Entity Linkage for Heterogeneous, Uncertain, and Volatile Data

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation von

M.Sc. Ekaterini Ioannou

geboren am 25. Juni 1980 in Lemesos, Zypern

Referent	**Prof. Dr. Wolfgang Nejdl** Institut für Verteilte Systeme Wissensbasierte Systeme (KBS) Universität Hannover Deutschland
Ko-Referent	**Prof. Dr. Yannis Velegrakis** Department of Information Engineering and Computer Science (DISI) University of Trento Italy
Vorsitz	**Prof. Dr. Kurt Schneider** Institut für Praktische Informatik Universität Hannover Deutschland
Tag der Promotion	**15. April 2011** Forschungszentrum L3S Hannover, Deutschland

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Wolfgang Nejdl for giving me the opportunity to join L3S and the freedom to pursue my research ideas. L3S is an exciting working environment, and for this I would like to also thank my friends and colleagues.

I would also like to express my gratitude to Prof. Yannis Velegrakis for the collaboration and discussions. His always provided me with several interesting comments and ideas for improving and extending the work.

I would like to thank my co-authors Dr. Peter Fankhauser, Dr. Claudia Niederée, George Papadakis, Odysseas Papapetrou, and Dr. Dimitrios Skoutas. Our join research work and discussions were always a pleasure. I thank Claudia for the support, guidance, and of course for the fun times we had while working on projects and papers. I am especially grateful to Odysseas for his dual role as a co-author and as a husband, which made everything easier and more pleasant.

My special appreciation goes to my parents for their constant support, and especially my brother and sister for their understanding, encouragement, and everyday companionship through the duration of my studies.

Zusammenfassung

Moderne Anwendungen und Datenkollektionen im Web werden heute oft durch die neuartige Kombination existierender Anwendungen und Datenkollektionen erstellt. Häufig benutzen die dabei verwendeten Quellen unterschiedliche Bezeichner, um dasselbe Objekt der realen Welt, wie z.B. einen Künstler, eine Konferenz oder eine Organisation, zu identifizieren. Obwohl eine Vielzahl von Ansätzen zur Duplikaterkennung (Entity Linkage) existieren, sind diese nicht zur effektiven Bearbeitung der Daten im "Web of Data" mit dessen neuen Charakteristiken geeignet. Zu diesen Charakteristiken zählen der hohe Grad der Heterogenität, der sich aus der autonomen Erzeugung und den verwendeten Datenmodellen ergibt, die Unsicherheit in den Daten, welche durch die verwendeten Erzeugungs- und Extraktionsprozesse und den unterschiedlichen Grad der Zuverlässigkeit von Quellen bedingt ist, sowie die unbeständige Natur der Daten, welche in der häufigen Interaktion von Nutzern und externen Anwendungen mit den Daten begründet ist.

Diese Dissertation stellt eine neue, alternative Methode zur Duplikaterkennung für Daten mit Heterogenität, Unsicherheiten und hoher Volatilität vor. Die Methode beruht auf dem Konzept probabilistischer Verlinkungsdatenbanken, welche sowohl die Entitäten der ursprünglichen Kollektionen als auch die möglichen Verlinkungen zwischen den Entitäten, wie sie sich durch Techniken zur Duplikaterkennung ergeben, abbilden.

Es wurden Lösungen für zwei zentrale Bereiche probabilistischer Verlinkungsdatenbanken entwickelt. Der erste Bereich dient der Bearbeitung von Anfragen zum effizienten Auffinden von Anfrageergebnissen. Der vorgestellte Ansatz zur Anfragenbearbeitung berücksichtigt nicht nur die Entitäten und die probabilistischen Verlinkungen, sondern auch die Unsicherheiten, welche in den Entitäten und in der Erkennung von Duplikaten existieren.

Der zweite zentrale Bestandteil probabilistischer Verlinkungsdatenbanken, der in dieser Arbeit betractet wird, dient der Verarbeitung der Entitäten zur Duplikaterkennung, d.h. zur Erkennung probabilistischer Verlinkungen zwischen Entitäten. Zum Umgang mit der Heterogenität und Volatilität der Daten führt dieser Teil auf inkrementelle und adaptive Techniken ein, welche nicht nur die direkt verfügbare textuelle Information, sondern auch die daraus ableitbare semantische Information berücksichtigen. Sowohl die Effektivität als auch die Effizienz der entwickelten Algorithmen wird durch experimentelle Evaluation für Daten mit den beschriebenen Charakteristiken gezeigt.

Schlagworte: Datenintegration, Duplikaterkennung, Probabilistischer Verlinkungsdatenbanken.

Abstract

A plethora of collections is nowadays created by merging data from a variety of different applications and information sources. These sources often use different identifiers for data that describe the same real world object, for example an artist, a conference, an organization. The large number of existing entity linkage approaches are not designed for the characteristics of modern applications and Web data. These includes data *heterogeneity* that is due to the lack of uniform standards, *uncertainty* resulting in imperfections in the extraction process or the reliability of the sources, and the *volatile* nature of the data due to constant modifications through interactions with users or external applications.

This dissertation introduces a novel methodology to address the entity linkage problem for heterogeneous, uncertain, and volatile data. The methodology is based on a probabilistic linkage database, which is able to simultaneously capture the entities from the original collection data and the possible linkages between entities, as these are generated by a number of the existing entity linkage techniques.

The probabilistic linkage database consists of two main components. The first is related to efficient query processing. The proposed query mechanism does not only consider the entities and the probabilistic linkages, but it also handles the uncertainty present in them.

The second component is related to the processing of the entity data for generating probabilistic linkages between entities. In order to handle the heterogeneity and the volatile nature of the data, this part focuses on incremental and adaptive techniques that consider not only the available textual information but also their inferred semantics. Both effectiveness and efficiency of the introduced algorithms are illustrated through an experimental evaluation that involves real world data.

Keywords: Data Integration, Entity Linkage, Probabilistic Linkage Database.

Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

The modern Web has grown from a publishing place of well structured data and HTML pages for companies and experienced users, into a vivid publishing and data exchange community in which everyone can participate, both as a data consumer and as a data producer. Unavoidably, the data available on the Web became highly heterogeneous, ranging from highly structured and semi-structured to highly unstructured user-generated content, reflecting different perspectives and structuring principles. The full potential of such data can only be realized by combining information from multiple sources. For instance, the knowledge that is typically embedded in monolithic applications can be outsourced, and thus used also in other applications [DKP+09]. Numerous systems nowadays are already actively utilizing existing content from various sources such as WordNet or Wikipedia. Some well known examples of such systems include DBPedia, Freebase, Spock, and DBLife.

A major challenge during combining and querying information from multiple heterogeneous sources is **entity linkage**, i.e., the ability to detect whether two pieces of information correspond to the same real world object [Len02, EIV07]. The task is also important in data cleaning applications [DJ03] and can be found in the literature under different names, such as merge-purge [HS98], entity identification [MVB08], deduplication [SB02], data matching [BMC+03, DLLH03], reference reconciliation [DHM05], or resolution [BGMM+09]. This topic has received considerable research attention with many interesting results relying on different methodologies, such as string similarity metrics [CRF03, BMC+03], entity inner-relationships [KM06, DHM05], and clustering [BG04b].

Unfortunately, existing approaches for entity linkage assume that data is relatively static. Thus, they typically perform data processing off-line in order to have the results readily available at query time. In modern Web applications, where data may at any time change not only their syntax or structure but also their semantics [Vel08], these techniques are so effective or efficient [EIV07]. This calls for entity linkage techniques that consider and deals with the special characteristics of such data.

This dissertation introduces a novel approach for addressing the entity linkage problem for heterogeneous, uncertain, and volatile data. Section 1.1 presents the motivation for this work. Section 1.2 discusses the related challenges. Section 1.3 provides an overview of the approach introduced in this dissertation, and Section 1.4 summarizes its contributions. Section 1.5 presents the structure of this dissertation.

1.1 Motivation

Consider a system created for monitoring and integrating data from multiple heterogeneous data sources on the Web. The basic data exchange unit of the system is an entity, composed of an identifier and a number of attribute name-value pairs describing the properties of the real world object the entity represents.

The first part of Figure 1.1 illustrates three entities existing in the system. The top two are referring to the story of Harry Potter and the Chamber of Secrets. The first entity has been extracted through text analysis of Wikipedia articles. Since entity extraction from text is not always accurate, the extracted entity attributes are accompanied with some probabilities reflecting the amount of confidence on the existence of these attributes. In the figure, this confidence is illustrated by the numbers next to the attribute values. The second entity has been extracted from a set of online bookstore databases. A number of these databases contain outdated or inconsistent data, thus, the attributes of the entity are also probabilistic. Finally, the third entity has been extracted by a corpus of news archives. For reasons similar to those of the first entity, its attributes have also some confidence associated to them.

Since the system needs to handle volatile data, we expect continuous appearance of new entities and these entities that need to be integrated with the data already present in the system. The second part of Figure 1.1 illustrates two additional entities, which the system needs to integrate. Similar to the entities already existing in the system, these two entities also have a set of attribute name-values

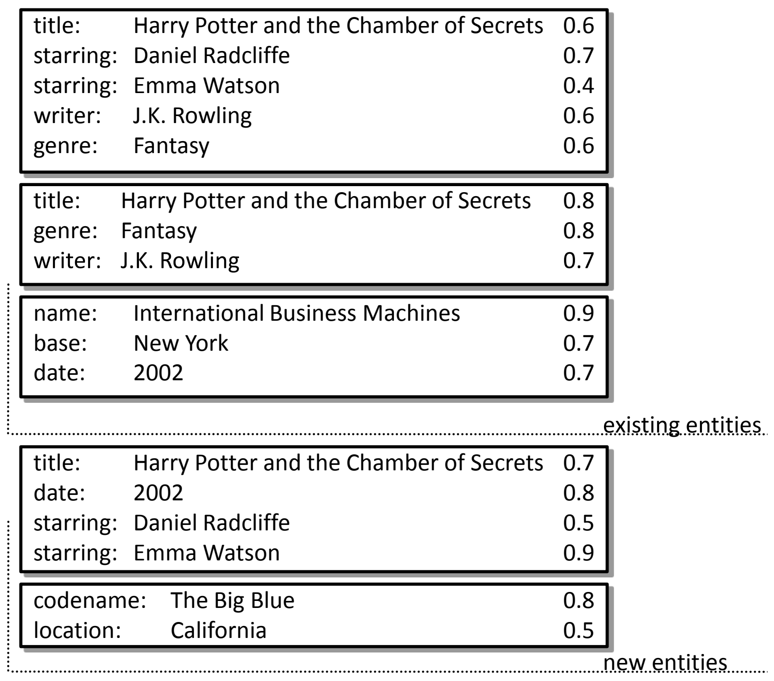

title: Harry Potter and the Chamber of Secrets 0.6
starring: Daniel Radcliffe 0.7
starring: Emma Watson 0.4
writer: J.K. Rowling 0.6
genre: Fantasy 0.6
title: Harry Potter and the Chamber of Secrets 0.8
genre: Fantasy 0.8
writer: J.K. Rowling 0.7
name: International Business Machines 0.9
base: New York 0.7
date: 2002 0.7
existing entities
title: Harry Potter and the Chamber of Secrets 0.7
date: 2002 0.8
starring: Daniel Radcliffe 0.5
starring: Emma Watson 0.9
codename: The Big Blue 0.8
location: California 0.5
new entities

Figure 1.1: A small fraction of the new entities that should be integrated with the entities already existing in the system.

pairs, each with some confidence value.

A traditional entity linkage methodology [EIV07] would simply use a predefined threshold and accept the merging of these entities when their computed similarity is above this threshold. For the entities of Figure 1.1, this might mean merging the first two entities. In this situation, a new entity would be created using the data from both entities, which might also involve the removal of some name-value pairs when these are considered as redundant, conflicting, or replicas.

There are two main issues with the traditional entity linkage methodologies. The first is that the system will return no results if asked to return an entity described using some of the attributes that were removed during the merging. The second is that the arrival of a new entity might cause the system to get into a stage that does not accurately reflect reality, since the system is now limited to two options: either decide that the new entity describes the same real world object as one of those that already exist in the system, or that the new entity is not among the already existing entities. This unfortunately ignores the possible options that would arise from reviewing any of the previous merging decisions. For instance, consider a system that has previously performed a merging between entities e_a and e_b, and that it now needs to process the new entity e_c. Merging entity e_c with e_a might provide a better solution than mering it with the previously merging of e_a with e_b. Of course, an alternative methodology for considering all possible options would be to maintain the original entities and at each addition re-execute an entity linkage technique over the original entities, ignoring the results from previous executions. Unfortunately, this approach has a prohibitively high computational cost and typically can not be applied.

1.2 Challenges

An effective and efficient solution for the entity linkage problem needs to consider the characteristics and challenges that appear in such scenarios. More specifically:

Challenge 1 - Volatile Data. Information spaces created by combining Web data or extracted data describing resources constantly change and evolve through interactions with users or external applications. Therefore, the knowledge available to the entity linkage techniques is subject to data reduction, addition, and modification. This implies the need for supporting an incremental computation and adaptation of the linkage information.

Challenge 2 - Heterogeneous Information. Effectively addressing the entity linkage problem implies the ability to handle highly heterogeneous data. The most common methodology to detect entity linkages is based on observing similarities between the attribute values from the entities. However, this assumes that entities describing the same real world objects would have the same, or at least similar, attribute values. Another methodology relies on identifying and facilitating semantic information, such as relationships between the entities. For example, co-authoring relationship in publications increases the belief that two authors describe the same object. Using a combination of these two methodologies can significantly improve entity linkage.

Challenge 3 - Diversity in Requirements. A wide variety of applications can be executed on top of integrated information spaces. Each application might have different requirements for the entity linkage solution. For example, one application might need only certain linkages (e.g., searching), whereas other applications might accept uncertain linkages based on only few evidences (e.g., recommendations that will further process the results). Therefore, entity linkages should be accompanied with a metric, indicating the belief that the corresponding entities describe the same real world object, based on the evidence that is currently present in the information space.

Challenge 4 - Data Uncertainty. Apart from the uncertainty in the linkage information, data uncertainty also appears for other reasons. One example is data uncertainty that comes directly from the extraction process, due to the very low quality that typically accompanies the unstructured data of such applications [GS06]. Another example is the uncertainty introduced when building structures for processing the data, e.g., social network analysis [AR07]. These approaches typically affect the quality of data, which is then reflected through probabilities. Unfortunately, incorporating uncertainty in a system may break a number of assumptions that many entity linkage techniques rely upon. Thus, performing entity linkage over uncertain data is a major challenge.

1.3 Summary of the Approach

The methodology we follow, takes into consideration heterogeneity, uncertainty, and the volatile nature of the data. It is based on maintaining the linkage information among the entities. As an example, let us consider the entities in Figure 1.1. It is easy to see that the first two entities may represent the same real world object, for instance the first entity may represent the actual movie whereas the second entity a DVD with the respective movie. Given that we do not have enough evidence to support a definite decision on whether these entities represent the same real world object or not, we do not perform a merging between them. We compute and store a probabilistic linkage connecting the two entities. The addition of the new entities requires only the computation of the linkages or (in some cases) the recomputation of the probability of existing linkages.

Figure 1.2 illustrates the computed linkages through the interconnecting dotted lines and alongside their probabilities. As depicted in the figure, these entities have three probabilistic linkages, two among the movie entities that are labeled e_1-e_3, and one among the company entities that are labeled as e_4 and e_5. Once we reach a final decision that two or more entities are linked, we can replace them by an equivalent entity consisting of the union of their attributes.

Consider now a user looking for the IBM consulting corporation. As is typically the case in dataspaces [HFM06], queries are expressed as a series of attribute name-value pairs. Thus the user sends the following query to the system:

$$\langle \text{name=“International Business Machines”, base=“New York”} \rangle$$

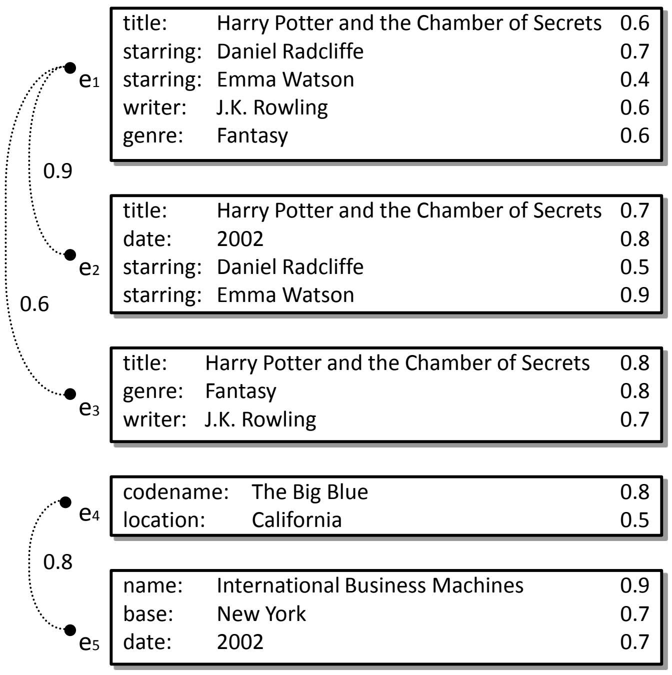

e1
title: Harry Potter and the Chamber of Secrets 0.6
starring: Daniel Radcliffe 0.7
starring: Emma Watson 0.4
writer: J.K. Rowling 0.6
genre: Fantasy 0.6
0.9
0.6
e2
title: Harry Potter and the Chamber of Secrets 0.7
date: 2002 0.8
starring: Daniel Radcliffe 0.5
starring: Emma Watson 0.9
e3
title: Harry Potter and the Chamber of Secrets 0.8
genre: Fantasy 0.8
writer: J.K. Rowling 0.7
e4
codename: The Big Blue 0.8
location: California 0.5
0.8
e5
name: International Business Machines 0.9
base: New York 0.7
date: 2002 0.7

Figure 1.2: Entities and their probabilistic linkage information.

Clearly among the five entities e_1, e_2, e_3, e_4, and e_5, only the fourth satisfies these two conditions. Of course, since the attributes of the specific entity exist with some uncertainty, specified by the respective probabilities, the existence of the entity in the query answers should also be probabilistic. A significant amount of research has been carried out in the area of the probabilistic databases [DS07b] on specifying the semantics and on the development of efficient query answering techniques for this kind of scenarios.

Assume now that a user is interested in the works of J.K. Rowling in the year 2002. She sends to the system the following query:

$$\langle \text{writer=“J.K. Rowling”, year=“2002”} \rangle$$

None of the three entities in Figure 1.2 contain both attribute name-value pairs as specified in the query, thus, any probabilistic database approach will return an empty set as an answer. However, the linkage information between entity e_1 and e_2 indicates that they may represent the same real world object. If they do, then they can be both merged into one entity, say e_{12} that contains as attributes the union of the attributes e_1 and e_2. That entity will satisfy both the conditions of the last query, and should be part of the answer set, even though it is not one of the three entities that are actually stored in the repository.

In a similar situation, assume that the user sends the query:

$$\langle \text{writer=“J.K. Rowling”, genre=“Fantasy”} \rangle$$

A complete answer to the query should take into consideration all the different cases that may exist based on the entity linkages. In particular, a complete answer should contain three entities, namely, entity e_1, entity e_3, and the entity e_{13} which is the merging of entities e_1 and e_3. Each of these entities should of course be in the answer set of the query with some degree of belief, based on the belief of the linkages and the belief of the attributes *writer* and *genre*.

The answer set for this query, could also contain entities e_{12} and e_{123}, which are created by the merging of entity e_2 with e_1, and entity e_2 with e_1 and e_3, respectively. Including in the merging the attributes of e_2 will create entities that have additional attributes, such as *date=2002*. We consider such additional attributes as redundant, since the user did not requested them through the query. Our basic principle is that we do not want to produce results that are not required, and therefore no merging should take place unless it is justified by the query given by the user, the linkages, and the attributes composing the entities.

Our approach creates the entities mergings by using available entity linkages. Since the linkages are probabilistic, for an effective query mechanism we need to take into consideration all the different combinations that may occur. Each such combination will partially contribute to the answer set. However, materialization of all combination will lead to exponential increase of the data, which is inefficient to generate and store. Instead, query processing at run time takes into consideration the related probabilistic linkages, computes the different combinations, along with their respective probability of existence, and then generates the answer set by merging the data produced from each combination.

1.4 Contributions

The main focus of this dissertation is to efficiently and effectively address the entity linkage problem as this appears in heterogeneous, uncertain, and volatile data. This is achieved by allowing data integration systems to generate and maintain probabilistic linkage information, and perform entity-aware query processing over their data and thus retrieve answers to queries that reflect the corresponding real world objects. This methodology avoids pitfalls that may result from the one-time a-priori merging decisions, as performed by traditional entity linkage techniques. Furthermore, it can support highly volatile data more efficiently. The reason is that since no merging decisions have taken place, the only updates required are on the linkages related to new data incorporated in the system, or modified data. The following paragraphs summarize the main contributions introduced in this dissertation. Each contribution is accompanied by the publications in which the specific contribution was originally introduced.

I. Modeling Entities and Linkages

In an effort to address the entity linkage problem for volatile data, we introduce a model for representing entities and linkages that aims at bringing together two worlds: the world of entity linkage and the world of probabilistic databases. The novelty of this data model is that it uses a generic entity-based representation model for highly heterogeneous data that supports the simultaneous representation of possible linkages between entities alongside the original data, as generated by a number of the existing entity linkage techniques. This means that no data merging is performed in advance, but the outcome of the entity linkage algorithms, i.e., the pairs of entities possibly representing the same real world object with the belief of that being true, are stored in the data. The outcome is a database that contains uncertainty not only on the attributes of the entities, but also on their linkages.
This work was introduced in the following publications:

[INNV10] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, Yannis Velegrakis. *On-the-Fly Entity-Aware Query Processing in the Presence of Linkage*. In Proceedings of the VLDB Endowment (PVLDB), Vol. 3, No. 1, pages 429-438, 13-17 Sep. 2010, Singapore.

[INNV11] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, Yannis Velegrakis. *LinkDB: A Probabilistic Linkage Database System.* In Proceedings of the ACM SIGMOD International Conference on Management of Data, 12-16 June 2011, Athens, Greece.

[Ioa09] Ekaterini Ioannou. *Entity-Aware Query Processing for Heterogeneous Data with Uncertainty and Correlations.* In Joint EDBT/ICDT Ph.D. Workshop, March 2009, St.-Petersburg, Russia.

[SI] Slawek Staworko, Ekaterini Ioannou. *Management of inconsistencies in data integration*. Chapter to be included in Dagstuhl Follow-up Series on Data Exchange, Integration, and Streams.

II. Entity Linkage Detection

In order to cope with the variations in the entity descriptions, we introduce techniques for detecting the possible linkages between entities. Unfortunately, the typical situation is that we do not have enough evidences to support a definite decision on whether the linked entities represent the same real world object or not. For that reason, the linkage information among the entities is probabilistic. Therefore, for each linkage we compute a probability that indicates our belief according to the evidences currently available in the information space. The clear separation of the entities and linkages enables the incremental update of linkages when new information becomes available. The benefits of this methodology are further extended as the introduced techniques do not need to reprocess data for recomputing linkages as would have been done in traditional approaches, but maintain internal structures that can be updated to reflect the new data.
This contribution was introduced in the following publications:

[INN08] Ekaterini Ioannou, Claudia Niederée, Wolfgang Nejdl. *Probabilistic Entity Linkage for Heterogeneous Information Spaces.* In Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE), pages 556-570, June 2008, Montpellier, France.

[IPSN10] Ekaterini Ioannou, Odysseas Papapetrou, Dimitrios Skoutas, Wolfgang Nejdl. *Efficient Semantic-Aware Detection of Near Duplicate Resources.* In Proceedings of the 7th Extended Semantic Web Conference (ESWC), pages 136-150, 30 May - 03 June 2010, Heraklion, Greece.

Furthermore, the following additional publications are related to this specific contribution:

[MPC+10] Enrico Minack, Raluca Paiu, Stefania Costache, Gianluca Demartini, Julien Gaugaz, Ekaterini Ioannou, Paul-Alexandru Chirita, Wolfgang Nejdl. *Leveraging Personal Metadata for Desktop Search - The Beagle++ System.* In Journal of Web Semantics, Vol. 8, No. 1, pages 37-54, 2010.

[PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Peter Fankhauser. *Efficient Entity Resolution for Large Heterogeneous Information Spaces.* In Proceedings of the 4th International Conference on Web Search and Data Mining (WSDM), February 2011, Hong Kong, China.

III. Query Answering under Probabilistic Linkages

We introduce a methodology to efficiently compute the answers for entity queries relying on the introduced model that contains a set of probabilistic linkages. Query answers reflect the entity linkage and entity representation information, with special emphasis given to the computation of the probabilities of the possible worlds based on the data and the matching uncertainty.

Generating entities by combining probabilistic linkages has several benefits. First, it produces additional valid query answering results compared to those of entity linkage and probabilistic databases, which cannot be simulated with previous techniques. An interesting feature is that reasoning about the entity linkages is done on the fly, meaning that some query results may not be explicitly represented in the database but might be a product of the reasoning which is based on the data as well as on the query conditions, i.e., by considering the union of all the attributes of the structures to be merged with corresponding probabilities [WMK+09].
This work was introduced in the following publications:

[INV] Ekaterini Ioannou, Claudia Niederée, Yannis Velegrakis. *Searching Web 2.0 Data through Entity-Based Aggregation.* To be submitted at Communications of the ACM (CACM).

[INNV10] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, Yannis Velegrakis. *On-the-Fly Entity-Aware Query Processing in the Presence of Linkage.* In Proceedings of the VLDB Endowment (PVLDB), Vol. 3, No. 1, pages 429-438, 13-17 Sep. 2010, Singapore.

[INV10] Ekaterini Ioannou, Claudia Niederée, Yannis Velegrakis. *Enabling Entity-Based Aggregators for Web 2.0 data.* In Proceedings of the 19th International World Wide Web Conference (WWW), pages 1119-1120, 26-30 April 2010, Raleigh, NC, USA.

[MBB+10] Z. Miklós, N. Bonvin, P. Bouquet, M. Catasta, D. Cordioli, P. Fankhauser, J. Gaugaz, E. Ioannou, H. Koshutanski, A. Mana, C. Niederée, T. Palpanas,

H. Stoermer. *From Web Data to Entities and Back.* In 22nd International Conference on Advanced Information Systems Engineering (CAiSE), pages 302-316, June 2010, Hammamet, Tunisia.

[ISB$^+$09] Ekaterini Ioannou, Saket Sathe, Nicolas Bonvin, Anshul Jain, Srikanth Bondalapati, Gleb Skobeltsyn, Claudia Niederée, Zoltán Miklós. *Entity Search with NECESSITY.* In Web and Databases Workshop (WebDB) co-located with ACM SIGMOD, June 2009, Providence, Rhode Island.

1.5 Structure of the Dissertation

The organization of this dissertation is as follows. Chapter 2 presents and discusses existing approaches that are related to the techniques presented in this dissertation. Chapter 3 introduces and explains the data model, which includes the representation of entities and linkages, as well as the mechanism for dealing with data uncertainty. Chapter 4 explains the mechanism for dealing with probabilistic linkage information through entity-aware query processing. Chapter 5 introduces two techniques for detecting linkages between entities and computing their probabilities. The first technique is for detecting linkages using evidences collected from the data. The second technique focuses on efficiently detecting linkages in RDF data. Chapter 6 presents an entity aggregation framework that provides a complete mechanism for addressing the entity linkage problem using the linkage detection and entity-aware query processing. Finally, Chapter 7 concludes the dissertation, and provides an overview of current and future work.

Chapter 2

Related Work

The work presented in this thesis is closely related to the task of entity linkage, which is used for identifying and merging entity referring to the same real world objects. Typically, the entity linkages techniques compute some similarity between the data that indicates the belief they have that this data are for the same object, and then merge the data that have a similarity value exceeding a predefined threshold. This whole process is performed offline, and thus at run-time query answering is simply performed over the merged data. A significant amount of research proposals focusing on efficiently and effectively addressing this challenge already exist. They can be found in the literature under different names, such as merge-purge [HS98], deduplication [SB02], entity identification [MVB08], reference reconciliation [DHM05], or entity resolution [WMK+09].

The existing approaches follow various directions to deal with entity linkage, based of the causes of the entity linkage problem. More specifically, the main causes can be summarized as follows:

Text variations. One of the most common source for the entity linkage problem is the text variations, i.e., using similar string for the same objects. Variations can appear due to introduced spelling mistakes, or due to the use of acronyms (e.g., "ICDE" for 'International Conference on Data Engineering"), or abbreviations (e.g., "J. Web Sem." for "Journal of Web Semantics").

Local knowledge. The lack of a global coordination for identifier assignment forces each source to create and use its own identifiers for the entities. In addition, each source uses text values in entities in a way most adequate for its purpose. For instance, in an entity representing a publication an author is representing by its email address, but in an entity representing a user of an application the user maybe represented by her full name.

Evolving nature of data. Another source for the different entity descriptions is the evolving nature of the data. As time passed, data describing the entities is added, removed, or modified [RVMB09]. For example, the famous ex-lady of the US was born as "Jacqueline Lee Bouvier" but this was later changed to "Jackie Kennedy" and then to "Jackie Onassis".

New functionality. Entity creation can be done by various tools. For example, extractors that automatically or semi-automatically extract entities from resources such as Web pages, content used in applications, and documents. In many cases we also have the direct reuse of existing content (e.g., Wikipedia) mainly through import of (partial) existing datasets for building new datasets (see DBPedia, Freebase, Spock, and DBLife), as data is of interest not only to the application by which it was created, but can also be reused by other applications [DKP+09].

The following paragraphs present and discuss existing approaches grouped into four categories according to the information used in the processing: (i) atomic similarity methods for comparing strings (Section 2.1), (ii) similarity methods for comparing sets of strings (Section 2.2), (iii) methods incorporating inner-relationship information (Section 2.3), and (iv) methods related to uncertain data management (Section 2.4). A complete overview of the existing work in this domain, can be found in surveys [DH05, GD05, EIV07] and related workshops/tutorials [KSS06, OS99].

2.1 Atomic Similarity Methods

The first category includes methods that compute similarity between two entity representations, where each representation is a single word or a small sequence of words, e.g., "John D. Smith" vs. "J. D. Smith", "Transactions on Knowledge and Data Engineering" vs. "IEEE Trans. Knowl. Data Eng.". The linkage and merging of entity descriptions is performed when these methods detect high resemblance between the text values found in the descriptions. Differences in strings (i.e., single words or sequence of words) are a common situation that are typically resulted from misspellings and naming variants due to the use of abbreviations, acronyms, etc.

The first group of methods that belong to the category with atomic similarity methods are based on the characters composing the string. These methods compute the similarity between two strings as a cost that indicates the total number of the operations needed to convert the first string to the second string. The basic method of edit distance, named Levenshtein distance [Lev66], counts the number of character deletions, additions, and modifications that are required for converting the first to the second string. The variations of this method extend it with additional aspects, such as operation cost depending on the character's location, and consideration of additional operations, including open gap, and extend gap [Nav01]. The Jaro [Jar89] computes similarity by considering the overlapping characters in the two strings along with their locations, and it suitable to small strings, e.g., first and last names. An extension of this method is the Jaro-Winkler [Win99] that gives higher weight to matching prefix, which increase the applicability of the approach to names. A second group of methods are the ones that compute the similarity between collection of words. Known methods from this group are the Jaccard similarity coefficient, and the TF/IDF similarity [SM86].

Fuzzy matching similarity [CGGM03] is another approach of this category. It is a generalized edit distance similarity that combines transformation operations with edit distance methods. Another method is the Soundex similarity. This method converts each word into a phonetic encoding by assigning the same code to the string parts that sound the same. The similarity between two words is then calculated as the difference between the corresponding phonetic encodings of these words. Finally, [CRF03, BMC$^+$03] describe and provide an experimental comparison of various basic similarity methods used for matching names.

Although, the existing approaches are successful in identifying the similar string, the idea of linking entities based on their string similarity is only partly correct, since the concepts to which the context of these entities refer is totally ignored. For example, consider two people with the exact same name. Using a similarity method from this category would result in incorrectly linking the entities of these people. For this reason, these methods are typically used only as part of the initial steps of more sophisticated matching approaches, in order to identify potential matches which are then further processed.

2.2 Entities as sets of data

In contrast to the previous category, the methods of this category focus on dealing with entities that are provided as sets of data. As such, these methods extend the methods of the previous category since they combine basic string methods with more complicated methodologies.

The first group of methods for this category are those that consider each relation (i.g., record) as an entity. The approaches suggested in [KMS04] and [Coh00] concatenate all data composing each relation and create strings, which they then compare using one of the string similarity methods. One of the most known methods of this category is the merge-purge [HS98], aiming in identifying whether two relational records refer to the entity. Merge-purge considers every database relation (i.e., record) as an entity, and first sorts the relation using the different available column names, uses the sorting to easy compare between similar information, and then merges the records according to the found resemblances.

The approaches proposed in [TKM02] and [DLLH03] aim at matching entities by discovering possible mappings from one entity to another entity. More specifically, in [TKM02] a mapping is identified by applying a collection of *transformations*, such as abbreviation, stemming, and initials. For the same purpose, Doan et al. [DLLH03] apply *profilers*, which are described as predefined rules with knowledge about specific concepts. Profilers are created by various sources, such as domain experts, learned from training data, or constructed from external data.

Cohen et al. [CR01] use methods for string similarity (presented in the previous category) to create techniques to adaptively modify the document similarity metrics. Li et al. [LMR05] also focus in handling multiple types of entities, addressing the problem as this appears in the context of the text documents.

As noted in [EIV07], one mechanism for enhancing the processing efficiency is data blocking. Instead of comparing each entity with all other entities, the entities are placed in blocks and are compared only with the ones inside their block. The challenge is to create blocks of entities that are most likely to refer to the same real world objects. Relevant methods typically associate each entity with a *Blocking Key Value* (BKV) summarizing the values of selected attributes and then operate exclusively on it. The Sorted Neighborhood approach [HS95], for instance, sorts entities according to their BKV and then slides a window of fixed size over them, comparing the records it contains. The StringMap method [JLM03] maps the BKV of each record to a multi-dimensional Euclidean space, and employs suitable data structures for efficiently identifying pairs of similar records. Alternatively, the q-grams based blocking presented in [GIJ$^+$01] builds overlapping clusters of records that share at least one q-gram (i.e., sub-string of length q) of their BKV. Canopy clustering [MNU00] employs a cheap string similarity metric for building high-dimensional overlapping blocks, whereas the Suffix Arrays approach, coined in [AO05] and enhanced in [dVKCC09], considers the suffixes of the BKV instead.

2.3 Facilitating inner-relationships

This category includes methods that identify linkages between two entity descriptions by discovering and exploiting inner-relationships among entities. These inner-relationships can be seen as links, or associations between the entities and parts of the entity data. As an example consider co-authorship in publications, which a widely used inner-relationships. By knowing that a publication has α, β, and γ as authors, and another publication has β', and γ as authors, we can increase our belief that β describe that same author as β'.

To capture the inner-relationships found inside an entity collection, the methods of this category model the collection into a supportive structure. For instance, the approach in [ACG02] uses dimensional hierarchies, and the approached in [BG04a] and [KMC05] uses graphs. Ananthakrishna et al. [ACG02] exploit dimensional hierarchies to detect fuzzy duplicates in dimensional tables. The hierarchies are build by following the links between the data from one table to data other tables. Entities are matched when the information along these generated hierarchies is found similar. Getoor et al. [BG04a, BG04b] model the metadata as a graph structure. The nodes in this graph is the information describing the entities, and edges are the inner-relationships between entities. The algorithm uses the edges from the graphs to cluster the nodes, and the clusters detected are then used to identify the common entities.

In [KMC05, KM06], the entity collection is also modeled as a graph following a similar methodology as the previous methods. These methods also generate other possible relationships to represent the candidate matches between entities. The additional relationships became edges that enhance the generated graph. Then, graph theoretic techniques are applied for analyzing the relationships in the graph and deciding the possible entity linkages. Other approaches follow a different methodology to create their internal supportive structures. In [PD04], the nodes represent the possible matches between two entities (and not one node representing one entity) and the edges the inner-relationships between the possible linkages, i.e., matches between entities. The relationships from the structure are then used to decide the existence of nodes (matches between entities), and information encapsulated in identified matches is propagated to the rest of the structure.

Some approaches have been proposed in the area of metadata management. The TAP system [GM03] uses a process named *Semantic Negotiation* to identify common descriptions (if any) between the different resources. These common descriptions are used to create a unified view of the data. Benjelloun et al. [BGMM+09] identify the different properties on which the efficiency of such algorithm depends on, and introduce different algorithms to address the possible combinations of the found properties.

Another well-know algorithm is the *Reference Reconciliation* [DHM05]. Here, the authors begin their computation by identifying possible associations between entities by comparing the corresponding entity descriptions. The information encoded in the found associations is propagated to the rest of the entities in order to enrich their information and improve the quality of final results. [AMNR+06] is a modified version of the reference reconciliation algorithm which is focused on detecting conflict of interests in paper reviewing processes.

2.4 Methods in Uncertain Data

Uncertain data management approaches deal with a variation of the entity linkage problem. More specifically, they consider the existence of more than one entities (modeled as relational relations) for the same real world object [RDS07, DS07c]. So, for each real world object the database contains a small set of possible entities, each coming with a probability that indicates the belief we have that this is the correct one.

Dong et al. [DHY07] investigate the use of the probabilistic mappings between the attributes of the contributing sources with a mediated schema. Applying this method on entities would have considered the possible mappings between the attribute names as given by contributing sources with a mediated schema S. This means that an attribute of entities α, β, and γ is mapped to an attribute from S with a probability to show the uncertainty of each mapping. Querying the mediated schema S is then based on these mappings. However, it does not really reflect the expected answer, since the expectation is to merge the data of the entities that describe the same objects and thus they should be merged accordantly. In fact, the probabilistic schema mappings described in this approach, can become an input to our approach by representing them as entity linkage information.

The approach in [AFM06] is more similar to the one presented in the thesis, since the focus is not on the schema information but on the actual data. The authors assume that the duplicate tuples for each entity are given. In our motivating example (Chapter 2) this means that linkages do not have probabilities (i.e., we know if they exist or not) and that all tuples describing alternative attributes have the same identifier, e.g.,:

Entity Identifier	Alternative Attribute	Probability
id_x	a_{10}	p_1
id_x	a_{20}	p_2

The tuples that represent the alternative attributes are considered as disjoined. This means that only one tuple for each identifier can be part of the final resulted entity.

Other related approaches are Dataspaces [HFM06] and Trio [ABS+06]. The main focus of these approaches is to create database systems that support uncertainty along with inconsistency and lineage. To some extend these systems also deal with duplicate tuples and uncertain data. The part of these systems that is responsible for providing correct answers over uncertain data which represent duplicated tuples faces similar issues as our approach. Our approach addresses more challenges of heterogeneous data, mainly by considering linkage/matching on the data (not only on schema information), and also correlations between entities.

Another important aspect of our approach is the efficient management of uncertainty in data; a topic that has received a lot of attention recently. Dalvi and Suciu [DS07a] used the notion of possible worlds to introduce query semantics for independent probabilistic data and presented how to efficiently evaluate queries. The approach by Sen et al. [SD07] moved towards defining and using different correlations, for example that existence of one tuple implies or disallows the existence of another tuple. The methodology identified used in probabilistic databases, for performing efficient query execution is also followed in our approach. We however also extend this methodology to provide its incorporation into a two level processing, i.e., generating possible words inside the generate possible worlds (as explained in Chapter 3).

Chapter 3

Probabilistic Linkage Database

This chapter introduces and explains the modeling of the information related to addressing the entity linkage problem. To effectively model highly heterogeneous information spaces we need a simple and flexible model. Strongly typed data models require full knowledge of the data schema, thus, they are not suitable for highly heterogeneous data. XML had been a success story in this direction, but its strict hierarchical structure caused some limitations, especially during metadata modeling [SV07]. For being able to represent relational, XML, RDF, and object oriented data without significant loss of information, we have chosen to go with a graph-based model that is typically used in dataspaces [HFM06], RDF [LC08], while it is also aligned with the notion of concepts [DKP+09].

The model we use is based on the notion of an attribute, which is a unit describing some characteristics of a real world entity. Each attribute is composed of a name and a value. More formally, assuming the existence of an infinite set of identifiers $\mathcal{O}$, an infinite set of names $\mathcal{N}$ and an infinite set of atomic values $\mathcal{V}$, an attribute is a pair $\langle n,v \rangle$, with $n \in \mathcal{N}$ and $v \in \mathcal{V} \cup \mathcal{O}$. Let $\mathcal{A} = \mathcal{N} \times \{\mathcal{V} \cup \mathcal{O}\}$ represent the infinite set of all the possible attributes.

Definition 3.1. *An* information space IS *is a finite set of attributes, i.e.,* $IS \subseteq \mathcal{A}$. *These attributes correspond to the* n *resources included in the information space, and thus* $IS = \bigcup_{i=1}^{n} attr(r_i)$ *where* $attr(r_i)$ *provides the attributes included in the integration system for resource* r_i. ∎

The value of an attribute can be atomic, for example, a string or an integer, but it can also be an identifier. The ability to use an identifier as an attribute value allows the support of relationships. This enables the modeling of complex data models, such as the RDF data generated for describing desktop resources [MPC+10].

The resources included in the information space contain descriptions of entities, which describe real world objects. For example, the data provided for a publication resource will contain entities for describing the authors of the publication and the conference. Each entity has its own identifier, assigned to the entity by the source that created it. Ideally, the same identifier should be used whenever entities describe the same real objects. However, since the identifier assignment is not globally coordinated, multiple identifiers are used for single real world entities. Therefore, even if identifiers provide an appropriate formalism for distinguishing entities, the entity linkage problem is still present (Chapter 2).

Our goal is to effectively and efficiently manage the *entity linkage problem* in the information space, i.e., discover and merge the entities describing the same real world object. To achieve this we introduce the probabilistic linkage database. Section 3.1 introduces and explains how we model entities and linkages, and Section 3.2 how we deal with the data uncertainty.

3.1 Entities and Linkages

The fundamental component of our data model is the entity, a design artifact used to represent a real world object. To model entities we adopt a flexible and probabilistic approach. It has the ability to handle highly heterogeneous data, and after its introduction in the context of dataspaces [DH07] started being used in applications. Its popularity is also based on its similarities to the human way of thinking, which unlocks the potential of developing integration applications for the modern web. A model based on similar ideas can also be found in the literature as concept model [DKP+09]. Furthermore, the plethora of existing data structures, makes data hard to describe to regular users. This, in combination with the fact that the users often have vague ideas of what they are actually looking for, preferring a more exploratory nature of interaction with the integration systems, leads unavoidably into a simple boolean query language with probabilistic answers.

An entity is a data structure consisting of a unique identifier and a set of attributes describing its characteristics. Since each entity is distinguished by its unique identifier, for the rest of the document, the terms entity and entity identifier will be considered equivalent.

Definition 3.2. *An* entity *e is a tuple $\langle id, A\rangle$ where $id{\in}O$ is the* entity identifier *and $A{\subseteq}\mathcal{A}$, finite, and referred to as the set of* entity attributes. ∎

To deal with the entity linkage problem we create and maintain a database that is a collection of entities. Each entity is partially modeling some part of a real world object through its corresponding attributes. For being able to identify the possible modelings of the same object we also maintain a collection of supporting evidences.

Definition 3.3. *A supporting evidence s is a tuple $\langle a_i, a_j, \phi\rangle$, where $a_i{\in}IS$, $a_j{\in}IS$, and ϕ a function that reports similarity between a_i and a_j.* ∎

As explained in Chapter 1, the new entities that will be given for integration could describe the same real world objects as the entities already existing in the information space. To overcome the absent of complete knowledge regarding the entity matches, we use the collected supporting evidences to compute an entity match P(e_i=e_j) for all pairs of entities in the information space. Each match expresses the probability with which the specific entities correspond to the same real world object, and we thus say that to two entities are linked. In the remaining document, we will also use notation l_{e_i,e_j} to represent P(e_i=e_j).

Example 3.1. *Consider again the first two entities from Figure 1.1 for time x, which we will name entity e_α and e_β. Both entities, i.e., e_α and e_β, could describe the same real world object. We represent the probability for this through P(e_α=e_β). As we show in Section 5.1, we compute this probability using supporting evidences. Two examples for the specific probability are: s_1=⟨"writer: J. K. Rowling", "writer: J. K. Rowling", $StringSim(.)$⟩, and s_2=⟨"genre: Fantasy", "genre: Fantasy", $StringSim(.)$⟩, where $StringSim$ is a string similarity function, e.g., Jaro [Jar89] or Jaro-Winkler [Win99]. In this situation, the attributes from the two entities are identical, however this does not happens for all entities.*

Table 3.1: Notation used throughout Chapters 3-6.

Notation	*Description*
a_i=$\langle n$,$v\rangle$	Attribute, i.e., a name-value pair
r_i=$\{a_j\}$	Resource (e.g., a publication), set of attributes
$IS=\bigcup_{i=1}^{n} attr(r_i)$	Information space
e_i=$\langle id, A\rangle$	Entity, i.e., an identifier and a set of attributes
$\langle a_i, a_j, \phi\rangle$	Supporting evidence
l_{e_i,e_j}=P(e_i=e_j)	Linkage, i.e., possible match between two entities
D=$\langle\mathcal{E}$,$\mathcal{L}$, p^a,$p^l\rangle$	Probabilistic linkage database
plw(D)	Possible l-worlds for D
$\mathcal{L}^{sp}$	Linkage specification
f_i	Factor, i.e., pairwise linked entities

Definition 3.4. *A* linkage database *is a tuple* $\langle\mathcal{E},\mathcal{L}\rangle$*, where* $\mathcal{E}$ *is a finite set of entities and* $\mathcal{L}$ *is a linkage assignment on* $\mathcal{E}$*. A* linkage assignment *over a set E is a binary relation* $L\subseteq E\times E$ *that is commutative, transitive, symmetric, and reflexive. Two entities* e_1,$e_2$$\in\mathcal{E}$ *of a database* $\langle\mathcal{E}, \mathcal{L}\rangle$ *are said to be* linked*, denoted as* l_{e_1,e_2}*, if* $(e_1, e_2)\in \mathcal{L}$*. A maximal group of pairwise linked entities forms a* factor. ∎

A linkage assignment can be equivalently expressed either through explicit enumeration of the binary relationships or through a set of groups of entities, with each group representing a factor. For instance, given entities e_1, e_2, …, e_5, the set $\{\{e_1, e_2, e_3\}, \{e_4, e_5\}\}$ describes a linkage assignment with two factors. The first factor consists of entities e_1, e_2, and e_3, and the second of the e_4, and e_5. The alternative representation is through the set of linkages that consists of the set of all pairwise relationships in each factor.

Since two or more linked entities model parts of the same real world object, they can be replaced by a third representing the information of both. In a linkage database this action needs to be followed by the respective update of every reference to these entities found in linkages or attributes, to the newly created entity.

Definition 3.5. *A* merge *of a set of entities* e_1, e_2, …, e_n, *, denoted as merge(*e_1, e_2, …, e_n*), is a new entity* e_{new}=$\langle id, A\rangle$ *where id is a new identifier and* A=$\cup_{i=1}^{n} A_i$*, with* A_i *representing the attributes of the entity* e_i*.*
The result of a merge *m of entities* e_1,e_2,…,e_n *into* e_{new} *in a linkage database D, is a new linkage database D′ constructed by:*

1. *eliminating from D all entities* e_1,e_2,…,e_n*, introducing entity* e_{new}*,*

2. *for $k, m{=}1..n$ eliminating all linkages of the form (e_k, e_m),*

3. *replacing any linkage of the form (e_k, e) (respectively (e, e_k)) with (e_{new}, e) (respectively (e, e_{new})), and*

4. *replacing every entity attribute of the form $\langle na, e_k\rangle$ with $\langle na, e_{new}\rangle$. This relationship between D and D' is denoted as $D\overset{m}{\rightarrow}D'$.*

The core *of a linkage database D is a database D_c such that there is a sequence m_1, m_2, ..., m_m of merge operators such that $D\overset{m_1}{\rightarrow}D_1\overset{m_2}{\rightarrow}\ldots\overset{m_m}{\rightarrow}D_c$ and no other merge is possible on D_c.* ■

By definition, the set of linkages in a core is always empty. It can be shown that the core of a linkage database is always unique but different linkage databases may have the same core. We now show that the core of a linkage database D_c is always unique as derived from D through a sequence of merge operators, i.e., $D\overset{m_1}{\rightarrow}D_1\overset{m_2}{\rightarrow}\ldots\overset{m_k}{\rightarrow}D_c$. The proof is by induction on the results for the size of merge operators with the smallest number.

Hypothesis: Let E denotes the entities in D, i.e., $\{e_1,e_2,\ldots,e_n\}$. We have to perform two merge operators, merge m_1 on the entities in S_a that results in entity e_a, and merge m_2 on the entities in S_b that results in entity e_b.

Induction base: Entity sets S_a and S_b are subsets of E, and by definition the entities in the merge operators are mutually exclusive, so $S_a \cap S_b = \{\}$. The two possible sequences are as follows:
(1) $D \overset{m_1}{\rightarrow} D\text{-}S_a\cup\{e_a\} \overset{m_2}{\rightarrow} D\text{-}S_a\cup\{e_a\}\text{-}S_b\cup\{e_b\} = D\text{-}S_a\text{-}S_b\cup\{e_a,e_b\}$
(1) $D \overset{m_2}{\rightarrow} D\text{-}S_b\cup\{e_b\} \overset{m_1}{\rightarrow} D\text{-}S_b\cup\{e_b\}\text{-}S_a\cup\{e_a\} = D\text{-}S_a\text{-}S_b\cup\{e_a,e_b\}$
Both sequences result in the same linkage database.

Induction step: We now show that if the hypothesis holds, then it also holds for an arbitrary number of merge operations. So, applying a set merge operators M on a linkage database D in any sequence produces the same core linkage database D_c.

Let $\{m_1, m_2, \ldots, m_k\}$ denote an initial merge sequence. We can create a new sequence M' by swapping two nearby merge operators insider M, i.e., $\{\ldots, m_{i+1}, m_i, \ldots\}$. Given the induction hypothesis, we know that the resulted D_c will be the same. With iterative swapping of merge operators in the created sequence, we can generate various sequences (and eventually all possible sequences), with each one of this sequences resulting in the same D_c. □

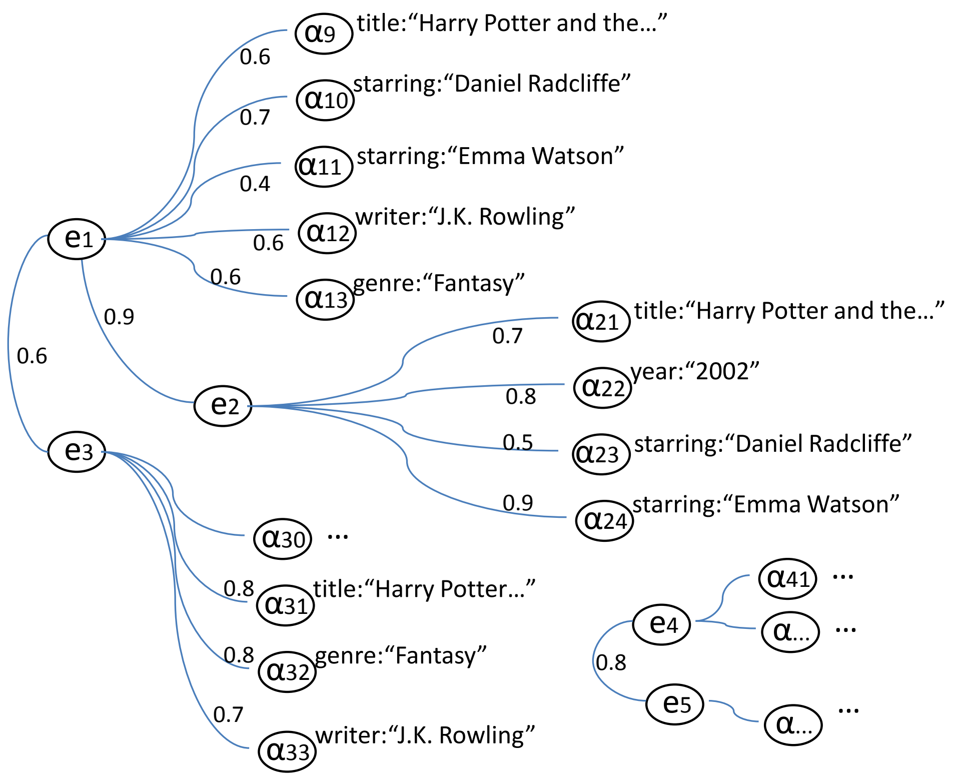
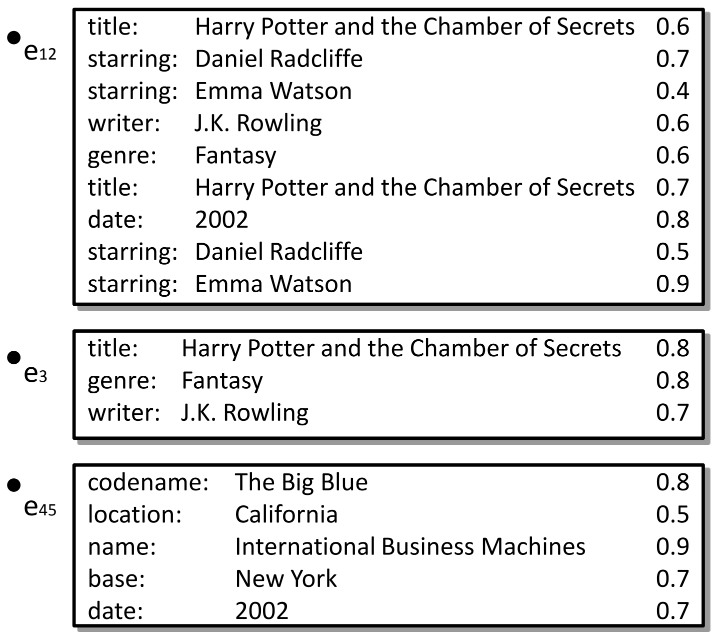

α9 title:"Harry Potter and the..."
0.6
α10 starring:"Daniel Radcliffe"
0.7
α11 starring:"Emma Watson"
0.4
α12 writer:"J.K. Rowling"
0.6
α13 genre:"Fantasy"
0.6
e1
0.9
0.6
e2
e3
α21 title:"Harry Potter and the..."
0.7
α22 year:"2002"
0.8
α23 starring:"Daniel Radcliffe"
0.5
α24 starring:"Emma Watson"
0.9
α30 ...
α31 title:"Harry Potter..."
0.8
α32 genre:"Fantasy"
0.8
α33 writer:"J.K. Rowling"
0.7
e4
e5
0.8
α41 ...
α... ...
α... ...

Figure 3.1: A fraction of a probabilistic linkage database, including some of the entities and linkage information that were illustrated in Figure 1.2.

3.2 Dealing with Uncertainty

To capture the uncertainty that may exist on the data, we adopt and extend the idea of the probabilistic databases. First, we associate to each entity attribute a value between 0 and 1. In the absence of linkages, this results into a traditional probabilistic database [DS07b]. A probabilistic database D represents a set of possible worlds, each being a database in which only a fraction of the attributes in D are present in each entity. An attribute probability indicates the likelihood that an attribute[1] is present in a randomly selected possible world. In the presence of linkages among the entities, we have a probabilistic database with linkage relationships. A probabilistic database with linkage relationships is also representing a set of possible databases, i.e., a set of possible worlds. This set is the set of possible worlds of its core.

[1]By abuse of terminology, the term attribute refers to the pair attribute name-attribute value.



●e12

title:	Harry Potter and the Chamber of Secrets	0.6
starring:	Daniel Radcliffe	0.7
starring:	Emma Watson	0.4
writer:	J.K. Rowling	0.6
genre:	Fantasy	0.6
title:	Harry Potter and the Chamber of Secrets	0.7
date:	2002	0.8
starring:	Daniel Radcliffe	0.5
starring:	Emma Watson	0.9

●e3

title:	Harry Potter and the Chamber of Secrets	0.8
genre:	Fantasy	0.8
writer:	J.K. Rowling	0.7

●e45

codename:	The Big Blue	0.8
location:	California	0.5
name:	International Business Machines	0.9
base:	New York	0.7
date:	2002	0.7

Figure 3.2: The core of the l-world of the database in Figure 1.2 generated by the linkage specification $\{l_{e_1,e_2}, l_{e_4,e_5}\}$.

We push the idea of the probabilistic databases beyond the traditional definition, by introducing uncertainty also on the linkages among the entities. This uncertainty exists naturally from the entity identification or deduplication techniques. The result is a new form of database, referred to as a probabilistic linkage database, which is a linkage database with probabilities associated on its linkage relationships and entity attributes.

Definition 3.6. *A* ***probabilistic linkage database*** *is a tuple* $\langle\mathcal{E},\mathcal{L}, p^a,p^l\rangle$*, where* $\mathcal{E}$ *is a set of entities,* $\mathcal{L}$ *is a linkage assignment on* $\mathcal{E}$*, and* p^a*,* p^l *are attribute and linkage probability assignment functions respectively. In particular,* $p^l|\mathcal{L}\mapsto[0,1]$ *and* $p^a|B\mapsto[0,1]$ *with* $B=\{a \mid \exists\langle id,A\rangle\in\mathcal{E} \wedge a\in A\}$. ∎

Example 3.2. *Consider again the entities from Figure 1.2. The probabilities shown in the figure correspond to the attribute* p^a *and linkage* p^l *probability assignment functions. An alternative illustration of the corresponding probabilistic linkage database for a fraction the entities and linkage information is shown in Figure 3.1.*

The Attributes table contains the list of the attributes of the entities in the database. Table Entities records the assignment of attributes to entities along with their probability value. Tables Entities and Attributes contain all the information needed to construct the probabilistic entities of the database. The linkage information among these entities is recorded in the Linkage table.

Due to the probabilities on the linkages, a probabilistic linkage database models a number of different probabilistic databases with linkage relationships. Each such database is generated from the probabilistic linkage database by selecting a fraction of its linkages. We refer to these probabilistic databases with linkage assignments as possible linkage worlds, or possible l-worlds for short. The set of all possible l-worlds of a probabilistic linkage database D is denoted by $plw(D)$.

A possible l-world of a probabilistic linkage database is specified by a linkage specification which determines what linkage relationships should be kept and what should be dropped. Not all the specifications are semantically meaningful. For instance, a specification that accepts a linkage between entities e_1 and e_2, and between e_2 and e_3 but not one between e_1 and e_3 is not semantically meaningful since the latter contradicts the first two from which it can be inferred that e_1 and e_3 are linked due to the transitivity property.

Definition 3.7. *Given a probabilistic linkage database $D=\langle\mathcal{E},\mathcal{L}, p^a,p^l\rangle$, a* linkage specification *is a linkage assignment $\mathcal{L}^{sp}\subseteq\mathcal{L}$ such that $\forall x\in\mathcal{L}^{sp}$: $p^l(x)\neq 0$. The* boolean expression *of a linkage specification is the expression $\bigwedge_{1,n} c_k$, where*

$$c_k = \begin{cases} x \neq y & \textit{if}\, (x,y)\in\mathcal{L} \wedge (x,y)\notin\mathcal{L}^{sp} \\ x = y & \textit{if}\, (x,y)\in\mathcal{L}^{sp} \end{cases}$$

A linkage specification is invalid *if its boolean expression is always false.* ∎

As an example, consider a database with a linkage assignment $\mathcal{L}=\{l_{e_1,e_2}, l_{e_2,e_3}, l_{e_1,e_3}\}$, and a linkage specification $\mathcal{L}^{sp}=\{l_{e_1,e_2}, l_{e_2,e_3}\}$. Independently of what the probabilities of the linkages are, the boolean expression of $\mathcal{L}^{sp}$ is $(e_1=e_2) \wedge (e_2=e_3) \wedge (e_1\neq e_3)$ which is always false, thus the linkage specification is invalid. In our work we consider l-worlds constructed by valid linkage specifications only. By definition, given a probabilistic linkage database, and a possible l-world of it, there is only one linkage specification defining this possible l-world.

We will use the symbol f_i to refer to the i-th factor, and $\mathcal{L}^{sp}_{f_i}$ to denote all the

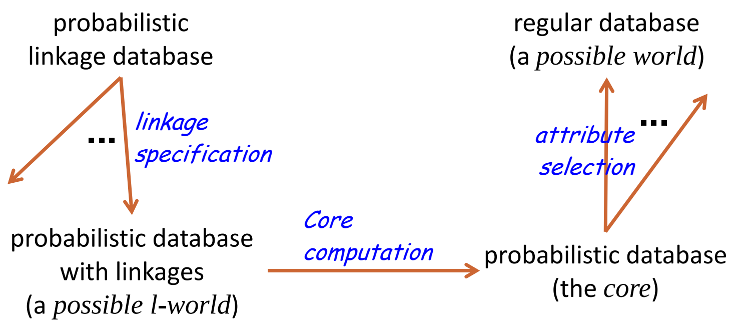

probabilistic linkage database
linkage specification
...
probabilistic database with linkages (a possible l-world)
Core computation
regular database (a possible world)
attribute selection
...
probabilistic database (the core)

Figure 3.3: The different kinds of probabilistic databases.

possible linkage specifications between its entities. The k-th assignment in $\mathcal{L}^{sp}_{f_i}$ is denoted by $\mathcal{L}^{sp}_{f_i}(k)$. Since the factors are independent of each other, the probability of a possible l-world W can be computed by the product of the probabilities of the factor assignments:

$$Pr(W) = \prod_{i=1}^{n} Pr(\mathcal{L}^{sp}_{f_i}(.)) \tag{3.1}$$

Example 3.3. *Consider the probabilistic linkage database of Figure 1.2 and the linkage specification $\{l_{e_1,e_2}, l_{e_4,e_5}\}$. The specification generates a possible l-world that is exactly as the database illustrated in Figure 1.2, but without the probabilities on the dotted lines and without the dotted lines between entities. Its core will be the one illustrated in Figure 3.2. Entity e_{12} is the result of the merge of e_1 and e_2, while entity e_{45} is the result of the merge of entities e_4 and e_5. Note that our model allows duplication on the attributes, thus, the fact that the same attribute name/value pair appears twice in an entity is not a problem. Elimination of this kind of duplication and consideration of dependencies among the attributes can be handled at a later stage.*

Figure 3.3 provides a graphical explanation of the relationships among the different types of databases defined here. As shown, by starting from the probabilistic linkage database and applying the different possible linkage specifications, we generate a number of probabilistic databases with linkages, which are also named possible l-worlds. Through the core computation, each of this possible l-world is mapped to a probabilistic database. By selecting the attributes in a probabilistic database, we generate a number of possible worlds, which are basically the regular databases and correspond to the entities in the original probabilistic linkage database.

Our data model and approach can be applied on the results of various entity linkage techniques and not just the ones introduces in this dissertation (Chapter 5). Therefore, it is important to note here that, in general, traditional entity linkage techniques measure beliefs. Some recent works explain how to turn these beliefs into probabilities [AFM06, DHY07, EIV07]. We consider this task outside the focus of the current work. We assume that this information has been computed by some data analysis tools [BNV07] or some other form of linkage discovery algorithms [DHY07], and has been provided to our framework as input. Another important note is regarding the meaning of a probabilistic linkage between two entities. A linkage represents the belief that the two entities are linked, independently of any other third entity. It is not a global belief. This means that through different linkage paths, different linkage beliefs may be computed. For instance, consider the simple example of three entities e_1, e_2 and e_3, with the following linkages between them: l_{e_1,e_2}=0.3, l_{e_2,e_3}=0.5 and l_{e_1,e_3}=0.8. Through transitivity, from the linkages l_{e_1,e_2} and l_{e_2,e_3}, it can be inferred a belief that entity e_1 and e_3 are linked with probability 0.3×0.5=0.15 which is different from the 0.8 direct linkage l_{e_1,e_3}. The way all these different probabilities are combined together to form the global belief of linkage between e_1 and e_3 is up to the query mechanism. It can be, for instance, the maximum value, their sum, or something else. In our system, this situation is taken care through the probability computation of the factor, that will be presented later.

For a query language we have adopted a flexible formalism that covers the needs of the emerging case of concept databases [DKP+09]. In particular, a query is a conjunction of attribute name-value pairs in which the user describes the characteristics, i.e., attributes, that the retrieved entities are expected to satisfy. An answer to a query is a set of entities. Consider a probabilistic linkage database D, and a query Q:$a_1 \wedge \ldots \wedge a_n$, with each a_i being an expression of the form $name_i$=$value_i$. An entity e is in the answer set of Q if there is a possible world W of a probabilistic database D_p, such that D_p is the core of a possible l-world $W^{\mathcal{L}}$ of the database D, entity e is in W and it contains an attribute $name_i$ with value $value_i$ for every i=1..n. In other words, the answer to a query is the union of the answers over all the possible worlds of all the possible l-worlds. Each entity in the answer set of a query is accompanied with a probability, which represents the belief we have that this entity will be selected among all the possible worlds of all the possible l-worlds of the probabilistic linkage database. This probability is computed based on the attribute and linkage probabilities.

3.3 Summary

In this chapter we introduced a generic representation model for highly heterogeneous data that supports the simultaneous representation of the same real world object under different formats. The model is based on the concept of entity which records its information through a number of attribute name-value pairs. The model probabilistic that can record uncertainty not only on the data values but also on the linkages among the entities.

Chapter 4

Dealing with Probabilistic Linkages

In this chapter we introduce entity-aware query processing that is responsible for the evaluation of a query Q over a probabilistic linkage database $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$. The semantics of query answering as presented and explained in Chapter 3, suggest an evaluation strategy that consists of the computation of all possible l-worlds, and for each such l-world the computation of its possible worlds, and then an evaluation of the query Q over each such world. For each entity in the answer set, the number of its appearances in the possible worlds can be computed to determine its probability. It is clear that such an evaluation is prohibitively expensive both in terms of space and time.

Instead of the brute-force evaluation, we propose here an alternative evaluation strategy that is based on a novel technique that avoids high computational cost and requires no materialization of worlds. The general high level idea, may seem to be similar to the evaluation of queries in probabilistic databases [SD07], but the existence of linkages makes the problem fundamentally different. Existing models for *correlated tuples in probabilistic databases* [SD07], for instance, cannot handle this situation. Despite the correlation they support among tuples, they still consider the tuples independent structures and manage them as such. In our case the linked entities are merged into one.

A distinguishing feature of our approach is that it restricts the computation to only those possible l-worlds and their corresponding linkage specifications that are meaningful for the query at hand. As a consequence, it is not simply the entities in the answer set that depend on the query, but also their structure, i.e., the attributes

Algorithm 4.1: Entity-Aware Query Evaluation

```
Input:  Query $Q$
Output: Set R of Entities satisfying query conditions
1  $LA \leftarrow$ findRequiredLinkageAssignments($Q$);
2  $PLW \leftarrow \emptyset$;
3  $R \leftarrow \emptyset$;
4  foreach $la \in LA$ do
5      $W \leftarrow$ findPossibleLWorlds($la$);
6      foreach $w \in W$ do
7          $w.prob \leftarrow$ calculateLWorldProbability($la$);
8          $PLW \leftarrow PLW \cup \{w\}$;
9      end
10     foreach $plw \in PLW$ do
11         $E \leftarrow$ evaluateQuery($plw$, $Q$);
12         foreach $e \in E$ do
13             $e.prob \leftarrow$ combineProb($e.prob$, $plw.prob$);
14             $R \leftarrow R \cup \{e\}$;
15         end
16     end
17 end
```

they contain: adding an extra condition to a query may trigger the consideration of additional linkage specifications, which —in turn— may result in additional attributes for the entities in the answer set. This feature has two major benefits. First, it makes the whole process computationally cheaper, since only the required merges take place. Second, it avoids overwhelming the user with answers containing long lists of attributes originating from different possibly linked entities that are outside the users' interests.

Algorithm 4.1 sketches an overview of query evaluation. The algorithm does not generate the possible l-worlds, but identifies ans uses only those related to the query that should be processed. The steps of the algorithm are listed here, and explained in more details in the remaining of this chapter:

1. We build an index on all the factors (Section 4.1).

2. Since there is a one-to-one correspondence between possible l-worlds and linkage specifications, and between linkage specifications and entity merges, we start by finding the entity merges required in order to generate an answer to the query at hand (Section 4.2)

3. From the merges we find the linkage assignments that need to be considered, and from these assignments the possible l-worlds. Then the probability of each possible l-world is computed (Section 4.3).

4. Finally, the possible worlds of each l-world are generated alongside their own probability, which is combined with the probability of the respective l-world and then included in the answer set of the query (Section 4.4).

4.1 Representing & Indexing Factors

A commonly used approach [AKO09, DS07b, RS08, SD07] in answering queries over probabilistic data is to partition the data into a series of disjoint/independent groups. These groups can be found in the literature under names such as *factors* [SD07] or *components* [AKO09]. The possible combinations of these groups generate all the possible worlds.

This idea is not directly applicable to our case. The transitive property of linkage may generate additional correlation, i.e., dependencies, that are equally important for the correct identification of the possible worlds.

We do, however, follow a similar idea to the one of managing uncertain data with correlations [SD07], and as a first step, we divide the set of entities into sets of connected components, i.e., factors (Definition 3.4). To compute all possible l-worlds of a database, we need to consider all the possible valid linkage specifications. This number can easily get large to make the computation intractable. Based on the fact that no linkage exists between entities in different factors, we can improve the situation by considering each factor independently.

Each possible l-world is based on some linkage specification within each factor. Thus, the set of possible l-worlds can be derived by combining the alternative linkage specifications withing each factor.

$$plw(\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle) \quad = \quad \mathcal{L}^{sp}_{\mathfrak{f}_1} \times \mathcal{L}^{sp}_{\mathfrak{f}_2} \times \ldots \times \mathcal{L}^{sp}_{\mathfrak{f}_n}$$

The probability of a generated l-world is computed using Formula 3.1, and is based on the assignments of its linkage specifications. The probability of assignment $\mathcal{L}^{sp}_{\mathfrak{f}_i}(k)$ is based on the probabilities of the linkages it uses. We compute it by considering only the linkages that can not be derived, and thus use set $M \subseteq \mathcal{L}^{sp}_{\mathfrak{f}_i}$ that does not include more than once the same information. This probability is computed as $\prod_{l_i \in M}$ ($p^l(l_i)$) · $\prod_{l_i \in (\mathcal{L}^{sp} - M)}$ (1 - $p^l(l_i)$). In this work, we selected to remove linkages that had a lower probability than their corresponding derived one. However, other options can also be incorporated.

Example 4.1. *In the database of Figure 1.2, the set of entity linkages is $\mathcal{L}$ = {l_{e_1,e_2}, l_{e_1,e_3}, l_{e_4,e_5}} in which two independent factors can be identified: $\mathfrak{f}_1$ = {e_1,e_2,e_3} and $\mathfrak{f}_2$ = {e_4,e_5}. The first contains entities e_1, e_2, and e_3 with linkages $\mathcal{L}^{sp}$={l_{e_1,e_2}, l_{e_1,e_3}}, and the second contains e_4 and e_5 with linkages $\mathcal{L}^{sp}$={l_{e_4,e_5}}. The sets of possible linkage specifications with the respective probabilities of the l-world they specify are:*

Factor $\mathfrak{f}_1$ = {e_1,e_2,e_3}			Factor $\mathfrak{f}_2$ = {e_4,e_5}	
$\mathcal{L}^{sp}_{\mathfrak{f}_1}(1)$={$l_{e_1,e_2}$,$l_{e_1,e_3}$}	0.9×0.6=0.54		$\mathcal{L}^{sp}_{\mathfrak{f}_2}(1)$={$l_{e_4,e_5}$}	0.8
$\mathcal{L}^{sp}_{\mathfrak{f}_1}(2)$={$l_{e_1,e_2}$}	0.9×(1-0.6)=0.36	×	$\mathcal{L}^{sp}_{\mathfrak{f}_2}(2)$={}	(1-0.8)=0.2
$\mathcal{L}^{sp}_{\mathfrak{f}_1}(3)$={$l_{e_1,e_3}$}	0.6×(1-0.9)=0.06			
$\mathcal{L}^{sp}_{\mathfrak{f}_1}(4)$={}	(1-0.9)×(1-0.6)=0.04			

Considering all the possible combination of the above individual linkage specifications of factors, the linkage specifications of the whole database can be constructed. Each such specification, specifies a possible l-world. The following table provides these l-worlds (through the linkages that each one considers) alongside the respective entity merges that need to take place in the computation of the core of the l-world. The meaning of the notation e≡e′ is that in the core computation of the respective l-world the merge of entities e and e′ needs to take place.

Possible l-world	*Required Merges*	*Probability*
I_1= {l_{e_1,e_2}, l_{e_1,e_3},l_{e_4,e_5}}	e_1≡e_2≡e_3, e_4≡e_5	0.54 × 0.8 = 0.432
I_2= {l_{e_1,e_2}, l_{e_1,e_3}}	e_1≡e_2≡e_3, e_4, e_5	0.54 × 0.2 = 0.108
I_3= {l_{e_1,e_2}, l_{e_4,e_5}}	e_1≡e_2, e_3, e_4≡e_5	0.36 × 0.8 = 0.288
I_4= {l_{e_1,e_2} }	e_1≡e_2, e_3, e_4, e_5	0.36 × 0.2 = 0.072
I_5= {l_{e_1,e_3}, l_{e_4,e_5}}	e_1≡e_3, e_2, e_4≡e_5	0.06 × 0.8 = 0.048
I_6= {l_{e_1,e_3}}	e_2, e_1≡e_3, e_4, e_5	0.06 × 0.2 = 0.012
I_7= {l_{e_4,e_5}}	e_1, e_2, e_3, e_4≡e_5	0.04 × 0.8 = 0.032
I_8= {}	e_1, e_2, e_3, e_4, e_5	0.04 × 0.2 = 0.008

The sum of the probabilities of the possible l-worlds in the above table is 1. In certain cases, the sum could have been less. This is because of the fact that certain *linkage specifications are not valid and are not considered. This would have been the case in the specific example, for instance, if there was also a linkage between entities e_2 and e_3.*

To avoid recomputing the factors every time, we create an index structure that is dynamically maintained. The index is based on the idea of equivalence classes. Actually, each factor is an equivalence class. When the data is modified and new linkages are introduced or old are eliminated, changes occur on the equivalence class memberships, and thus on the factors.

4.2 Deciding the Entity Merges

Clearly, not all the possible l-worlds need to be created every time a new query needs to be answered. If the core of a possible l-world contains no entity that satisfies all the attributes requested in the query, then it is certain that every possible world of the core will return no answer to the query. To avoid these l-worlds, we exploit the list of factors that have been precomputed and indexed. From all the possible factors, only those that for every attribute mentioned in the query, contain at least one entity satisfying that attribute, are considered.

The above step already provides a considerable reduction to the number of linkages and entities that need to be processed. However, merging all the entities in each of the selected factors may result into entities with a large number of attributes. We exploit the linkage specifications to push the optimization even further by considering only the linkage specifications of each factor that are between entities satisfying at least one attribute from those in the query. Furthermore, it is required that the union of the attributes of the factor entities involved in the linkages of the linkage specification to be a superset of the set of attributes in the query. In practice the above selections and the merges are actually computed in one step.

Algorithm 4.2 provides the steps we follow. For each query attribute we create a set E_i with all the entities satisfying the specific attribute. Then we create the cartesian product of these sets, with the extra requirement that the entities should belong to the same factor. Since an entity may belong to more than one E_i set, we involve a duplicate elimination step at the end.

Example 4.2. *Consider the query Q: starring="Emma Watson" $\wedge$ date=2002 on our usual probabilistic linkage database example of Figure 1.2. Only entities e_1 and*

Algorithm 4.2: Generate Entity Merges

```
Input:  Query $Q := \langle a_1, a_2, \ldots, a_k \rangle$, Database $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$
Output: Entity Merges M
1 foreach $a_i$ in $Q$ do
2     $E_i \leftarrow \{e \mid e{=}\langle id, A \rangle \wedge e{\in}\mathcal{E} \wedge a_i{\in}A\}$;
3 end
4 $N \leftarrow \{(e_1, \ldots e_n) \mid \forall i{=}1..n\colon e_i{\in}E_i \wedge \forall i{=}2..n\colon factor(e_{i-1}){=}factor(e_i)\}$;
5 $M \leftarrow \{eliminateDuplicates(m) \mid m \in N\}$ ;
```

e_2, both belonging to factor f_1, satisfy the first attribute starring="Emma Watson", thus, the list E_1={f_1−e_1, f_1−e_2} is constructed. Similarly, the list E_2={f_1−e_2, f_2−e_5} is also constructed for the attribute date=2002 of the query. The cartesian product of these two lists, with the additional condition of agreement on the factors gives the pairs: ⟨f_1−e_1,f_1−e_2⟩ and ⟨f_1−e_2,f_1−e_2⟩ which becomes ⟨f_1−e_2⟩. This suggest one merging of e_1 and e_2 and one that considers e_2 with no merging. Both make sense since we cannot distinguish between the attribute starring="Emma Watson" of e_1 and that of e_2. However, notice that although e_3 also belongs to factor f_1, it is not considered, since it contains no attribute that has been asked by the query.

4.3 Computing l-world probabilities

Having decided the merges that need to be performed for satisfying the given query, the next step is to compute the probabilities of the respective l-worlds. Equation 3.1 under the conditions from a merge that needs to take place, becomes:

$$Pr(I|c_m) = \prod_{i=1}^{m} Pr(\mathcal{L}^{sp}_{f_i} \mid c_m) \tag{4.1}$$

where c_m are the conditions describing merge m. Recall, however, that a merge may be true in many possible l-worlds. There are two alternatives that one can follow. The first is to compute the probability of the merge as the sum of the probabilities of all l-worlds that satisfy this merge. The second is to compute and consider only the maximum of these probabilities. The latter requires significantly less computation time, since it only needs to identify the l-world with the highest probability. For systems that simply use that probability as a ranking mechanism for the entities before displaying them to the user, this second option is typically sufficient.

The algorithm for computing the maximum probability is based on the algorithm for finding shortest paths in graphs. In particular, provided the entity linkages $\mathcal{L}^{sp}_{f_i}$, we generate a weighted undirected graph G as follows: every entity participating in a linkage of l_{e_i,e_j} becomes a node of the graph. Each linkage l_{e_i,e_j} becomes an edge that connects the nodes representing entities e_i and e_j. The weight of such an edge is given by the probability of the respective linkage.

An entity merging $merge(e_1,e_2,\ldots,e_n)$ corresponds to a spanning tree that connects all entities $e_1,e_2,\ldots,e_n$. Computing the merging that maximizes the probability is similar to computing the maximum connected component of the graph that has the highest total probability (i.e., multiplication of the probabilities of its edges). Since the nodes of the graph correspond to the entities of a factor, they are all connected, thus, the maximum connected component will include all the nodes of the graph. To compute it, we rank the edges in decreasing order of their linkage probability. Initially, all the entities (i.e., nodes) are marked as not-visited. The highest ranked edge is first selected and the two nodes it connects are marked as visited. Then as a list of edges is considered the subset of the edges that have one endpoint marked visited and one non-visited. The one with the highest probability is selected and its non-visited endpoint is marked as visited. The same step is repeated until all the nodes in the graph have been marked as visited. The probability of the merge is the multiplication of the probabilities of the edges that have been used in this process, and by construction this probability is the maximum.

4.4 Possible worlds and their probabilities

Each possible world from an l-world essentially represents a different combination over the attributes of the entities participating in a specific l-world. For instance, consider the data in Figure 3.2, and in particular the attributes involved in $merge(e_1,e_2)$. The entity $merge(e_1,e_2)$ needs to include all attributes from entities e_1 and e_2, as shown in Figure 4.1. Two issues need to be taken into consideration. One is the probabilities of the attributes, specifically in the case of duplication, and the other is the dependencies that may exist among them.

The attributes that appear in real world datasets are not always independent. The correlations (i.e., dependencies) between attributes that need to be considered in attribute merge, strongly depend on the nature of the sources and their datasets. Our framework is able to handle such correlations in a uniform manner. The following paragraphs provide more details for generating worlds with two possible

Table 4.1: Possible worlds for *merge*($e_1 \equiv e_2$) for exclusive attributes.

aid.	**name**	**value**	**p**
• a_{10}	starring	Daniel Radcliffe	0.7
⋄ a_{11}	starring	Emma Watson	0.4
a_{12}	writer	J.K. Rowling	0.6
a_{13}	genre	Fantasy	0.6
• a_{20}	starring	Daniel Radcliffe	0.5
⋄ a_{21}	starring	Emma Watson	0.9

Possible Worlds

(1)	(2)	(3)	(4)
a_{10}	a_{20}	a_{10}	a_{20}
a_{11}	a_{11}	a_{21}	a_{21}
a_{12}	a_{12}	a_{12}	a_{12}
a_{13}	a_{13}	a_{13}	a_{13}

collections:

A. Independent Attributes One option is to assume no correlation and thus no restrictions on which attributes to include in the resulted entity merge. This case results is only one world given by the union of all attributes:

$$merge(e_1, \ldots, e_n) = \langle id', \cup_{i=1}^{n} e_i.A \rangle$$

B. Exclusive Attributes In certain cases, the attributes originating from different entities participating in the entity merge are exclusive. This requires that only one occurrence of such an attribute to be in the entity resulted by the merge. A typical example of such an attribute are the distinct attribute names, e.g., a person can have only one name. Other examples are the attributes with the same name but similar (semantically or syntactically) values, e.g., attributes a_{11} and a_{21} from Figure 4.1. A simple method is to cluster the exclusive attributes from each entity, i.e., $M = \{\{e_1.\alpha_i, e_1.\alpha_j, \ldots\}\}$. We can then use this set to generate worlds with these correlations:

$$merge(e_1, \ldots, e_n) = \langle id', A \rangle, where$$

$$A \subseteq (M_1 \times M_2 \times \ldots \times M_m) \cup \{\alpha \mid \alpha \notin \cup_{i=1}^{m} M_i.\alpha\}$$

The overall probability of a possible world depends on the probability of the attributes included or not included in the world. It is computed as the product of probability p^{α} when attribute α is part of the world and $(1 - p^{\alpha})$ when attribute α is not part of it:

Table 4.2: Linkages in the Cora datasets.

Entity Linkages (under threshold t)					
t=0.52	t=0.58	t=0.62	t=0.68	t=0.72	t=0.78
12,440	12,012	10,775	6,394	5,985	4,184

$$Pr(\, e' \mid \quad merge(e_1, e_2, \ldots, e_n)) = Pr(\langle \mathcal{E}, \mathcal{L}_c, p^a, p^l_c \rangle) \times \\ \prod_{\alpha \in e'.A} p^{\alpha} \times \prod_{\alpha \notin e'.A \;\&\; \alpha \in e_i.A} (1 - p^{\alpha}) \qquad (4.2)$$

Example 4.3. *Figure 4.1 shows the attributes involved in merge(e_1,e_2). The exclusive attributes are given by set M={{α_{10},α_{20}},{α_{11}, α_{21}}}. Figure 4.1 shows the four generated possible worlds, and their probability is computed according to above formula.*

4.5 Experimental Evaluation

For the experimental evaluation we used a JAVA 1.6 implementation of our approach, which we will refer to as **EAQP**. We also used a JAVA implementation of two additional methodologies described in the following paragraphs.

4.5.1 Approaches under Consideration

Entity Linkage Technique (ELA)

This implementation is the methodology currently followed by existing entity linkage techniques. Once a linkage algorithm is applied on the data (e.g., reference reconciliation [DHM05], or entity resolution [WMK+09]), a list of the matched data along with the derived matching probability is generated [EIV07, KSS06]. Then, the entity linkage technique uses a predefined threshold to accept and keep only the linkages those probability is higher than this threshold.

The data of the accepted linkages are consider to exist (i.e., no probabilities), and thus used for creating the finally integrated dataset in which the data found to describe the same real world entity is merged. Matched data is merged using the approach described in [WMK+09]. Following this approach, the entities in the dataset are not directly merged and updated, but for each entity we maintain all

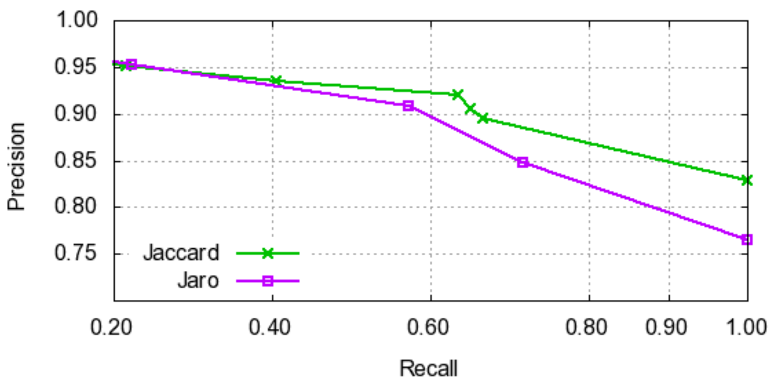


Figure 4.1: Precision-recall for the movie datasets.

matched data, and use them when we need to retrieve the entity from the dataset. Therefore, during query processing, an entity will be retrieved even when the query contains data that are not in the final entity representation, since this data is presented in the entity's matched data.

Probabilistic DBs Technique (PDBT)

A couple of recent approaches have been introduced for addressing the uncertainty appearing with linkage algorithms [ABS+06, AFM06]. They consider duplicate tuples as alternative representations for the same real world object. These techniques expect that the alternative representations for each entity are known and that only one of them can exist in the final integrated data, i.e., alternative representations are disjoin events. Furthermore, an entity is basically described by the information encapsulated in a single record, and not a set of records.

id	attr.
e_1	a
e_1	b
e_1	c
e_1	d
e_2	...

attr.	name	value	prob.
a	*starring*	Daniel Radcliffe	pr
a	*starring*	Radcliffe, Daniel	pr
b	*starring*	Emma Watson	pr
b	*starring*	Watson, Emma (II)	pr
...			

The existing techniques can not represent and handle the uncertainty on the entity linkage information (Chapter 1). For being able to perform a comparison of our EAQP approach with PDBT, we converted the problem as it can be represented by the technique introduced in [AFM06]. More specifically, we assume that

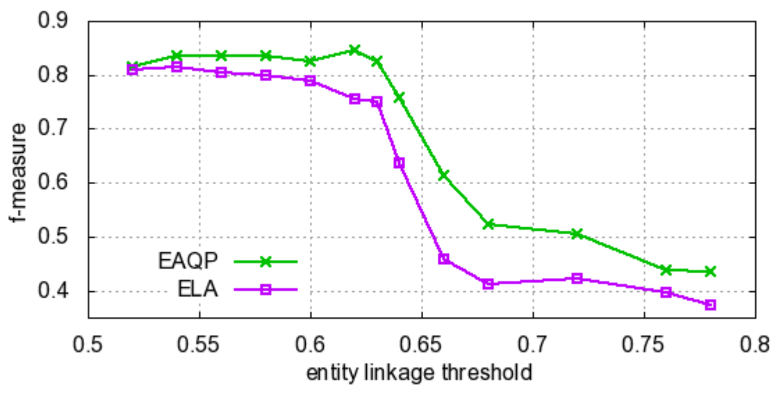


Figure 4.2: F-measure for EAQP and ELA.

the attributes of the entity (i.e., characteristics) are given and that each of these attributes have a set of alternative representations. The above tables show how this representation would apply on the data from Figure 1.2.

Although this representation allows us to provide a comparison between the time required for query processing between the EAQP and the PDBT technique, we still need to note the differences in their semantics: (i) PDBT needs to be provided with the exact linkages for then handling possible alternatives for attributes, and (ii) EAQP is able to handle other types of conditions and is not restricted to disjoin events.

4.5.2 Datasets

Cora Dataset

It is a collection of publications and authors from CiteSeer[1], that is typically used to evaluate linkage techniques [AFM06, DHM05, PD04]. It contains 9,774 author descriptions that refer to 2,882 real world objects. We generated entity linkages between authors (i.e., entities) using the probabilistic entity linkage algorithm [INN08]. Figure 4.2 shows the number of linkages for different linkage thresholds. Precision and recall of the generated entity linkages are similar to the ones generated by other algorithms such as [DHM05, PD04]. For our approach we did not apply a threshold in order to obtain all the linkages, even those with low probabilities.

[1]http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz

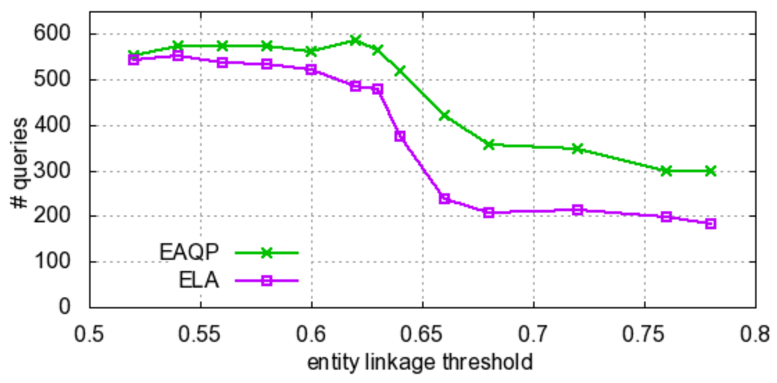


Figure 4.3: Query success of EAQP and ELA.

Movie Dataset

For evaluating the efficiency we needed a sufficiently large dataset and also linkages coming from different linkage techniques. We generated such a dataset by integrating data describing 13, 435 movies coming from two real world systems: 23, 182 *IMDb* movies (relational data) and 28, 040 *DBpedia* movies (RDF data). We converted both datasets to our data model and stored them in a relational database. For generating the entity linkages we compared the movie titles using two standard string similarity methods [CRF03], *Jaccard* and *Jaro*. Figure 4.1 plots the precision-recall graph resulting when using these techniques to link entities, with *Jaccard* being more successful in linking *IMDb* to *DBpedia* movies than *Jaro*. As expected, for both techniques we see the typical dependency between precision and recall. Linkage techniques always have to make a trade-off between the two. In our experiments we investigate how our approach addresses this issue.

All evaluations are reported on the average of 800 queries, constructed by randomly selecting object attributes.

4.5.3 Evaluation Results

Effectiveness

In our first experiment we examine the result quality of our approach using the Cora dataset. We processed the 800 queries and compared the results returned by EAQP and by ELA. Evaluation of the effectiveness is based on the ground truth of the Cora Dataset. For ELA, queries are evaluated over the already pre-merged en-

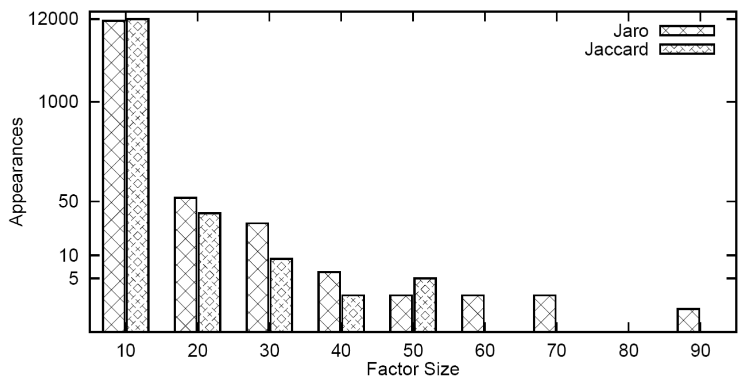


Figure 4.4: Data statistics for the factors.

tities (based on the respective entity linkage threshold). Selecting a low threshold (*t*=0.6) will provide linkages with a high recall but low precision, whereas selecting a higher threshold (*t*=0.8) will provide linkages with significantly higher precision, but with lower recall. So by increasing the threshold only linkages with high probabilities are accepted and the number of linkages, thus, is reduced (see Figure 4.2 for the exact numbers). We examine the behavior of EAQP as well as of ELA for increasing linkage threshold values.

Figure 4.2 shows the average F-measure (weighted harmonic mean of precision and recall) of the 800 queries for various entity linkage thresholds. As expected, when moving towards higher thresholds, the entity linkage technique accepts less and less linkages. This makes the technique unable to find the entities described by the queries. EAQP exhibits a higher F-measure than ELA for the entire range of considered linkage threshold values. The difference is especially high for linkage threshold between threshold values 0.65 and 0.75, where EAQP is still able to identify many of the searched entities. For instance, for *t*=0.66 EAQP returned the correct entity for around 10% more queries than ELA. This is because EAQP can find connecting linkages to construct the entity described in the query, even if the linkage probability is below the threshold. ELA had to reject these linkages due to their low threshold.

Figure 4.3 shows the numbers of queries that were correctly answered for different linkage thresholds. As shown query processing with our approach returns the correct results to more queries than ELA. In additional experiments we performed, we noticed that the entities returned by EAQP were with higher confidence (i.e.,

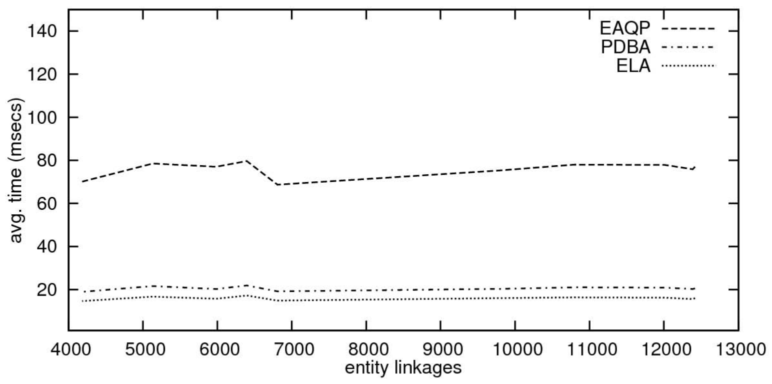


Figure 4.5: Query answering time.

with higher probability) than the entities returned by ELA. For instance, for *t*=0.6 EAQP returned 421 correct answers whereas ELA returned 238 correct answers. For 91 answers, the entities of EAQP had higher probability than the entities of ELA.

Efficiency

The core of our approach is based on generating factors by grouping entities which are pairwise linked. During query processing we select the factors to construct the entities relevant for the query. The number of entities in the factors influences the execution time of our approach. In this experiment we examined this influence using the Movie dataset. We computed the size of the generated factors as the number of entity linkages contained in the factors. We then constructed the histogram of factor sizes. Figure 4.4 shows appearances per factor sizes as generated by both entity linkage techniques, *Jaro* and *Jaccard*. It can be seen that only few factors have a large size, which means less overall processing time.

Our final evaluation was to measure the time required for EAQP, and also to compute the overhead that a system will have for offering this additional functionality. Figure 4.5 shows the average time taken to answer queries with EAQP, PDBA, and ELA. We show time over different number of entity linkages in dataset. As expected there is an increase in the time required by our approach, but this is relatively small and it remains under 70 milliseconds. Furthermore, time does not increase as the dataset gets larger. On the contrary, query time remains stable even when the data size doubles. This behavior is justified by the effective grouping of

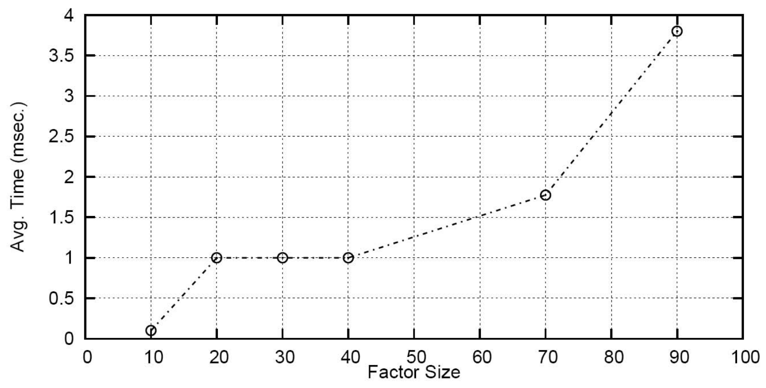


Figure 4.6: Average time for computing the possible world with the maximum probability over factor sizes.

linkages into factors that takes place and allows the algorithm to easily detect and use only a small subset of the linkages during query processing. As expected the additional time required by the PDBA is lower than that of EAQP, since PDBA does not cover the full semantics of our approach, especially not considering linkage probabilities. This clearly leads to a much lower number of possible worlds to be considered, which is reflected in the smaller increase in run time.

Time to retrieve possible worlds

As we presented in the above text, our approach separates linkages into factors and query precessing is performed on the related factors. The size of the factors influences the time required for processing queries, so we now investigate the effectiveness for factors of different sizes.

For this experiment, we measured the time needed to identify the possible world with the highest probability in respect to the factor size (Section 4.3). Figure 4.6 shows the average time required for processing queries over different factor sizes. As expected, for larger factor sizes the algorithm requires more time than for smaller factor sizes, which however still remains below 4 milliseconds. For small factor sizes (i.e., 20-40 entity linkages) that constitute the dominating majority among the factors, the algorithm requires around 1 millisecond.

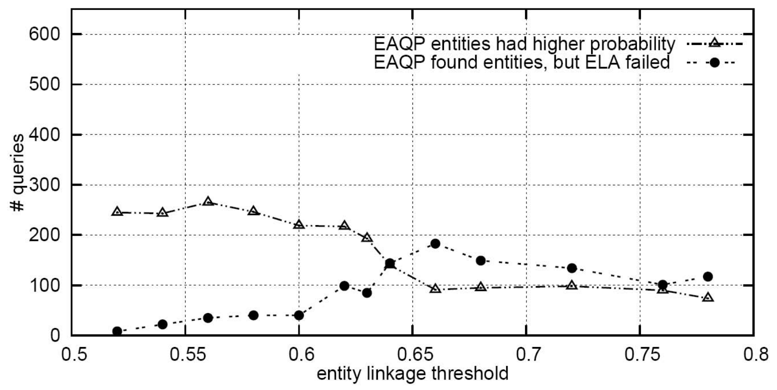


Figure 4.7: The number of queries, from the ones shown in Figure 4.3, in which EAQP entities had higher probability, and identified entities whereas the ELA failed.

Improvements over ELA

We further analyzed the results of the evaluation related to effectiveness, and identified two situations in which EAQP performs better than ELA. The first is that our approach has less failures, i.e., empty result set as an answer to queries. For instance, for *t*=0.6 EAQP was able to return the correct answers for the 150 queries in which ELA did not return anything.

The second situation is that there are cases in which the entities returned by EAQP were with higher confidence (i.e., with higher probability) than the entities returned by ELA. As shown in Figure 4.3, for *t*=0.6 EAQP returned 421 correct answers whereas ELA returned 238 correct answers. For 91 answers, EAQP had higher probability that ELA. Figure 4.7 presents the exact numbers for these two situations for various entity linkage thresholds. As shown by the results of the evaluation, EAQP exhibits a higher effectiveness than ELA. There are of course cases in which both approaches return the same answer set. For these cases, we can view the additional processing time of EAQP as a disadvantage in comparison to ELA.

4.6 Summary

We have introduced a novel approach that allows on-the-fly entity-aware query processing in the presence of linkage information. Our approach can be applied on various data formats and structures using a generic entity representation. We explained how query processing can be performed efficiently over the entities and their possible linkages as these are generated by existing entity linkage techniques. Special focus was given on handling the uncertainty that appears in the entity linkage information as well as in the entity data. Our experimental evaluation confirmed that incorporating entity-aware query processing in a system makes the system able to better handle the entity linkage problem, and able to provide query answers that reflect the possible entity solutions for the current data. The approach has a small overhead in time required for processing queries, but due to our efficient processing strategy this cost remains low and constant even for large datasets with a large amount of entity linkages.

Chapter 5

Detecting Entity Linkages

Detecting and generating probabilistic entity linkages requires computing the similarity between entities. As explained in Chapter 2, comparing two entities based only on their textual values may not be sufficient. For instance, two people entities, or two news articles entities that are written by different authors with different writing styles, may not have a very high similarity when compared using a bag of words representation, while they may still refer to the same entities and qualify as entities referring to the same real world object. In contrast to existing techniques, our focus is to address this problem while also dealing with the special characteristics and challenges described in Chapter 1.

This chapter introduces and explains two techniques for detecting probabilistic linkages. Both techniques considers not only the entity textual values but also the available relationships. The first technique (Section 5.1) detects linkages using the evidences collected from the information space for computing the probability for each possible linkage between entities. The second technique (Section 5.2) focuses RDF data, and it combines an indexing scheme for similarity search with the RDF representations of the resources. This techniques is provided with probabilistic analysis that allows applications to configure it for satisfying their specific quality requirements.

5.1 Linkages based on evidences

The strong relationships between the entities of information spaces are a valuable source of evidences for entity linkages. Existing techniques operating in such information spaces, such as [DHM05] and [BM05], did not restrict themselves to entity attributes only but also systematically exploit the context of the entity, taking into account associations with other entities. For instance, Dong et al. [DHM05] use association properties of entities in combination with normal attributes for computing record linkage. Also, Bekkerman et al. [BM05] exploit the link structure of Web pages about persons as an indicator for entity (person) relationships. The technique we introduce now is most similar to the technique presented in [DHM05], which uses entity context in the form of relationships and propagation of matching evidence information. However, we go further by also using an incremental computation and adaptation of entity linkage information, while achieving comparable precision and recall performance.

Our approach focuses on managing entity linkage information in heterogeneous information spaces using probabilistic methods. We create and maintain a Bayesian network for modeling the evidences that support the possible matches between entities along with the interdependencies between them. This enables us to flexibly update the network when new information becomes available, and to cope with the different requirements imposed by applications build on top of information spaces. The probabilities for all entity linkages are then easily retrieved through the inference over this Bayesian network.

We begin the description of this technique by providing a brief overview over Bayesian Networks and Inference (Section 5.1.1). Then, we present the structure of the network (Section 5.1.2), followed by the procedure for incremental computation and maintenance of the network (Section 5.1.3). Finally, we report the results of the experimental evaluation (Section 5.1.4), which show good precision and recall on real-life data collections.

5.1.1 A Brief Review of Bayesian Networks and Inference

Bayesian Networks

Bayesian networks [Pea88, Jen01] are probabilistic graphical models for reasoning under uncertainty, using cause-effect relationships modeled as a directed acyclic graphs. Each node in the graph represents a variable with two or more possible states. Each edge from parent node X to child node Y, represents a cause-effect relationship with X being the cause and Y the effect, whenever the state of Y is directly influenced by the state of X. Each node X is accompanied with a *local probability distribution* $P(X \mid U_1, .., U_m)$, showing the conditional probability of all states in X given the states of its parents U_1,...,U_m. Nodes without parents are associated with an unconditional $P(X)$, representing prior probabilities.

Bayesian networks represent the *joint probability distribution* over all variables, defined as the product of all local probability distributions, as follows:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid parent(X_i)),$$

where $P(X_i \mid parent(X_i))$ corresponds to the local probability distribution of node X_i, and $parent(X_i)$ to the parent nodes of X_i.

Probabilistic Inference

Bayesian networks successfully determine the conditional probabilities of cause nodes based on the current probabilities of the effect nodes, a task called *probabilistic inference*. Given any new effects (evidences), probabilistic inference recomputes the probability of the cause nodes which are responsible for these effects.

One well-known algorithm for probabilistic inference is *message-passing* by Pearl [Pea88]. It allows the information collected for a single node to propagate to other nodes through the node connections. Pearl's algorithm is iterative, and in each iteration calculates the belief of a node based on messages exchanged by the node X with its parents $U_1, ..., U_m$ and its children $Y_1, ..., Y_n$. When node X is activated, and receives all messages $\pi_X(U_i)$ from its parents, and $\lambda_{Y_j}(X)$ from its children, it calculates its own belief as:

$$\begin{aligned} BEL(X) &= \alpha\lambda(X)\pi(X), \quad \text{where} \\ &\alpha \text{ is a normalization constant,} \\ \pi(X) &= \sum_{U_1,...,U_m} P(X|U_1, ..., U_m) \prod_{i=1}^{m} \pi_X(U_i), \text{ and} \\ \lambda(X) &= \prod_{j=1}^{n} \lambda_{Y_j}(X) \,. \end{aligned}$$

After calculating its belief, node X computes and sends new messages $\lambda_X(U_i)$ to its parents, and $\pi_{Y_j}(X)$ to its children. These messages are:

$$\begin{aligned} \pi_{Y_j}(X) &= \alpha\, \pi(X) \prod_{k \neq j} \lambda_{Y_k}(X) \\ \lambda_X(U_i) &= \sum_X \lambda(X) \sum_{U_k:k\neq i} P(X|U_1, ..., U_k) \prod_{k\neq i} \pi_x(U_k) \end{aligned}$$

metadata for publ. #77 ($M(r_{77})$)**={...,**
⟨ file:///P77, type, ‘publication’⟩,
⟨ file:///P77, title, ... ⟩,
⟨ file:///P77/a_1, name, `K. Marriott`⟩,
⟨ file:///P77/a_2, name, `P. J. Stuckey`⟩}

metadata for publ. #127 ($M(r_{127})$)**={...,**
⟨ file:///P127/a_1, name, `‘Marriott, K’`⟩,
⟨ file:///P127/a_2, name, `‘Sndergaard, H’`⟩,
⟨ file:///P127/a_3, name, `‘Kelly, A’`⟩}

metadata for email #128 ($M(r_{128})$)**={...,**
⟨ file:///E128/to_1, name, `Kelly A.` ⟩,
⟨ file:///E128/from, name, `Sndergaard H.`⟩,
⟨ file:///E128/to_2, name, `Stuckey P.`⟩}

(a)

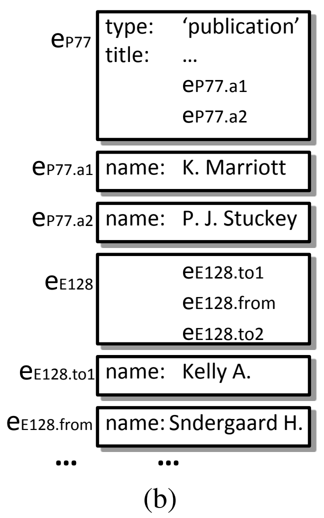


Figure 5.1: (a) Metadata extracted for desktop resources, and (b) an illustration of the corresponding entities following the suggested data model (Section 3.1).

5.1.2 Structure of the Bayesian Network

Figure 5.1(a) shows the metadata generated for describing two publication resources and one email resource[MPC+10]. Following the data model of Section 3.1 we can generate the corresponding entities. Some of these entities are as shown in Figure 5.1(b), and since this metadata does not contain probabilities we expect that each attribute is with belief 1.0. Due to this, the attribute probabilities in the entities are omitted from the figure.

The goal of our algorithm, is to compute the probability of each possible entity linkage, i.e., P(e_i=e_j) providing the probability for the match between entities e_i and e_j. This is done by constructing and maintain a Bayesian network with entities and supporting evidences for all linkages. Thus, information is encoded into the network using the following node types:

Linkage Nodes

These nodes represent a possible entity linkage, e.g., l_4=P($e_{P77.a1}$ = $e_{P127.a1}$). As explained in Section 3, the information space $\mathcal{D}$ does not have the information to directly compute linkages, thus we can not specify the states of entity nodes. The probability of their states is computed through probabilistic inference based on the cause-effect relationships in the network.

Evidence Nodes

These nodes represent evidences for the linkage nodes, and it is the first type of nodes we use to represent evidence for linkage nodes. Each evidence node represents a supporting evidence $s(a_i, a_j, \phi)$, with a_i and a_j being attributes from the two entities participating in the possible linkage. Evidence nodes form the prior probabilities in our network, since they have no parent nodes upon which they depend. The unconditional probability distribution is determined by the similarity function ϕ used for creating them. In our current approach, we rely on the comparison of literals from a_i and a_j to measure compatibility between the corresponding properties. An extension of our approach for operation in more heterogeneous contexts is the inclusion of a comparison of the properties themselves and the inclusion of these similarities into the aggregation of the matching probabilities. This can be achieved by the introduction of further evidence nodes.

Direct-Relation Nodes

These nodes rely on the fact that two entities are related when their descriptions contain references to the same entities. It is the second type of nodes that influence the states of linkage nodes. For example, possible linkage P($e_{P77.a1} = e_{P127.a1}$) implies a relation between entities e_{P77} and e_{P127}, encoded in the direct-relation node dir-rel(e_{P77}, e_{P127}). Finding evidences for more shared references between these entities (e.g., additional common authors) increases the belief for the relation of e_{P77} with e_{P127}, and consequently the belief in the corresponding possible linkage. Direct-relation nodes are the effect we can observe for entity nodes and for deductive-relation nodes (explained bellow). The local probability distribution of their states is directly influenced by their relationships with these node types.

Deductive-Relation Nodes

These nodes represent the indirect relation between two entities, inferred by combining the information of two nodes, either direct-relation or deductive-relation. Combining direct-relation nodes dir-rel(e_{P77}, e_{P127}) and dir-rel(e_{P77}, e_{E128}) for example, implies a new relation between e_{P127} with e_{E128} (due to the common resource e_{127}), which we encode in deductive-relation node del-rel(e_{P127}, e_{E128}). The local probability distribution for this node type is computed using the two nodes from which this node is inferred.

Table 5.1: The possible cause-effect relationships used in our Bayesian network.

Effect Nodes:	(1) Evidence	(2) Direct-Rel.	(3) Deductive-Rel.
Cause nodes:			
(1) Linkage	√	√	
(2) Ded.-Rel.		√	√

An important aspect of the Bayesian network is the cause-effect relationships between the nodes of the Bayesian network. We have explained these together with the different types of nodes. Table 5.1 gives a summary.

5.1.3 Incremental Computation of the Network

To compute the probability of possible linkages we collect evidences, positive and negative. We then calculate their probability by constructing a Bayesian network, modeling linkages and related evidence. Starting point for this computation is an incrementally growing metadata set (i.e., entities), which are added to $\mathcal{D}$. We have to update the Bayesian network incrementally, after addition of new metadata (i.e., entities).

Upon addition of a new metadata, and thus a new set of entities, the algorithm performs four steps. Mere details and explanations are provided in the following paragraphs.

1. Processes the entity descriptions contained in the added metadata. By comparing the attributes from the new entities with the attributes of previous entities, the algorithm identifies similarities, and updates the network with new evidence and linkage nodes.

2. Creates direct-relation nodes to represent the effects we observed that could cause these new linkage nodes.

3. Analyzes the updated network and generates new information about the relation of entities, represented using deductive-relation nodes.

4. Performs probabilistic inference on the Bayesian network, and generates the updated probabilities for the possible linkages.

Step 1 - Adding Entity & Evidence Nodes.

The new metadata contains one or more entity descriptions. In the first step, the algorithm updates the Bayesian network with new evidences generated using these entities. We start with similarity computations to identify resemblance between attributes from the new entity $e_{new.k}$ with compatible attributes from the existing entities $e_{exists.m}$. In the current version of our algorithm, similarities are detected using two functions. The first algorithm is *String Similarity*, detecting string resemblance between literals of attributes[1]. The second algorithm is *Soundex Similarity*, which detects the resemblance in pronunciation between literals[2]. Whenever similarity is above a given threshold, we consider it as supporting evidence for the possible linkage P($e_{new.k}$ = $e_{exists.m}$).

An evidence node is created for each similarity identified. Since the current version of our algorithm includes two similarity algorithms, we create one or two suporting evidence nodes for each possible linkage. All evidence nodes have three states, *Good*, *Moderate*, and *Poor*, which we set based on computed similarity.

A linkage node is created to represent the identified match P($e_{new.k}$ = $e_{exists.m}$), if such node does not yet exists. The relation between the newly created supporting evidence nodes with the linkage node is represented by introducing cause-effect relationships. All linkage nodes have two possible states, *Exists* to indicate that the corresponding linkage is true, and *Exists_Not* to indicate that the linkage is not true. This probability for each assignment is done based on probabilistic inference.

Step 2 - Adding Direct-Relation Nodes.

Direct-relation nodes represent the observed effect that linkage nodes could cause. These node are created by using only information from the linkages as follows: for each linkage P($e_{new.k}$ = $e_{exists.m}$) we extract its entities, and use them to create a direct-relation node del-rel(e_{new},e_{exists}), if this node does not yet exist. If there is more than one linkage referring to the same two entities, we represent this through a cause-effect relationships created between the linkage nodes and the corresponding direct-relation node. The direct-relation nodes have two possible states, *Yes* to indicate that the two entities are related, and *No* to indicate that the entities are not related. The probabilities of these states are again computed by probabilistic inference.

[1]For String Similarity we use the JaroWinkler method from the SecondString API [CRF03].
[2]For Soundex Similarity we use the Apache Codec API.

Step 3 - Adding Deductive-Relation Nodes.

This step analyzes the current status of the network to extract indirect relations between the entities. The underlying idea is similar to the one represented by the direct-relation nodes. To identify possible indirect relations, our algorithm inspects the direct-relation and deductive-relation nodes. Each node is considered as a transitive, binary relation (b-relation) between the two participating entities. For example, dir-rel(e_{P77},e_{P127}) corresponds to b-relation (e_{P77},e_{P127}), and dir-rel(e_{P77},e_{E128}) to b-relations (e_{P77},e_{E128}). The algorithm extracts more relations by transitively combining b-relations: b-relation (e_{P127},e_{P127}) for example is the transitive combination of our two previous b-relations. We encode the new b-relation using a ded-rel node, for example del-rel(e_{P127},e_{P127}).

Since computing transitive b-relations is a recursive process, we need an appropriate stopping criterion. In the current version of our algorithm, we enforce a fixed ratio between linkage nodes and deductive-relation nodes. This approach allows us handle specific characteristics possibly present in $\mathcal{D}$. If $\mathcal{D}$ contains only few linkages, the algorithm will be forced to search for evidence by incorporating many deductive-relation nodes. On the other hand, if $\mathcal{D}$ contains a relatively big number of linkages, the algorithm will include only a small subset of them, enough to increase the belief for the specific node without overloading the network with nodes.

Step 4 - Updating the Linkages.

Once the network is updated with nodes representing new possible linkages and supporting evidences, we need to recalculate the probability for the states of each node. This task is performed through probabilistic inference which updates all nodes according to the current status of the network. To minimize the time needed for doing this, we execute probabilistic inference only on the newly added nodes and nodes related to them.

As explained in Section 5.1.1, computing the probability of a node requires information from its neighbor nodes. Computed results are propagated back to the neighbor nodes to allow them to recompute their probability. For example, consider node R from the Bayesian network of Figure 5.2. Once node R is activated, and receives messages $\lambda_{r_1}(R)$, $\lambda_{r_2}(R)$, $\pi_R(l1)$, and $\pi_R(l2)$ from its parent and children nodes, it computes: (i) its own belief as $BEL(R) = \alpha\lambda(R)\pi(R)$ (marked as eq. 5.1 in the following equation list), and (ii) new messages to send to its parent nodes

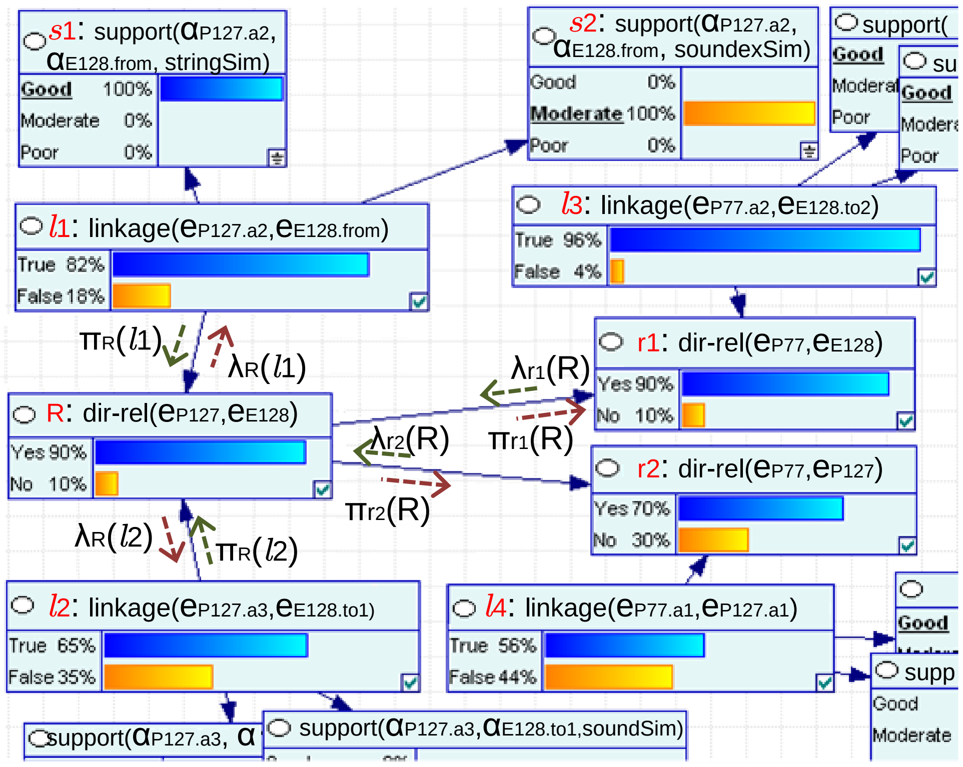


Figure 5.2: An illustration of the Bayesian network for the data of Figure 5.1.

(eq. 5.2), and children nodes (eq. 5.3). These messages are as follow:

$$\lambda(R) = \lambda_{r_1}(R)\lambda_{r_2}(R), \text{ and } \pi(R) = P(R|l1, l2)\pi_R(l1)\pi_R(l2) \tag{5.1}$$

$$\lambda_R(l1) = P(R|l1, l2)\pi_R(l2)\lambda(R) \tag{5.2}$$

$$\pi_{r2}(R) = \pi(R)\lambda_{r1}(R) \tag{5.3}$$

The message computation in this example shows the main benefit of using cause-effect relationships between nodes. Although node $l1$ is not directly connected to node $l2$, the algorithm is able to propagate information from one node to the other, through their cause-effect relationships with node R. Consequently, a high belief of node $l2$ affects the belief of node R (eq. 5.1), which is reflected in the message node R sends to node $l1$ (eq. 5.2). Finally, node $l1$ is affected when it recomputes its belief using the message sent to it by node R.

After executing probabilistic inference, we have an updated set of matches that

Entity Linkage information:

⟨ el:///linkage_p127.a2_p128.from,	entity,	file:///P127/a_2 ⟩
⟨ el:///linkage_p127.a2_p128.from,	entity,	file:///P128/from ⟩
⟨ el:///linkage_p127.a2_p128.from,	belief,	0.96 ⟩
...		

Figure 5.3: Part of the resulted entity linkage information that was generated by our algorithm, for the data shown in Figure 5.1.

reflect the metadata present in the information space. Different representations of the results matches are possible (i.e., include or do not include the probability of each match), and the selected representation depends on the needs of the specific system. Figure 5.3 shows one possible representation for the results of the metadata from Figure 5.1. In this example, the matches are used for generating additional metadata to represent each match in the Bayesian network using the corresponding object representations and belief.

5.1.4 Experimental Evaluation

We evaluated our approach using a JAVA implementation of our entity linkage algorithm, including all features we described in the previous sections. For performing probabilistic inference[3]on the Bayesian network we used the *jSMILE API*[4], and for creating a database to store internal information we used *MySQL* 5.0[5]. The following paragraphs present the effectiveness of our algorithm on two datasets, the Cora and a PIM dataset.

Cora Dataset

The *Cora dataset*[6] is a collection of publications collected from CiteSeer. Each publication contains title and author names, using different forms for the names (e.g., ‘J. Antonisse’, ‘Antonisse , H. J.’, ‘Antonisse’, ‘Jim Antonisse’). The dataset was manually processed, each publication author is accompanied with an identifier that indicates the corresponding real-world entity.

[3]For efficiency reasons we use the ‘Backward simulation’ algorithm; a modified version of Pearl’s algorithm that performs approximate inference.

[4]http://genie.sis.pitt.edu/

[5]http://www.mysql.com/

[6]We used the version from http://www.cs.umd.edu/~indrajit/ER/index.html

We processed the Cora dataset and converted each publication into RDF triples. Our process generated 14392 attributes describing title and authors for 1563 resources (publications). A total of 2882 attributes described authors, with 9768 matches between these authors. The number of matches for author references ranged between 1 and 43, with an average of 3,39. Following the definitions of Section 3, we use as entity $e_{A_i.C_k}$, the attributes describing author A_i as given in the attributes generated for publication C_k. The task of our algorithm is then to compute the probability for entity linkages of author A_i from publication C_k with author A_j from publication C_n, represented by P($e_{A_i.C_k}$=$e_{A_j.C_n}$).

The goals of our Cora dataset experiments were twofold: (i) evaluate the effectiveness of our algorithm in identifying the entities, and (ii) compare the effectiveness of our algorithm with the effectiveness of the basic similarity functions we use for generating the evidence nodes. We measured effectiveness, as usual in information retrieval, by calculating precision and recall. These measures were calculated with respect to the actual real world objects, as specified by the unique identifier given for the authors of each publication in the Cora dataset.

We executed the experiments, by adding the attributes generated from the Cora dataset incrementally into an information space, which uses our algorithm for entity linkage. After adding triples for 100 publications, we performed probabilistic inference on the Bayesian network generated by our algorithm. The following table shows the number of the entity linkages that correspond to the different numbers of publications.

Publications	1000	1100	1200	1300	1400	1563
Linkages	4129	4620	5050	6036	7337	9774

Entity Linkage Effectiveness.

Figure 5.4 shows the plot for precision, and Figure 5.5 the plot for recall. Both plots are for different probability thresholds and for several publications groups. The plots do not include groups that contain less than 1000 publications because the number of the corresponding matches is too small. Small values of the probability threshold (θ <0.4) are not included in the plots since the results are similar to θ=0.4.

As shown in Figures 5.4 and 5.5, our algorithm is able to maintain the same values for precision and recall for the different probability thresholds. For lower probability thresholds (i.e., θ=0.4, and θ=0.5) we see that recall is very high and precision is already quite satisfactory (around 0.9). Moving toward higher proba-

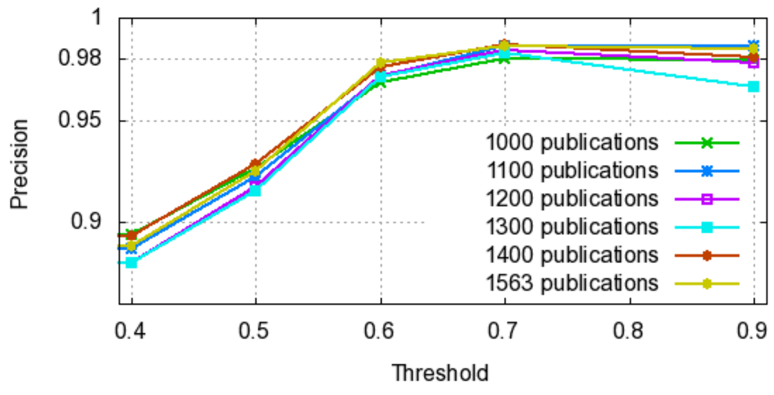


Figure 5.4: Precision vs. different thresholds.

bility thresholds (i.e. θ=0.6, θ=0.7,) we see precision values increasing and, as expected, decreasing recall values. Precision does not 'automatically' increase with groups that have more publications —more data, and thus entities are available— but rather reflects the belief we have for the entities in our current data.

The results of these plots follow exactly the behavior explained in the analysis of our algorithm. It is clear that external algorithms are able to control the precision/recall of the entities by selecting an appropriating value of the probability threshold. For example, an application that needs only very certain linkages will choose a high probability threshold, whereas an application that accepts uncertain-linkages a lower.

We also used our Cora dataset experiments to compare with previous approaches described in the literature. The authors of [DHM05] reported precision 0.994 with recall 0.985, the authors of [PD04] had precision 0.842 with recall 0.909. To compare these numbers with our results, we considered only linkages generated by our algorithm that exceed a preselected low probability threshold (e.g., θ=0.5). As shown in our two plots, these linkages have high precision and high recall, similar to the ones given by these other algorithms. Our algorithm offers two additional advantages: (i) identified matches do not alter original data, (ii) our algorithm is able to further classify these linkages according to the belief we have for their existence.

Comparison with basic similarity functions.

In this experiment we performed a comparison of the effectiveness of our algorithm with the basic similarity functions used for generating evidence nodes. The algo-

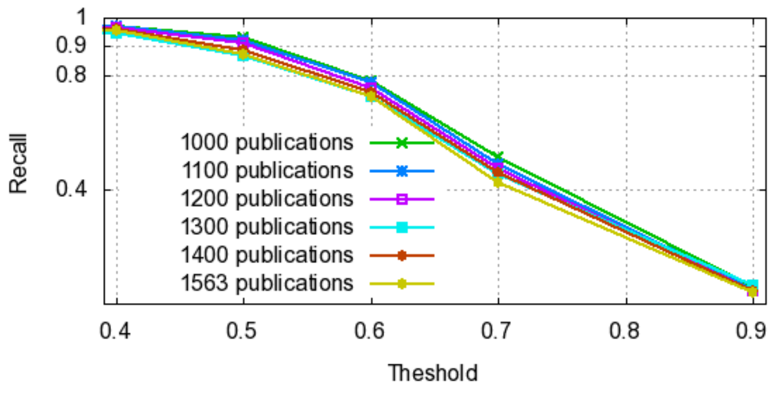


Figure 5.5: Recall vs. different thresholds.

rithms we considered were Soundex Similarity and String Similarity, as described in Section 5.1.3.

Table 5.2 shows precision and recall values given by the two similarity functions on different publications groups. In all cases we assume that the real world objects are the ones those probability is above threshold 0.7. Our evaluation shows our entity linkage technique clearly improves effectiveness of the basic similarity functions.

PIM Dataset

As a second dataset for evaluating our algorithm we use metadata generated in a personal information management environment, for desktop resources. As there is no publicly available PIM dataset, we created a suitable collection of metadata by simulating the behavior of a PIM application.

Our PIM dataset included metadata describing desktop resources from three groups:

- The first group contains publications, randomly selected from the DBLP system, to simulate arbitrary publications downloaded from the Web. This group resulted in metadata describing 700 imported resources, with 1326 triples (i.e., attributes) corresponding to authors.

- The second group contains publications imported into the PIM environment from the DBLP system for which one of the authors is our co-worker at L3S. The results of this import were metadata for 250 resources, with 480 triples

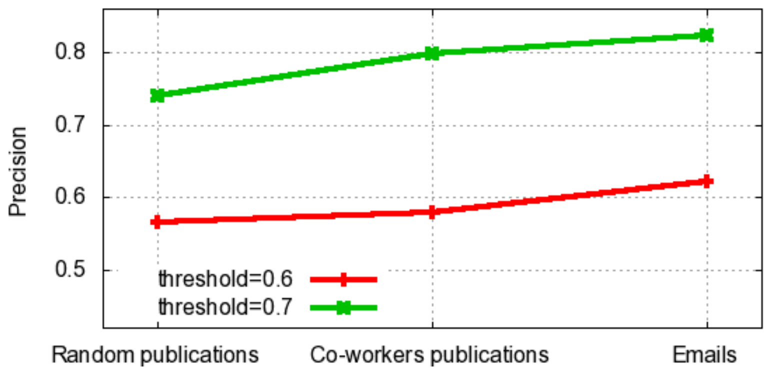


Figure 5.6: (a) Precision, and (b) Recall vs. different thresholds.

(i.e., attributes) describing authors.

- The third group contains personal emails from one of the author's email client. Our goal was to identify and link authors from the publications with the corresponding person sending emails. The entity linkage problem in this case is somehow limited since persons are usually accompanied with email addresses which can act as unique identifiers for them. For this reason, we applied our entity linkage algorithm only on the existing email address, person name pairs. To capture the connections between these people, we randomly selected a small portion of available emails. This group contained metadata for 200 resources, with 400 triples describing people. This group contains the most heterogeneous data, since each person has various email addresses and name variants.

We evaluated our algorithm on this PIM dataset, adding data incrementally into an information space, which uses our algorithm for entity linkage. We performed probabilistic inference on the Bayesian network generated by our algorithm, after adding the data of these three groups. Figure 5.6 shows the precision of the results generated by our algorithm. As shown, adding emails does not reduce the precision of the generated results, which is what we would have excepted since the emails contain the most heterogeneous data. The results indicate that our algorithm is able to handle the heterogeneous instances of persons referenced in emails, and successfully link them with the author instances gained from the publications.

Table 5.2: Precision/Recall of the entity linkage, and the basis similarity functions we used for generating the evidence nodes (θ=0.7).

Publications	Entity Linkage	String Sim.	Soundex Sim.
200	0.219/0.969	0.892/0.081	0.482/0.362
400	0.218/0.977	0.422/0.065	0.246/0.346
600	0.358/0.982	0.329/0.05	0.181/0.220

5.2 Linkages based on RDF structure

In the Semantic Web resources are annotated with metadata in the form of RDF statements. These annotations can be made manually or (semi-)automatically using tools for natural language processing and information extraction, such as the Calais Web Service [Ope] or metadata extractors [MPC+10], which identify and extract from unstructured text entities, facts, relationships, and events, and provides them in the RDF format. This structured and semantically rich information can be exploited to more accurately identify entity linkages between resources. Existing approaches that deal with the problem of efficiency in similarity search, e.g., [BCG05, GIM99, MJS07], do not operate on structured data.

The introduced technique is named *RDFsim*. Short for *"RDF Data Similarity"*, RDFSim performs semantic-aware and efficient detection of entity linkages by combining indexing schemes for similarity search with the RDF representations of the resources, i.e., entities. The following paragraphs present the formalization and an overview of the technique (Section 5.2.1). We then present the steps of RDFsim in detail, starting from how it semantically represent RDF resources (Section 5.2.2). Next, we introduce the indexing structure that is employed from the RDFsim (Section 5.2.3). We then explain the use of this indexing structure for querying possible duplicate resources, and thus retrieving the probabilistic entity linkages (Section 5.2.4). Finally, we report the results of the experimental evaluation (Section 5.2.5).

5.2.1 Overview of RDFsim

A resource in the Semantic Web is described by a set of RDF triples of the form $(subject, predicate, object)$, where $subject$ is a URI identifying a resource, $predicate$ is a URI representing a property of the resource, and $object$ represents the value of this property, which can be either a literal or a URI identifying another resource.

These triples form a graph, where the nodes correspond to subjects and objects, and the edges correspond to predicates. When a node is not identified by a URI (i.e., blank nodes), we use the node id information that is provided. Hence, each resource is represented by an RDF graph R, constructed from the RDF triples which describe this resource.

Let $\mathcal{R}$ be the set of all available resources, each one describing an entity. Function $sim : \mathcal{R} \times \mathcal{R} \rightarrow [0,1]$ computes the similarity between two resources, based on their RDF graphs. We define possible duplicate resources as follows:

Definition 5.1. *Given two resource descriptions R_1 and R_2, a similarity function sim, and a similarity threshold $minSim$, then these two resources are possible duplicates if $sim(R_1, R_2) \geq minSim$.* ■

Given a potentially large set of resources $\mathcal{R}$, the problem we focus on is to efficiently identify all pairs of possible duplicate resources in $\mathcal{R}$. A straightforward solution to this problem is to first perform a pairwise comparison between all the resources, and then to select those pairs having similarity above the given threshold. However, this is not scalable with respect to the number of resources, and hence not suitable for performing this task under time restrictions (e.g., online processing), or when the set of resources $\mathcal{R}$ is dynamic.

To address this problem efficiently, we need to avoid the pairwise comparisons of resources. For this purpose, we propose a method that relies on Locality Sensitive Hashing (LSH) [GIM99]. First, each resource is converted into the internal representation used by the algorithm, which is then indexed in an index structure based on LSH. This index structure allows us to efficiently detect the possible duplicates of a given resource, with probabilistic guarantees.

Probabilistic Entity Linkages

RDFsim enables the efficiently retrieval of the possible duplicates for any given resource, i.e., entity. If we have an entity we can first make a collection with all its possible duplicates, and then use an entity linkage technique for computing the probability between the given entity with each of the entities in the retrieved collection. We therefore can efficiently retrieve the linkage probability for an entity.

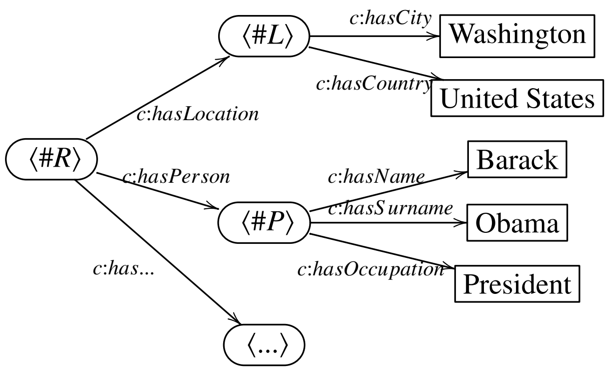


Figure 5.7: Representation of resources for RDFsim takes into consideration the semantic structure.

5.2.2 Resource Representation

Our method emphasizes on semantic-aware detection of possible duplicate resources, i.e., it operates on the RDF representation of the resources. As this information is often not available a priori, a pre-processing step may be required to extract semantic information for the resources. There are several tools that can be used for this purpose, such as the Calais Web Service [Ope] (see Section 5.2.5 for more details). Subsequently, ontology mapping methods can be applied to handle the cases where different vocabularies are used by different sources. In addition, some metadata may be deliberately filtered out by the application, as they may not be relevant to the task of possible duplicate detection. For example, in the case of the news aggregation scenario, an article identifier assigned to the article by the particular agency publishing it should not be taken into consideration when searching for possible duplicate articles.

Once the RDF graph describing the resource has been constructed, it needs to be transformed to a representation that is suitable for indexing in an index based on LSH, while preserving the semantic information for the resource. For this purpose, the algorithm applies a transformation of the RDF graph of each resource R_x as follows: each RDF triple is represented as a concatenation of the predicate and the object. In the case that the object is a literal, then the predicate is concatenated with the literal. In the case that the object is itself the subject of another RDF triple, e.g., R_y, then the predicate is concatenated with the representation $rep(R_y)$ of R_y, which is generated recursively. During this recursive generation, cycles are detected and broken. This process is illustrated by the following example.

Example 5.1. *Consider the RDF graph shown in Figure 5.7. The representation of the nodes L and P are the following:*

$$rep(L) = \{\text{“c:hasCity, Washington”, “c:hasCountry, United States”}\}$$

$$rep(P) = \{\text{“c:hasName, Barack”, “c:hasSurname, Obama”,}$$
$$\text{“c:hasOccupation, President”}\}$$

Then, the representation of the resource R is generated recursively using the representations of the resources under R (e.g., L and P) as follows:

$$rep(R) = \{\text{“c:hasLocation, L”, “c:hasPerson, P”}, \dots\} \cup rep(L) \cup rep(P)$$

Notice that some resources may have large and complex RDF graphs (e.g., large documents), which leads to large representations. However, this does not constitute a problem since these representations do not need to be maintained in main memory. Instead, the representation of each resource is only computed and used once, as an intermediate step for the purpose of hashing it in the index structure.

Along with the resource representation, our algorithm also needs a similarity method (see Definition 5.1) that is used for computing the similarity between two RDF representations. For the purpose of this work we apply one of the standard similarity measures, Jaccard coefficient. However, the algorithm and the underlying LSH index can incorporate other measures, and there have already been analytic results which enable LSH on different distance measures [Cha02], for example for the cosine similarity.

5.2.3 Indexing Structure

The index used by the algorithm is based on the Locality Sensitive Hashing (LSH) approach of [GIM99]. The main idea behind LSH is to hash points from a high dimensional space using a hash function h such that, with high probability, nearby points have similar hash values, while dissimilar points have significantly different hash values, i.e., for a distance function $D(\cdot,\cdot)$, distance thresholds (r_1, r_2), and probability thresholds (pr_1, pr_2):

- if $D(p,q) \leq r_1$, then $Pr[h(p) = h(q)] \geq pr_1$
- if $D(p,q) > r_2$, then $Pr[h(p) = h(q)] < pr_2$

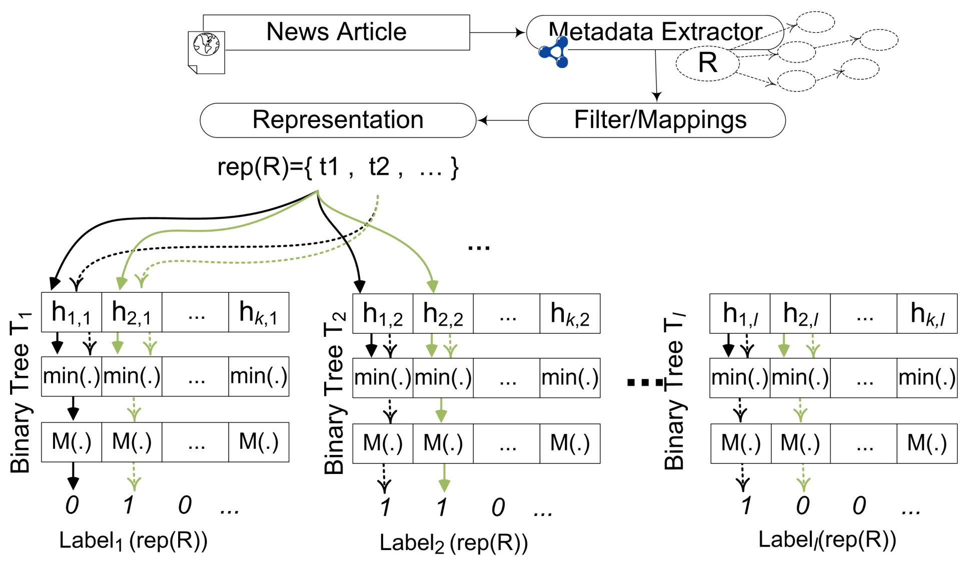


Figure 5.8: An illustration of the process followed for generating the labels of RDF resources, which are used for inserting these resources into the indexing structure.

More specifically, we use an indexing structure $\mathcal{I}$ that consists of l binary trees, denoted with $\mathcal{T}_1$,$\mathcal{T}_2$, ..., $\mathcal{T}_l$. To each tree, we bind k hash functions, randomly selected from a family of locality sensitive hash functions $\mathcal{H}$. We denote the hash functions bound to tree $\mathcal{T}_i$ as $h_{1,i}, h_{2,i}, \ldots, h_{k,i}$.

Figure 5.8 shows the process we follow for indexing resources. When a new resource R_x arrives, first its representation $rep(R_x)$ is computed as described above. Recall that $rep(R_x)$ consists of a set of terms (i.e., the elements of the set $rep(R_x)$). We compute l labels of length k. Each label corresponds to a binary tree. The algorithm computes the label of $rep(R_x)$ for each tree $\mathcal{T}_j$ as follows:

- It hashes all the terms in $rep(R_x)$ using each hash function $h_{i,j}(\cdot)$ that is attached to the binary tree $\mathcal{T}_j$.
- It detects the minimum hash value produced by $h_{i,j}(\cdot)$ over all terms in $rep(R_x)$, denoted as $min(h_{i,j}(\cdot))$.
- It maps $min(h_{i,j}(\cdot))$ to a bit 0 or 1 with consistent mapping $\mathcal{M} \mapsto [0,1]$. This resulting bit is used as the i'th bit of the label of $rep(R_x)$.

The same map $\mathcal{M}$ is used for all the binary trees. Any mapping function can be used, for example *mod* 2, as long as it returns 0 and 1 with equal probability.

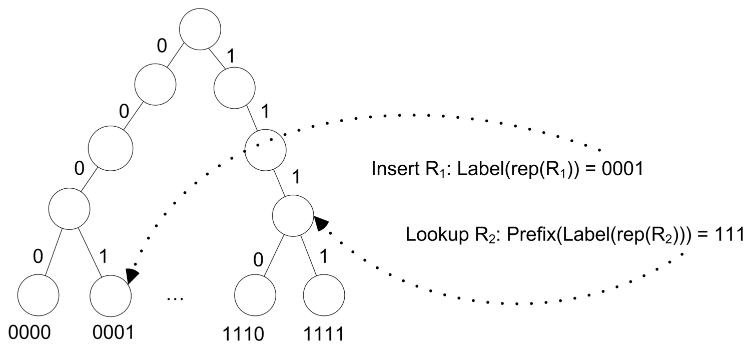

0
1
0
1
0
1
0
1
0
1
0000
0001
...
1110
1111
Insert R1: Label(rep(R1)) = 0001
Lookup R2: Prefix(Label(rep(R2))) = 111

Figure 5.9: Inserting and searching for resources in a tree of the algorithm.

After computing the l labels of a resource, the algorithm inserts the resource in the inverted index. Let $Label_i(rep(R_x))$ denote the binary label computed from R_x for the binary tree $\mathcal{T}_i$. Then, R_x is inserted in the tree using $Label_i(rep(R_x))$ as its path. For example, if $Label_i(rep(R_x)) = 0001$, then R_x is inserted at the node with the specific path in tree $\mathcal{T}_i$ (see Figure 5.9).

5.2.4 Retrieving Linkages

Executing a query for possible duplicate resources is similar to the process described above for indexing a resource. Let R_q denote the resource for which we want to search for possible duplicates, and $minSim$ the minimum similarity between the query R_q and another resource $R_p \in \mathcal{R}$ for considering the two resources as possible duplicates. Our method provides a trade-off between performance and recall, expressed by the minimum probability *minProb* that each possible duplicate of R_q is found.

First, we create the labels for the query $Label_1$(rep(R_q)), $Label_2$(rep(R_q)),..., $Label_l$(rep(R_q)), which correspond to each of the l trees $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_l$. Assume now that we are interested only for *exact* matches of R_q, i.e., exact duplicates. Then, the query would be executed by performing a lookup of each label in the corresponding tree, selecting the resources indexed in the identified nodes, and examining whether each of these resource is an exact duplicate of R_q. Notice that due to the hashing and mapping functions employed during the indexing process, several resources may be indexed under the same node, hence the last step in the aforementioned process is required to filter out false positives.

Since in our case we are interested in finding the possible duplicates of R_q, we need to relax the selection criterion in order to retrieve resources that are not exact matches but highly similar to R_q. Recall that due to the property of Locality Sensitive Hashing, similar resources are indexed at nearby nodes in the tree with high probability. Hence, the selection criterion can be relaxed by performing a lookup not for the entire label but only for a prefix of it, of length k'. The question that arises is how to determine the appropriate value for k'. Setting a high value for k' leads to a stricter selection, and hence some possible duplicates may be missed. On the other hand, a low value for k' retrieves a large result set, from which false positives need to be identified and filtered out, thus reducing the performance of query execution. For example, in the extreme case where $k' = 1$, half of the resources from each tree are retrieved, leading to a very large result set. Consequently, k' should be set to the maximum value that still allows for possible duplicate resources to be detected with probability equal or higher than the requested *minProb*. Once k' has been determined, we retrieve from each tree the resources with the same prefix to the respective label of R_q, which results in the set of candidate possible duplicates for R_q, denoted by $\mathcal{ND}_{cand}(R_q)$. Then, for each resource in $\mathcal{ND}_{cand}(R_q)$, we compute its similarity to R_q, filtering out those resources having similarity lower than $minSim$. In the following, we provide an analysis on how to determine the right value for k'.

The appropriate value k' of the prefix length to be used for the lookup during query execution is determined by the values of *minProb* and $minSim$. We assume that the index comprises l binary trees, and labels of total length k ($k' \leq k$). The computation is based on the following theorem.

Theorem 5.1. *Let $sim(P, Q)$ denote the Jaccard similarity of two resources P, Q, based on their respective representations $rep(P)$ and $rep(Q)$. The corresponding labels $Label_i(rep(P))$ and $Label_i(rep(Q))$, $i = 1 \dots l$, of the two resources are equal with probability $Pr[Label_i(rep(P)) = Label_i(rep(Q))] = \left(\frac{1+sim(P,Q)}{2}\right)^k$. Furthermore, the probability that the two resources have at least one common label is $1 - \left(1 - \left(\frac{1+sim(P,Q)}{2}\right)^k\right)^l$.*

Proof. As explained in Section 5.2.3, each bit in the label is computed by (a) hashing all terms of the representation using a hash function from a family of LSH functions $\mathcal{H}$, (b) getting the minimum hash value over all terms, and (c) mapping it to binary. Let $min(h_{i,j}(rep(P)))$ denote the minimum value of the hash function $h_{i,j}$ over all the terms of $rep(P)$, and $\mathcal{M}(min(h_{i,j}(rep(P))))$ the result of the mapping function. The labels $Label_j(rep(P))$ and $Label_j(rep(Q))$ of the two resources P and Q will have the same corresponding bit i if either of the following holds:

(a) $min(h_{i,j}(rep(P))) = min(h_{i,j}(rep(Q)))$, or

(b) $min(h_{i,j}(rep(P))) \neq min(h_{i,j}(rep(Q)))$ and $\mathcal{M}(min(h_{i,j}(rep(P))))=\mathcal{M}(min(h_{i,j}(rep(Q))))$

The probability of (a) is directly related to the similarity of the two representations [BCFM98], and is equal to:

$$Pr[min(h_{i,j}(rep(P))) = min(h_{i,j}(rep(Q)))] = sim(rep(P), rep(Q)) \tag{5.4}$$

The probability of (b) equals to:

$$\left(1 - Pr[min(h_{i,j}(rep(P))) = min(h_{i,j}(rep(Q)))]\right)/2$$

Since the two cases are mutually exclusive, the probability that either (a) or (b) is true is the sum of the two probabilities, and equals to:

$$\frac{1 + sim(P,Q)}{2}$$

For two resources to have the same label i, then all bits $1, 2, \ldots, k$ of the two labels must be equal. The probabilities are independent, therefore:

$$Pr[Label_i(rep(P)) = Label_i(rep(Q))] = \left(\frac{1 + sim(P,Q)}{2}\right)^k \tag{5.5}$$

Then, the probability that the two resources have at least one common label is:

$$\begin{aligned} & Pr[\exists i : Label_i(rep(P)) = Label_i(rep(Q))] && (5.6)\\ & = 1 - Pr[\neg\exists i : Label_i(rep(P)) = Label_i(rep(Q))] \\ & = 1 - \left(1 - \left(\frac{1 + sim(P,Q)}{2}\right)^k\right)^l \end{aligned}$$

□

Following directly from Equation 5.6, we can compute the value of k' as:

$$k' = \left\lfloor -\frac{\log\left(1 - (1 - minProb)^{1/l}\right)}{\log(2) - \log(1 + minSim)} \right\rfloor \tag{5.7}$$

The number of trees l comprising the index and the length k of each label are set during the initialization of the algorithm. Higher values of l allow the algorithm to also use longer prefixes of length k' for querying, which results to fewer false positives, and consequently to lower cost for retrieving the candidate possible duplicate resources and comparing them to the query. However, as l increases, there is an extra cost imposed for maintaining the additional trees. For tuning these parameters l and k, one needs to have some knowledge regarding the queries and the distribution of the resources to be indexed. If this information is not available, one can choose values that are large enough to support a wide range of queries, while still having a good performance. For our experiments, we experimented with different combinations of l and k, and we observed that an index with $l = 20$ and $k = 50$ enabled the algorithm to answer queries efficiently, for probabilistic guarantees as high as 98% and minimum similarity as low as 0.8. By further increasing l and k one can enable stricter probabilistic guarantees and lower similarity thresholds, albeit with a higher cost for maintaining the index.

Maintenance of the binary trees

Another aspect of our approach is the implementation details for the binary trees. In practice, a fast and memory-efficient implementation of binary trees is important for the efficiency and scalability of the algorithm. In this work, we consider two techniques: (i) a main memory binary tree implementation, and (ii) an implementation using a relational database.

An efficient main memory implementation of binary trees has been proposed in [BCG05] for solving the approximate k-nearest neighbor problem. To reduce the amount of required memory, the binary trees are represented as PARTICIA tries [Mor68]. PARTICIA representation reduces the memory requirements by replacing long paths in the tree with a single node, which represent these paths. This compression technique makes the number of tree nodes linear to the number of resources stored in it. Since there are l trees, the total memory requirements will be $O(n * l)$, where n is the number of indexed resources.

Several other binary tree disk-based implementations were also considered and discussed in [BCG05]. For the algorithm, there is no benefit using one of these implementations compared to a relational database. In fact, commercial relational databases build their indices using some of these implementations, therefore we do not need to reimplement them from scratch for the algorithm. We instead use them indirectly via the relational database, and thereby benefit from efficient generic database optimizations, e.g., caching.

Parameter Configuration

For running a possible duplicate query, the user first selects the minimum similarity between two resources for considering them as possible duplicates. Additionally, for each query, the user may choose the trade-off between the completeness of the results and the system's efficiency, expressed with probabilistic guarantees. In this section, we show how given an existing inverted index with l trees, and labels of total length k, the algorithm decides on the length of the query labels k', which will satisfy the probabilistic guarantees requested from the user at the minimum cost.

Clearly, there is a trade-off between the query cost and the quality of the results. Query cost includes the cost of retrieving all documents that have common labels with the query, and the cost of comparing all retrieved documents with the query, for filtering out the false positives and keeping only the true possible duplicates. This cost is influenced from the length of the labels k: when k is small the number of false positives increases. For example, in the extreme case where $k = 1$, half of the documents would be retrieved from each tree for a query. On the other hand, when k is large, the cost decreases, but the probability that each possible duplicate will be detected is also reduced. Since k is fixed when the inverted index is first initialized, the algorithm changes the length of the query labels k' instead. Based on the required probabilistic guarantees and the minimum similarity for considering two resources as possible duplicates, the algorithm finds the optimal k' that satisfies the required probabilistic guarantees. The optimal k' for the given index is the maximum value which satisfies the required probabilistic guarantees.

5.2.5 Experimental Evaluation

In this section, we describe a prototype implementation that uses the algorithm to identify possible duplicate news articles. We then report the results of our experimental evaluation using the news articles collected by our prototype application.

Prototype Implementation

To test our approach on a real-world scenario, we consider a news aggregation service, which aims at providing a unified view over the articles published on the Web by various news agencies, identifying and grouping together all possible duplicate articles. In particular, we have implemented a prototype in Java 1.6 that uses the algorithm to index the RDF representations extracted from incoming news articles and to entity linkages.

The application operates on a large collection of RDF data extracted from real-world news articles. In particular, we crawl news articles from the Google News Web site, which links to articles from various news agencies, such as BBC, Reuters, and CNN. For each newly added news article, we use the *OpenCalais* Web service [Ope] to extract the RDF statements describing the information available in it[7]. OpenCalais analyzes the text of the news articles and identifies entities described in this text, such as people, locations, organizations, and events, providing an RDF representation of the information in the article.

For the implementation of the binary trees required for indexing the RDF representations of the articles, there are two alternatives that can be used: (a) a main memory binary tree implementation, or (b) an implementation on secondary storage, e.g. a relational database. An efficient main memory implementation of binary trees has been presented in [BCG05] for solving the approximate k-nearest neighbor problem. The binary trees are represented as PATRICIA tries [Mor68], which reduces the amount of required memory by replacing long paths in the tree with a single node representing these paths. This compression technique makes the number of tree nodes linear to the number of resources stored in it. In our case, since there are l trees, the total memory requirements will be $O(n \times l)$, where n is the number of indexed resources. However, although accessing the main memory is much faster compared to secondary storage, this approach is limited by the capacity of main memory, and hence it is not suitable for a large number of RDF resources.

Hence, in our implementation, we have used a relational database, in particular MySQL 5, to efficiently store and retrieve all the resources with a given label. The resources are stored in a relational table $\mathcal{I}$ as tuples of the form ($resource\_id$, $tree\_id$, $hash\_value$). The algorithm needs to find all labels that share the same prefix of length k' with the query, where $k' \leq k$. This can be efficiently executed in a relational database using SQL operators, e.g., the $LIKE$ operator in MySQL. Hence, all the resources with prefix v from the tree t can be retrieved using the following expression:

$$\pi_{resource\_id}(\sigma_{tree\_id=t \text{ and } hash\_value \text{ LIKE } 'v\%'}(\mathcal{I}))$$

The size of the database is $O(n \times l)$, where n is the number of resources, and

[7]The RDF schema for the Web service output is available at:
http://www.opencalais.com/documentation/calais-web-service-api

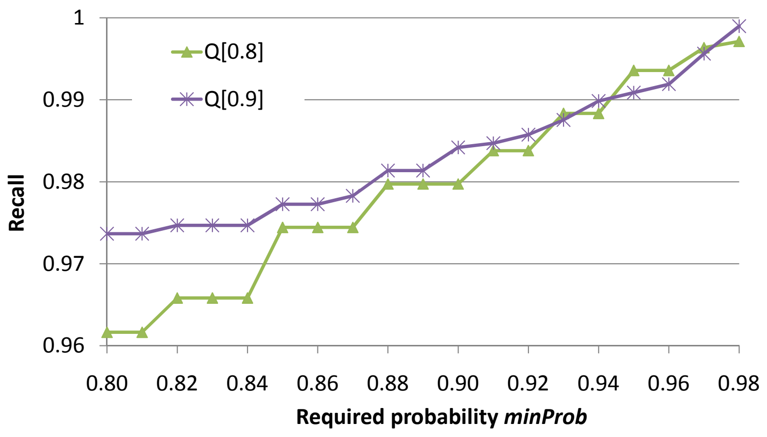


Figure 5.10: Probabilistic guarantees vs. recall.

the complexity of querying is $O(\log(n))$ per tree, i.e.,$O(l \times \log(n))$ in total.

Upon receiving a keyword query, the application identifies the news articles containing these keywords. Then, for each of the found news articles, it retrieves its possible duplicates. Based on the possible duplicates, it groups the news articles and it returns these groups as the answer to the query. In addition, for each group, we also generate a data cloud that summarizes the entities found in these news articles, taking into consideration the frequency of appearance of these entities in the articles.

Evaluation Results

The purpose of the experiments was to evaluate the algorithm with respect to quality and efficiency, for executing queries for possible duplicate resources. Efficiency was measured as the average time required to execute each query, and quality was measured with recall, i.e., the number of possible duplicates detected, divided by the number of total possible duplicates in the repository. Note that precision is always 1, since RDFsim includes a filtering step that filters out false positives, as described in Section 5.2.4. All the experiments were executed on a server using 1 Gb RAM and one Intel Xeon 2.8GHz processor.

As testbed, we have used the prototype described in Section 5.2.5. The data set consisted of 94.829 news articles, with a total of 2.711.217 entities, described as RDF statements[8], and it was stored in a MySQL 5 database, residing at the same

[8]http://out.l3s.uni-hannover.de:8898/rdfsim/data.html

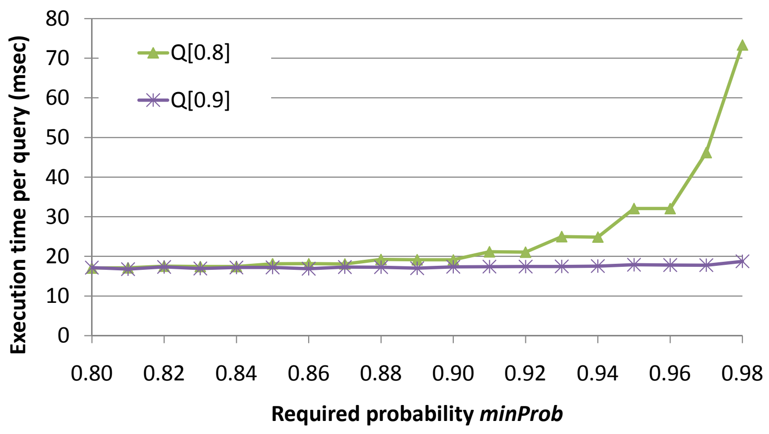


Figure 5.11: Probabilistic guarantees vs. average query execution time.

machine.

We indexed all the news articles using 20 binary trees ($l = 20$), and labels of length 50 ($k = 50$). The ground truth for the experiments was constructed by applying an exhaustive search to detect all pairs of articles that have pairwise similarity above a threshold $minSim$. With $Q[minSim]$, we denote the set of resources that have at least one possible duplicate for the threshold $minSim$. For each article in $Q[minSim]$, we detected the possible duplicate articles. All queries were repeated for different required probabilistic guarantees, expressed as the minimum probability $minProb$ that each possible duplicate article with the query will be returned, with $minProb \in [0.8, 0.98]$.

Figure 5.10 plots the average recall for the queries, for $minSim = 0.8$ and $minSim = 0.9$. As expected, recall increases with the required probability $minProb$. This is due to the fact that when $minProb$ increases, the algorithm chooses a smaller length k' for the prefixes of the query labels (see Section 5.2.4), and thereby the query retrieves a larger number of candidates. However, it is not necessary to set $minProb$ to very high values in order to get high recall; for our dataset, a value of $minProb = 0.9$ already results in recall over 0.98, which satisfies the practical requirements for most applications.

We also note that the recall is always higher than the value of $minProb$, which verifies that the probabilistic guarantees of the algorithm, described in Section 5.2.4, are always satisfied. In fact, the difference between the actual recall and the expected recall (the recall guaranteed by $minProb$) is notable, especially for low $minProb$ values. This happens because $minProb$ controls the probability that

Table 5.3: Values of k' for different combinations of similarity and probability.

minSim/ minProb	0.80	0.82	0.84	0.86	0.88	0.90	0.92	0.94	0.96	0.98
0.8	24	23	23	22	21	21	20	19	18	16
0.9	49	48	47	46	44	43	41	39	37	33

each possible duplicate will be retrieved, under the assumption that all possible duplicates have similarity $minSim$ with the query. However, in practice most of the possible duplicates have similarity higher than $minSim$. Therefore, the individual probability that these possible duplicates are retrieved ends up to be higher than $minProb$, and the overall quality of the results is better than the one expected according to the value of $minProb$.

With respect to efficiency, Figure 5.11 shows the average execution time per query, for varying $minProb$ values. The measured time includes the total time required to answer the query, i.e., generating the labels for the query, detecting and retrieving the candidate possible duplicates, and comparing all retrieved possible duplicates with the query to filter out the false positives. We see that for all configurations, the average execution time is small, always below 100 msec per query. Note that, if exhaustive comparisons are used instead for detecting the possible duplicate resources, the time required is around 1 minute per query.

We also see that the average execution time for the queries in $Q[0.9]$ is always less than the corresponding time for the queries in $Q[0.8]$. This is due to the effect of the similarity threshold $minSim$ on k': for a higher $minSim$ value, the algorithm can choose a higher value for k', thereby avoiding many false positives and reducing the execution cost significantly. For example, for $minProb = 0.8$, the algorithm sets k' to 49 for $minSim = 0.9$, whereas the corresponding k' value for $minSim = 0.8$ is only 24. Table 5.3 shows the different combinations of the values of these parameters.

As expected, the execution time increases as the requested probability $minProb$ increases. This is also due to the lower k' value chosen by the algorithm for answering queries with higher $minProb$ values. This effect is more noticeable for $minSim = 0.8$, since the lower $minSim$ value causes an additional reduction to k', and increases the false positives significantly. For $minSim = 0.9$, the effect of increasing the probabilistic guarantees $minProb$ is not so noticeable because the value of k' remains high, i.e., $k' \geq 33$, and therefore the algorithm does not retrieve many false positives. However, even for queries with very high requirements, e.g., $minProb = 0.98$ and $minSim = 0.8$, the execution time is less than 80 msec per query. Summarizing, the experimental results verify the probabilistic guarantees offered by the algorithm and confirm the effectiveness of the algorithm for detecting possible duplicate resources in large RDF repositories in real-time and for configurable requirements.

5.3 Summary

We addressed the problem of identifying and linking heterogeneous entities referring to the same real world objects. The first technique algorithm uses a Bayesian network to explicitly model evidences supporting possible matches between different references, along with interconnections between these matches. The algorithm runs incrementally and does not modify existing data. The second technique focuses on dealing with data from the Semantic Web. It utilizes the RDF representations of resources to detect possible linkages by taking into consideration the semantics and structure in the entity descriptions. For efficient processing it employs an index using LSH which allows to avoid the need for a large number of pairwise similarity computations. We provided a probabilistic analysis that allows to configure the algorithm according to specific quality requirements of users or applications. Our evaluations showed that both techniques successfully achieves our goal of efficiently and effectively linking data in heterogeneous information spaces.

Chapter 6

Entity Aggregation Framework

This chapter introduces an integration infrastructure for entity-based aggregation of data from various sources. As explained in the previous chapters, the plethora of existing entity linkage techniques suggests the inability of a single matching technique to globally address the entity linkage problem [EIV07]. To address this specific aspect of the problem, our infrastructure incorporates an extensible set of matching modules, each based on a different matching technique. This provides the ability to address the entity linkage problem over a large domain of data characteristics.

More specifically, the introduced infrastructure is equipped with an Entity-Name-System component (ENS) that acts as a repository for entities. ENS consists of an entity store that allows searching over a large number of entities, and a matching framework that is responsible for the selection of the most appropriate matching technique, or combination of the probabilistic entity linkages resulted from more than one matching modules. The architecture of the aggregator is illustrated in Figure 6.1. Its core components are the matching, the merging, and the aggregation support. The following paragraphs describe these components.

Matching Component. The main functionality of this component is to efficiently and effectively match an entity against the entities stored in the ENS repository. Such a match denotes that the entity in the given query and the entity from the ENS repository describe the same real world object. The details of the matching component are provided in Sections 6.1 and 6.2.

Merging Component. If a match to an entity is found by the matching component in the ENS repository of the aggregator, the entity search query must be merged with the entity found in the ENS repository. The merging component

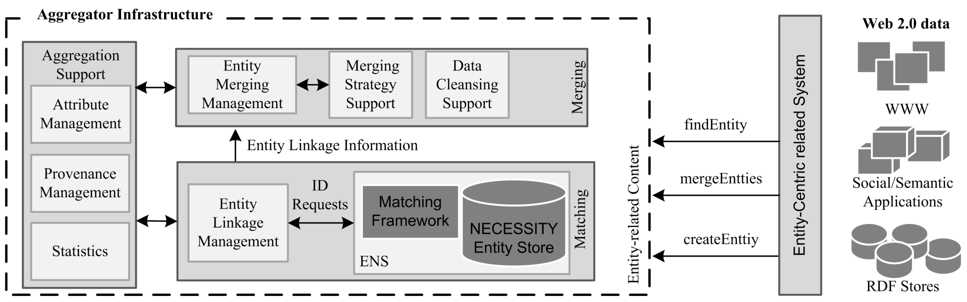


Figure 6.1: The architecture of the aggregator.

is responsible for this task, and it involves decisions on what will be the set of attributes and alternative identifiers of the merged entity.

Aggregator Support Component. This component of the aggregator's infrastructure includes services for managing knowledge and statistics about attribute names, as a mechanism to alleviate the absence of complete and uniform schema information. Such knowledge could for example be used to detect conflicts and inconsistencies in attribute values from the entities, and thus, can be used both in the merging and the matching component. Furthermore, it contains services for storing and assessing confidence information about the sources that can be used, for instance, in the merging component to assist in deciding which attributes to keep in the merged entity.

In addition, an aggregator has also an *Entity-Centric related System* that makes the requests with the entity descriptions collected from various sources. For example, this can be a system that uses extracted information from WWW web-pages, with one of the many currently available information extraction tools, such as *Cogito*[1] and *OpenCalais*[2]. This system can be also retrieving information from other applications through mashup services, such as *Yahoo! Pipes*[3], or through other kinds of source wrappers.

The entity-centric related system poses function requests to the aggregator infrastructure. In the case of *findEntity* function this is an entity search query providing the attributes of the entity in request, as explained in Chapter 3. This query is first given to the matching component, which retrieves the set of entities from the

[1]http://www.expertsystem.net/page.asp?id=1515&idd=200
[2]http://www.opencalais.com
[3]http://pipes.yahoo.com/pipes/

ENS repository that are found to possibly match the specific query.

The matching component relies on the functionality of the ENS. There exist two possibilities for the ENS: (i) a local ENS service that is developed exclusively for the aggregator, and (ii) a global ENS public service that already exists, such as the one offered by the OKKAM system[4]. The latter has the benefit of enabling global, cross-system, re-usability of entity identifiers, which can trigger a network effect for easing entity-based integration, especially when the ENS is already in use by other systems.

The output of the matching component is sent to the merging component, which continues in merging together the attributes of the entity found to match in the ENS with the entity described in the query that was provided as an input to the matching component. Merging expands the ENS entity with the information from the query. The set of alternative identifiers is enhanced with the identifier and any alternative identifiers of the entity from the query. In addition, the set of attributes is updated with the information from the query's attributes. The default approach for this is to take the union of all the available attributes in the two entities. More complex solutions (as also explained in Section 4.4) are also supported in order to address: (i) updates in the entity status, (ii) different levels of confidence of the sources, (iii) conflicting attributes, (iv) freshness, and (v) inconsistencies.

Consider the challenges of integrating Web 2.0 data. Such data do not have a fixed schema and may consist of name-value attributes, attributes with only values, or a mix of both. Furthermore, the data given for integration may contain only one entity, or a collection of entities that are somehow related to each other, such as those present in the repositories of Social applications. Despite the many existing results in entity linkage techniques (i.e., as discussed in Chapter 2), no solution has been found to work for all situations.

The aggregator leverages this problem by using a generic methodology for matching that incorporates an extendable set of matching modules, each focusing on addressing a specific situation. Matching queries with entities is performed through a series of steps that involve the retrieval of a small set of matching candidates from the potentially very large entity store, and the further processing of these candidates either by the most promising module given the specific query data (i.e., matching technique), or by a combination of different modules. The infrastructure for entity-based aggregation was evaluated in respect to the idea of using various matching techniques and scalability of the entity store. Section 6.1 explains

[4]http://fp7.okkam.org/

the entity store component, and Section 6.2 the matching framework. Section 6.3 reports the results of the experimental evaluation.

6.1 NECESSITY Entity Store

For an entity store we introduce and use NECESSITY, which provides a repository of entities along with an index for efficient entity retrieval. The reason for including an entity store in our framework is to reduce the set of entity candidates in order to provide less data to the matching modules, since the matching algorithms are typically performing a lot of heavy operations.

The NECESSITY entity store is essentially a search engine which returns a ranked list of entities that are relevant to the specified query. It is composed of two main parts, as shown in Figure 6.2. The first part is implemented as a key-value Voldemort repository[5] and it is able to maintain a large number of entities. Scalability in Voldemort is obtained by simply adding more servers when more entities are added to the repository, which means that the remaining components of the aggregator are left untouched. Moreover, this key-value repository supports linear scalability since it is designed from scratch to scale by simply adding new nodes.

The second part of NECESSITY is the inverted index, implemented as a Solr Brocker[6], which uses Lucene[7] for full-text indexing and search. When a new query arrives, the Solr broker will assign the query to some of its shards, collect their $top - k$ entities, and then aggregate them to identify the k best entities. The resulting entities are then returned to the matching framework and correspond to the candidate entities that could be a match with the given user/application query.

6.2 Matching Framework

The matching framework is responsible for receiving the queries requesting an entity, and controlling the evaluation flow. Figure 6.2 illustrates the main components of the matching framework.

Given a request describing an entity, the matching component invokes the entity search planner. This first analyses the given entity request to generate an initial query for the entity store, and identifies the matching module or modules that could

[5] http://project-voldemort.com/
[6] http://lucene.apache.org/solr/
[7] http://lucene.apache.org/

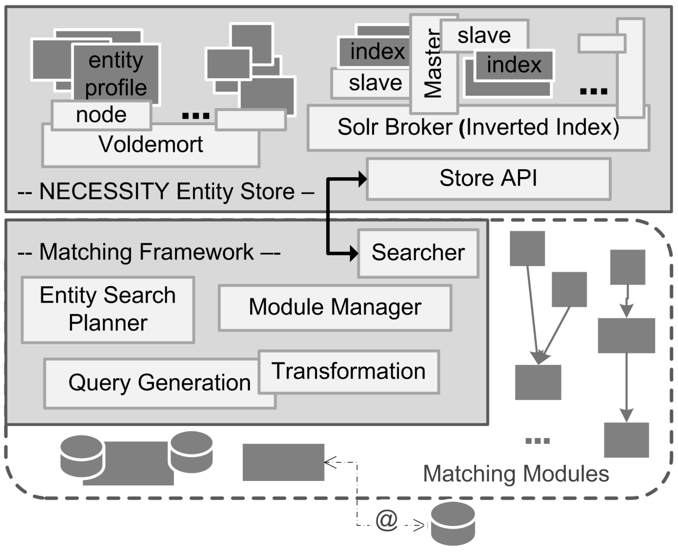


Figure 6.2: Matching queries with entities is performed through a series of processes, and may involve either the most promising module (i.e., matching technique) for the given query, or a combination of different matching modules.

process it. The initial query is then revised by the selected matching module(s) and send to the entity store. The store processes the query and returns a small set of entity candidates. These candidate entities are then given to the module(s) for performing matching and identifying the entity that corresponds to the given query. The following paragraphs provide the details for the main parts involved in this process.

Matching Modules

Individual matching modules implement their own methods for matching queries with entities. Naturally, the algorithm of each module will focus on a specific matching task. For example, we can have matching modules focusing on entities that do not contain attribute names, or for entities that contain inner-relationships. As shown in Figure 6.2, modules may also use a local database for storing their internal data, or even communicate with external sources for retrieving information useful for their matching algorithm.

In addition to the individual modules, the matching framework can also contain modules that do not compute matches directly, but by combining the results of other modules. The current version of aggregators can handle the following two types of combination modules:

- Sequential Process: The matching module invokes other modules in a sequence, and to each module it gives the results of the previously invoked modules. Thus, each module refines the entity matches it received. The resulting entity matches are the ones returned by the last module.

- Parallel Process: The matching module invokes a set of matching modules at the same time. Each module returns its entity matches (i.e., entity linkages), and thus this combination module needs to combine their results.

To be able to choose the right modules for each query, the matching framework maintains the profiles of the modules. These profiles contain not only the module description and classification, but also information on their matching capabilities. For example, the average time required for processing queries, and the query formats that they can handle.

Both entity linkage techniques presented in Chapter 5 are possible matching modules and can be used in the the matching framework of the entity-based aggregation infrastructure. As explained, these techniques generate probabilistic entity linkage information during the processing of new entities that are given to the infrastructure. This information is maintained in the *Entity Linkage Management* component.

Query Generation

The entity search planner needs to generate a query for the storage. More specifically, for the NECESSITY entity store this corresponds to a Lucene query. Since the entity store offers very efficient but restricted search functionality, this step might also require the generation of more that one queries, with the final entity candidate list being the merging of the entity candidates returned by the entity store for all generated queries.

As shown in Algorithm 6.1, the query can be enhanced and refined by the matching modules according to their needs. This might involve query transformations on the schema level, for adapting from attributes used by the user/applications to attributes available in the repository, or to include attribute name alternatives, or to relax the query to the most frequent naming variants.

Algorithm 6.1: Evaluation of entity search queries.

```
    Input:   Query $Q_e := \{A\}$
    Output: Entities E := $\{\langle id_1, A_1\rangle, \ldots, \langle id_\kappa, A_\kappa\rangle\}$
 1  $Q_s \leftarrow$ generateStoreQuery($Q_e$) ;
 2  $MM \leftarrow$ selectMatchingModule($Q_s$) ;
 3  $Q'_s \leftarrow MM$.updateQuery($Q_s$) ;
 4  $MC \leftarrow$ retrieveEntityCandidates($Q'_s$) ;
    /* ability here to also relax query - if needed (e.g.,
       when |$MC$|=0)                                          */
 5  for $candidate \in MC$ do
 6      score $\leftarrow$ MM.match($Q_e$, candidate) ;
        /* a module has the capability to also invoke
           additional modules                                  */
 7      if acceptable($Q_e$, candidate, score) then
 8          R.add(⟨ candidate, score⟩);
 9      end
10  end
11  orderList(R); // based on score;
12  return $R$
```

Module Section & Combination

The planner needs to select the best suitable matching module to perform the entity matching for the given entity. Different options for module selection are possible:

- The aggregator explicitly picks a matching module, a selection which for example is based on previous experience or because it has the knowledge that a specialized matching module is more effective when integrating the data of a specific Social application.

- The matching module is selected based on information in the entities to be integrated. This may include requirements with respect to performance or supported entity types.

- The module is selected based on an analysis of the data in the entity to be integrated, for example existence or not of attribute names.

The entity-aware query processing presented in Chapter 4 is a possible mechanism for the combination of matching module results and can be used in the matching framework of the entity-based aggregation infrastructure. In this case, the query processing of the infrastructure will be using the probabilistic linkage information from the Entity Linkage Management component, which are stored by the individual matching modules.

6.3 Experimental Evaluation

This section presents the results of our experimental evaluation. The evaluation was performed using a JAVA 1.6 prototype of the entity-based aggregator (Section 6.3.1). The focus our experiments was to evaluate the effectiveness of the aggregator (Section 6.3.2) and the scalability of the entity store (Section 6.3.3).

6.3.1 Prototype's modules, section, and combination

With our current aggregator's implementation we aim in effectively handling entity queries coming mainly from information extractors. For this we include two modules, namely "Group Linkage" and "Eureka".

The currently embedded module selection is rule-based, and proceeds as follows: We process and analyze the given query. If the query specifies a module or the entity type correspond to a specific module, then this module is selected. We then check whether the query contains attribute information, or it is composed only by keywords. For the first case we select the "Eureka Module", and for the latter the "Group Linkage". This selection criteria was concluded by a series of evaluation using these two modules.

Group Linkage

This module adopts to our aggregator the algorithm suggested in [OKLS07]. According to this algorithm, an entity matches a candidate when we detect a large fraction of similarities between predicates from the entity with attribute value pairs from the matching candidates. To use this algorithm we consider the given entity Q and the matching candidate C as the two entities. The matching probability between entity Q and matching candidate C is given by:

$$MP(Q,C) = \frac{\sum\limits_{\forall a_i \in Q \forall a_j \in C} \begin{cases} sim(a_i, a_j) & if \;\; sim > t \\ 0 & otherwise \end{cases}}{|Q| + |C| - matched\_pairs,} \tag{6.1}$$

where $|C|$ gives the number of name-value attribute pairs in the matching candidate, $|Q|$ the number of predicates in the entity, and *matched_pairs* the number of

Table 6.1: A small fragment of the entity queries from Historical Records.

Historical Records
Don Henderson British actor
Jacques-Yves Cousteau French explorer
Fred Zinnemann Austrianborn director
John Carew Eccles Australian neurophysiologist recipient of the Nobel Prize in Physiology or Medicine
George Wald American scientist recipient of the Nobel Prize in Physiology or Medicine

$sim(a_i, a_j)$ higher that threshold t.

Eureka Matching

The matching algorithm of the Eureka module computes the overlap of the predicates in the query with the attributes of the matching candidates. As an initialization step, the algorithm creates a small local inverted index as follows: Each term (i.e., word) in the values from the name-value pairs composing the query become keys in a hash table. We then process the information in each matching candidate and when we identify a candidate containing one of these values, we add the candidate's identifier with the attribute values to the list of entities of the corresponding key. The score $MP(Q, C)$ between entity Q and matching candidate C is computed by:

$$MP(Q,C) = \sum_{\forall a_1 \in Q,\, \forall a_2 \in C} \begin{cases} 1 \times importance(a_1.name), \\ \quad if \;\; a_1.name = a_2.name \;\&\; a_1.value \in a_2.value \\ 0.5 \times importance(a_1.name), \\ \quad if \;\; a_1.name = null \;\&\; a_1.name \in a_2.name \end{cases} \tag{6.2}$$

where *importance* is a weight that reflects the importance of a specific attribute name, e.g., *name* is more important than attribute *data* for entity e_5 of Figure 1.2.

6.3.2 Effectiveness of Aggregation

Datasets

Many applications typically contain local lists of entities, for example members of organizations, or historic records in online newspapers[8]. The *Historical Records* dataset is composed of such a list of entities. In particular, we use the list maintained by Wikipedia for two reasons: (i) it is large, and (ii) the Wikipedia URLs allow us to retrieve the ground truth for our evaluation.

We crawled all available historical records from Wikipedia and used the collected data about the entities directly in the integration process, i.e., no extraction process. A small fragment of these entity queries are shown in Table 6.1. Our process resulted in $3,381$ entities, out of which 2,202 were in the ENS and $1,179$ not in the ENS. In our experiments we used all these entities.

The second dataset is based on the data from Blogosphere, which is one of the most popular sources of Web 2.0 data. There exist applications that focus on monitoring new entities of *Web blogs* for creating new services. To evaluate the capabilities of our aggregation approach on this data, we crawled entities from a few blogs[9] and used the OpenCalais extractor to retrieve the described entities. A small fragment of these entity queries is shown in Table 6.2.

This process resulted to a total of $19,426$ entities. We randomly selected some entities from this dataset and manually identified their corresponding aggregator's identifiers. These identifiers were used to automatically evaluate the decisions of the entity-based integration. For this experiment we detected 200 entities that already existed in the ENS, and another 200 that were new entities.

Methodology

We performed our evaluation under the following scenario: given a system which uses our aggregation process with an already sufficiently large entity store, we integrate the entities from applications and sources described above.

As an existing large store we used the entities extracted and stored in the entity repository of the OKKAM project. This provided the entity store with $6,865,392$ entities, which included people and organizations from Wikipedia[10], and a large

[8]Examples: `http://en.wikipedia.org/wiki/January_1/`, and `http://www.bbc.co.uk/history/historic_figures/`
[9]Example: `http://www.barackoblogger.com/`
[10]`http://www.wikipedia.org/`

Table 6.2: A small fragment of the entity queries from Web Blogs.

Web Blogs
name="John Quincy Adams"
name="Barbara A. McKinzie"
name="Ted Kennedy Speech"
name="Theola Labb??-DeBose"
name="Geoff McFetridge"
name="Hillary Rodham Clinton"
name="Shepard Fairey-Yes We"

amount of geographical entities (e.g., describing countries, cities, roads, mountains, rivers) extracted from GeoNames[11]. These entities were extracted using Cogito and follow the entity representation model described in [BCPS08], which —in its simplest form— corresponds to a set of name-value attributes and conforms to the entity model described in Chapter 3.

For creating a ground truth for the evaluation, the entities we wanted to integrate were equipped with the corresponding entity identifier used within the entity store (and the aggregator), if the specific entity was already described/integrated in the data of the aggregator. For performing the evaluation, we posed requests with each of the queries from the datasets using the functionality of the aggregator's matching component, and checked whether the returned identifier was the one given by the ground truth (Case: Existing Entity). In case the specific entity was not yet available in the aggregator (Case: New Entity), there should be no identifier returned by the system.

We measured the overall accuracy of integration decisions as the sum of correct "Creation decisions" (when a new entity was encountered) and correct "Merge decisions" (when an existing entity was encountered), divided by the total number of decisions.

We measure accuracy for two possible system configurations: (1) automatic, and (2) semi-automatic. A system configured with the automatic configuration proceeds to entity creations and merges automatically, without requiring feedback from the user. A system configured to semi-automatic proceeds to entity creations and entity merges only when it has strong evidence for them. When the evidence is weaker, the system provides up to three entities, which could match the requested entity and asks the user to make the final decision.

[11]http://www.geonames.org/

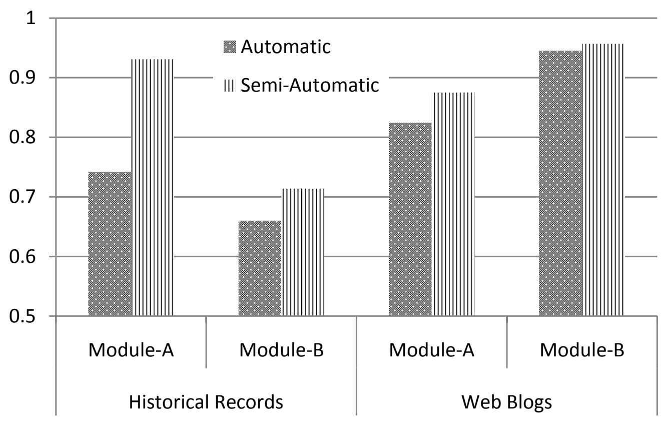


Figure 6.3: Accuracy of matching modules for two different entity datasets.

Comparison of Modules Effectiveness

We integrated the entities from datasets "Historical Records" and "Web Blogs" using our matching modules. Figure 6.3 shows the overall accuracy for both the modules, corresponding to an automatic and a semi-automatic configuration.

As we can see from Figure 6.3, on the "Historical Records" dataset the accuracy of module-A is higher than the accuracy of module-B. On the "Web Blogs" dataset we observe the opposite behavior, that the accuracy of module-B is higher than the accuracy of module-A. Considering again the differences between the entity descriptions composing these datasets (Tables 6.1 and 6.2), confirms that there is no single module (i.e., algorithm) that is able to perform the best matching for all possible descriptions - a statement that is also reflected by the plethora of matching approaches already suggested in the literature. The ability to incorporate various algorithms enables our aggregation process to effectively perform entity search over various data formats coming from various sources and applications.

6.3.3 System Scalability

Entity Dataset

In order to show the scalability of the entity store of our aggregator, we needed a dataset that would be large and also composed of real data. We generated such a dataset using the Census[12] database. Using this synthetic database, we generated

[12] http://www.census.gov/genealogy/names/

75 million person entities which follow the distribution of frequently occurring first names and surnames of US citizen from the 1990. Each entity was describing a citizen and was composed by the following attribute names: first name, last name, age, gender, language, and city.

Methodology

Our large-scale experiment was conducted using eight servers of Intel Core2 Quad CPU@2.40GHz with 8GB RAM. The installation of the aggregator included two Solr shards and four Voldemort nodes. Two additional machines were used as clients.

We added the entities from this dataset into the aggregator, and thus indexed and stored them in the storage layer. We then evaluated our system at different storage sizes, and more precisely when the aggregator's store contained 25 million, 50 million, and 75 million entities. For each evaluation, we had several clients (i.e., simulating users) that posed queries on the system. Each client performed ten sets of queries with 100 to 1000 queries per set, i.e, a step of 100 for every request.

Evaluation Results

We used the above methodology to study the behavior of the aggregator when the number concurrent users increases. Figure 6.4 shows the number of requests per seconds, averaged over the ten sets, which correspond to the number of the concurrent users. The figure contains three series, one for each of the considered storage sizes, i.e., 25, 50, and 75 Millions. Each experiment was reproduced 5 times, using random and unique queries.

As we can see from the results, even with a large number of entities in the aggregator, the entity store scales almost linearly with the increase of concurrent clients. For example, in the case of 25 Millions entities indexed and stored, 163 requests per second are processed when 10 clients are concurrently querying the entity store. Because of the good scalability properties of our architecture, the total throughput increases with the number of concurrent clients: 255 requests per second are processed when 40 clients make concurrent queries. Following the typical behavior of indexing, the average throughput decreases when the index size gets larger. So, while the time to retrieve an entity from Voldemort remains constant, independent of the size of the store, the query processing time at the Solr shards is directly affected by the size of the index.

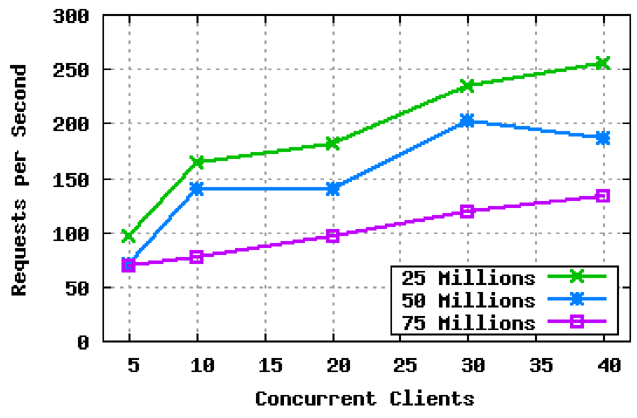


Figure 6.4: Average request throughput of the entity store with concurrent users.

6.4 Summary

In this chapter we presented a new approach for enabling aggregators to perform entity-based integration, leading to more efficient and effective integration for data from various sources and applications, such as Social and Semantic Web systems. In particular we equipped aggregators with an Entity-Name-System, which offers storage and matching functionality for entities. The matching functionality is based on a generic framework that allows incorporating and combining of an expandable set of matching modules. The entity aggregation framework follows the data model introduced in Chapter 3, and can incorporate the entity linkages techniques explained in Chapters 4 and 5.

Chapter 7

Conclusions

7.1 Summary

This dissertation introduced a new methodology to address the entity linkage problem for information spaces created by combining data from various applications. More specifically, the focus was on handling collections with heterogeneous, uncertain, and volatile data. The following paragraphs summarize the main contributions and explain how these overcome the challenges of such data collections, as these are explained in Section 1.4.

We propose a generic data model for entities and linkages between entities. The model is able to capture and represent the highly heterogeneous data as these appear in several applications, for example Web data, and results from data extractors. The model is probabilistic, enabling the incorporation of the uncertainty on the entity data and on the linkages. Relying on this model, we proposed an advanced entity-based query mechanism that exploits linkage information and uncertainty for retrieving entities. The query processing does not maintain the merged entities but operates directly over the linkage information, thereby enabling the update of the linkage information when new data is integrated. In addition, we described how the probabilistic entity linkages can be detected and generated, and how they address the data heterogeneity. The introduced entity linkage techniques are incremental, focusing on volatile data and satisfying the requirements of different applications. An additional contribution is the entity aggregation framework, which explains how the introduced techniques are incorporated into a single extensible framework for integrating and searching for entities.

7.2 Ongoing and Future Work

The following paragraphs describe and discuss interesting aspects of ongoing work and future research.

Scaling Entity Linkage to Large Collections

We have recently witnessed an enormous growth in the volume of structured and semi-structured data sets available on the Web. One methodology to address the entity linkage problem in such large volumes of data is through blocking techniques, i.e., by separating the data into blocks and comparing only the data inside each block. The high dynamics, the loose schema binding, and the heterogeneity of (semi-)structured Web data, impose new challenges for generating and processing blocks. Existing blocking approaches become inapplicable because they rely on the homogeneity of the considered data and on a-priory known schemata. The challenge here is therefore to create methods that can scale to large, noisy, and heterogeneous data collections. We are currently investigating the possibility to combine an attribute-agnostic mechanism for building blocks with intelligent block processing techniques that boost blocks with high expected utility, propagate knowledge about identified matches between entities, and preempt the process when it becomes too expensive. The initial results, reported in [PINF11], show that this approach is both effective and efficient.

Provenance for Entity Linkage

From the time that the data are given for integration until the time they are returned as the result of a query, a number of *functions/transformations* are applied. This includes the processing of the data for integration purposes, the generation of the entity related information (from the linkage techniques), and the usage of the data during query processing. In addition to the successful final results, many tasks also require to know the origin of the data, namely the provenance information. Provenance for our situation should provide information on two levels. The first is provenance for the original heterogeneous data, such as the source that provided the data. The second level is for the resulted entity-aware data. For the latter, we consider information such as the data used for a specific entity representation, or the considerations that lead to a specific entity linkage.

Considering the above levels, we are planning to work on extending our model for including the provenance information. Consequently, an extension of the query language to allow querying for provenance and to adjust query processing is also required.

Incremental and Adaptive Entity Linkage Index

As explained in this dissertation, there are two main methodologies for addressing the entity linkage problem. The first methodology performs data processing off-line in order to have the merged entities readily available at query time. The second methodology maintains the probabilistic linkage information and during query processing use them on-the-fly to decide and compile the merged entities. We are currently investigating the possibility of having a technique in between the two methodologies, combining the best of their aspects. For instance, having the merged entities readily available at query time is beneficial for entities that are highly popular and frequently requested. On the other hand, for entities requested very rarely, or even never, we do not need to generate the merged entities or their entity linkages a-priori. We want to investigate an adaptive model which balances the information generated a-priori and on-the-fly by considering the popularity of each entity as well as its volatility.

Bibliography

[ABS+06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

[ACG02] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[AFM06] Periklis Andritsos, Ariel Fuxman, and Renée J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[AKO09] Lyublena Antova, Christoph Koch, and Dan Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB Journal*, 18(5):1021–1040, 2009.

[AMNR+06] Boanerges Aleman-Meza, Meenakshi Nagarajan, Cartic Ramakrishnan, Li Ding, Pranam Kolari, Amit P. Sheth, Ismailcem Budak Arpinar, Anupam Joshi, and Tim Finin. Semantic analytics on social networks: Experiences in addressing the problem of conflict of interest detection. In *WWW*, pages 407–416, 2006.

[AO05] Akiko N. Aizawa and Keizo Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, pages 30–39, 2005.

[AR07] Eytan Adar and Christopher Re. Managing uncertainty in social networks. *IEEE Data Engineering Bulletin*, pages 15–22, 2007.

[BCFM98] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.

[BCG05] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[BCPS08] Barbara Bazzanella, Junaid Ahsenali Chaudhry, Themis Palpanas, and Heiko Stoermer. Towards a general entity representation model. In *SWAP*, 2008.

[BG04a] Indrajit Bhattacharya and Lise Getoor. Deduplication and group detection using links. In *LinkKDD*, 2004.

[BG04b] Indrajit Bhattacharya and Lise Getoor. Iterative record linkage for cleaning and integration. In *DMKD*, pages 11–18, 2004.

[BGMM$^+$09] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.

[BM05] Ron Bekkerman and Andrew McCallum. Disambiguating web appearances of people in a social network. In *WWW*, pages 463–470, 2005.

[BMC$^+$03] Mikhail Bilenko, Raymond J. Mooney, William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[BNV07] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Inferring xml schema definitions from xml data. In *VLDB*, pages 998–1009, 2007.

[CGGM03] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD Conference*, pages 313–324, 2003.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 327–336, 2002.

[Coh00] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3):288–321, 2000.

[CR01] W. Cohen and J. Richman. Learning to match and cluster entity names. In *MF/IR Workshop co-located with SIGIR*, 2001.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb co-located with IJCAI*, pages 73–78, 2003.

[DH05] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

[DH07] Xin Dong and Alon Y. Halevy. Indexing dataspaces. In *SIGMOD Conference*, pages 43–54, 2007.

[DHM05] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[DHY07] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

[DJ03] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley, 2003.

[DKP$^+$09] Nilesh N. Dalvi, Ravi Kumar, Bo Pang, Raghu Ramakrishnan, Andrew Tomkins, Philip Bohannon, Sathiya Keerthi, and Srujana Merugu. A web of concepts. In *PODS*, pages 1–12, 2009.

[DLLH03] AnHai Doan, Ying Lu, Yoonkyong Lee, and Jiawei Han. Object matching for information integration: A profiler-based approach. In *IIWeb co-located with IJCAI*, pages 53–58, 2003.

[DS07a] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

[DS07b] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[DS07c] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[dVKCC09] Timothy de Vries, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[GD05] Lise Getoor and Christopher P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

[GIJ$^+$01] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 432–442, 1999.

[GM03] Ramanathan V. Guha and Rob McCool. TAP: a semantic web platform. *Computer Networks*, 42(5):557–577, 2003.

[GS06] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[HFM06] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[HS98] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[INN08] Ekaterini Ioannou, Claudia Niederée, and Wolfgang Nejdl. Probabilistic entity linkage for heterogeneous information spaces. In *CAiSE*, pages 556–570, 2008.

[INNV10] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.

[INNV11] Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis. LinkDB: A probabilistic linkage database system. In *SIGMOD Conference*, 2011.

[INV] Ekaterini Ioannou, Claudia Niederée, and Yannis Velegrakis. Searching web 2.0 data through entity-based aggregation. Technical Report.

[INV10] Ekaterini Ioannou, Claudia Niederée, and Yannis Velegrakis. Enabling entity-based aggregators for web 2.0 data. In *WWW*, pages 1119–1120, 2010.

[Ioa09] Ekaterini Ioannou. Entity-aware query processing for heterogeneous data with uncertainty and correlations. In *Joint EDBT/ICDT Ph.D. Workshop*, 2009.

[IPSN10] Ekaterini Ioannou, Odysseas Papapetrou, Dimitrios Skoutas, and Wolfgang Nejdl. Efficient semantic-aware detection of near duplicate resources. In *ESWC*, pages 136–150, 2010.

[ISB$^+$09] Ekaterini Ioannou, Saket Sathe, Nicolas Bonvin, Anshul Jain, Srikanth Bondalapati, Gleb Skobeltsyn, Claudia Niederée, and Zoltán Miklós. Entity search with NECESSITY. In *WebDB Workshop co-located with SIGMOD*, 2009.

[Jar89] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *American Statistical Association*, 84, 1989.

[Jen01] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[JLM03] Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, 2003.

[KM06] Dmitri V. Kalashnikov and Sharad Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2):716–767, 2006.

[KMC05] Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM SDM*, 2005.

[KMS04] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible string matching against large databases in practice. In *VLDB*, pages 1078–1086, 2004.

[KSS06] Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD Conference*, pages 802–803, 2006.

[LC08] Ming Zhong 0002, Mengchi Liu, and Qian Chen. Modeling heterogeneous data in dataspace. In *IRI*, pages 404–409, 2008.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[Lev66] VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[LMR05] Xin Li, Paul Morie, and Dan Roth. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine*, 26(1):45–58, 2005.

[MBB$^+$10] Zoltán Miklós, Nicolas Bonvin, Paolo Bouquet, Michele Catasta, Daniele Cordioli, Peter Fankhauser, Julien Gaugaz, Ekaterini Ioannou, Hristo Koshutanski, Antonio Maña, Claudia Niederée, Themis Palpanas, and Heiko Stoermer. From web data to entities and back. In *CAiSE*, pages 302–316, 2010.

[MJS07] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *WWW*, pages 141–150, 2007.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[Mor68] Donald R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 1968.

[MPC$^+$10] Enrico Minack, Raluca Paiu, Stefania Costache, Gianluca Demartini, Julien Gaugaz, Ekaterini Ioannou, Paul-Alexandru Chirita, and Wolfgang Nejdl. Leveraging personal metadata for desktop search: The Beagle^{++} system. *Journal of Web Semantics*, 8(1):37–54, 2010.

[MVB08] Alexis Morris, Yannis Velegrakis, and Paolo Bouquet. Entity identification on the semantic web. In *SWAP*, 2008.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[OKLS07] Byung-Won On, Nick Koudas, Dongwon Lee, and Divesh Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007.

[Ope] Open Calais. http://www.opencalais.com/.

[OS99] Aris M. Ouksel and Amit P. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5–12, 1999.

[PD04] Parag and P. Domingos. Multi-relational record linkage. In *MRDM Workshop co-located with KDD*, pages 31–48, 2004.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

[RDS07] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[RS08] Christopher Re and Dan Suciu. Managing probabilistic data with MystiQ: The can-do, the could-do, and the can't-do. In *SUM*, pages 5–18, 2008.

[RVMB09] Flavio Rizzolo, Yannis Velegrakis, John Mylopoulos, and Siarhei Bykau. Modeling concept evolution: A historical perspective. In *ER*, pages 331–345, 2009.

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[SD07] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[SI] Slawek Staworko and Ekaterini Ioannou. Management of inconsistencies in data integration. Chapter to be included in Dagstuhl Follow-up Series on Data Exchange, Integration, and Streams.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[SV07] Divesh Srivastava and Yannis Velegrakis. Intensional associations between data and metadata. In *SIGMOD Conference*, pages 401–412, 2007.

[TKM02] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.

[Vel08] Yannis Velegrakis. On the importance of updates in information integration and data exchange systems. In *DBISP2P*, 2008.

[Win99] William Winkler. The state of record linkage and current research problems, 1999.

[WMK$^+$09] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.