Miller, T. Leek, and R. M. Schwartz. A Hidden Markov Model Information Retrieval System. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–221, 1999. ↪ 24, 170

[136] G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995. ↪ 27, 160

[137] J. Min, J. Ahn, and C. Chung. Efficient Extraction of Schemas for XML Documents. *Inf. Process. Lett.*, 85:7–12, January 2003. ↪ 118

[138] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. ↪ 30, 47, 158

[139] R. Navigli. Word Sense Disambiguation: A Survey. *ACM Comput. Surv.*, 41:10:1–10:69, February 2009. ↪ 28

[140] S. Nestorov, S. Abiteboul, and R. Motwani. Infering Structure in Semistructured Data. *SIGMOD Record*, 26(4):39–43, 1997. ↪ 118

[141] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchial Data. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 79–90, 1997. ↪ 118, 122

[142] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic Linkage of Vital Records. *Science*, 130:954–959, 1959. ↪ 15

[143] A.-C. N. Ngomo and S. Auer. LIMES A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2312–2317, 2011. ↪ 45, 46, 111

[144] A.-C. N. Ngomo, J. Lehmann, S. Auer, and K. Höffner. RAVEN - Active Learning of Link Specifications. In *ISWC'11 Workshop on Ontology Matching*, 2011. ↪ 111

[145] N. Okazaki and J. Tsujii. Simple and Efficient Algorithm for Approximate Dictionary Matching. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 851–859, 2010. ↪ 44

[146] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 1099–1110, 2008. ↪ 164

[147] C. Pesquita, D. Faria, A. O. Falcao, P. Lord, and F. M. Couto. Semantic Similarity in Biomedical Ontologies. *PLoS Comput Biol*, 5(7):e1000443, 7 2009. ↪ 27

[148] K. Q. Pu, O. Hassanzadeh, R. Drake, and R. J. Miller. Online Annotation of Text Streams with Structured Entities. In *Proc. of the ACM Conf. on Information and Knowledge Management (CIKM)*, pages 29–38, 2010. ↪ 42

[149] K. Q. Pu and X. Yu. Keyword Query Cleaning. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):909–920, 2008. ↪ 42

[150] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Int'l Journal on Very Large Data Bases*, 10(4):334–350, 2001. ↪ 40, 87, 111

[151] J. W. Ratclif. Pattern Matching: The Gestalt Approach. *Dr. Dobbs Journal*, pages 46–51, July 1988. ↪ 44

[152] P. Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995. ↪ 27

[153] P. Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999. ↪ 27, 28

[154] S. Robertson. Understanding Inverse Document Frequency: On Theoretical Arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004. ↪ 170

[155] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *The Text REtrieval Conference (TREC)*, 1994. ↪ 24, 95

[156] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. T. Jr, S. Auer, J. Sequeda, and A. Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. Technical report, W3C RDB2RDF incubator group, 2009. ↪ 4

[157] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975. ↪ 23, 95

[158] N. Seco, T. Veale, and J. Hayes. An Intrinsic Information Content Metric for Semantic Similarity in WordNet. In *ECAI'2004, the 16th European Conference on Artificial Intelligence*, 2004. ↪ 27

[159] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *J. Data Semantics IV*, pages 146–171, 2005. ↪ 40

[160] N. Sioutos, S. de Coronado, M. W. Haber, F. W. Hartel, W. Shaiu, and L. W. Wright. NCI Thesaurus: A Semantic Model Integrating Cancer-Related Clinical and Molecular Information. *Journal of Biomedical Informatics*, 40(1):30–43, 2007. ↪ 71

[161] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. GORDIAN: Efficient and Scalable Discovery of Composite Keys. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 691–702, 2006. ↪ 122

[162] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. ↪ 21

[163] C. Swamy. Correlation Clustering: Maximizing Agreements Via Semidefinite Programming. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 526–527, New Orleans, Louisiana, USA, 2004. ↪ 36

[164] S. Tejada, C. A. Knoblock, and S. Minton. Learning Object Identification Rules for Information Integration. *Information Systems*, 26(8):607–633, 2001. ↪ 29, 39

[165] O. Udrea, L. Getoor, and R. J. Miller. Leveraging Data and Structure in Ontology Integration. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 449–460, 2007. ↪ 40

[166] C. Umans. Hardness of Approximating Sigma$_2^{\mathrm{p}}$ Minimization Problems. In *Symp. on Foundations of Computer Science (FOCS)*, pages 465–474, 1999. ↪ 33

[167] V. S. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. *J. Web Sem.*, 4(1):14–28, 2006. ↪ 43

[168] S. van Dongen. *Graph Clustering By Flow Simulation*. PhD thesis, University of Utrecht, 2000. ↪ 36

[169] R. Vernica, M. J. Carey, and C. Li. Efficient Parallel Set-Similarity Joins Using MapReduce. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 495–506, 2010. ↪ 164

[170] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and Maintaining Links on the Web of Data. In *Proc. of the Int'l Semantic Web Conference (ISWC)*, pages 650–665, 2009. ↪ 83

[171] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient Approximate Entity Extraction with Edit Distance Constraints. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 759–770, 2009. ↪ 41

[172] R. H. Warren and F. W. Tompa. Multi-column Substring Matching for Database Schema Translation. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 331–342, 2006. ↪ 111

[173] M. Waterman, T. Smith, and W. Beyer. Some Biological Sequence Metrics. *Advances in Mathematics*, 20(3):367–387, 1976. ↪ 21

[174] M. Weis and F. Naumann. DogmatiX Tracks down Duplicates in XML. In *ACM SIGMOD Int'l Conf. on Mgmt. of Data*, pages 431–442, 2005. ↪ 41

[175] M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster. Industry-Scale Duplicate Detection. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1253–1264, 2008. ↪ 41

[176] D. T. Wijaya and S. Bressan. Ricochet: A Family of Unconstrained Algorithms for Graph Clustering. In *Proc. of the Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 153–167, Brisbane, Australia, 2009. ↪ 34

[177] P. Willett, editor. *Document Retrieval Systems*. Taylor Graham Publishing, London, UK, UK, 1988. ↪ 24

[178] W. 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 Methods, American Statistical Association*, pages 354–359, 1990. ↪ 16, 21

[179] W. E. Winkler. Improved Decision Rules In The Fellegi-Sunter Model Of Record Linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 274–279, 1993. ↪ 16, 17

[180] W. E. Winkler. Methods for Record Linkage and Bayesian Networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002. ↪ 17

[181] W. E. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006. ↪ 16

[182] W. E. Winkler and Y. Thibaudeau. An Application Of The Fellegi-Sunter Model Of Record Linkage To The 1990 U.S. Decennial Census. Technical report, Research Report Series RR91/09, US Bureau of the Census, 1991. ↪ 16

[183] D. S. Wishart, C. Knox, A. C. Guo, S. Shrivastava, M. Hassanali, P. Stothard, Z. Chang, and J. Woolsey. DrugBank: A Comprehensive Resource for in Silico Drug Discovery and Exploration. *Nucleic Acids Res*, 34:668–672, 2006. ↪ 6, 113

[184] S. Wölger, K. Siorpaes, T. Bürger, E. Simperl, S. Thaler, and C. Hofer. Interlinking Data - Approaches and Tools. Technical report, Semantic Technology Institute (STI), March 2011. ↪ 47, 158

[185] Y. Yang, N. Bansal, W. Dakka, P. G. Ipeirotis, N. Koudas, and D. Papadias. Query by Document. In *Proc. of the Int'l Conf. on Web Search and Web Data Mining(WSDM)*, pages 34–43, 2009. ↪ 42

[186] S. H. Yeganeh, O. Hassanzadeh, and R. J. Miller. Linking Semistructured Data on the Web. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, 2011. ↪ 113

[187] J. X. Yu, L. Qin, and L. Chang. Keyword Search in Databases. *Synthesis Lectures on Data Management*, 1(1):1–155, 2009. ↪ 42

[188] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On Multi-Column Foreign Key Discovery. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):805–814, 2010. ↪ 88

[189] A Java API for Google Spell Checking Service. http://code.google.com/p/google-api-spelling-java/. [Online; accessed 10-11-2011]. ↪ 126

[190] Alignment API and Alignment Server . http://alignapi.gforge.inria.fr/. [Online; accessed 31-10-2011]. ↪ 44

[191] Amazon Mechanical Turk. https://www.mturk.com/. [Online; accessed 31-10-2011]. ↪ 157

[192] Apache Hadoop Project. http://hadoop.apache.org/. [Online; accessed 31-10-2011]. ↪ 163

[193] BibBase. http://www.bibbase.org/. [Online; accessed 10-11-2011]. ↪ 149

[194] BibSonomy. http://www.bibsonomy.org/. [Online; accessed 31-10-2011]. ↪ 149

[195] CiteSeerX. http://citeseer.ist.psu.edu/. [Online; accessed 31-10-2011]. ↪ 149

[196] citeulike. http://www.citeulike.org/. [Online; accessed 31-10-2011]. ↪ 149

[197] D2R Server publishing the DBLP Bibliography Database. http://www4.wiwiss.fu-berlin.de/dblp/. [Online; accessed 31-10-2011]. ↪ 149

[198] difflib - Helpers for computing deltas. http://docs.python.org/library/difflib.html. [Online; accessed 14-10-2011]. ↪ 44

[199] EPrints. http://www.eprints.org/. [Online; accessed 31-10-2011]. ↪ 149

[200] FLAMINGO Package (Approximate String Matching) - Release 2.0 (October 14, 2008). http://flamingo.ics.uci.edu/releases/2.0/. [Online; accessed 31-10-2011]. ↪ 44

[201] Gleaning Resource Descriptions from Dialects of Languages (GRDDL) . http://www.w3.org/TR/grddl/. [Online; accessed 10-11-2011]. ↪ 120

[202] PubZone. http://www.pubzone.com/. [Online; accessed 31-10-2011]. ↪ 149

[203] ID3 Audio File Data Tagging Format. http://www.id3.org/. [Online; accessed 31-10-2011]. ↪ 6, 113

[204] JavaScript Object Notation (JSON). http://www.json.org/. [Online; accessed 14-10-2011]. ↪ 6

[205] Linked Data Movie Quiz by Guille and Jorge Lamb. http://10k.aneventapart.com/1/Uploads/310/. [Online; accessed 31-10-2011]. ↪ 143

[206] List of RDF Converter Tools, W3C Wiki. http://www.w3.org/wiki/ConverterToRdf. [Online; accessed 31-10-2011]. ↪ 119

[207] Mendeley. http://www.mendeley.com/. [Online; accessed 31-10-2011]. ↪ 149

[208] pylevenshtein - A fast implementation of Levenshtein Distance (and others) for Python . http://code.google.com/p/pylevenshtein/. [Online; accessed 14-10-2011]. ↪ 44

[209] RefWorks. http://www.refworks.com/. [Online; accessed 31-10-2011]. ↪ 149

[210] RKB Explorer. http://www.rkbexplorer.com/. [Online; accessed 31-10-2011]. ↪ 149

[211] SecondString Project Page. http://secondstring.sourceforge.net/. [Online; accessed 31-10-2011]. ↪ 44

[212] SimMetrics. http://sourceforge.net/projects/simmetrics/. [Online; accessed 31-10-2011]. ↪ 44

[213] SimPack Project Page. http://www.ifi.uzh.ch/ddis/simpack.html. [Online; accessed 31-10-2011]. ↪ 44

[214] SWRC Ontology. http://ontoware.org/swrc/. [Online; accessed 31-10-2011]. ↪ 149, 155, 156

[215] The Bibliographic Ontology. http://bibliontology.com/. [Online; accessed 14-10-2011]. ↪ 155, 156

[216] The Facebook Open Graph Protocol. http://developers.facebook.com/docs/opengraph/. [Online; accessed 10-11-2011]. ↪ 162

[217] The GeoNames Geographical Database. http://www.geonames.org/. [Online; accessed 12-July-2011]. ↪ 5

[218] The Toronto Link Discovery Project Homepage. http://dblab.cs.toronto.edu/project/linkdiscovery. [Online; accessed 14-10-2011]. ↪ 86, 112

[219] The xCurator Project Homepage. http://dblab.cs.toronto.edu/project/xcurator. [Online; accessed 31-10-2011]. ↪ 133

[220] Web Reference Database (refbase). http://www.refbase.net/. [Online; accessed 31-10-2011]. ↪ 149

[221] WordNet::Similarity. http://wn-similarity.sourceforge.net/. [Online; accessed 31-10-2011]. ↪ 44