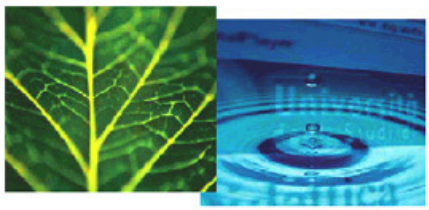
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DIT - University of Trento

EXPLOITING THE VOLATILE NATURE OF DATA AND INFORMATION IN EVOLVING REPOSITORIES AND SYSTEMS WITH USER GENERATED CONTENT

Siarhei Bykau

Advisor:

Prof. Yannis Velegrakis

Università degli Studi di Trento

April 2013

Dedicated to the memory of my mother.

Abstract

Modern technological advances have created a plethora of an extremely large, highly heterogeneous and distributed collection of datasets that are highly volatile. This volatile nature makes their understanding, integration and management a challenging task.
One of the first challenging issues is to create the right models that will capture not only the changes that have taken place on the values of the data but also the semantic evolution of the concepts that the data structures represent. Once this information has been captured, the right mechanisms should be put in place to enable the exploitation of the evolution information in query formulation, reasoning, answering and representation. Additionally, the continuously evolving nature of the data hinders the ability of determining the quality of the data that is observed at a specific moment, since there is a great deal of uncertainty on whether this information will remain as is. Finally, an important task in this context, known as information filtering, is to match a specific piece of information which is recently updated (or added) in a repository to a user or query at hand.
In this dissertation, we propose a novel framework to model and query data which have the explicit evolution relationships among concepts. As a query language we present an expressive evolution graph traversal query language which is tested on a number of real case scenarios: the history of Biotechnology, the corporate history of US companies and others. In turn, to support query evaluation we introduce an algorithm using the idea of finding Steiner trees on graphs which is capable of computing answers on-the-fly taking into account the evolution connections among concepts.
To address the problem of data quality in user generated repositories (e.g. Wikipedia) we present a novel algorithm which detects individual controversies by using the substitutions in the revision history of a content. The algorithm groups the disagreements between users by means of a context, i.e. the surrounding content, and by applying custom filters. In the extensive experimental evaluation we showed that the proposed ideas lead to high effectiveness on a various sources of controversies.
Finally, we exploit the problem of producing recommendations in evolving repositories by focusing on the cold start problem, i.e. when no or little past information about the users and/or items is given. In the dissertation we present a number of novel algorithms which cope with the cold-start by leveraging the item features using the k-neighbor classifier, Naive Bayes classifier and maximum entropy principle. The obtained results enable recommender systems to operate in rapidly updated domains such that news, university courses and social data.

Keywords
Evolving data, recommender systems, data quality.

Acknowledgements

I'd like to thank my advisor Prof. Yannis Velegrakis for his patience and continuous support. His guidance helped me not only with my PhD studies but also to learn important life lessons. Without a doubt, this dissertation would not have been possible unless Yannis' help, encouragement and optimism. It has been a great honor to be his first PhD student.

Throughout my four years of PhD study I was really fortunate to work with some brilliant people. It is a great pleasure to thank Prof. John Mylopoulos for his inspirational advices and suggestions, Flavio Rizzolo for his help in my early research, the Papyrus project team and especially Nadzeya Kiyavitskaya and Chrisa Tsinaraki for the interesting and productive collaboration, Georgia Koutrika for her guidance and help, Divesh Srivastava and Flip Corn for extremely important and valuable lessons on how to research and make things done.

I'd like to thank my friends who became my family in Italy: Ivan Tankoyeu, Viktoria Ambrushkevich, Heorhi Raik, Dina Shakirova, Liubou Zhauniarovich, Mikalai Tsytsarau, Maksim Khadkevish, Ludmila Polovinko. My special thanks go to Yury Zhauniarovich for his endless support and valuable advices.

Finally, I wish to express my love and gratitude to my beloved family and Maryna Hramkova for their understanding and endless love through all my studies.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Modern technological advances have created a plethora of an extremely large, highly heterogeneous and distributed collection of datasets that are highly volatile. This volatile nature makes their understanding, integration and management a challenging task.

In this section, we introduce the problems of modeling, querying and exploiting the volatile nature of data in evolving repositories and systems with user generated contents. Section 1.1 presents a context and motivation for that problems. Then, in Section 1.2 we discuss the key research problems and our contributions which are presented in this dissertation. Finally, the structure of the dissertation is given in Section 1.3.

1.1 Motivation

Modern information systems produce a tremendous amount of data which is highly heterogeneous and distributed. Moreover, such data is not static and undergoes many transformations on the way. To cope with the volatile nature of data one need to solve a number of problems. In the following we motivate the three problems addressed in the dissertation, namely how to model and query evolving data, how to improve the data quality of evolving user-generated repositories and how to personalize rapidly changing data.

Modeling and querying of data evolution. To capture not only the changes that have taken place on the values of the data but also the semantic evolution of the concepts that the data structures represent there is a need to create right models. Conceptual modeling languages – including the ER Model, UML class diagrams and Description Logics – are all founded on a notion of "entity" that represents a "thing" in the application domain. Although the state of an entity can change over its lifetime, entities themselves are atomic and immutable [Gregersen and Jensen, 1999, Soo, 1991, Buneman et al., 2002, Chawathe et al., 1999, Lutz et al., 2008]. Unfortunately, this feature prevents existing modeling languages from capturing phenomena that involve the evolution of an entity into something else, such as a caterpillar becoming a butterfly, or Germany splitting off into two Germanies right after WWII. In these cases, there is a general agreement that an entity evolves into one or more different entities. Moreover, there is a strong relationship between the two, which is not captured by merely deleting one entity and then creating another. For example, the concept of Whale evolved over the past two centuries in the sense that whales were considered some sort of fish, but are now recognized as mammals.

The aim of this dissertation is to *model evolution*, not only at the instance and class levels but also across levels: instances may evolve into classes and vice verse.

Having the data model chosen, the next problem is to make use of that data or in other words to enable the user to pose queries. The query language should support the user in formulating queries in an expressive and meaningful way. To *query data evolution* we need to exploit a large number of possible instances of data which are produced by evolution relationships, e.g. all Germanies are considered as the same country across its history or every ancestor of modern Germany is an individual country. This is done in a similar fashion to the one that probabilities in a probabilistic database are defining possible worlds [Dalvi and Suciu, 2007]. Each such instance, models a way that concepts associated trough some evolution path can be merged into one. The evaluation of a query over an evolution database will have to take into consideration all these possible instances (possible worlds). In contrast to the case of probabilistic databases, the evolution relationships are forming a graph that can be used for the efficient computation of answers across possible worlds. To find all the answers, a connectivity discovery algorithm needs to be executed, but algorithms of this kind have an exponential cost.

Data quality in evolving repositories. An important issue in user generated evolving repositories is the *data quality*. The continuously evolving nature of the data hinders the ability of determining the quality of the data that is observed at a specific moment, since there is a great deal of uncertainty on whether this information will remain as is. Such, Wikipedia, as an extremely success story of collaboratively edited content, rises the situations where there are incompatible viewpoints of the same issue, as often happens in real life. Ideally, a dispute will eventually converge to a community-accepted consensus after undergoing discussion between the disagreeing authors. However, disputes can sometimes degenerate into so-called edit wars which disrupt progress towards improving the article. To obtain better quality articles in such repositories it is important to address the problem of automatically identifying controversies [Vuong et al., 2008, Chin et al., 2010, Yasseri et al., 2012]. As an example, consider the Wikipedia entry for "Caesar salad". At one time, one group of authors argued that the name derives from the Roman emperor Julius Caesar, while another group argued that it was from an Italian-American restauranteur named Caesar Cardini. The controversy resulted in back-and-forth arguments between a large number of different authors until a consensus was reached to attribute the creation to Caesar Cardini.

Identifying controversies is important for several reasons: to distinguish them from vandalism, giving disputes a chance to be aired out rather than squelched; to set appropriate edit policies and raise the bar for supporting evidence when needed; and for enabling the reader a more nuanced understanding of content. For Wikipedia specifically, controversy detection plays a significant role. It helps maintaining Wikipedia's neutral point of view policy, which strives for objectivity, meaning that articles should be fair to all sides (though not necessarily equal, in the case of minority views), and that controversial content requires more rigorous documentation. It also helps identify and protect sensitive pages, for instance by allowing only accounts that have been active for some duration to make changes, as well as tightly restricting the frequency of reverts blocking any editor for some time period who attempts to revert more than once in a 24-hour period.

Personalization for evolving items/users. Another important task which arises in evolving user-generated repositories is *personalization* or the task of producing recommendations. A difficult problem commonly faced by recommender systems where items and users are continuously evolved is the cold-start problem [Schein et al., 2002]. In that problem, recommendations are required for new items (new-item cold-start) or users for whom little or no information has yet been acquired (new-user cold-start) [Schein et al., 2002]. Unfortunately, most recommendation algorithms fail because they cannot build the user's profile or identify users with similar preferences. In this dissertation, we address the cold-start problem where new items and new users continuously appear (we refer to it as new-item/user cold start). The new-item/user cold start is a setting which is common in several domains. For example, in news recommendations, new or unknown users appear constantly in the system while all users are interested in new content [Liu et al., 2010]. As another example, consider a system which recommends academic courses to students [Parameswaran et al., 2010]. On one hand, users (students) stay in the university for a short time (4 years for bachelors) hence they never use the recommender system for long enough. This phenomenon makes it even more difficult to have enough historical ratings even in the long run. On the other hand, courses are dynamic and every quarter (semester) they evolve, e.g., new instructors, updated material, new requirements, and so on. In such environment, the problem of cold start is very severe because the system cannot boost the recommendation algorithm even after some time of operation.

1.2 Research Issues and Contributions

Modeling and querying of data evolution. To address the problem of data evolution in heterogeneous environments the dissertation focuses on the problem of concept evolution which has not been studied comprehensively yet. We aim at supporting the queries of the following form: How has a concept evolved over time? From what (or, to what) other concepts has it evolved? What other concepts have affected its evolution over time? What concepts are related to it through evolution and how? These kinds of queries are of major importance for interesting areas like history, entity management, life sciences, and others. Such, historians may be interested in the evolution of countries such as Germany, with respect to territory, political division or leadership. Alternatively, they may want to study the evolution of scientific concepts, e.g., how the concept of biotechnology has evolved from its humble beginnings as an agricultural technology to the current notion that is coupled to genetics and molecular biology.

In this dissertation we propose a conceptual model [Rizzolo et al., 2009] enhanced with the notion of a lifetime of a class, individual and relationship. The temporal model is an extension of the temporal model proposed in [Gutiérrez et al., 2005] with consistency conditions and additional constructs to model merges, splits, and other forms of evolution among concepts. To support querying we introduce an *evolution-aware query language* that supports queries on the lifetime of concepts and describe its formal semantics.

In order to support query evaluation over such concept evolution we propose the following. Due to the computation complexity of the evolution-aware query language we reduce it to a less complex version, namely *evolution unaware query language*. More specifically, the semantics of evolution unaware query language defines how to retrieve concepts that may not be present in the database but can be formed dynamically through merges of concepts representing different

evolution phases. For example, a query about the patents of AT&T would normally return only the patents filled directly to AT&T but not those of the AT&T ancestors (e.g. Bell Labs). Taking into account the merges, spin-offs and acquisitions of AT&T we aim at finding all the patents in the *history* of AT&T.

Answering queries of the evolution unaware language is a challenging task as well. The evolution information recorded in the data is intensionally specifying a large number of possible instances in a similar fashion to the one that probabilities in a probabilistic database are defining possible worlds. Each such instance, models a way that concepts associated trough some evolution path can be merged into one. The evaluation of a query over an evolution database will have to take into consideration all these possible instances (possible worlds). In contrast to the case of probabilistic databases, the evolution relationships are forming a graph that can be used for the efficient computation of answers across possible worlds. To deal with that, we introduce the idea of finding a Steiner forest as a mean of finding the optimal merges that need to be done to generate the possible instances and present indexing techniques for its efficient implementation. The proposed algorithm allowed us to shift the complexity from the exponential to the parameterized, i.e. we have the polynomial time in the size of an evolution graph and exponential time in the size of a query. In turn, as the experimental evaluation showed we are able to produce answers in an online fashion.

The findings on the modeling and querying of data evolution resulted in a research prototype, called *TrenDS* [Bykau et al., 2013c], which is a part of the European Union research project Papyrus[1].

Data quality in evolving repositories. In user generated repositories, the very first example is Wikipedia, the content can be freely updated by any user and therefore it may evolve rapidly. As a result, this content is fresh but sometimes in a corrupted state due to vandalism or disputes. Therefore, the data quality problem for such repositories is a challenging problem. To address that problem, we concentrate on the controversy identification problem. Controversy identification is a challenging task due to the large number of versions in the history of a document. For example, at the time of writing, Wikipedia contained 4,105,700 entries and was growing at a rate of 25,000 new entries per month; the total number of versions among these entries was approximately 570M and growing at a rate of 5M edits per month.

In contrast to prior work on detecting controversial pages in Wikipedia at the page level, we focus on identifying *fine-grained* controversies pinpointing the specific controversial topics, locations within a page and occurrences in time. We observe that controversies often arise from conflicting points of view or opinions on the same topic. Unfortunately, segmenting a page into distinct topics is a very difficult problem in the natural language processing and machine learning literature. Moreover, tracking a topic over time is particularly challenging, even for a minor rewording, since its location within a page can change (say, from the introduction to the biography section). We formalize a syntactic notion of controversy based on the assumption that the associated dispute is expressed as back-and-forth substitutions of content embedded within a similar context (rather than deletions, which was the basis of previous work). We have investigated two different models for that purpose: one sees the document as (space-separated) "words"; the other sees the document as a sequence of the hyperlinks (to other Wikipedia pages) contained within a document, ignoring the text, multimedia and links to external Web sites.

[1] http://www.ict-papyrus.eu

While the former uses a superset of the latter, and therefore has access to more information, the latter exploits the labeling (and, therefore, normalization) of semantic concepts implicit in links, since most real-world entities and concepts now have a Wikipedia page and Wikipedia pages often contain a high density of hyperlinks, as is encouraged in the Wikipedia style guide. We tested our algorithms on a significantly large fraction of Wikipedia and demonstrated that controversies can be effectively identified from a document's version history in a scalable manner. Finally, our experiments have shown that the link model identifies more semantically meaningful edits, and as a consequence favors precision, while the word model favors recall.

Recommendations for evolving items/users. Another important task which arises in evolving user-generated repositories is personalization or the task of producing recommendations. In this dissertation, we address the cold-start problem which arises in dynamic repositories since new items and new users continuously appear (we refer to it as *new-item/user cold start*). The new-item/user cold start is a novel severe cold-start setting which has not been addressed so far and it is common in several domains. For example, in news recommendations, new or unknown users appear constantly in the system while all users are interested in new content [Liu et al., 2010]. As another example, consider a system which recommends academic courses to students [Parameswaran et al., 2010]. On one hand, users (students) stay in the university for a short time (4 years for bachelors) hence they never use the recommender system for long enough. This phenomenon makes it even more difficult to have enough historical ratings even in the long run. On the other hand, courses are dynamic and every quarter (semester) they evolve, e.g., new instructors, updated material, new requirements, and so on. In such environment, the problem of cold start is very severe because the system cannot boost the recommendation algorithm even after some time of operation.

Unfortunately, existing approaches to the cold-start problem are not applicable to the new-item/user cold start because they are designed to tackle one type of cold-start problem, new users or new items, but not both. Furthermore, in practice it has been shown that users are typically reluctant to engage into any interview process for providing their preferences [Liu et al., 2010]. We device three methods to produce recommendations for such repositories, namely *Similarity-based*, *Feature-based* and *Maximum entropy* methods. Each of the method leverages item features to produce recommendations in the following ways. The similarity-based method assumes that more similar items contribute to the recommendation more. The feature-based one considers every feature independently and finds how they contribute to an item's score. Finally, the maximum entropy employs the maximum entropy principle to find such a distribution of ratings over features which conforms to the observed values and doesn't introduce any additional bias.

Using an extensive experimental evaluation we show how the proposed methods perform. Due to the (lack of) assumptions of our problem setting, i.e., we use no external data or input, we cannot directly compare to other approaches that operate on such assumptions. As a baseline we use an adapted version of the Pairwise preference regression method [Park and Chu, 2009] which is the closest competitor.

1.3 Structure of the Dissertation

Chapter 2 discusses the works which are related to the topics of dissertation. In particular, we present an overview of temporal data models from different communities such as Entity Relationship (ER), semi-structured data, Description Logics, ontologies, and others. Then, we present the existing works on probabilistic databases and Steiner trees which are relevant to the query evaluation part of the dissertation. Next, we discuss the data quality problems in user-generated repositories. Finally, we present the existing literature on recommendation systems and, in particular, the cold start problem.

Chapter 3 introduces the problem of modeling and querying data evolution. We define the temporal data model along with the evolution relationships and their semantics. Moreover, an expressive query language, denoted the evolution aware query language [Rizzolo et al., 2009][Bykau et al., 2012], based on graph traversing is proposed. We present the query evaluation strategy [Bykau et al., 2011][Bykau et al., 2012] for the evolution unaware query language (which is a reduced version of the evolution aware language) along with the Steiner forest algorithm which is used for an efficient computation of answers. Finally, the evolution management research prototype called TrenDS [Bykau et al., 2013c] is discussed.

Chapter 4 focuses on the problem of data quality in user generated repositories. In particular, we propose a novel algorithm of controversy identification in Wikipedia [Bykau et al., 2013a] which analyzes the evolution history of a Wikipedia page. The chapter contains the details of the algorithm along an extensive experimental evaluation where we study a number of trade-offs with respect to the possible data models and the algorithm parameters.

Chapter 5 deals with the problem of personalization in evolving repositories. We define the problem of new-user and new-item cold start and propose a number of recommendation algorithms [Bykau et al., 2013b]. Then, we describe an experimental evaluation where we study both the efficiency and effectiveness of the algorithms on the two real datasets, namely the dataset of academic courses and the movie rating dataset.

Finally, we conclude in Chapter 6 and list the key contribution of the dissertation. Also we discuss the future directions of the work.

1.4 Published Papers

1. [Bykau et al., 2013c] Bykau, S., Rizzolo F., Velegrakis Y. A Query Answering System for Data with Evolution Relationships (SIGMOD), 2013

2. [Bykau et al., 2013a] Bykau, S., Korn, F., Strivastava, D., Velegrakis, Y. Controversy Detection in Wikipedia, 2013 (ready for submission)

3. [Bykau et al., 2013b] Bykau, S., Koutrika, G., Velegrakis, Y. On Dealing with the Permanent Coping with the Persistent Cold-start Problem, 2013 (ready for submission)

4. [Bykau et al., 2012] Bykau, S., Mylopoulos, J., Rizzolo, F., & Velegrakis, Y. On Modeling and Querying Concept Evolution. Journal on Data Semantics (JoDS), 2012

5. [Bykau et al., 2011] Bykau, S., Mylopoulos, J., Rizzolo, F., & Velegrakis, Y. Supporting Queries Spanning Across Phases of Evolving Artifacts using Steiner Forests. Conference

on Information and Knowledge Management (CIKM), 2011

6. [Katifori et al., 2011] Katifori, A., Nikolaou, C., Platakis, M., Ioannidis, Y., Tympas, A., Koubarakis, M., Sarris, N., et al. The Papyrus digital library: discovering history in the news. Theory and Practice of Digital Libraries (TPDL), 2011

7. [Bykau et al., 2010] Bykau S., Kiyavitskaya N., Tsinaraki C., Velegrakis Y. Bridging the gap between heterogeneous and semantically diverse content of different disciplines. (FlexDBIST), 2010.

8. [Presa et al., 2009] Presa, A., Velegrakis Y., Rizzolo R., and Bykau S. Modelling Associations through Intensional Attributes.International Conference on Conceptual Modeling (ER), 2009.

9. [Rizzolo et al., 2009] Rizzolo F., Velegrakis Y., Mylopoulos J., and Bykau S. Modeling concept evolution: a historical perspective. International Conference on Conceptual Modeling (ER), 2009.

Chapter 2

Related Work

The section discusses the related work of the topics which are presented in the dissertation. In particular, Section 2.1 discusses the areas of data evolution modeling and querying along with the limitations of the existing approaches. Section 2.2 describes the problem of data quality in systems with evolving user-generated contents (e.g. Wikipedia). We highlight the differences between our approach and the previous ones. Finally, in Section 2.3 we present the principles of recommender systems and the problems which arise in the presence of evolving items and users. We indicate the novelties of our solution in comparison to the related approaches.

2.1 Modeling and Querying of Data Evolution

The section focuses on two fields which are related to data evolution. First, we discuss the existing models of temporal and evolving data in Section 2.1.1 and then we present the works on possible world semantics and Steiner trees which are related to the querying of data evolution (Section 2.1.2).

2.1.1 Modeling Data Evolution

A large number of ways have been proposed to cope with evolving (changing) data in recent years. On the one hand, researches deal with the change modeling by means of temporal data models and, on the other hand, they use domain oriented frameworks to cope with data evolution.

Temporal Models

Temporal data models have achieved a significant attention in many research communities and from a number of perspectives such as the relational data model, Entity-Relationship model, semi-structured data, knowledge representation field. In the following, we discuss those perspectives in detail.

Temporal data management is extensively studied in the *relational* paradigm and a large number of approaches on modeling, querying, indexing and storing the relational data with time have been proposed (see a detailed survey [Soo, 1991]). Versioning approaches store the information of the entire document at some point in time and then use edit scripts and change logs to reconstruct versions of the entire document. In contrast, [Buneman et al., 2002]

and [Rizzolo and Vaisman, 2008] maintain a single temporal document from which versions of any document fragment (even single elements) can be extracted directly when needed.

The *Entity-Relationship* (ER) temporal modeling aims to enrich the ER model with a temporal dimension. [Gregersen and Jensen, 1999] describes and compares the existing temporal ER models according to the predefined set of parameters such as the valid/transaction time support, explicit/implicit temporal constructs, multiple time granularities, compatibility issues and others. This work sharps the core ideas in the time modeling which have been used not only for the ER modeling but also in other areas of data and knowledge management.

In the domain of *semi-structured* data, one of the first models for managing historical information was proposed by Chawathe et al. [Chawathe et al., 1999]. In turn, a versioning scheme for XML was proposed in [Chien et al., 2001]. In temporal models for XML, the time dimension is supported through the time intervals assigned to both XML nodes and containment/reference edges. Such, Rizollo et al. [Rizzolo and Vaisman, 2008] proposed a model for the temporal XML where to deal with the life spans of nodes and edges, the temporal XML defines a set of restrictions which have to be fulfilled in order to have a consistent model. In addition, Rizollo et al. present the algorithms of the consistency checking along with their computational complexities. The querying of temporal XML documents is supported by the TXPath query language, an extension of XPath, which is enriched with temporal statements. In order to support querying, Rizollo et al. define the continuous path, i.e. an XML path which is valid during a particular interval of time. Focusing on the continuous paths of a document, Rizollo et al. proposed an index which supports query evaluation.

In the Description Logics (DLs), temporal assertions are considered at both TBox and ABox levels (see a survey in [Lutz et al., 2008]) allowing the modelers to reason over temporal statements. Basically, the temporal DLs are a composition of some non-temporal DL and two special operators: `at the next moment` and `eventually`. Using these two building blocks one can produce more expressive operators. The whole family of temporal DLs can be classified in several directions: the point-based versus interval-based time, the explicit/implicit temporal dimension, the external/internal point of view on time [Artale and Franconi, 2001]. The main problem of using DLs for temporal modeling is to find the best trade-off between the expressivity of a logical formalism and its tractability.

A study of *evolution constraints* modeled in DL is presented in [Artale et al., 2007]. The idea of this work is to identify and formalize possible constraints over evolving data by means of DLs and allow for the reasoning about them. To fulfill this goal the authors classify the changes which may appear in temporal models.

- Dynamic integrity constraints. These kinds of constraints are applied to attributes. For example, we may require that the attribute `Salary` of a person can only increase its value or, for a multivalued attribute, the number of values can only increase (the number of diplomas of a person).
- Status constraints. These types of constraints pose restrictions on the life cycle of an object. As an example, we can define that an active object of some class may become a disabled member of it but not vice verse (the detailed explanation of the statuses is below).
- Transition constraints. They appear when we need to restrict the possible memberships of an object throughout its lifetime. For instance, an object of a person can only change its participation in the following order. It starts to be a member of the class `Student`, then it becomes an instance of the class `Worker`, and finally it goes to the class `Retired`.

- Evolution constraints embedded into relationships. Relationships can specify evolutionary constraints itself. The `generation relationships` [Gupta and Hall, 1992] define the fact that some object generate another. For instance, a company's department can split into two others, so that we may say that one department "cause" the existence of others. In turn, `synchronization relationships` [Parent et al., 2006] enforce Allen's temporal constraints [Allen, 1983] between the participating objects. For example, the relationship which assigns a person to some project forces the participating objects to overlap in time.

To support the aforementioned constrains the authors define a set of temporal class statuses which characterize the life cycles of their objects. `Scheduled` - an object is not a type of the class, but at some point of time it will be, `Active` - an object is an instance of the class, `Suspended` - captures the temporal inactive relation (an employee taking a temporary leave of absence is an example of what can be considered as a suspended employee), `Disabled` - an object is not an instance of the class and, moreover, can not be it at any point of time in the future. Using these statuses one can formally define constraints on changes.

As a sequel of the previous work, Artale et al. published the paper [Artale et al., 2008] which studies the `part-of` relation in terms of the lifecycles of objects. In particular, the authors define the following topology of the `part-of` relation. The `mandatory part` is the part which must exist for some object (like a heart for a human) during its life time, but it is not crucial to be the same object (transplantation of hearts). In contrast, the `essential part` is not only a part which must exist for an object, but also it needs to be the same across the whole life time of the object (e.g. the brains of a human). The `immutable part` is called a conditionally essential part which means that an object belongs to some class till it has this part. Through the notions of class statutes, Artale el al. formally describe these types of `part-of` relation, what, in turn, presents a good use case of the framework described in [Artale et al., 2007].

Ontologies [Guarino, 1995] are well-know formalism to describe data with the ability to infer (reason) at class and instance levels. Although ontologies, as a comparatively new research branch in knowledge management, have many similarities in temporal modeling with the corresponding approaches in ER, XML and other communities, they have their own distinctions. The survey [Flouris et al., 2008] classifies the whole variety of ontology changes such as mapping, matching, evolution, articulation, versioning, and others. The aforementioned classification of ontology changes pays more attention on the semantic properties of evolution comparing with the data-oriented approaches. For example, it shows the reasons why changes may appear, what changes coming in an integration process (how to change an ontology in order to resolve the heterogeneity between the source and target), how to handle different version of ontologies at different time points, and others.

Similarly to the versioning in databases, *ontology versioning* studies the problem of maintaining changes in ontologies by creating and managing different variants of it [Klein and Fensel, 2001]. An approach to model revision in RDF ontologies has been presented in [Konstantinidis et al., 2007].

Gutierrez et al. propose a *temporal extension for the Resource Description Framework* (RDF) in [Gutiérrez et al., 2005]. In fact, the paper presents a framework which incorporates temporal reasoning into RDF. The authors define the semantics of explicit temporal operators

in the terms of the non-temporal RDF. The time dimension is presented by an assignment of validity intervals to triples. Furthermore, the paper proposes the sound and complete inference system what's entailment does not yield any extra asymptotic time complexity with respect to standard RDF graphs. In addition to that, a sketch of temporal query language is defined.

Most of the methods presented in the temporal models are founded on the notion of "entity" that represents a "thing" in the application domain. Although the state of an entity can change over its lifetime, entities themselves are atomic and immutable. Unfortunately, this feature prevents existing modeling languages from capturing phenomena that involve the evolution of an entity into something else. In our case, we focus on the evolution of a concept that spans different concepts (e.g., student to professor, research lab to independent corporate concept).

Evolution Frameworks

Schema evolution [Lerner, 2000] studies changes that appear in a relational schema. In particular, a schema can undergo the following changes: a new attribute is inserted or an old one is deleted, the type of an attribute is changed, the bounds of a range type are changed, and so forth. In the scope of such structural changes some challenging problems are arisen. For instance, during the integration between heterogeneous data sources a structural change over one of the schema should be properly taken into account in the existing mappings [Velegrakis et al., 2004]. The *view maintenance* deals with a problem of an automatic view update in the case of a change in the underlying schemas. To cope with this problem a view update algorithm has been proposed in [Kotidis et al., 2006]. In both cases, the changes occur at the structural level whereas in our work we are interested in the semantics of those changes.

A composite approach to model and query *evolving data in digital archives* has been proposed by Berberich et al. [Berberich et al., 2009]. The authors show that digital archives containing the data about historical events may potentially not be processed properly because of the changes in the terminology. For example, if one asks about the documents related to St.Petersburg, the result list will be empty for the period from 1924 to 1991 because the city had the name Leningrad that time. The authors claim that there is a vast number of such terminology changes in the history, so that there is a clear need to be able to find and use them. To enable the relations between different terms through time the authors propose the following. First, a measure to compare semantic similarities between terms is suggested. Two concepts which co-occur with the same set of terms may be treated as semantically related concepts. For example, we know that the concept `Walkman` appears together with such terms like player, mp3, music, and others and, moreover, the term `iPod` co-occurs with approximately the same set of terms, as a result we may conclude that `Walkman` and `iPod` are used in the same contexts for their respective times. On that basis the authors introduce the statistical way to measure such kind of similarities. Second, to support the query answering, the authors developed the query reformulation method that rewrites user queries according to the existing similarities between the terms. Third, an experimental study has been carried out to study the effectiveness and applicability of the framework. The above work is highly related to our approach to model the concept evolution. However, there are some fundamental differences. Firstly, we focus on the semantics of the evolution changes, i.e. what exactly happens during the evolutions in terms of the temporal and mereological constraints whereas the terminology evolution work address only one kind of change. Secondly, while the terminology evolution supports the keyword searching, we provide a query language (and its evaluation algorithm) which is capable of asking structural questions

about the evolution history (e.g. using attributes and associations of entities).

A framework which deals with *changes in the semi-structured data model* is proposed by Chawathe et al. [Chawathe et al., 1999]. As a motivating example the authors use a web page which is based on a semistructured data (HTML) and undergoes several changes along its lifetime. Although the simple storage of all version of a web page allows users to query different states (versions), the authors focus on the ability to pose queries about the changes itself. For example, one may be interested in finding all restaurants whose entree price changed or wants to be notified when some kind of change appears. To support such kind of queries, the change annotations are introduced. In particular, a user can explicitly specify what and when evolved: one can create a new node or update the existing one, a new arc (relationship) or update the existing one. Managing such meta-information gives the possibility to effectively trace the history of data across a series of document versions. The query answering is done through the query rewriting when the constructs asking about the changes are presented as normal statements. In addition, it is shown that this kind of rewriting do not yield additional complexity. In contrast to our work, Chawathe et al. aim at recording and managing changes that are taking place at the values of the data whereas our approach focuses on how concept evolve/mutate into another.

Kauppinen et al. present a new vision on the evolution of ontologies in [Kauppinen and Hyvonen, 2007]. The authors study the very similar problem to that in [Berberich et al., 2009], however narrowing their domain to the Geographical Information Systems (GIS). They show that the museum artifacts should be annotated with geo terms (concepts) which, in turn, have cross temporal dependencies. For instance, asking about nowadays Germany artifacts, it is impossible to infer that also West and East Germany ones should be taken into account because of the Germany history. The paper proposes several novel ideas to deal with evolving concepts. Firstly, it introduces the explicit spatial bridges between concepts. One can express facts about splitting, merging, evolving of concepts. Secondly, the authors provides the semantics of such global bridges by means of local ones. The local bridges connect two sets of concepts allocated in different snapshots. Basically, it specifies how the concepts from one snapshot cover or covered by another. For this purpose the authors define two sets: `covers` and `coveredBy`. For example, the merge of East and West Germany can be translated into the local bridges as following. The territory of new-formed Germany consists of that ones of East and West Germany in the proportion of 0.3 and 0.7, therefore we can write that:

$$covers = \{\langle Germany; EastGermany\rangle = 1; \langle Germany; WestGermany\rangle = 1\} \tag{2.1}$$

$$coveredBy = \{\langle Germany; EastGermany\rangle = 0.3; \langle Germany; WestGermany\rangle = 0.7\} \tag{2.2}$$

Kauppinen et al. formally create definitions: the merge occurs when several independent regions form a new one, split - a region is divided exhaustively into several distinct regions, etc. Finally, the paper gives the way of reasoning about such global bridges in order to produce rankings of results. The algorithm which supports such kind of reasoning is presented. For instance, knowing that 30% of Germany covers East Germany one can conclude that all artifacts related to Germany may be associated with East Germany with coefficient of 0.3. The main difference between the work of Kauppinen et al. and ours is that Kauppinen et al. focus on the GIS domain and therefore it is hard to model causality and types of evolution involving abstract concepts beyond geographical concepts.

Entity linkage [Ioannou et al., 2010, Li et al., 2011] is a problem faced by many modern web storages, i.e. there can be many data records which refer to the same entity (e.g. in DBLP there authors who changed their names and thus there is a need to merge the information about the same entity). To address this problem one need a way to decide whether two records describe the same entity or not. In [Li et al., 2011] Li et al. address the problem of entity linkage from a temporal perspective, i.e. one entity is represented as many data records distributed in time. By using temporal decay and clustering Li et al. link records with the accuracy higher than the existing linkage methods. The above work differs from the work of the dissertation in the following aspects. First, we focus on the semantics of evolution, i.e. what happens with those entities (split, merge, etc.) whereas the temporal linkage considers only one type; we focus on the query evaluation whereas the temporal linkage focus is on the discovery of entity histories (for that purpose we use the trademark dataset, the details are in Section 3.7). To address the entity linkage problem [Ioannou et al., 2010] uses the possible world semantics which is discussed in Section 2.1.2 more in detail.

2.1.2 Query Answering on Evolving Data

To discuss the related work of the area of query evaluation on evolving data we focus on the probabilistic databases and Steiner trees which are the essential parts of our evaluation algorithm.

Possible world semantics has been intensively used in many areas such as probabilistic databases, entity linkage, data integration, and others. Informally, possible world semantics defines a set of possible worlds each of which is a valid instance of database under some condition. Then, a query is evaluated over all that possible instances and the results are combined into one answer set with some certainties (scores) attached. In the next, we go over the most related applications of possible world semantics.

A seminal paper [Dalvi and Suciu, 2007] on probabilistic databases introduces the semantics of queries where matches can be uncertain, i.e. not all the conditions of a query are necessarily satisfied. This results in answers which are ranked with respect to the relevance to the original query. In the area of entity linkage, possible world semantics is used to model uncertain linkages [Ioannou et al., 2010]. By considering that linkages have some level of uncertainty, the query answering is reduced to the problem of testing all possible combinations of linkages and finding those that most likely answer the query semantics. In data integration, probabilistic mappings are supported through possible world semantics [Dong and Halevy, 2007]. In this case the connections between the data sources and the mediated schema are defined with some level of confidence. [Dong and Halevy, 2007] provides the algorithm for efficiently computing the answers and study its complexity. The above approaches are similar to our case in the sense that we can treat evolution relationships as a source of uncertainty which allows us to generate possible database instances. However, the query evaluation and optimization in our approach is fundamentally different.

In the following, we discuss the problem of *Steiner tree*, i.e. how to find such a tree on a graph which covers a predefined set of nodes (called terminals) possibly using additional nodes (called Steiner nodes). Note, that in contrast to the polynomial complexity of the problem of spanning tree [Neetil et al., 2001] where there are no additional nodes, the problem of Steiner tree becomes NP-hard. Existing approaches for finding *Steiner trees* can be divided into two

groups: approximate solutions [Bhalotia et al., 2002, Kacholia et al., 2005, He et al., 2007] and exact, with the latter been based on dynamic programming [Dreyfus and Wagner, 1972, Molle et al., 2005, Ding et al., 2007, Kimelfeld and Sagiv, 2006]. In this section we consider only exact solutions since approximate solutions cannot be applied in our case (see details in Section 3.4.3).

Historically, a first algorithm was proposed by [Dreyfus and Wagner, 1972] and it is based on the decomposition property of a Steiner tree, i.e. for an arbitrary terminal the tree is divided into two Steiner trees of smaller size and one shortest path. Using that decomposition property [Dreyfus and Wagner, 1972] builds a dynamic programming algorithm which computes the Steiner tree for every subset of terminals until the solution is found. To avoid the computation of all subtrees, [Molle et al., 2005] proposes the division of a Steiner tree into inclusion-maximal regions where each terminal is a leaf. In that case, the Dreyfus-Wagner algorithm is used only for subtrees of the size of the maximum region. [Ding et al., 2007, Kimelfeld and Sagiv, 2006] introduce a new way for the iterative computation of the solution to a Steiner tree problem by gradually growing and merging Steiner subtrees. In our work we proposed an extension of an algorithm from [Ding et al., 2007] that allows for the computation of a Steiner forest.

2.2 Data Quality in Evolving User-Generated Contents

In user-generated repositories (e.g. Wikipedia) a content can be updated by any user at any moment of time. On the one hand, this leads to an up-to-date information which reflects real-time changes in the world but on the other hand to vandalism, controversies, and stability issues. Therefore, the problem of data quality in such repositories is an important and challenging problem. In the following we classify the data quality problem into three sub problems which may overlap in some aspects, namely the content *stability* prediction [Druck et al., 2008, Papadakis et al., 2011, Adler and de Alfaro, 2007, Adler et al., 2008], the *vandalism* [Chin et al., 2010, Viégas et al., 2004] and *controversy* [Viégas et al., 2004, Vuong et al., 2008, Sepehri Rad and Barbosa, 2011, Yasseri et al., 2012, Sepehri Rad et al., 2012, Kittur et al., 2007, Brandes and Lerner, 2008] detection.

The methods which are used to address those problems can be grouped as follows. The *statistical* approaches [Liu and Ram, 2011, Vuong et al., 2008, Adler and de Alfaro, 2007, Adler et al., 2008, Yasseri et al., 2012] are based on gathering some statistics about the content or/and authors and experimentally showing the correlation between the statistics and the aforementioned data quality characteristics. The *learning* methods [Druck et al., 2008, Chin et al., 2010, Papadakis et al., 2011, Sepehri Rad and Barbosa, 2011, Sepehri Rad et al., 2012, Kittur et al., 2007] define the features of collaboratively created content and train some classifiers to detect vandalism, controversies, and so on. The *visualization* method [Viégas et al., 2004, Adler et al., 2008, Brandes and Lerner, 2008] aims at building a visual representation of the data such that the user can easily detect the patterns of interest (e.g. vandalism, controversies). As the data quality ground truth the existing solutions use the Wikipedia meta-information (Wikipedia templates, the lists of controversial and featured articles) [Liu and Ram, 2011, Vuong et al., 2008, Sepehri Rad and Barbosa, 2011], the Wikipedia administrator elections [Sepehri Rad et al., 2012], the longevity of content [Druck et al., 2008, Papadakis et al., 2011, Adler and de Alfaro, 2007, Adler et al., 2008] or the user feedback [Chin et al., 2010, Viégas et al., 2004, Yasseri

et al., 2012].

In this section, we discuss all three data quality problems and since in the dissertation we concentrate on the problem of controversy detection, we take a closer look at the controversy problem. Moreover, we discuss the limitations of existing controversy detection works with respect to our approach.

2.2.1 Stability

The *stability* of content (or an edit) is defined as its ability to stay on a page for a comparatively long time without being altered or modified. The more stable content is of a primary importance for the Wikipedia readers because it is more trustful.

The machine learning techniques are widely used to predict the stability of the Wikipedia content. Such, the problem of predicting the level of stability of an edit is studied by Druck et al. [Druck et al., 2008]. For each edit, the authors identify features such that registered vs. not-registered user edits, the number of reverts done by the user performing an edit, addition of a link or a text deletion and so on. By using a discriminatively-trained probabilistic log-linear model, the approach classifies edits into three classes: the revert, longevity (that the edit will stay on the page) 6 hours, longevity 1 day. The paper discusses the formal notion of an edit quality, the analysis of features which are the most relevant to the data quality of edits. Although the authors used a probabilistic log-linear model, the solution is not scalable to be used at run-time. Similar to [Druck et al., 2008] Paradakis et al. [Papadakis et al., 2011] addresses the problem of entity stability in collaboratively edited data sources. More specifically, Papadakis et al. pursue the goal of detecting entity attribute-value pairs which will not be changed in next revisions. The solution relies on N-Grams and machine learning techniques.

The notion of trust and stability are closely connected because a more trustful content tends to be unaltered for a longer period of time. Adler et al. [Adler and de Alfaro, 2007, Adler et al., 2008]. focus on the trust of Wikipedia pages by measuring the longevity of edits. More specifically, in [Adler and de Alfaro, 2007] Adler et al. define the notion of author reputation and design an algorithm which computes the reputation values of Wikipedia authors. In the nutshell, a Wikipedia author increases its reputation if her edits are not changed by the subsequent edits. This is interpreted as a sign of the approval that the edit should remain on the page and thus the user who did it deserves that her reputation grows. By analyzing the revision history of a page, the WikiTrust system computes the reputation values for all users. In the experimental evaluation, Adler et al. show that the reputation values of authors have a good predictability of the text stability.

In the subsequent paper [Adler et al., 2008], Adler et al. extend their previous work by analyzing the trust of text which is measured through its stability, i.e. the ability of text to stay on the page without being altered or removed. The work leverages the author reputation in order to compute the text trust where the main assumption is that a high reputation author is likely to produce a trustable text. The work results in an online system[1] which allows the Wikipedia readers to see the trust of the text of any page of Wikipedia at run-time.

The correlation between various patterns of collaboration and quality of Wikipedia articles is analyzed in [Liu and Ram, 2011]. The patterns are determined as clusters of pages where

[1] http://www.wikitrust.net/

contributors with specific roles performed specific kinds of actions. For example, starters, contributors focusing on sentence creation, dominate sentence insertions and causal contributors did a lot of sentence modifications. By identifying 5 clusters and using the quality labels of Wikipedia articles, the authors statistically prove the correlation between them. Although the work has no direct practical applicability, its theoretical findings about collaboration patterns can be used for building approaches to improve the quality of Wikipedia.

2.2.2 Vandalism

Since Wikipedia allows the authors to freely edit almost any page (there are a few most popular pages where only registered authors can modify the content) some of them intentionally create irrelevant, nonsense or even harmful content. This activity is enormously widespread and known as *vandalism*. The methods of vandalism detection plays a very important role in Wikipedia since they allow the Wikipedia administrators and readers to quickly identify vandalized pages.

The problem of vandalism detection was extensively studied by Chin et al. [Chin et al., 2010]. The authors identify different kinds of vandalism based on the type of action (delete, insert, change, revert), the type of change (format, content), the scale of editing (mass deletion or mass insertion). The proposed solution uses machine learning techniques to classify the blanking, large-scale editing, graffiti and misinformation types of vandalism by means of the text features such that the number of known words, perplexity value, number of known bigrams and so on.

A visualization approach for vandalism detection is proposed by Viegas et al. [Viégas et al., 2004]. It aims at visualizing the history of a Wikipedia page in order to study its conflict, vandalism and cooperation patterns. The underlying data model is based on text where the basic building block is a sentence delimited by tokens. By consecutively parsing the revisions of a page at the sentence level the system plots colored flows of sentences from one revision to another. This allows the user to observe abnormal content changes (e.g. oscillating flow represents reverts, etc.)

2.2.3 Controversy

The prolonged public disagreements or heated discussions, known as *controversy*, plays a significant role in Wikipedia. Detecting the controversial content is of a primary importance both for the Wikipedia authors and readers since usually that kind of content presents some important topics that the community haven't developed a single opinion on. Since in the dissertation we concentrate on the problem of controversy detection in the following we discuss the existing approaches along with their limitations in detail.

An approach to detect controversies using the statistics of deletions is proposed by Vuong et al. [Vuong et al., 2008] According to Vuong et al., the basic element of a controversy is a dispute when someone disagrees on the present content and deletes it. A dispute is more controversial if its authors caused less controversies in the past. Conversely, a dispute is more controversial if it appears in a less controversial page. Using the above assumptions, known as the Mutual Reinforcement Principle, Vuong et al. provide the formulas to compute the level of controversy of a page. The experimental evaluation shows that dispute statistics has a higher correlation

with the page controversiality than the baseline approaches like the number of revisions, the number of unique authors and others. Our work is fundamentally different to [Vuong et al., 2008] since we focus on detecting individual controversies whereas Vuong et al. classify entire pages. Moreover, our building block is substitutions, i.e. when authors correct information that they consider incorrect and propose an alternative information, whereas Vuong et al. use insertion/deletions which are more related to vandalism.

Another recent study of conflicts at the page level is proposed by Yasseri et al. [Yasseri et al., 2012]. Similarly to [Chin et al., 2010] (the method for vandalism detection which is discussed in Section 2.2.2), Yasseri et al. use the revert statistics of a page to detect controversies at the page level. With the help of extensive experimental evaluation, the work shows that reverts are a good indicator of the page controversiality.

Machine learning techniques were also widely used to identify controversies, Such a method based on Support Vector Machines (SVM) to predict conflicts was proposed by Kittur et al. [Kittur et al., 2007]. As input features Kittur et al. used the number of revisions, page length, unique editors, anonymous edits, etc. As a ground truth the paper used controversial templates. A method to learn the positive and negative attitudes between Wikipedia authors using the history of their interactions is proposed by Sepehri et al. [Sepehri Rad et al., 2012] In particular, it builds the author profiles by gathering the statistics of three kinds: the author individual features, directional and mutual features. The training data is obtained by analyzing the Wikipedia administrator elections where the votes are interpreted as positive/negative attitudes. Sepehri et al. experimentally show that the proposed classifier leads to a high accuracy in predicting the author attitudes to each other and, in addition, the classifier is capable of accurately identifying a controversial content. The main limitation of the above controversy detection solutions is that they focus on controversies at the page level, i.e. they answer the question of how controversial is a particular page. However, our focus is on individual controversies. We aim at looking for the place on a page and time of a controversy along with its topic.

A recent work [Sepehri Rad and Barbosa, 2011] on controversy detection proposes to identify controversial arguments of a page at the section/paragraph level by using the user interaction statistics. More specifically, the authors focus on the two metrics of controversiality, namely the bipolarity of the edit graph of a page and the classifier trained on the featured and controversial articles. Two desirable properties are introduced for that metrics, i.e. the discrimination of featured and controversial articles and monotonicity. The latter means that when the controversial arguments are removed the controversy metrics should decrease its value. The experiments show that only the classifier has both the monotonicity and discrimination properties whereas the bipolarity doesn't change its value during the removal of arguments. Although Sepehri et al. [Sepehri Rad and Barbosa, 2011] introduce the idea of controversial argument, they report only preliminary statistical results which cannot be used as a practical method of individual controversy detection.

Finally, a number of approaches [Kittur et al., 2007, Brandes and Lerner, 2008] aim at visualizing the data in order to assist the user in identifying controversies.

2.3 Recommendation Systems

In this section we discuss the work which is related to the problem of recommendation in evolving user-generated repositories. We start by introducing the domain of recommender systems and then we discuss the recommendation strategies when items and users rapidly evolve. Finally, we present the recommendation methods which are based the idea of maximum entropy since in the dissertation we use it as a main solution method.

Recommender systems suggest items of interest to users, like products in Amazon and movies in Netflix, based on the users explicit and implicit preferences, the preferences of other users, and user and item attributes. Depending on the type of data upon which recommendations are based, recommender systems can be broadly divided into three categories: content-based filtering [Mladenic, 1999], collaborative filtering [Su and Khoshgoftaar, 2009] and hybrid [Burke, 2002].

Historically, a first approach was *content-based filtering* which leverages the existing data about items and users. More specifically, it aims at building users' profile, the detailed information about user's preferences, by analyzing her previous item contents and ratings. To make a recommendation, the algorithm matches the user's profile with the available items finding the item which is closest to the user's profile. *Collaborative filtering* makes a recommendation in two steps. First, it searches for the users who have rated similar items with similar ratings. Those users are likely to have similar tastes with the user the algorithm produces a recommendation to. Second, by looking at the items which obtained high ratings from the similar users and are not taken by the current user, the collaborative filtering approach produces a recommendation. Finally, *hybrid* approaches combine content-based and collaborative filtering characteristics.

In evolving repositories both items and users may change rapidly. As a consequence, the system cannot accumulate enough historical information about the users and items. This fact prevents the existing recommendation algorithms from operating properly. The above problem is known as the *cold-start problem* [Schein et al., 2002] and it have received a lot of attention. In the next, we discuss three research directions which address the cold-start setting.

First, a natural approach to solving the cold-start problem is to elicit new user preferences by collecting users' responses progressively through an initial interview process'[Rashid et al., 2002]. An effective model for designing the interview process should ask the interview questions adaptively, taking into account the user's responses when asking the next question in the interview process. Decision trees have been found to work especially well for this purpose [Golbandi et al., 2010]. Another approach is to use some external source. Social sites have been used as a source of information both for new users and new items. In the first case, the preferences of a new user can be inferred by the preferences of her social or trust network [Guy et al., 2009, Jamali and Ester, 2010]. On the other hand, social tags have been used to augment information about new items [Eck et al., 2007]. Ontologies are yet another source of information for new items [Middleton et al., 2002]. Finally, a third approach for solving the cold-start problem is to utilize the features of users or items. The content features can be used to capture the similarities between users and items and thus reduce the amount of data required to make accurate predictions [Agarwal and Chen, 2010, Agichtein et al., 2008, Schein et al., 2002]. Latent Factor Analysis (LDA) has been recently used [Agarwal and Chen, 2010] for that purpose, and has the advantage that it takes into consideration item features even in the presence of a cold start.

Although this setting is very close to ours there is a fundamental difference. In particular, the method produces a prediction for user i and item j as the multiplication of vectors which represent the affinity of item j to the K different latent topics and the affinity of user i to the same set of topics. In our case, we have a new user whose affinity vector is not know. However, if we consider the case when the user affinity vector is the same for all users and items are represented as a bag of words (features) we can employ the Latent Dirichlet Allocation (LDA) to find the topics of items and therefore improve the feature connectedness. We analyze this method further in the experimental Section 5.5.

In the dissertation, we use the idea of *maximum entropy* to address the cold-start problem and in the following we discuss how the notion of maximum entropy was used in the recommendation community and how we are different. A number of works [Jin et al., 2005, Pavlov et al., 2004] used the maximum entropy principle to produce recommendations. Such, Pavlov et al. [Pavlov et al., 2004] employed the maximum entropy to model the probability distribution of the items in a user session. In turn, Jin et al. [Jin et al., 2005] modeled both the navigation history and the previous ratings of a user using the maximum entropy distribution thus unifying the ideas of the content-based and collaborative filtering. However, we use the maximum entropy in a different way, in particular, while in [Jin et al., 2005] and [Pavlov et al., 2004] the maximum entropy principle is used to model user profiles, in our case the maximum entropy allows us to mine and use patterns of item features which have the highest impact on the ratings.

Chapter 3

Modeling and Querying of Data Evolution

In this chapter we introduce, motivate and address the problem of concept evolution, i.e. how to capture not only the changes that have taken places on the values of the data but also the semantic evolution of the concepts that the data structure represents. We start by introducing the idea of concept evolution in Section 3.1. Section 3.2 describes our motivating examples. In Section 3.3 we present a modeling framework for evolution-aware queries. A formal semantics of query answering in the presence of evolution for queries that are evolution-unaware is discussed in Section 3.4. Section 3.5 presents the architecture of the system we have developed that incorporates the evolution-unaware evaluation methods. Two case studies which involve evolution are described in Section 3.6. Finally, the description of a number of experiments for evaluating our evolution-unaware query answering mechanism is shown in Section 3.7.

The work presented in the chapter is published in [Rizzolo et al., 2009], [Bykau et al., 2011], [Bykau et al., 2012] and the demo version is presented in [Bykau et al., 2013c].

3.1 Concept Evolution

Conceptual modeling languages – including the ER Model, UML class diagrams and Description Logics – are all founded on a notion of entity that represents a thing in the application domain. Although the state of an entity can change over its lifetime, entities themselves are uniformly treated as atomic and immutable. Unfortunately, this feature prevents existing modeling languages from capturing phenomena that involve the mutation of an entity into something else, such as a caterpillar becoming a butterfly, or the evolution of one entity into several (or vice versa), such as Germany splitting off into two Germanies right after WWII. The same difficulty arises when we try to model evolution at the class level. Consider the class Whale, which evolved over the past two centuries in the sense that whales were considered a species of fish, whereas today they are known to be mammals. This means that the concept of Whale-as-mammal has evolved from that of Whale-as-fish in terms of the properties we ascribe to whales (e.g., their reproductive system). Clearly, the (evolutionary) relationship between Whale-as-fish and Whale-as-mammal is of great interest to historians who may want to ask questions such as "Who studied whales in the past 3 centuries?" We are interested in modeling and querying such evolutionary relationships between entities and/or classes. In the sequel, we refer to both entities and classes as concepts.

In the database field, considerable amount of research effort has been spent on the development of models, techniques and tools for modeling and managing state changes for a concept, but no work has addressed the forms of evolution suggested above. These range from data manipulation languages, and maintenance of views under changes [Blakeley et al., 1986], to schema evolution [Lerner, 2000] and mapping adaptation [Velegrakis et al., 2004, Kondylakis and Plexousakis, 2010]. To cope with the history of data changes, temporal models have been proposed for the relational [Soo, 1991] and ER [Gregersen and Jensen, 1999] models, for semi-structured data [Chawathe et al., 1999], XML [Rizzolo and Vaisman, 2008] and for RDF [Gutiérrez et al., 2005]. Almost in its entirety, existing work on data changes is based on a state-oriented point of view. It aims at recording and managing changes that are taking place at the values of the data.

In the next sections, we present a framework for modeling the evolution of concepts over time and the evolutionary relationships among them. The framework allows posing new kinds of queries that previously could not have been expressed. For instance, we aim at supporting queries of the form: How has a concept evolved over time? From what (or, to what) other concepts has it evolved? What other concepts have affected its evolution over time? What concepts are related to it through evolution and how? These kinds of queries are of major importance for interesting areas which are such as those discussed below.

History. Modern historians are interested in studying the history of humankind, and the events and people who shaped it. In addition, they want to understand how systems, tools, concepts of importance, and techniques have evolved throughout history. For them it is not enough to query a data source for a specific moment in history. They need to ask questions on how concepts and the relationships that exist between them have changed over time. For example, historians may be interested in the evolution of countries such as Germany, with respect to territory, political division or leadership. Alternatively, they may want to study the evolution of scientific concepts, e.g., how the concept of biotechnology has evolved from its humble beginnings as an agricultural technology to the current notion that is coupled to genetics and molecular biology.

Entity Management. Web application and integration systems are progressively moving from tuple and value-based towards entity-based solutions, i.e., systems where the basic data unit is an entity, independently of its logical representation [Dong et al., 2005]. Furthermore, web integration systems, in order to achieve interoperability, may need to provide unique identification for the entities in the data they exchange [Palpanas et al., 2008]. However, entities do not remain static over time: they get created, evolve, merge, split, and disappear. Knowing the history of each entity, i.e., how it has been formed and from what, paves the ground for successful entity management solutions and effective information exchange.

Life Sciences. One of the fields in Biology focuses on the study of the evolution of species from their origins and throughout history. To better understand the secrets of nature, it is important to model how the different species have evolved, from what, if and how they have disappeared, and when. As established by Darwin's theories, this evolution is very important for the understanding of how species came to be what they are now, or why some disappeared.

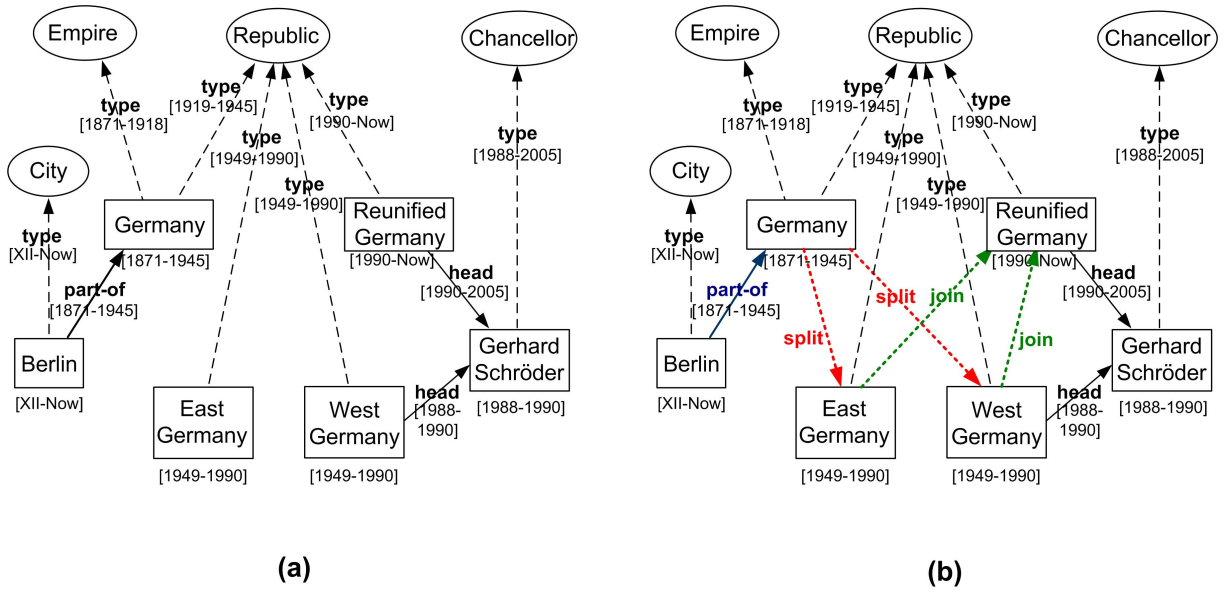


Figure 3.1: The concepts of Germany, in a temporal model **(a)** and in our evolutionary model **(b)**

3.2 Evolution Scenarios

Consider a historian database that records information about countries and their political governance. A fraction of the database modeling a part of the history of Germany is illustrated in Figure 3.1 (a). In it, the status of Germany at different times in history has been modeled through different individuals or through different concepts. In particular, Germany was an empire until 1918 and a republic from 1919 to 1945. This change is modeled by the multiple instantiations of Germany relative to Empire and Republic. Shortly after the end of the World War II, Germany was split in four zones[1] that in 1949 formed East and West Germany. These two entities lasted until 1990, when they were merged to form the republic of Germany as we know it today. This entity is modeled through the individual as Reunified Germany.

To model the validity constraints on the states of Germany during different periods, we use a temporal model similar to temporal RDF [Gutiérrez et al., 2005]. The model associates to each concept a specific time frame. The time frames assigned to the individuals that model different Germanies are illustrated in Figure 3.1 through intervals. The same applies to every property, instance and subclass relationship. Note, for example, how the instantiation relationship of Germany to Empire has a temporal interval from 1871 to 1918, while the one to Republic has a temporal interval from 1918 to 1945. It is important for such a database to contain no inconsistencies, i.e., situations like having an instantiation relationship with a temporal interval bigger than the interval of the class it instantiates. Although temporal RDF lacks this kind of axiomatization, a mechanism for consistency checking needs to be in place for the accurate representation of concepts and individuals in history. Our solution provides an axiomatization of the temporal RDF that guarantees the consistency of the historical information recorded in a database.

[1]The information of the four zones is not illustrated in the Figure 3.1

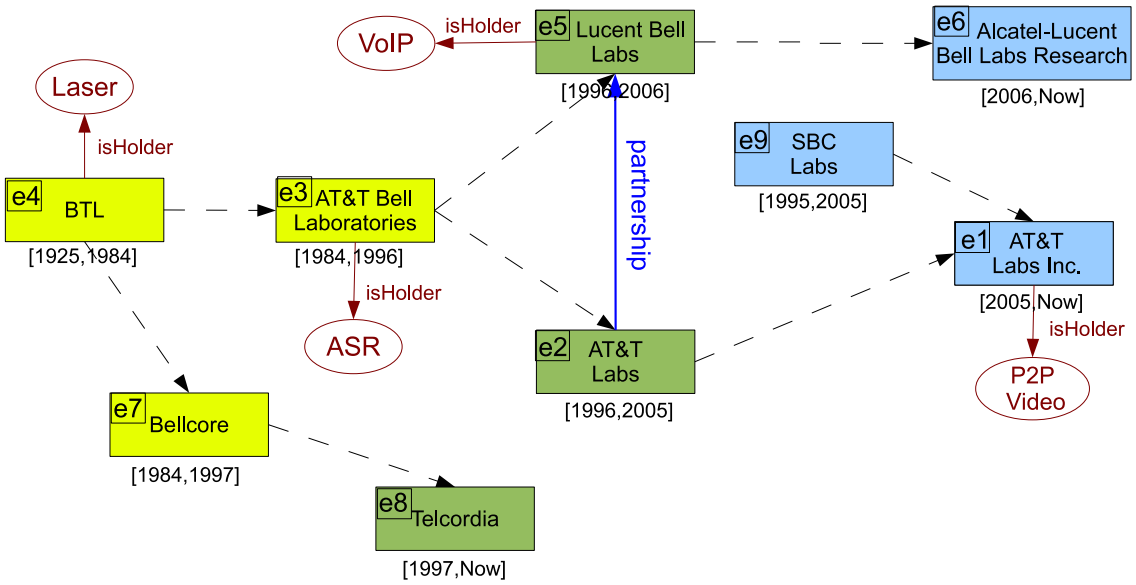


Figure 3.2: The history of the AT&T Labs.

Consider now the case of a historian who wants to study the history of Germany. The historian may be interested, for example, in all the leaders and constituents of Germany throughout its history. Using traditional query mechanisms, the historian will only be able to retrieve the individual Germany. Using keyword based techniques, she may be able to also retrieve the remaining individuals modeling Germany at different times, but this under the assumption that each such individual contains the keyword "Germany". Applying terminology evolution [Tahmasebi et al., 2008], it is possible to infer that the four terms for Germany, i.e., Germany, East Germany, West Germany, and Reunified Germany refer to related concepts regardless of the keywords that appear in the terms. Yet, in neither case the historian will be able to reconstruct how the constituents of Germany have changed over time. She will not be able to find that the East and West Germany were made by splitting the pre-war Germany and its parts, neither that East and West Germany were the same two that merged to form the Reunified Germany. We propose the use of explicit constructs to allow the modeling of evolution in databases, as exemplified in Figure 3.1(b) introducing split, join and part-of relationships.

Consider the case when the user cannot formulate queries using evolution constructs or if the conceptual model that the user has in mind in terms of evolution granularity is different from the one that actually exists in the data repository. Normally it will not be possible to answer any queries. To better illustrate the problem, let us consider the history of AT&T, a company that over the years has gone through a large number of break-ups, mergers and acquisitions. We consider a new evolution graph (see Figure 3.2) for evolution-unaware queries because it is more simple and has data peculiarities which allow us to demonstrate the details of our approach. The company was founded in 1925 under the name Bell Telecommunication Laboratories (BTL). In 1984, it was split into Bellcore (to become Telcordia in 1997) and AT&T Bell Laboratories. The latter existed until 1996 when it was split into Bell Labs, that was bought by Lucent, and to

AT&T Labs. The Lucent Bell Labs became Alcatel-Lucent Bell Labs Research in 2006 due to the take-over of Lucent by Alcatel. Furthermore, due to the take-over of AT&T by SBC in 2005, the AT&T Labs were merged with the SBC Labs to form the new AT&T Inc. Labs. Despite being research labs of different legal entities, Lucent and AT&T Labs have actually maintained a special partnership relationship. All these different labs have produced a large number of inventions, as the respective patents can demonstrate. Examples of such inventions are VoIP (Voice over Internet Protocol), the ASR (Automatic Speech Recognition), the P2P (Peer-to-Peer) Video and the laser. A graphical illustration of the above can be found in Figure 3.2 where the labs are modeled by rectangles and the patents by ovals.

Assume now a temporal database that models the above information. For simplicity we do not distinguish among the different kinds of evolution relationships (split, merger, and so on). Consider a user who is interested in finding the lab that invented the laser and the ASR patent. It is true that these two patents have been filed by two different labs, the AT&T Bell Labs and the AT&T Labs Inc. Thus, the query will return no results. However, it can be noticed that the latter entity is an evolution of the former. It may be the case that the user does not have the full knowledge of the way the labs have changed or, in her own mind, the two labs are still considered the same. We argue that instead of expecting from the user to know all the details of how the concept has evolved and the way the data has been stored, which means that the user's conceptual model should match the one of the database, we'd like the system to try to match the user's conceptual model instead. This means that the system should have the evolution relationships represented explicitly and take them into account when evaluating a query. In particular, we want the system to treat the AT&T Bell Labs, the AT&T Labs Inc, and the AT&T Labs as one unified (virtual) entity. That unified entity is the inventor of both the laser and the ASR, and should be the main element of the response to the user's query.

Of course, the query response is based on the assumption that the user did not intend to distinguish between the three aforementioned labs. Since this is an assumption, it should be associated with some degree of confidence. Such a degree can be based, for instance, on the number of labs that had to be merged in order to produce the answer. A response that involves 2 evolution-related entities should have higher confidence than one involving 4.

As a similar example, consider a query asking for all the partners of AT&T Labs Inc. Apart from those explicitly stated in the data (in the specific case, none), a traversal of the history of the labs can produce additional items in the answer, consisting of the partners of its predecessors. The further this traversal goes, the less likely it is that this is what the user wanted; thus, the confidence of the answer that includes the partners of its predecessors should be reduced. Furthermore, if the evolution relationships have also an associated degree of confidence, i.e., less than 100% certainty, the confidence computation of the answers should take this into consideration as well.

3.3 Supporting Evolution-Aware Queries

This section studies queries which are agnostic to evolution details, namely the evolution-aware queries. In particular, first we introduce the temporal database (Section 3.3.1), second we describe the evolution modeling technique (Section 3.3.2), finally we provide a formal definition of the query language (Section 3.3.3) and briefly talk about its evaluation strategy (Sec-

tion 3.3.4).

3.3.1 Temporal Databases

We consider an RDF-like data model. The model is expressive enough to represent ER models and the majority of ontologies and schemas that are used in practice [Lenzerini, 2002]. It does not include certain OWL Lite features such as *sameAs* or *equivalentClass*, since these features have been considered to be outside of the main scope of this work and their omission does not restrict the functionality of the model.

We assume the existence of an infinite set of *resources* $\mathcal{U}$, each with a *unique resource identifier* (URIs), and a set of *literals* $\mathcal{L}$. A *property* is a relationship between two resources. Properties are treated as resources themselves. We consider the existence of the special properties: rdfs:type, rdfs:domain, rdfs:range, rdfs:subClassOf and rdfs:subPropertyOf, which we refer to as type, dom, rng, subc, and subp, respectively. The set $\mathcal{U}$ contains three special resources: rdfs:Property, rdfs:Class and rdf:Thing, which we refer to as Prop, Class and Thing, respectively. The semantics of these resources as well as the semantics of the special properties are those defined in RDFS [W3C, 2004]. Resources are described by a set of triples that form a database.

Definition 3.3.1. *A* database Σ *is a tuple* $\langle U, L, T\rangle$*, where* $U \subseteq \mathcal{U}$*,* $L \subseteq \mathcal{L}$*,* $T \subseteq \mathcal{U}\times\mathcal{U}\times\{\mathcal{U}\cup\mathcal{L}\}$*, and* U *contains the resources* rdfs:Property*,* rdfs:Class*, and* rdf:Thing*. The set of* classes *of the database* Σ *is the set* $C=\{x \mid \exists\langle x, \text{type}, \text{rdfs:Class}\rangle \in T\}$*. Similarly, the set of* properties *is the set* $P=\{x \mid \exists\langle x, \text{type}, \text{rdfs:Property}\rangle \in T\}$*. The set* P *contains the RDFS properties* type*,* dom*,* rng*,* subc*, and* subp*. A resource* i *is said to be an* instance *of a class* $c \in C$ *(or of type* c*) if* $\exists\langle i, \text{type}, c\rangle \in T$*. The set of* instances *is the set* $I=\{i \mid \exists\langle i, \text{type}, y\rangle \in T\}$*.* ∎

A database can be represented as a hypergraph called an *RDF graph*. In the rest of the chapter, we will use the terms database and RDF graph equivalently.

Definition 3.3.2. *An* RDF graph *of a database* Σ *is an hypergraph in which nodes represent resources and literals and the edges represent triples.* ∎

Example 3.3.3. *Figure 3.1(a) is an illustration of an RDF Graph. The nodes* Berlin *and* Germany *represent resources. The edge labeled* part-of *between them represents the triple* ⟨Berlin,part-of,Germany⟩*. The label of the edge, i.e.,* part-of*, represents a property.*

To support the temporal dimension in our model, we adopt the approach of Temporal RDF [Gutiérrez et al., 2005] which extends RDF by associating to each triple a time frame. Unfortunately, this extension is not enough for our goals. We need to add time semantics not only to relationships between resources (what the triples represent), but also to resources themselves by providing temporal-varying classes and individuals. This addition and the consistency conditions we introduce below are our temporal extensions to the temporal RDF data model.

We consider time as a discrete, total order domain $\mathbb{T}$ in which we define different granularities. Following [Dyreson et al., 2000], a *granularity* is a mapping from integers to *granules* (i.e., subsets of the time domain $\mathbb{T}$) such that contiguous integers are mapped to non-empty granules and granules within one granularity are totally ordered and do not overlap. Days and months

are examples of two different granularities, in which each granule is a specific day in the former and a month in the latter. Granularities define a lattice in which granules in some granularities can be aggregated in larger granules in *coarser* granularities. For instance, the granularity of months is coarser than that of days because every granule in the former (a month) is composed of an integer number of granules in the latter (days). In contrast, months are not coarser (nor finer) than weeks.

Even though we model time as a point-based temporal domain, we use intervals as abbreviations of sets of instants whenever possible. An ordered pair $[a, b]$ of time points, with a, b granules in a granularity, and $a \leq b$, denotes the closed interval from a to b. As in most temporal models, the current time point will be represented with the distinguished word *Now*. We will use the symbol $\mathcal{T}$ to represent the infinite set of all the possible temporal intervals over the temporal domain $\mathbb{T}$, and the expressions $i.start$ and $i.end$ to refer to the starting and ending time points of an interval i. Given two intervals i_1 and i_2, we will denote by $i_1 \sqsubseteq i_2$ the containment relationship between the intervals in which $i_2.start \leq i_1.start$ and $i_1.end \leq i_2.end$.

Two types of temporal dimensions are normally considered: *valid* time and *transaction* time. Valid time is the time when data is valid in the modeled world whereas transaction time is the time when data is actually stored in the database. Concept evolution is based on valid time.

Definition 3.3.4. *A* temporal database Σ_T *is a tuple* $\langle U, L, T, \tau \rangle$*, where* $\langle U, L, T \rangle$ *is a database and* τ *is function that maps every resource* $r \in U$ *to a temporal interval in* $\mathcal{T}$*. The temporal interval is also referred to as the* lifespan *of the resource. The expressions* $r.start$ *and* $r.end$ *denote the start and end points of the interval of* r*, respectively. The* temporal graph *of* Σ_T *is the RDF graph of* $\langle U, L, T \rangle$ *enhanced with the temporal intervals on the edges and nodes.* ∎

For a temporal database to be semantically meaningful, the lifespans of the resources need to satisfy certain conditions. For instance, it is not logical to have an individual with a lifespan that does not contain any common time points with the lifespan of the class it belongs to. Temporal RDF does not provide such a mechanism, thus, we are introducing the notion of a consistent temporal database.

Definition 3.3.5. *A* consistent temporal database *is a temporal database* $\Sigma_\tau = \langle U, L, T, \tau \rangle$ *that satisfies the following conditions:*

1. $\forall r \in L \cup \{\mathsf{Prop}, \mathsf{Class}, \mathsf{Thing}, \mathsf{type}, \mathsf{dom}, \mathsf{rng}, \mathsf{subc}, \mathsf{subp}\}$: $\tau(r) = [0, Now]$*;*
2. $\forall \langle d, p, r \rangle \in T : \tau(\langle d, p, r \rangle) \sqsubseteq \tau(d)$ *and* $\tau(\langle d, p, r \rangle) \sqsubseteq \tau(r)$*;*
3. $\forall \langle d, p, r \rangle \in T$ *with* $p \in \{\mathsf{type}, \mathsf{subc}, \mathsf{subp}\}$: $\tau(d) \sqsubseteq \tau(r)$.

∎

Intuitively, literals and the special resources and properties defined in RDFS need to be valid during the entire lifespan of the temporal database, which is $[0, Now]$ (Condition 1). In addition, the lifespan of a triple needs to be within the lifespan of the resources that the triple associates (Condition 2). Finally, the lifespan of a resource has to be within the lifespan of the class the resource instantiates, and any class or property needs to be within the lifespan of its superclasses or superproperties (Condition 3).

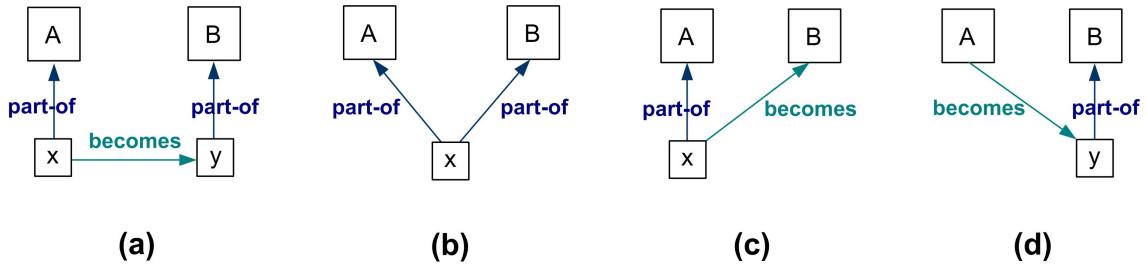


Figure 3.3: Liaison examples

3.3.2 Modeling Evolution

Apart from the temporal dimension that was previously described, two new dimensions need to be introduced to successfully model evolution: the mereological and the causal.

Mereology [Keet and Artale, 2008] is a sub-discipline in philosophy that deals with the ontological investigation of the part-whole relationship. It is used in our model to capture the parthood relationship between concepts in a way that is carried forward as concepts evolve. Such a relationship is modeled through the introduction of the special property part-of, which is reflexive, antisymmetric and transitive. A property part-of is defined from a resource x to a resource y if the concept modeled by resource x is part of the concept modeled by resource y. Note that the above definition implies that every concept is also a part of itself. When x is a part of y and $x \neq y$ we say that x is a *proper part* of y. Apart from this special semantics, part-of behaves as any other property in a temporal database. For readability and presentation reasons, we may use the notation $x \overset{\text{part-of}}{\longrightarrow} y$ to represent the existence of a triple $<x,\text{part-of},y>$ in the set T of a temporal database Σ_τ.

To capture causal relationships, i.e., the interdependency between two resources, we additionally introduce the notion of becomes, which is an antisymmetric and transitive relation. For similar reasons as before, we use the notation $x \overset{\text{becomes}}{\longrightarrow} y$ to represent the fact that $(x, y) \in$ becomes. Intuitively, $x \overset{\text{becomes}}{\longrightarrow} y$ means that the concept modeled by resource y originates from the concept modeled by resource x. We require that $\tau(x).end < \tau(y).start$.

To effectively model evolution, we also need the notion of a *liaison*. A liaison between two concepts is another concept that keeps the former two linked together in time by means of part-of and becomes relationships. In other words, a liaison is part of at least one of the concepts it relates and has some causal relationship to a part of the other.

Definition 3.3.6 (Liaison)**.** *Let A, B be two concepts of a temporal database with $\tau(A).start < \tau(B).start$, and x, y concepts for which $x \overset{\text{part-of}}{\longrightarrow} A$ and $y \overset{\text{part-of}}{\longrightarrow} B$. A concept x (or y) is said to be a* liaison between A and B *if either $x \overset{\text{becomes}}{\longrightarrow} y$ or $x = y$.* ∎

The most general case of a liaison is graphically depicted in Figure 3.3 (a). The boxes A and B represent the two main concepts whereas the x and y represent two of their respective parts. Figure 3.3 (b) illustrates the second case of the definition in which x and y are actually the same concept. Figure 3.3 (c) (respectively, (d)) shows the special case in which y (respectively, x) is exactly the whole of B (respectively, A) rather than a proper part of it.

To model the different kinds of evolution events that may exist, we introduce four evolution terms: join, split, merge, and detach.
[join] The join term, denoted as join$(c_1 \dots c_n, c, t)$, models the fact that every part of a concept c born at time t comes from a part of some concept in $\{c_1,\dots,c_n\}$. In particular:

- $\tau(c).start{=}t$;
- $\forall x$ s.t. $x \xrightarrow{\text{part-of}} c$: $\exists c_i$ s.t. x is a liaison between c_i and c, or $x = c_i$, with $1{\leq}i{\leq}n$.

[split] The split term, denoted as split$(c, c_1 \dots c_n, t)$, models the fact that every part of a concept c ending at time t becomes the part of some concept in $\{c_1,\dots,c_n\}$. In particular:

- $\tau(c).end{=}t$;
- $\forall x$ s.t. $x \xrightarrow{\text{part-of}} c$: $\exists c_i$ s.t. x is a liaison between c and c_i, or $x = c_i$, with $1{\leq}i{\leq}n$.

[merge] The merge term, denoted as merge(c, c', t), models the fact that at least a part of a concept c ending at a time t becomes part of an existing concept c'. In particular:

- $\tau(c).end{=}t$;
- $\exists x$ s.t. $x \xrightarrow{\text{part-of}} c'$ and x is a liaison between c and c'.

[detach] The detach term, denoted as detach(c, c', t), models the fact that the new concept c' is formed at a time t with at least one part from c. In particular:

- $\tau(c').start{=}t$;
- $\exists x$ s.t. $x \xrightarrow{\text{part-of}} c$ and x is a liaison between c and c'.

Note that in each evolution term there is only one concept whose lifespan has necessarily to start or end at the time of the event. For instance, we could use a join to represent the fact that different countries joined the European Union (EU) at different times. The information of the period in which each country participated in the EU is given by the interval of each respective part-of property.

We record the becomes relation and the evolution terms in the temporal database as *evolution triples* $\langle c, term, c' \rangle$, where $term$ is one of the special *evolution properties* becomes, join, split, merge, and detach. Evolution properties are *meta-temporal*, i.e., they describe how the temporal model changes, and thus their triples do not need to satisfy the consistency conditions in Definition 3.3.5. A temporal database with a set of evolution properties and triples defines an *evolution base*.

Definition 3.3.7. *An* evolution base Σ_T^E *is a tuple* $\langle U, L, T, E, \tau \rangle$*, where* $\langle U, L, T, \tau \rangle$ *is a temporal database,* U *contains a set of evolution properties, and* E *is a set of evolution triples. The* evolution graph *of* Σ_T^E *is the temporal graph of* $\langle U, L, T, \tau \rangle$ *enhanced with edges representing the evolutions triples.* ∎

The time in which the evolution event took place does not need to be recorded explicitly in the triple since it can be retrieved from the lifespan of the involved concepts. For instance, detach(Kingdom of the Netherlands, Belgium, 1831) is modeled as the triple: ⟨Kingdom of the Netherlands, detach, Belgium⟩ with $\tau(\text{Belgium}).start = 1831$.

For recording evolution terms that involve more than two concepts, e.g. the join, multiple triples are needed. We assume that the terms are indexed by their time, thus, the set of (independent) triples that belong to the same terms can be easily detected since they will all share the same start or end time in the lifespan of the respective concept. For instance, split(Germany, East Germany, West Germany, 1949) is represented in our model through the triples ⟨Germany, split, East Germany⟩ and ⟨Germany, split, West Germany⟩ with $\tau(\text{East Germany}).start = \tau(\text{West Germany}).start = 1949$.

Note that the evolution terms may entail facts that are not explicitly represented in the database. For instance, the split of Germany into West and East implies the fact that Berlin, which is explicitly defined as part of Germany, becomes part of either East or West. This kind of reasoning is beyond the scope of the current work.

3.3.3 Query Language

To support evolution-aware querying we define a navigational query language to traverse temporal and evolution edges in an evolution graph. This language is analogous to nSPARQL [Pérez et al., 2008], a language that extends SPARQL with navigational capabilities based on nested regular expressions. nSPARQL uses four different axes, namely $\mathbf{self}$, $\mathbf{next}$, $\mathbf{edge}$, and $\mathbf{node}$, for navigation on an RDF graph and node label testing. We have extended the nested regular expressions constructs of nSPARQL with temporal semantics and a set of five *evolution axes*, namely $\mathbf{join}$, $\mathbf{split}$, $\mathbf{merge}$, $\mathbf{detach}$, and $\mathbf{becomes}$ that extend the traversing capabilities of nSPARQL to the evolution edges. The language is defined according to the following grammar:

$$exp := \mathbf{axis} \mid \mathbf{t\text{-}axis} :: a \mid \mathbf{t\text{-}axis} :: [exp] \mid exp[I] \mid exp/exp \mid exp|exp \mid exp^*$$

where a is a node in the graph, I is a time interval, and $\mathbf{axis}$ can be either $\mathbf{forward}$, $\mathbf{backward}$, $\mathbf{e\text{-}edge}$, $\mathbf{e\text{-}node}$, a $\mathbf{t\text{-}axis}$ or an $\mathbf{e\text{-}axis}$, with $\mathbf{t\text{-}axis} \in \{\mathbf{self}, \mathbf{next}, \mathbf{edge}, \mathbf{node}\}$ and $\mathbf{e\text{-}axis} \in \{\mathbf{join}, \mathbf{split}, \mathbf{merge}, \mathbf{detach}, \mathbf{becomes}\}$.

The evaluation of an evolution expression exp is given by the semantic function $\mathcal{E}$ defined in Figure 3.4. $\mathcal{E}[\![exp]\!]$ returns a set of tuples of the form $\langle x, y, I\rangle$ such that there is a path from x to y satisfying exp during interval I. For instance, in the evolution base of Figure 3.1, $\mathcal{E}[\![\mathbf{self} :: \text{Germany}/\mathbf{next} :: \text{head}/\mathbf{next} :: \text{type}]\!]$ returns the tuple ⟨Germany, Chancellor, $[1988, 2005]$⟩. It is also possible to navigate an edge from a node using the $\mathbf{edge}$ axis and to have a nested expression $[exp]$ that functions as a predicate which the preceding expression must satisfy. For example, $\mathcal{E}[\![\mathbf{self}[\mathbf{next} :: \text{head}/\mathbf{self} :: \text{Gerhard Schröder}]]\!]$ returns ⟨Reunified Germany, Reunified Germany, $[1990, 2005]$⟩ and ⟨West Germany, West Germany, $[1988, 1990]$⟩.

In order to support evolution expressions, we need to extend nSPARQL triple patterns with temporal and evolution semantics. In particular, we redefine the evaluation of an nSPARQL triple pattern $(?X, exp, ?Y)$ to be the set of triples $\langle x, y, I\rangle$ that result from the evaluation of the evolution expression exp, with the variables X and Y bound to x and y, respectively.

$$
\begin{array}{rcl}
\mathcal{E}[\![\textbf{self}]\!] & := & \{\langle x, x, \tau(x)\rangle \mid x \in U \cup L\} \\
\mathcal{E}[\![\textbf{self}::r]\!] & := & \{\langle r, r, \tau(r)\rangle\} \\
\mathcal{E}[\![\textbf{next}]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle x, z, y\rangle \in T\} \\
\mathcal{E}[\![\textbf{next}::r]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle x, r, y\rangle \in T\} \\
\mathcal{E}[\![\textbf{edge}]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle x, y, z\rangle \in T\} \\
\mathcal{E}[\![\textbf{edge}::r]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle x, y, r\rangle \in T\} \\
\mathcal{E}[\![\textbf{node}]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle z, x, y\rangle \in T\} \\
\mathcal{E}[\![\textbf{node}::r]\!] & := & \{\langle x, y, \tau(t)\rangle \mid t = \langle r, x, y\rangle \in T\} \\
\mathcal{E}[\![\textbf{e-edge}]\!] & := & \{\langle x, \textbf{e-axis}, [0, Now]\rangle \mid t = \langle x, \textbf{e-axis}, z\rangle \in E\} \\
\mathcal{E}[\![\textbf{e-node}]\!] & := & \{\langle \textbf{e-axis}, y, \tau(t)\rangle \mid t = \langle z, \textbf{e-axis}, y\rangle \in E\} \\
\mathcal{E}[\![\textbf{e-axis}]\!] & := & \{\langle x, y, [0, Now]\rangle \mid \exists\, t = \langle x, \textbf{e-axis}, y\rangle \in E\} \\
\mathcal{E}[\![\textbf{forward}]\!] & := & \bigcup_{\textbf{e-axis}} \mathcal{E}[\![\textbf{e-axis}]\!] \\
\mathcal{E}[\![\textbf{backward}]\!] & := & \bigcup_{\textbf{e-axis}} \mathcal{E}[\![\textbf{e-axis}^{-1}]\!] \\
\mathcal{E}[\![\textbf{self}::[exp]]\!] & := & \{\langle x, x, \tau(x) \cap I\rangle \mid x \in U \cup L, \exists \langle x, z, I\rangle \in \mathcal{P}[\![exp]\!], \tau(x) \cap I \neq \emptyset\} \\
\mathcal{E}[\![\textbf{next}::[exp]]\!] & := & \{\langle x, y, \tau(t) \cap I\rangle \mid t = \langle x, z, y\rangle \in T, \exists \langle z, w, I\rangle \in \mathcal{P}[\![exp]\!], \tau(t) \cap I \neq \emptyset\} \\
\mathcal{E}[\![\textbf{edge}::[exp]]\!] & := & \{\langle x, y, \tau(t) \cap I\rangle \mid t = \langle x, y, z\rangle \in T, \exists \langle z, w, I\rangle \in \mathcal{P}[\![exp]\!], \tau(t) \cap I \neq \emptyset\} \\
\mathcal{E}[\![\textbf{node}::[exp]]\!] & := & \{\langle x, y, \tau(t) \cap I\rangle \mid t = \langle z, x, y\rangle \in T, \exists \langle z, w, I\rangle \in \mathcal{P}[\![exp]\!], \tau(t) \cap I \neq \emptyset\} \\
\mathcal{E}[\![\textbf{axis}^{-1}]\!] & := & \{\langle x, y, \tau(t)\rangle \mid \langle y, x, \tau(t)\rangle \in \mathcal{E}[\![\textbf{axis}]\!]\} \\
\mathcal{E}[\![\textbf{t-axis}^{-1}::r]\!] & := & \{\langle x, y, \tau(t)\rangle \mid \langle y, x, \tau(t)\rangle \in \mathcal{E}[\![\textbf{t-axis}::r]\!]\} \\
\mathcal{E}[\![\textbf{t-axis}^{-1}::[exp]]\!] & := & \{\langle x, y, \tau(t)\rangle \mid \langle y, x, \tau(t)\rangle \in \mathcal{E}[\![\textbf{t-axis}::[exp]]\!]\} \\
\mathcal{E}[\![exp[I]]\!] & := & \{\langle x, y, I \cap I'\rangle \mid \langle x, y, I'\rangle \in \mathcal{E}[\![exp]\!] \text{ and } I \cap I' \neq \emptyset\} \\
\mathcal{E}[\![exp/e\text{-}exp]\!] & := & \{\langle x, y, I_2\rangle \mid \exists \langle x, z, I_1\rangle \in \mathcal{E}[\![exp]\!], \exists \langle z, y, I_2\rangle \in \mathcal{E}[\![e\text{-}exp]\!]\} \\
\mathcal{E}[\![exp/t\text{-}exp]\!] & := & \{\langle x, y, I_1 \cap I_2\rangle \mid \exists \langle x, z, I_1\rangle \in \mathcal{E}[\![exp]\!], \exists \langle z, y, I_2\rangle \in \mathcal{E}[\![t\text{-}exp]\!] \text{ and} \\
 & & I_1 \cap I_2 \neq \emptyset\} \\
\mathcal{E}[\![exp_1|exp_2]\!] & := & \mathcal{E}[\![exp_1]\!]\ cup \mathcal{E}[\![exp_2]\!] \\
\mathcal{E}[\![exp^*]\!] & := & \mathcal{E}[\![\textbf{self}]\!] \cup \mathcal{E}[\![exp]\!] \cup \mathcal{E}[\![exp/exp]\!] \cup \mathcal{E}[\![exp/exp/exp]\!] \cup \ldots \\
 & & \\
\mathcal{P}[\![e\text{-}exp]\!] & := & \mathcal{E}[\![e\text{-}exp]\!] \\
\mathcal{P}[\![t\text{-}exp]\!] & := & \mathcal{E}[\![t\text{-}exp]\!] \\
\mathcal{P}[\![t\text{-}exp/exp]\!] & := & \{\langle x, y, I_1 \cap I_2\rangle \mid \exists \langle x, z, I_1\rangle \in \mathcal{E}[\![t\text{-}exp]\!], \exists \langle z, y, I_2\rangle \in \mathcal{E}[\![exp]\!] \text{ and} \\
 & & I_1 \cap I_2 \neq \emptyset\} \\
\mathcal{P}[\![e\text{-}exp/exp]\!] & := & \{\langle x, y, I_1\rangle \mid \exists \langle x, z, I_1\rangle \in \mathcal{E}[\![e\text{-}exp]\!], \exists \langle z, y, I_2\rangle \in \mathcal{E}[\![exp]\!]\} \\
\mathcal{P}[\![exp_1|exp_2]\!] & := & \mathcal{E}[\![exp_1|exp_2]\!] \\
\mathcal{P}[\![exp^*]\!] & := & \mathcal{E}[\![exp^*]\!] \\
 & & \\
t\text{-}exp & \in & \{\textbf{t-axis}, \textbf{t-axis}::r, \textbf{t-axis}::[exp], \textbf{t-axis}[I]\} \\
e\text{-}exp & \in & \{\textbf{e-axis}, \textbf{e-axis}::[exp], \textbf{e-axis}[I], \textbf{forward}, \textbf{backward}\}
\end{array}
$$

Figure 3.4: Formal semantics of nested evolution expressions

In particular:

$$
\begin{aligned}
\mathcal{E}[\![(?X, exp, ?Y)]\!] := \{(\theta(?X), \theta(?Y)) \mid\ & \theta(?X) = x \text{ and} \\
& \theta(?Y) = y \text{ and } \langle x, y, I\rangle \in \mathcal{E}[\![exp]\!]\}
\end{aligned}
$$

Our language includes all nSPARQL operators such as AND, OPT, UNION and FILTER with the same semantics as in nSPARQL. For instance:

$$
\mathcal{E}[\![(P_1 \text{ AND } P_2)]\!] := \mathcal{E}[\![(P_1)]\!] \bowtie \mathcal{E}[\![(P_2)]\!]
$$

where P_1 and P_2 are triple patterns and $\bowtie$ is the join on the variables P_1 and P_2 have in common.

A complete list of all the nSPARQL operators and their semantics is given by [Pérez et al., 2008].

3.3.4 Query Evaluation

The query language presented in the previous section is based on the concepts of nSPARQL and can be implemented as an extension of it by creating the appropriate query rewriting procedures that implement the semantics of Figure 3.4. Since our focus in query evaluation strategies is mainly on the evolution-unaware queries, we will not elaborate further on this.

3.4 Supporting Evolution-Unaware Queries

In this section we discuss the evolution-unaware queries. In particular, we present the model of evolution in Section 3.4.1, then we show a number of query evaluation techniques starting from naive ways and ending up with our solution (Section 3.4.2). In Section 3.4.3 we introduce the Steiner forest algorithm, which constitutes the core of our evaluation strategy, along with an optimization technique.

3.4.1 Modeling Evolution

To support queries that are unaware of the evolution relationships we need to construct a mechanism that performs various kinds of reasoning in a way transparent to the user. This reasoning involves the consideration of a series of data structures associated through the evolution relationships as one unified concept. For simplicity of the presentation, and also to abstract from the peculiarities of RDF, in what follows we use a *concept model* [Dalvi et al., 2009] as our data model. Furthermore, we do not consider separately the different kinds of evolution events but we consider them all under one unified relationship that we call *evolve*. This allows us to focus on different aspects of our proposal without increasing its complexity.

The fundamental component of the model is the *concept*, which is used to model a real world object. A concept is a data structure consisting of a unique identifier and a set of attributes. Each attribute has a name and a value. The value of an attribute can be an atomic value or a concept identifier. More formally, assume the existence of an infinite set of concept identifiers $\mathcal{O}$, an infinite set of names $\mathcal{N}$ and an infinite set of atomic values $\mathcal{V}$.

Definition 3.4.1. *An* attribute *is a pair* $\langle n,v \rangle$*, with* $n \in \mathcal{N}$ *and* $v \in \mathcal{V} \cup \mathcal{O}$*. Attributes for which* $v \in \mathcal{O}$ *are specifically referred to as* associations*. Let* $\mathcal{A} = \mathcal{N} \times \{\mathcal{V} \cup \mathcal{O}\}$ *be the set of all the possible attributes. A* concept *is a tuple* $\langle id, A \rangle$ *where* $A \subseteq \mathcal{A}$*, is finite, and* $id \in \mathcal{O}$*. The id is referred to as the* concept identifier *while the set* A *as the set of* attributes *of the concept.* ∎

We will use the symbol $\mathcal{E}$ to denote the set of all possible concepts that exist and we will also assume the existence of a Skolem function Sk [Hull and Yoshikawa, 1990]. Recall that a Skolem function is a function that guarantees that different arguments are assigned different values. Each concept is uniquely identified by its identifier, thus, we will often use the terms *concept* and *concept identifier* interchangingly if there is no risk of confusion. A *database* is a collection of concepts, that is closed in terms of associations between the concepts.

Definition 3.4.2. *A* database *is a finite set of concepts $E \subseteq \mathcal{E}$ such that for each association $\langle n, e' \rangle$ of a concept $e \in E$: $e' \in E$.* ∎

As a query language we adopt a datalog style language. A query consists of a head and a body. The body is a conjunction of atoms. An atom is an expression of the form $e(n_1{:}v_1, n_2{:}v_2, \ldots, n_k{:}v_k)$ or an arithmetic condition such as $=$, $\leq$, etc. The head is always a non-arithmetic atom. Given a database, the body of the query is said to be true if all its atoms are true. A non-arithmetic atom $e(n_1{:}v_1, n_2{:}v_2, \ldots, n_k{:}v_k)$ is true if there is a concept with an identifier e and attributes $\langle n_i, v_i \rangle$ for every $i{=}1..k$. When the body of a query is true, the head is also said to be true. If a head $e(n_1{:}v_1, n_2{:}v_2, \ldots, n_k{:}v_k)$ is true, the answer to the query is a concept with identifier e and attributes $\langle n_1{:}v_1 \rangle$, $\langle n_2{:}v_2 \rangle$, $\ldots$, $\langle n_k{:}v_k \rangle$.

The components e, n_i and v_i, for $i{=}1..k$ of any atom in a query can be either a constant or a variable. Variables used in the head or in arithmetic atoms must also be used in some non-arithmetic atom in the body. If a variable is used at the beginning of an atom, it is bound to concept identifiers. If the variable is used inside the parenthesis but before the ":" symbol, it is bound to attribute names, and if the variable is in the parenthesis after the ":" symbol, it is bound to attribute values. A variable assignment in a query is an assignment of its variables to constants. A *true assignment* is an assignment that makes the body of the query true. The answer set of a query involving variables is the union of the answers produced by the query for each true assignment.

Example 3.4.3. *Consider the query:*
$\$x(isHolder{:}\$y)$*:-*$\$x(name{:}'AT\&TLabsInc.', isHolder{:}\$y)$
that looks for concepts called "AT&T Labs Inc." and are holders of a patent. For every such concept that is found, a concept with the same identifier is produced in the answer set and has an attribute isHolder with the patent as a value. ∎

In order to model evolution we need to model the lifespan of the real world objects that the concepts represent and the evolution relationship between them. For the former, we assume that we have a temporal database, i.e., each concept is associated to a time period (see Section 3.3.1); however, this is not critical for the evolution-unaware queries so we will omit that part from the following discussions. To model the evolution relationship for evolution-unaware queries we consider a special association that we elevate into a first-class citizen in the database. We call this association an *evolution relationship*. Intuitively, an evolution relationship from one concept to another is an association indicating that the real world object modeled by the latter is the result of some form of evolution of the object modeled by the former. Note that the whole family of evolution operators from Section 3.3.2 is reduced to only one relationship.

In the next, with the abuse of notation we re-introduce the notion of evolution database with respect to the evolution-unaware queries. This allows us to focus only on the parts of our data model which are essential to the evolution-unaware queries. In Figure 3.2, the dotted lines between the concepts illustrate evolution relationships. A database with evolution relationships is an *evolution database*.

Definition 3.4.4. *An* evolution database *is a tuple $\langle E, \Omega \rangle$, such that $\langle E \rangle$ is a database and Ω is a partial order relation over E. An* evolution relationship *is every association $(e_1, e_2) \in \Omega$.* ∎

Given an evolution database, one can construct a directed acyclic graph by considering as nodes the concepts and as edges its evolution relationships. We refer to this graph as the *evolution graph* of the database.

Our proposal is that concepts representing different evolution phases of the same real world object can be considered as one for query answering purposes. To formally describe this idea we introduce the notion of *coalescence*. Coalescence is defined only on concepts that are connected through a series of evolution relationships; the coalescence of those concepts is a new concept that replaces them and has as attributes the union of their attributes (including associations).

Definition 3.4.5. *Given an evolution database* $\langle E,\Omega\rangle$*, The* coalescence *of two concepts* $e_1{:}\langle id_1, A_1\rangle$, $e_2{:}\langle id_2, A_2\rangle \in E$*, connected through an evolution relationship* ev *is a new evolution database* $\langle E',\Omega'\rangle$ *such that* $\Omega'=\Omega-ev$ *and* $E'=(E - \{e_1, e_2\}) \cup \{e_{new}\}$*, where* $e_{new}{:}\langle id_{new}, A_{new}\rangle$ *is a new concept with a fresh identifier* $id_{new}=Sk(id_1, id_2)$ *and* $A_{new}=A_1 \cup A_2$*. Furthermore, each association* $\langle n, id_1\rangle$ *or* $\langle n, id_2\rangle$ *of an concept* $e \in E$*, is replaced by* $\langle n, id_{new}\rangle$*. The relationship between the two databases is denoted as* $\langle E,\Omega\rangle \xrightarrow{ev} \langle E',\Omega'\rangle$ ∎

The Skolem function that we have mentioned earlier defines a partial order among the identifiers, and this partial order extends naturally to concepts. We call that order *subsumption*.

Definition 3.4.6. *An identifier* id_1 *is said to be* subsumed *by an identifier* id*, denoted as* $id_1 \dot{\prec} id$ *if there is some identifier* $id_x \neq id$ *and* $id_x \neq id_1$ *such that* $id=Sk(id_1, id_x)$*. A concept* $e_1=\langle id_1, A_1\rangle$ *is said to be* subsummed *by a concept* $e_2=\langle id_2, A_2\rangle$*, denoted as* $e_1 \dot{\prec} e_2$*, if* $id_1 \dot{\prec} id_2$ *and for every attribute* $\langle n, v_1\rangle \in A_1$ *there is attribute* $\langle n, v_2\rangle \in A_2$ *such that* $v_1=v_2$ *or, assuming that the attribute is an association,* $v_1 \dot{\prec} v_2$*.* ∎

Given an evolution database $\langle E,\Omega\rangle$, and a set $\Omega_s \subseteq \Omega$ one can perform a series of consecutive coalescence operations, each one coalescing the two concepts that an evolution relationship in the Ω_s associates.

Definition 3.4.7. *Given an evolution database* $D{:}\langle E,\Omega\rangle$ *and a set* $\Omega_s=\{m_1, m_2, \ldots, m_m\}$ *such that* $\Omega_s \subseteq \Omega$*, let* D_m *be the evolution database generated by the sequence of coalescence operations* $D \xrightarrow{m_1} D_1 \xrightarrow{m_2}, \ldots, \xrightarrow{m_m} D_m$*. The* possible world *of* D *according to* Ω_s *is the database* D_{Ω_s} *generated by simply omitting from* D_m *all its evolution relationships.* ∎

Intuitively, a set of evolution relationships specifies sets of concepts in a database that should be considered as one, while the possible world represents the database in which these concepts have actually been coalesced. Our notion of a possible world is similar to the notion of a possible worlds in probabilistic databases [Dalvi and Suciu, 2007]. Theorem 3.4.8 is a direct consequence of the definition of a possible world.

Theorem 3.4.8. *The possible world of an evolution database* $D{:}\langle E,\Omega\rangle$ *for a set* $\Omega_s \subseteq \Omega$ *is unique.* ∎

Due to this uniqueness, a set Ω_s of evolution relationships of a database can be used to refer to the possible world.

According to the definition of a possible world, an evolution database can be seen as a shorthand of a set of databases, i.e., its possible worlds. Thus, a query on an evolution database can be seen as a shorthand for a query on its possible worlds. Based on this observation we define the semantics of query answering on an evolution database.

$x	$y	Possible World	Answer	Cost
e1	P2P Video	$\emptyset$	e1(isHolder:"P2P Video")	0
Sk(e1,e2)	**P2P Video**	**e1,e2**	**Not generated**	**1**
Sk(e1,e2,e3)	**P2P Video**	**e1,e2,e3**	**Not generated**	**2**
Sk(e1,e2,e3)	ASR	e1,e2,e3	Sk(e1,e2,e3)(isHolder:"ASR")	2
Sk(e1,e2,e3,e4)	Laser	e1,e2,e3,e4	Sk(e1,e2,e3,e4)(isHolder:"Laser")	3
...	...	...	...	...

Table 3.1: A fraction of variable assignments for Example 3.4.11.

Definition 3.4.9. *The evaluation of a query q on an evolution database D is the union of the results of the evaluation of the query on every possible world D_c of D.* ∎

For a given query, there may be multiple possible worlds that generate the same results. To eliminate this redundancy we require every coalescence to be well-justified. In particular, our principle is that no possible world or variable assignment will be considered, unless it generates some new results in the answer set. Furthermore, among the different possible worlds that generate the same results in the answer set, only the one that requires the smaller number of coalescences will be considered. To support this, we define a subsumption relationship among the variable assignments across different possible worlds and we redefine the semantics of the evaluation of a query.

Definition 3.4.10. *Let h and h' be two variable assignment for a set of variables X. h' is said to be* subsumed *by h, denoted as $h' \subseteq h$ if $\forall x \in X$: $h(x)=h'(x)=constant$, or $h(x)=e$ and $h'(x)=e'$, with e and e' being concepts for which $e' \dot{\prec} e$ or $e=e'$.*
Given an evolution database D, let $\mathcal{W}$ be the set of its possible worlds. The answer to a query q is the union of the results of evaluating q on every database in $\mathcal{W}$. During the evaluation of q on a database in $\mathcal{W}$, true variable assignments that are subsumed by some other true variable assignment, even in other possible worlds, are not considered. ∎

It is natural to assume that not all possible worlds are equally likely to describe the database the user has in mind when she was formulating the query. We assume that the more a possible world differs from the original evolution database, the less likely it is to represent what the user had in mind. This is also in line with the minimality and well-justification principle described previously. We reflect this as a reduced confidence to the answers generated by the specific possible world and quantify it as a score assigned to each answer. One way to measure that confidence is to count how many evolution relationships have to be coalesced for the possible world to be constructed. The evolution relationships may also be assigned a weight reflecting the confidence to the fact that its second concept is actually an evolution of the first.

Example 3.4.11. *Consider again the query of Example 3.4.3 and assume that it is to be evaluated on the database of Figure 3.2. Table 3.1 illustrate a set of true variable assignments for some of the possible worlds of the database. The possible world on which each assignment is defined is expressed through its respective set Ω_s. The fourth column contains the result generated in the answer set from the specific assignment and the last column contains its respective cost, measured in number of coalesces that are required for the respective possible world to be*

generated from the evolution database. Note that the second and the third assignment (highlighted in bold), are redundant since they are subsumed by the first. ∎

The existence of a score for the different solutions, allows us to rank the query results and even implement a top-k query answering. The challenging task though is how to identify in an efficient way the possible worlds and more specifically the true variable assignments that lead into correct results.

3.4.2 Query Evaluation

In this section we present evaluation strategies for the evolution-unaware queries, namely the naive approach , the materializing all the possible worlds method, the materializing only the maximum world approach and, finally, the on-the-fly coalescence computations.

The naive approach

The straightforward approach in evaluating a query is to generate all the possible worlds and evaluate the query on each one individually. In the sequel, generate the union of all the individually produced results, eliminate duplication and remove answers subsumed by others. Finally, associate to each of the remaining answers a cost based on the coalescences that were performed in order to generate the possible world from which the answer was produced, and rank the answers according to that score. The generation of all possible worlds is a time consuming task. For an evolution database with an evolution graph of N edges, there are 2^N possible worlds. This is clearly a brute force solution, not desirable for online query answering.

Materializing all the possible worlds

Since the possible worlds do not depend on the query that needs to be evaluated, they can be pre-computed and stored in advance so that they are available at query time. Of course, as it is the case of any materialization technique, the materialized data need to be kept in sync with the evolution database when its data is modified. Despite the fact that this will require some effort, there are already well-known techniques for change propagation [Blakeley et al., 1986] that can be used. The major drawback, however, is the space overhead. A possible world contains all the attributes of the evolution database, but in fewer concepts. Given that the number of attributes are typically larger than the number of concepts, and that concepts associated with evolution relationships are far fewer than the total number of concepts in the database, we can safely assume that the size of a possible world will be similar to the one of the evolution database. Thus, the total space required will be 2^n times the size of the evolution database. The query answering time, on the other hand, will be 2^n times the average evaluation time of the query on a possible world.

Materializing only the maximum world

An alternative solution is to generate and materialize the possible world D_{max} generated by performing all possible coalescences. For a given evolution database $\langle E, \Omega \rangle$, this world is the one constructed according to the set of all evolution relationships in Ω. Any query that has

an answer in some possible world of the evolution database will also have an answer in this maximal possible world D_{max}. This solution work has two main limitations. First it does not follow our minimalistic principle and performs coalescences that are not needed, i.e., they do not lead to any additional results in the result set. Second, the generated world fails to include results that distinguish difference phases of the lifespan of a concept (phases that may have to be considered individual concepts) but the approach coalesces them in one just because they are connected through evolution relationships.

On-the-fly coalescence computations

To avoid any form of materialization, we propose an alternative technique that computes the answers on the fly by performing coalescences on a need-to-do basis. In particular, we identify the attributes that satisfy the different query conditions and from them the respective concepts to which they belong. If all the attributes satisfying the conditions are on the same concept, then the concept is added in the answer set. However, different query conditions may be satisfied by attributes in different concepts. In these cases we identify sets of concepts for each one of which the union of the attributes of its concepts satisfy all the query conditions. For each such a set, we coalesce all its concepts into one if they belong to the same connected component of the evolution graph. Doing the coalescence it is basically like creating the respective possible world; however, we generate only the part of that world that is necessary to produce an answer to the query. In more details, the steps of the algorithm are the following.

[Step 1: Query Normalization] We decompose every non-arithmetic atom in the body of the query that has more than one condition into a series of single-condition atoms. More specifically, any atom of the form $x(n_1{:}v_1, n_2{:}v_2, \ldots, n_k{:}v_k)$ is decomposed into a conjunction of atoms $x(n_1{:}v_1), x(n_2{:}v_2), \ldots, x(n_k{:}v_k)$.

[Step 2: Individual Variable Assignments Generation] For each non-arithmetic atom in the decomposed query, a list is constructed that contains assignments of the variables in the respective atom to constants that make the atom true. Assuming a total of N non-arithmetic atoms after the decomposition, let L_1, L_2, …, L_N be the generated lists. Each variable assignment actually specifies the part of the evolution database that satisfies the condition described in the atom.

[Step 3: Candidate Assignment Generation] The elements of the lists generated in the previous step are combined together to form complete variable assignments, i.e., assignments that involve every variable in the body of the query. In particular, the cartesian product of the lists is created. Each element in the cartesian product is a tuple of assignments. By construction, each such tuple will contain at least one assignment for every variable that appears in the body of the query. If there are two assignments of the same attribute bound variable to different values, the whole tuple is rejected. Any repetitive assignments that appear within each non-rejected tuple is removed to reduce redundancy. The result is a set of variable assignments, one from each of the tuples that have remained.

[Step 4: Arithmetic Atom Satisfaction Verification] Each assignment generated in the previous step for which there is at least one arithmetic atom not evaluating to true, is eliminated from

the list.

[Step 5: Candidate Coalescence Identification] Within each of the remaining assignments we identify concept-bound variables that have been assigned to more than one values. Normally this kind of assignment evaluates always to false. However, we treat them as suggestions for coalescences, so that the assignment will become a *true assignment* (ref. previous Section). For each assignment h in the list provided by Step 4, the set $\mathcal{V}_h$=$\{V_{x_1}, V_{x_2}, \ldots, V_{x_k}\}$ is generated, where V_x is the set of different concepts that variable x has been assigned in assignment h. In order for the assignments of variable x to evaluate to true, we need to be able to coalesce the concepts in V_x. To do so, these concepts have to belong to the same connected component in the evolution graph of the database. If this is not the case, the assignment h is ignored.

[Step 6: Coalescence Realization & Cost Computation] Given a set $\mathcal{V}_h$=$\{V_{x_1}, V_{x_2}, \ldots, V_{x_k}\}$ for an assignment h among those provided by Step 5, we need to find the minimum cost coalescences that need to be done such that all the concepts in a set V_i, for i=$1..k$, are coalesced to the same concept. This will make the assignment h a *true* assignment, in which case the head of the query can be computed and an answer generated in the answer set. The cost of the answer will be the cost of the respective possible world, which is measured in terms of the number of coalescences that need to be performed. Finding the required coalescences for the set $\mathcal{V}_h$ that minimizes the cost boils down to the problem of finding a Steiner forest [Gassner, 2010],

Example 3.4.12. *Let us consider again the query of Example 3.4.3. In Step 1, its body will be decomposed into two parts:* $\$x(name{:}'AT\&TLabsInc.')$ *and* $\$x(isHolder{:}\$y)$. *For those two parts, during Step 2, the lists* L_1=$\{\{\$x{=}e1\}\}$ *and* L_2=$\{\{\$x{=}e1,\ \$y{=}'P2PVideo'\},\ \{\$x{=}e3,\ \$y{=}'ASR'\},\ \{\$x{=}e4,\ \$y{=}'Laser'\},\ \{\$x{=}e5,\ \$y{=}'VoIP'\}\}$ *will be created. Step 3 creates their cartesian product* L=$\{\{\$x{=}e1,\ \$x{=}e1,\ \$y{=}'P2PVideo'\},\ \{\$x{=}e1,\ \$x{=}e3,\ \$y{=}'ASR'\},\ \{\$x{=}e1,\ \$x{=}e4,\ \$y{=}'Laser'\},\ \{\$x{=}e1,\ \$x{=}e5,\ \$y{=}'VoIP'\}\}$. *The only attribute bound variable is* $\$y$ *but this is never assigned to more than one different value at the same time so nothing is eliminated. Since there are no arithmetic atoms, Step 4 makes no change either to the list* L. *If for instance, the query had an atom* $\$y{\neq}'VOIP'$ *in its body, then the last element of the list would have been eliminated. Step 5 identifies that the last three elements in* L *have the concept bound variable* $\$x$ *assigned to two different values; thus, it generates the candidate coalesences:* V_1=$\{e1, e3\}$, V_2=$\{e1, e4\}$ *and* V_3=$\{e1, e5\}$. *Step 6 determines that all three coalescences are possible. Concepts* $e1$, $e2$ *and* $e3$ *will be coalesced for* V_1, $e1$, $e2$, $e3$ *and* $e4$ *for* V_2, *and the* $e1$, $e2$, $e3$ *and* $e5$ *for* V_3. ∎

3.4.3 Steiner Forest Algorithm

The last step of the evaluation algorithm for evolution unaware query language takes as input a set of concept sets and needs to perform a series of coalesce operations such that all the concepts within each set will become one. To do so, it needs to find an interconnect on the evolution graph among all the concepts within each set. Note that the interconnect may involve additional concepts not in the set that unavoidably will also have to be coalesced with those in the set. Thus, it is important to find an interconnect that minimizes the total cost of the coalescences. The cost of a coalescence operation is the weight of the evolution relationship that connects the

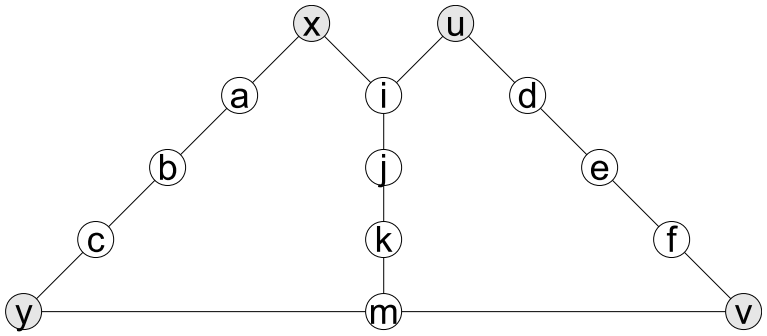


Figure 3.5: An illustration of the Steiner forest problem.

two concepts that are coalesced. Typically, that cost is equal to one, meaning that the total cost is actually the total number of coalescence operations that need to be performed. For a given set of concepts, this is known as the problem of finding the Steiner tree [Dreyfus and Wagner, 1972]. However, given a set of sets of concepts, it turns out that finding the optimal solution, i.e., the minimum cost interconnect of all the concepts, is not always the same as finding the Steiner tree for each of the sets individually. The specific problem is found in the literature as the Steiner forest problem [Gassner, 2010].

The difference in the case of the Steiner forest is that edges can be used by more than one interconnect. More specifically, the Steiner tree problem aims at finding a tree on an undirected weighted graph that connects all the nodes in a set and has the minimum cost. In contrast to the minimum spanning tree, a Steiner tree is allowed to contain intermediate nodes in order to reduce its total cost. The Steiner forest problem takes as input set of sets of nodes and needs to find a set of non-connected trees (branches) that make all the nodes in each individual set connected and the total cost is minimal, even if the cost of the individual trees are not always the minimal. We refer to these individual trees with the term *branches*. Figure 3.5 illustrates the difference through an example. Assuming that we have the graph shown in the figure and the two sets of nodes $\{x,y\}$ and $\{u,v\}$. Clearly, the minimum cost branch that connects nodes x and y is the one that goes through nodes a, b and c. Similarly the minimum cost branch that connects u and v is the one that goes through nodes e, f and g. Each of the two branches has cost 4 (the number of edges in the branch), thus, the total cost will be 8. However, if instead we connect all four nodes x, y, u and v through the tree that uses the nodes i, j, k and m, then, although the two nodes in each set are connected with a path of 5 edges, the total cost is 7.

Formally, the *Steiner forest* problem is defined as follows. Given a graph $G = \langle N, E \rangle$ and a cost function $f : E \rightarrow \mathbb{R}^+$, alongside a set of groups of nodes $\mathcal{V}= V_1, \ldots, V_L$, where $V_i \subseteq N$, find a set $C \subseteq E$ such that C forms a connected component that involves all the nodes of every group V_i and the $\sum_i f(c_i) \mid c_i \in C$ is minimal.

The literature contains a number of approximate solutions [Bhalotia et al., 2002, Kacholia et al., 2005, He et al., 2007] as well as a number of exact solution using Dynamic Programming [Dreyfus and Wagner, 1972, Ding et al., 2007, Kimelfeld and Sagiv, 2006] for the discovery of Steiner trees. However, for the Steiner forest problem (which is known to be NP-hard [Gassner, 2010]) although there are approximate solutions [Gassner, 2010], no optimal

Algorithm 1 Steiner tree algorithm

Require: graph G,$f : E \rightarrow \mathbb{R}^+$, groups $\mathcal{V}$= $V_1, \ldots, V_L$
Ensure: ST for each element in $flat(\mathcal{V})$

```
1: Q_T: priority queue sorted in the increasing order
2: Q_T ⇐ ∅
3: for all s_i ∈ maxflat(V) do
4:    enqueue T(s_i, {s_i}) into Q_T;
5: end for
6: while Q_T ≠ ∅ do
7:    dequeue Q_T to T(v, p);
8:    if p ∈ flat(V) then
9:       ST(p) = T(v, p)
10:   end if
11:   if ST has all values then
12:      return ST
13:   end if
14:   for all u ∈ N(v) do
15:      if T(v, p) ⊕ (v, u) < T(u, p) then
16:         T(u, p) ⇐ T(v, p) ⊕ (v, u);
17:         update Q_T with the new T(u, p);
18:      end if
19:   end for
20:   p_1 ⇐ p;
21:   for all p_2 s.t. p_1 ∩ p_2 = ∅ do
22:      if T(v, p_1) ⊕ T(v, p_2) < T(v, p_1 ∪ p_2) then
23:         T(u, p_1 ∪ p_2) ⇐ T(v, p_1) ⊕ T(v, p_2);
24:         update Q_T with the new T(u, p_1 ∪ p_2);
25:      end if
26:   end for
27: end while
```

algorithm has been proposed so far. In the current work we are making a first attempt toward that direction by describing a solution that is based on dynamic programming and is constructed by extending an existing Steiner tree discovery algorithm.

To describe our solution it is necessary to introduce the set $flat(\mathcal{V})$. Each element in $flat((\mathcal{V}))$ is a set of nodes created by taking the union of the nodes in a subset of $\mathcal{V}$. More specifically, $flat(\mathcal{V})$= $\{U \mid U$=$\bigcup_{V_i \in S} V_i$ with $S \subseteq \mathcal{V}\}$. Clearly $flat(\mathcal{V})$ has 2^L members. We denote by $maxflat(\mathcal{V})$ the maximal element in $flat(\mathcal{V})$ which is the set of all possible nodes that can be found in all the sets in $\mathcal{V}$, i.e., $maxflat(\mathcal{V})$=$\{n \mid n \in V_1 \cup \ldots \cup V_L\}$.

Our solution for the computation of the Steiner forest consists of two parts. In the first part, we compute the Steiner trees for every member of the $flat(\mathcal{V})$ set, and in the second part we use the computed Steiner trees to generate the Steiner forest on $\mathcal{V}$.

The state-of-the-art optimal (i.e., no approximation) algorithm for the Steiner tree problem is a dynamic programming solution developed in the context of keyword searching in relational data [Ding et al., 2007]. The algorithm is called the Dynamic Programming Best First (DPBF)

algorithm and is exponential in the number of input nodes and polynomial with respect to the size of graph. We extend DPBF in order to find a *set* of Steiner trees, in particular a Steiner tree for every element in $flat(\mathcal{V})$. The intuition behind the extension is that we initially solve the Steiner tree problem for the $maxflat(\mathcal{V})$ and continue iteratively until the Steiner trees for every element in $flat(\mathcal{V})$ has been computed. We present next a brief description of DPBF alongside our extension.

Let $T(v, \mathbf{p})$ denote the minimum cost tree rooted at v that includes the set of nodes $\mathbf{p} \subseteq maxflat(\mathcal{V})$ Note that by definition, the cost of the tree $T(s, maxflat(\mathcal{V}))$ is 0, for every $s \in maxflat(\mathcal{V})$.

Trees can be iteratively merged in order to generate larger trees by using the following three rules.

$$T(v, \mathbf{p}) = min(T_g(v, \mathbf{p}), T_m(v, \mathbf{p})) \tag{3.1}$$
$$T_g(v, \mathbf{p}) = min_{u \in N(v)}((v, u) \oplus T(u, \mathbf{p})) \tag{3.2}$$
$$T_m(v, \mathbf{p}_1 \cup \mathbf{p}_2) = min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset}(T(v, \mathbf{p}_1) \oplus T(v, \mathbf{p}_2)) \tag{3.3}$$

where $\oplus$ is an operator that merges two trees into a new one and $N(v)$ is the set of neighbour nodes of node v. In [Ding et al., 2007] it was proved that these equations are dynamic programming equations leading to the optimal Steiner tree solution for $maxflat(\mathcal{V})$ set of nodes. To find it, the DPBF algorithm employs the Dijkstra's shortest path search algorithm in the space of $T(v, \mathbf{p})$. The steps of the Steiner tree computation are shown in Algorithm 3.4.3. In particular, we maintain a priority queue Q_T that keeps in an ascending order the minimum cost trees that have been found at any given point in time. Naturally, a *dequeue* operation retrieves the tree with the minimal cost. Using the greedy strategy we look for the next minimal tree which can be obtained from the current minimal. In contrast to DPBF, we do not stop when the best tree has been found, i.e. when the solution for $maxflat(\mathcal{V})$ has been reached, but we keep collecting minimal trees (lines 7-10) until all elements in $flat(\mathcal{V})$ have been computed (lines 11-13). To prove that all the elements of $flat(\mathcal{V})$ are found during that procedure, it suffices to show that our extension corresponds to the finding of all the shortest paths for a single source in the Dijkstra's algorithm. The time and space complexity for finding the Steiner trees is $O(3^{\sum l_i}n + 2^{\sum l_i}((\sum l_i + logn)n + m))$ and $O(2^{\sum l_i}n)$, respectively, where n and m are the number of nodes and edges of graph G, and l_i is the size of the ith set V_i in the input of set $\mathcal{V}$ of the algorithm.

Once all the Steiner trees for $flat(\mathcal{V})$ have been computed, we use them to find the Steiner forest for $\mathcal{V}$. The Steiner forest problem has an optimal substructure and its subproblems overlap. This means that we can find a dynamic programming solution to it. To show this, first we consider the case for $L{=}1$, i.e., the case in which we have only one group of nodes. In that case finding the Steiner forest is equivalent to finding the Steiner tree for the single set of nodes that we have. Assume now that $L{>}1$, i.e., the input set $\mathcal{V}$ is $\{V_1, \ldots, V_L\}$, and that we have already computed all the Steiner forests for every set $\mathcal{V}' \subset \mathcal{V}$. Let $SF(\mathcal{V})$ denote the Steiner forest for an input set $\mathcal{V}$. We do not know the exact structure of $SF(\mathcal{V})$, i.e. how many branches it has and what elements of $\mathcal{V}$ are included in each. Therefore, we need to test all possible hypotheses of the forest structure, which are 2^L, and pick the one that has minimal cost. For instance, we assume that the forest has a branch that includes all nodes in V_1. The total cost of the forest with that assumption is the sum of the Steiner forest on V_1 and the Steiner forest for $\{V_2, \ldots, V_L\}$ which is a subset of $\mathcal{V}$, hence it is considered known. The Steiner forest on V_1 is actually a Steiner tree. This is based on the following lemma.

Algorithm 2 Steiner forest algorithm

Require: $G = \langle N, E\rangle$, $\mathcal{V}$= $\{V_1, \ldots, V_L\}$,$ST(s)$ $\forall s \in flat(\mathcal{V})$
Ensure: $SF(\mathcal{V})$
1: **for all** $V_i \in \mathcal{V}$ **do**
2: $SF(V_i) = ST(V_i)$
3: **end for**
4: **for** $i = 2$ to $L - 1$ **do**
5: **for all** $H \subset \mathcal{V}$ and $\mid H \mid = i$ **do**
6: $u \Leftarrow \infty$
7: **for all** $E \subseteq H$ and $E \neq \emptyset$ **do**
8: $u \Leftarrow min(u, ST(maxflat(E)) \oplus SF(H \setminus E))$
9: **end for**
10: $SF(H) \Leftarrow u$
11: **end for**
12: **end for**
13: $u \Leftarrow \infty$
14: **for all** $H \subseteq \mathcal{V}$ and $H \neq \emptyset$ **do**
15: $u \Leftarrow min(u, ST(maxflat(H)) \oplus SF(\mathcal{V} \setminus \mathcal{H}))$
16: **end for**
17: $SF(\mathcal{V}) \Leftarrow u$

Lemma 3.4.13. *Each branch of a Steiner forest is a Steiner tree.*

Proof. This proof is done by contradiction. Assuming that a branch of the forest is not a Steiner tree, it can be replaced with a Steiner tree and reduce the overall cost of the Steiner forest. This means that the initial forest was not minimal. □

We can formally express the above reasoning as:

$$SF(\mathcal{V}) = \min_{H \subseteq V}(ST(maxflat(H)) \oplus SF(\mathcal{V} \setminus H)) \tag{3.4}$$

Using the above equation in conjunction with the fact that $SF(\mathcal{V})$=$ST(V_1)$, if $\mathcal{V}$=$\{V_1\}$, we construct an algorithm (Algorithm 2) that finds the Steiner forest in a bottom-up fashion. The time and space requirements of the specific algorithm are $O(3^L - 2^L(L/2 - 1) - 1)$ and $O(2^L)$, respectively. Summing this with the complexities of the first part, it gives a total time complexity $O(3^{\sum l_i}n+2^{\sum l_i}((\sum l_i+logn)n+m))+3^L-2^L(L/2-1)-1)$ with space requirement $O(2^{\sum l_i}n + 2^L)$.

Query Evaluation Optimization

In the case of top-k query processing there is no need to actually compute all possible Steiner forests to only reject some of them later. It is important to prune as early as possible cases which are expected not to lead to any of the top-k answers. We have developed a technique that achieves this. It is based on the following lemma.

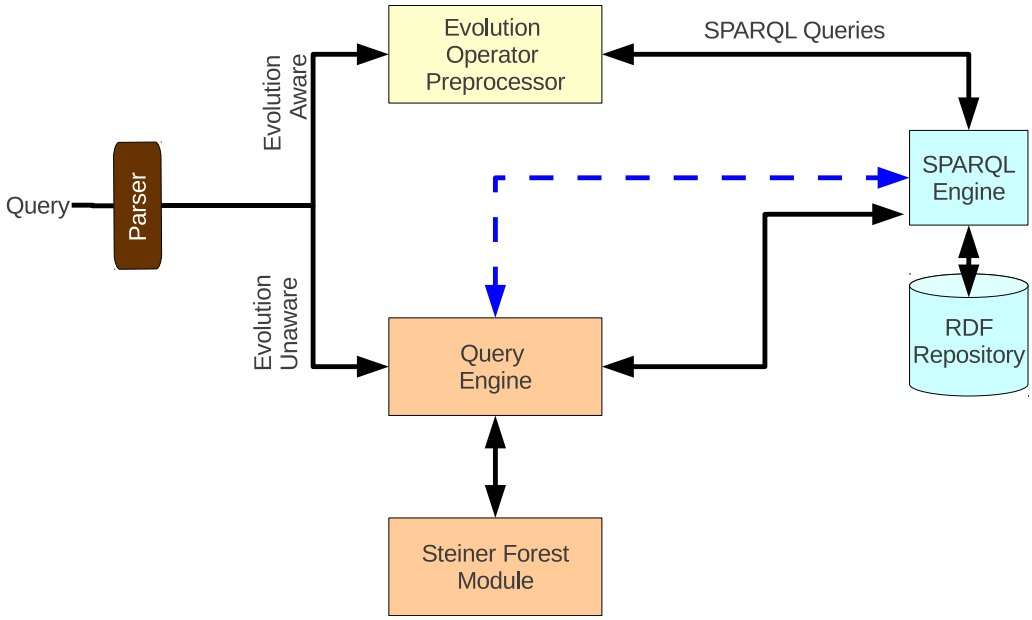


Figure 3.6: The TrenDS System Architecture

Lemma 3.4.14. *Given two sets of sets of nodes $\mathcal{V}'$ and $\mathcal{V}''$ on a graph G for which $\mathcal{V}' \subseteq \mathcal{V}''$:* $cost(SF(\mathcal{V}')) \leq cost(SF(\mathcal{V}''))$.

Proof. The proof is based on the minimality of a Steiner forest. Let SF(V') and SF(V'') be Steiner forests for V' and V'', with costs w_1 and w_2, respectively. If $cost(SF(\mathcal{V}'')) \leq cost(SF(\mathcal{V}'))$, then we can remove $V'' \backslash V'$ from V'' and cover V' with a smaller cost forest than $SF(V')$, which contradicts the fact that $SF(V')$ is a Steiner forest. □

To compute the top-k answers to a query we do the following steps. Assume that $B = \{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$ is a set of inputs for the Steiner forest algorithm. First, we find the $B_{min} \subseteq B$ such that for each $\mathcal{V}' \in B_{min}$ there is no $\mathcal{V}'' \in B$ such that $\mathcal{V}' \subset \mathcal{V}''$. Then, we compute the Steiner forest for each element in B_{min}. According to Lemma 3.4.14 and the construction procedure of B_{min} we ensure that the the Top-1 is among the computed Steiner forests. We remove the input which corresponds to that Top-1 answer from B and then we continue with the computation of Steiner forests to update B_{min}. The above steps are repeated until k answers have been found.

3.5 System Implementation

We have built a system in order to materialize the ideas described previously and see them in practice. The system has been implemented in Java, is called `TrenDS`, and the architecture of which is illustrated in Figure 3.6. It consists of four main components. One is the data repository for which it uses an RDF storage. The RDF storage has a SPARQL interface through which the data can be queried. It has no specific requirements thus, it can be easily replaced by any other RDF storage engine. The evolution relationships among the concepts are modeled as RDF attributes.

Once a query is issued to the system, it is first parsed by the *Query Parser*. The parser checks whether the query contains evolution related operators, in which case it forwards it to the *Evolution Operator Processor* module. The model is responsible for the implementation of the semantics of the evolution expressions as described in Figure 3.4. To achieve this it rewrites the query into a series of queries that contain no evolution operators, thus, they can be sent for execution to the repository. The results are collected back and sent to the user or the application that asked the query.

In the case in which we deal with the evolution-unaware queries, it is then sent to the *Query Engine* module. This module is responsible for implementing the whole evaluation procedure described in Section 3.4.2. In particular it first asks the repository and retrieves all the concepts that satisfy at least one of the query conditions (dotted line in Figure 3.6). Then it calls the *Steiner Forest* module to compute the different ways they can be coalesced in order to produce on-the-fly possible worlds. Finally, for each such a world, the results are generated and returned, alongside any additional fields that need to be retrieved by the repository. In the case in which only the top-k results are to be retrieved, the module activates the optimization algorithm described in Section 3.4.3 and prunes the number of Steiner forests that are to be computed.

Query Engine operates in four modes. The first is the *traditional* mode in which no evolution relationships are considered and the answer set consists of only those data elements that satisfy the query conditions. The second mode is the *evolution* mode in which the answer set is enhanced with answers satisfying the query conditions but do not exist in the data repository. These are data elements created through merges of existing entities in the repository that are related through evolution relationships. The third mode is the *top-k evolution* which is the same like the *evolution* mode apart from the fact that only the top-k results are returned. The sorting of the results is based on the Steiner forest that was used to associate the entities in the repository and merge them. Finally, the fourth mode, called *fully-collapsed*, executes the query against a repository where all entities connected through evolution relationships have been collapsed into one.

A screen-shot of the `TrenDS` is illustrated in Figure 3.7. It consists of a text field on which the user can write a query. On the left of the query, a pull down menu allows the user to select the method of computing the cost of result items. The main part of the screen is the list of the results found as an answer to the query. Theoretically, every entry in the results describe the binding of the query variables into values of the repository. This is the case in which no evolution information is considered. However, this is not the case if the evolution is taken into consideration. The presented entities may be virtual, i.e., created on-the-fly through a merge of entities associated with evolution relationships. As an example, consider the 5th entry in Figure 3.7. There is no entity `eid1-2`, neither entity `eid3-7`. The former has been constructed by the merging of the entities `eid1` and `eid2`, while the second by the merging of `eid3` and `eid7`. If the user wants to see the binding of variables to actual entities in the repository, she can do so by clicking on the entry which opens a small panel as shown in the figure. Note that TrenDS has variables bound to more than one data value, which can look strange at first sight. However, this is the feature of our approach and the actual reason that the merging of the entities on which the variable is bound is forced. In the specific example, variable $x is bound to entity `eid1` and `eid2`, while variable $y is bound to `eid3` and `eid7`. This leads to the merging (collapse) of the two former, creating the virtual entity `eid1-2` and

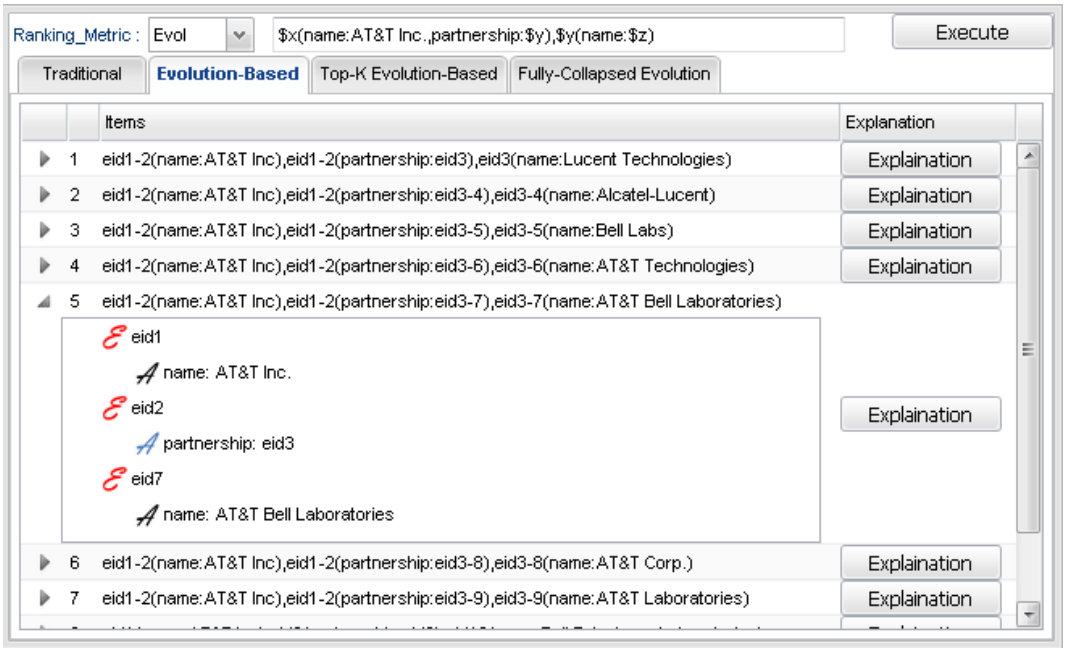


Figure 3.7: TrenDS Graphical User Interface

the two latter creating entity `eid3-7`. If the user wants additionally to understand how the Steiner forest between these entities looks like, she can click on the `Explanation` button on the right of the result entry which opens a graphical illustration.

In the following we describe some use cases which demonstrate the system capabilities.

Evolution Discovery Although it is not the main goal and feature of `TrenDS`, it has the ability to explore the data set of US companies (see Section 3.7 for the dataset details). By entering the name of a company (e.g. Oracle, Microsoft, Google), it is possible to see its evolution graph, i.e., from what companies it has been created, if any, what companies it has incorporated, what spin-offs were created by it, and to what companies it was broken-up, if any. For these evolution states, the reregistration of the trademarks is displayed to justify the inference about its evolution.

Query Evaluation with Evolution Semantics. The main part of the system is to use of evolution in query answering. The users can pose a query which will be executed and their results will be presented. The users have the ability to see the results of their queries as they would have been returned by a traditional search engine. Then, by selecting the appropriate tab the results when evolution is taken into consideration will be displayed. The additional results are highlighted and explained. For explanation, we use the Steiner forest that illustrates to the user in a graphical way the evolution merges that have led to the specific answer. Furthermore, the user is able to choose between seeing the top-k results of the fully collapsed evolution graph, and compare the time needed for generating each one.

Ranking Based on Different Metrics Merging entities that are (from the evolution perspective) far to each other is not as important as merging those that are close. The users of the the system have the ability to experiment with the different ranking functions which are based on the Steiner forest connecting the entities to be merged, and witness how the results are

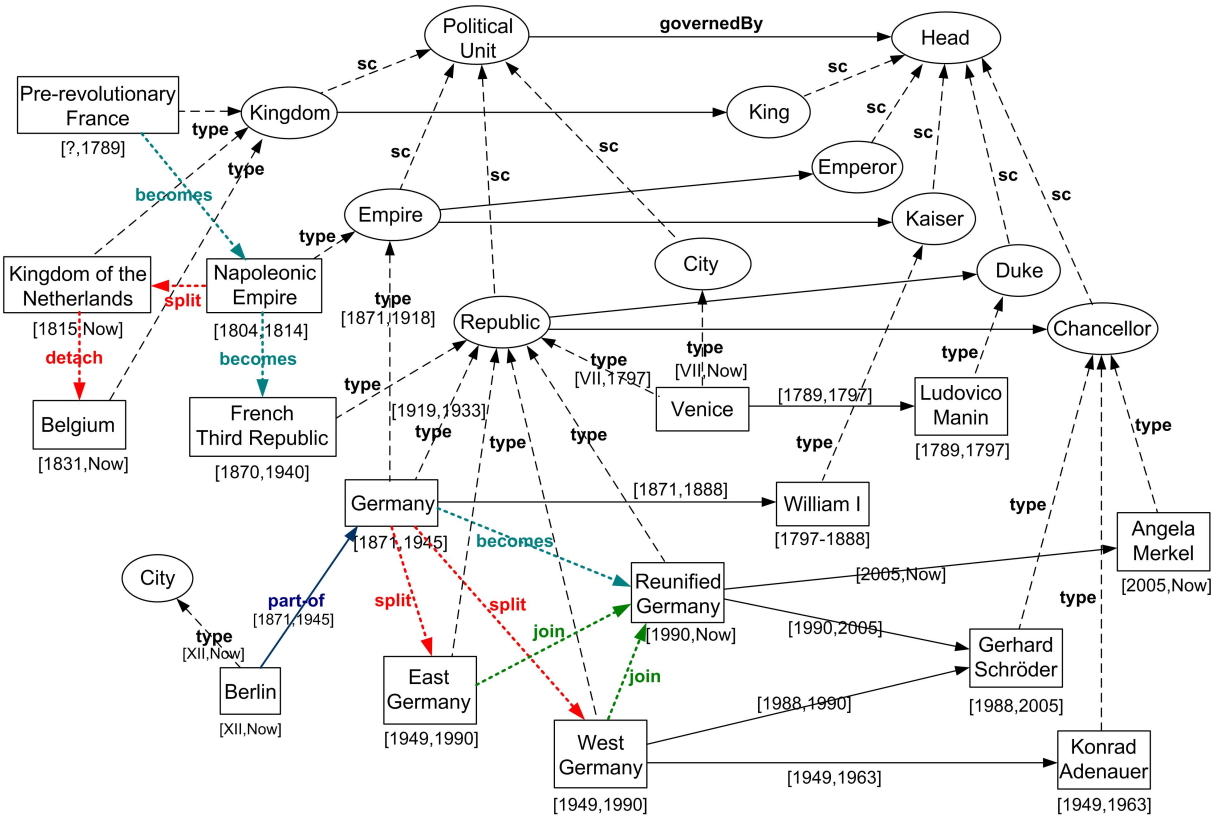


Figure 3.8: The evolution of the concepts of Germany and France and their governments (full black lines represent governedBy properties)

automatically affected by them.

3.6 Case Studies

We have performed two main case studies of modeling and querying evolution. Consider an evolution database which is an extension of the example introduced in Section 1 that models how countries have changed over time in terms of territory, political division, type of government, and other characteristics. Classes are represented by ovals and instances by boxes. A small fragment of that evolution base is illustrated as a graph in Figure 3.8.

Germany, for instance, is a concept that has changed several times along history. The country was unified as a nation-state in 1871 and the concept of Germany first appears in our historical database as Germany at instant 1871. After WWII, Germany was divided into four military zones (not shown in the figure) that were merged into West and East Germany in 1949. This is represented with two split edges from the concept of Germany to the concepts of West Germany and East Germany. The country was finally reunified in 1990, which is represented by the coalescence of the West Germany and East Germany concepts into Unified Germany via two merge edges. These merge and split constructs are also defined in terms of the parts of the concepts they relate. For instance, a part-of property indicates that Berlin was part of Germany during $[1871, 1945]$. Since that concept of Germany existed until 1945 whereas Berlin exists until today,

the part-of relation is carried forward by the semantics of split and merge into the concept of Reunified Germany. Consider now a historian who is interested in finding answers to a number of evolution-aware queries.

[Example Query 1]: How has the notion of Germany changed over the last two centuries in terms of its constituents, government, etc.? The query can be expressed in our extended query language as follows:

$\mathbf{Select}\ ?Y, ?Z, ?W$
$(?X, \mathbf{self}$ ::Reunified Germany$/$
$\mathbf{backward}^*[1800, 2000]/, ?Y)$ AND
$(?Y, \mathbf{edge}, ?Z)$ AND $(?Z, \mathbf{edge}, ?W)$

The query first binds $?X$ to Reunified Germany and then follows all possible evolution axes backwards in the period $[1800, 2000]$. All concepts bound to $?Y$ are in an evolution path to Reunified Germany, namely Germany, West Germany, and East Germany. Note that, since the semantics of an * expression includes **self** (see Figure 3.4), then Reunified Germany will also bind $?Y$. The second triple returns in $?Z$ the name of the properties of which $?Y$ is the subject, and finally the last triple returns in $?W$ the objects of those properties. By selecting $?Y, ?Z, ?W$ in the head of the query, we get all evolutions of Germany together with their properties.

[Example Query 2]: Who was the head of the German government before and after the unification of 1990? The query can be expressed as follows:

$\mathbf{Select}\ ?Y$
$(?X, \mathbf{self}$ ::Reunified Germany$/\mathbf{join}^{-1}[1990]/$
$\mathbf{next}$ :: head$[1990], ?Y)$ AND
$(?Z, \mathbf{self}$ ::Reunified Germany$/\mathbf{next}$ :: head$[1990], ?Y)$

The first triple finds all the heads of state of the Reunified Germany before the unification by following $\mathbf{join}^{-1}[1990]$ and then following $\mathbf{next}$ :: head$[1990]$. The second triple finds the heads of state of the Reunified Germany. Finally, the join on $?Y$ will bind the variable only to those heads of state that are the same in both triples, hence returning the one before and after the mentioned unification.

Consider now the evolution of the concept of biotechnology from a historical point of view. According to historians, biotechnology got its current meaning (related to molecular biology) only after the 70s. Before that, the term biotechnology was used in areas as diverse as agriculture, microbiology, and enzyme-based fermentation. Even though the term "biotechnology" was coined in 1919 by Karl Ereky, a Hungarian engineer, the earliest mentions of biotechnology in the news and specialized media refer to a set of ancient techniques like selective breeding, fermentation and hybridization. From the 70s the dominant meaning of biotechnology has been closely related to genetics. However, it is possible to find news and other media articles from the 60s to the 80s that use the term biotechnology to refer to an environmentally friendly technological orientation unrelated to genetics but closely related to bioprocess engineering. Not only the use of the term changed from the 60s to the 90s, but also the two different meanings coexisted in the media for almost two decades.

Figure 3.9 illustrates the evolution of the notion of biotechnology since the 40s. As in

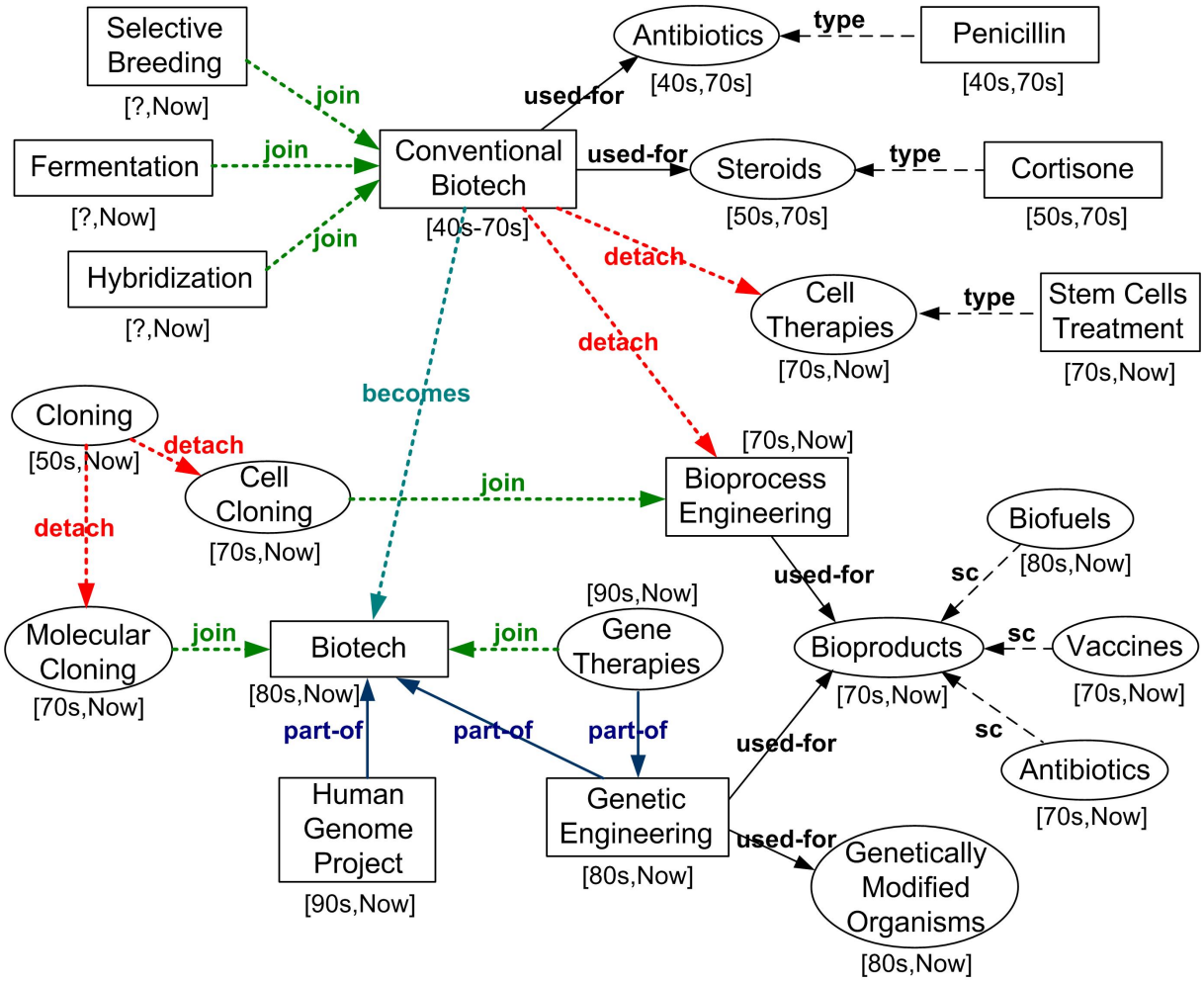


Figure 3.9: The evolution of the concept of Biotechnology

the previous example, classes in the evolution base are represented by ovals and instances by boxes. The used-for property is a normal property that simply links a technological concept to its products. The notions of Selective breeding, Fermentation and Hybridization existed from an indeterminate time until now and in the 40s joined the new topic of Conventional Biotech, which groups ancient techniques like the ones mentioned above. Over the next decades, Conventional Biotech started to include more modern therapies and products such as Cell Therapies, Penicillin and Cortisone. At some point in the 70s, the notions of Cell Therapies and Bioprocess Engineering matured and detached from Conventional Biotech becoming independent concepts. Note that Cell Therapies is a class-level concept that detached from the an instance-level concept. The three concepts coexisted in time during part of the 70s, the latter two coexist even now. During the 70s, the notion of Conventional Biotech stopped being used and all its related concepts became independent topics. In parallel to this, the new topic of Biotech started to take shape. We could see Biotech as an evolution of the former Conventional Biotech but using Genetic Engineering instead of conventional techniques. Concepts and terms related to the Biotech and Genetic Engineering topics are modeled with a part-of property. In parallel to this, the concept of Cloning

techniques started to appear in the 50s, from which the specialized notions of Cell Cloning and Molecular Cloning techniques detached in the 70s and joined the notions of Bioprocess Engineering and Biotech, respectively. The latter is an example of class-level concepts joining instance-level concepts.

[Example Query 3]: Is the academic discipline of biotechnology a wholly new technology branch or has it derived from the combination of other disciplines? Which ones and how? The query requires to follow evolution paths and return the traversed triples in addition to the nodes in order to answer the question of "how." The query is expressed in our language as follows:

$\mathbf{Select}\ ?Y, ?Z, ?W$
$(?X, \mathbf{self} :: \text{Biotechnology}/\mathbf{backward}^*, ?Y)$ AND
$(?Y, \mathbf{e\text{-}edge}/\mathbf{self}, ?Z)$ AND $(?Z, \mathbf{e\text{-}node}, ?W)$

The first triple binds $?Y$ to every node reachable from Biotechnology following evolution edges backwards. Then, for each of those nodes, including Biotechnology, the second triple gets all the evolution axes of which the bindings of $?Y$ are subjects whereas the third triple get the objects of the evolution axes. This query returns, (Biotech, $\mathbf{becomes}^{-1}$, Conventional Biotech), (Conventional Biotech, $\mathbf{join}^{-1}$, Hybridization), (Conventional Biotech, $\mathbf{join}^{-1}$, Fermentation), and (Conventional Biotech, $\mathbf{join}^{-1}$, Selective Breeding).

[Example Query 4]: Which scientific and engineering concepts and disciplines are related to the emergence of cell cloning? We interpret "related" in our model as being immediate predecessors/successors and siblings in the evolution process. That is, from a concept we first find its immediate predecessors by following all evolution edges backwards one step. We then follow from the result all evolution edges forward on step and we get the original concept and some of its siblings. Finally, we repeat the same process in the opposite direction following evolution edges one step, first forward and then backwards. Based on this notion, we can express the query as follows:

$\mathbf{Select}\ ?Y, ?Z, ?W$
$(?X, \mathbf{self} :: \text{Cell Cloning}, ?Y)$ AND
$(?Y, \mathbf{backward} \mid \mathbf{backward}/\mathbf{forward}, ?Z)$
AND $(?Y, \mathbf{forward} \mid \mathbf{forward}/\mathbf{backward}, ?W)$

The first triple will just bind Cell Cloning to $?Y$. The second triple follows the $\mathbf{detach}$ edge back to Cloning, and then the $\mathbf{detach}$ edge forward to Molecular Cloning. The third triple starts again from Cell Cloning and follows the $\mathbf{join}$ edge forward to Bioprocess Engineering and then the $detach$ edge backwards to Conventional Biotech. All these concepts will be returned by the query.

3.7 Evolution-Unaware Query Performance Evaluation

To evaluate the efficiency of the evolution-unaware query evaluation approach we performed two kinds of experiments. First we studied the behavior of the Steiner forest discovery algorithm in isolation, and then we evaluated the performance of the query answering mechanism we have developed and which uses internally the Steiner forest algorithm. We also studied

the improvements in performance with regard to our optimization technique for top-k query answering.

In the experiments we used both synthetic and real data. We used the term *non-evolution* data to refer to concepts and attributes, and the term *evolution* data to refer to the the evolution relationships and more generally to the evolution graph. We noticed that in the real datasets, the non-evolution data were much larger than the evolution data and we maintained a similar analogy during our synthetic data generation.

For the synthetic data generation we used the Erdös-Rényi graph generator [Johnsonbaugh and Kalin, 1991] which can produce random graphs for which the probability to have an edge between two nodes is constant and independent of other edges. Since many real world data follow a power law distribution, for the non-evolution synthetic data we used the Zipf's distribution. In our own implementation of the Zipfian distribution, as a rank we considered the number of occurrences of an attribute-value pair in the entire dataset (e.g., if the attribute $\langle State, CA\rangle$ appeared 15 time, its rank was 15). This allowed us to model the fact that there are few frequent attribute-value pairs and the majority are rare attributes. The real corpora that we used had similar properties. We will refer to the synthetic dataset generated using this method as *ER-SYNTH*.

For real dataset we used an extract from the trademark corpora which is available from the United States Patent and Trademark Office[2]. The trademarks are a kind of intellectual property which may belong to an individual or a company. If some change in ownership occurs the corresponding trademarks have to be re-registered accordingly. For instance, in the domain of corporate mergers, acquisitions and spin-offs, one can observe how the intellectual property is allocated among its owners throughout time. The United States Patent and Trademark Office provides a set of trademarks along with the lists of their owners. To illustrate this, let us take a standard character trademark "Interwise" which was initially registered by `Interwise Inc.` and then, after the `AT&T Corp.` acquisition, it became a property of `AT&T Corp.`. A modeller has two options to store this information: either she can consider both companies as one concept with two name attributes or create two concepts which own the same trademark. Note, that because of temporal consistency constraints we have to split the trademark into two separate concepts in the second modelling choice. With respect to the evolution world semantics we interpret these two modelling possibilities as two possible worlds, because in the second case `Interwise Inc.` and `AT&T Corp.` should be involved into an evolution relation. The dataset extracted from the UPTSO contained approximately 16K unique companies, 200K attributes. In this dataset we have discovered 573 evolution graphs of various sizes between 5 and 373. The example of discovered evolution graph is presented in Figure 3.10 which depicts the fraction of evolution of AT&T.

The attributes' frequency in the trademark dataset is distributed according to the Zipfian law, i.e. the most frequent attribute-value pair appear twice as often as the second most frequent attribute value pair, and so on. More specifically, the value of exponent of the found Zipfian distribution is approximately equal to 3. To make the dataset extracted from real data even richer, i.e., with components of higher complexity, we used two graph merging strategies. In the first one, a new evolution component is constructed by connecting two components through an artificial evolution relationship edge between two random nodes from the two components. We

[2]http://www.uspto.gov/

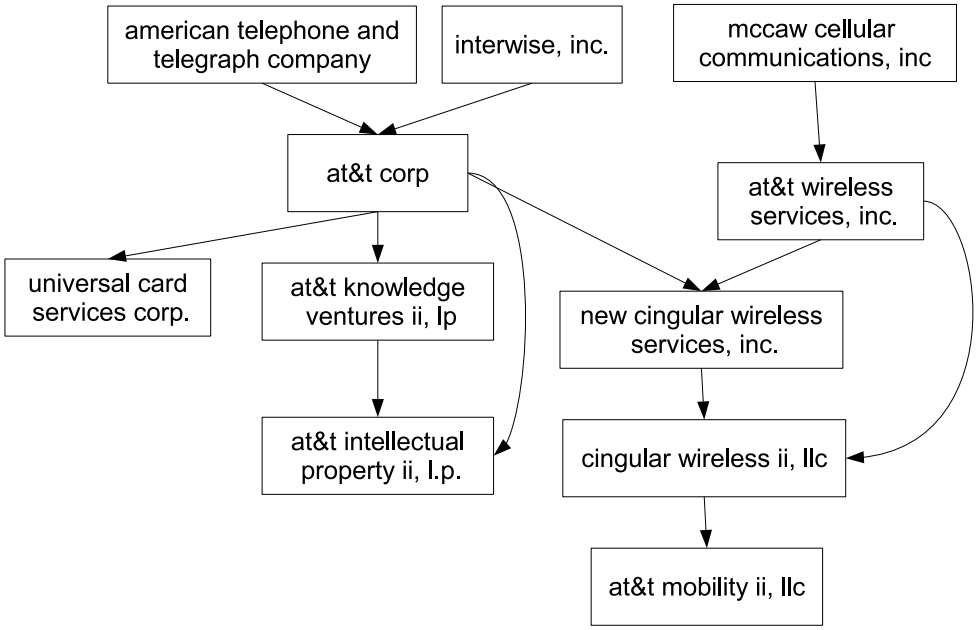


Figure 3.10: A fraction of the discovered evolution graph of AT&T

refer to this kind of merge as *CHAIN*, because it creates a chain of source graphs. In the second strategy, two components are merged by choosing an arbitrary node from one component and then adding evolution relationship edges to some random node of every other component. We refer to this method as *STAR*. Datasets generated using these methods will be denoted as *REAL-CHAIN* and *REAL-STAR*, respectively. The naive evaluation strategies that were described in Sections 3.4.2 and 3.4.2 are omitted from the discussion since their high complexity makes them practically infeasible to implement. In some sense, the naive case corresponds to the brute force way of finding a minimal Steiner forest, which is exponential in the size of evolution graph.

The experiments were all carried out on a 2.4GHz CPU and 4G memory PC running MS Windows Vista.

3.7.1 Steiner Forest

To study in detail the Steiner forest algorithm we performed two kinds of experiments. First, we studied the scalability properties of the algorithm for varying inputs, and then the behavior of the algorithm for graphs with different characteristics.

Scaling the Input. Recall that the input to the Steiner forest algorithm is the set $\mathcal{V}=\{V_1, \ldots, V_L\}$. In this experiment we studied the query evaluation time with respect to the size of the input. By size we considered two parameters: (i) the total number of elements in the sets of $\mathcal{V}$, i.e., the $\sum_{i=1}^{L} |V_i|$; and (ii) the number of the groups, i.e., the value L.

For the former, we started with L=1 and we scaled the $\sum |V_i|$ (which in this case is actually

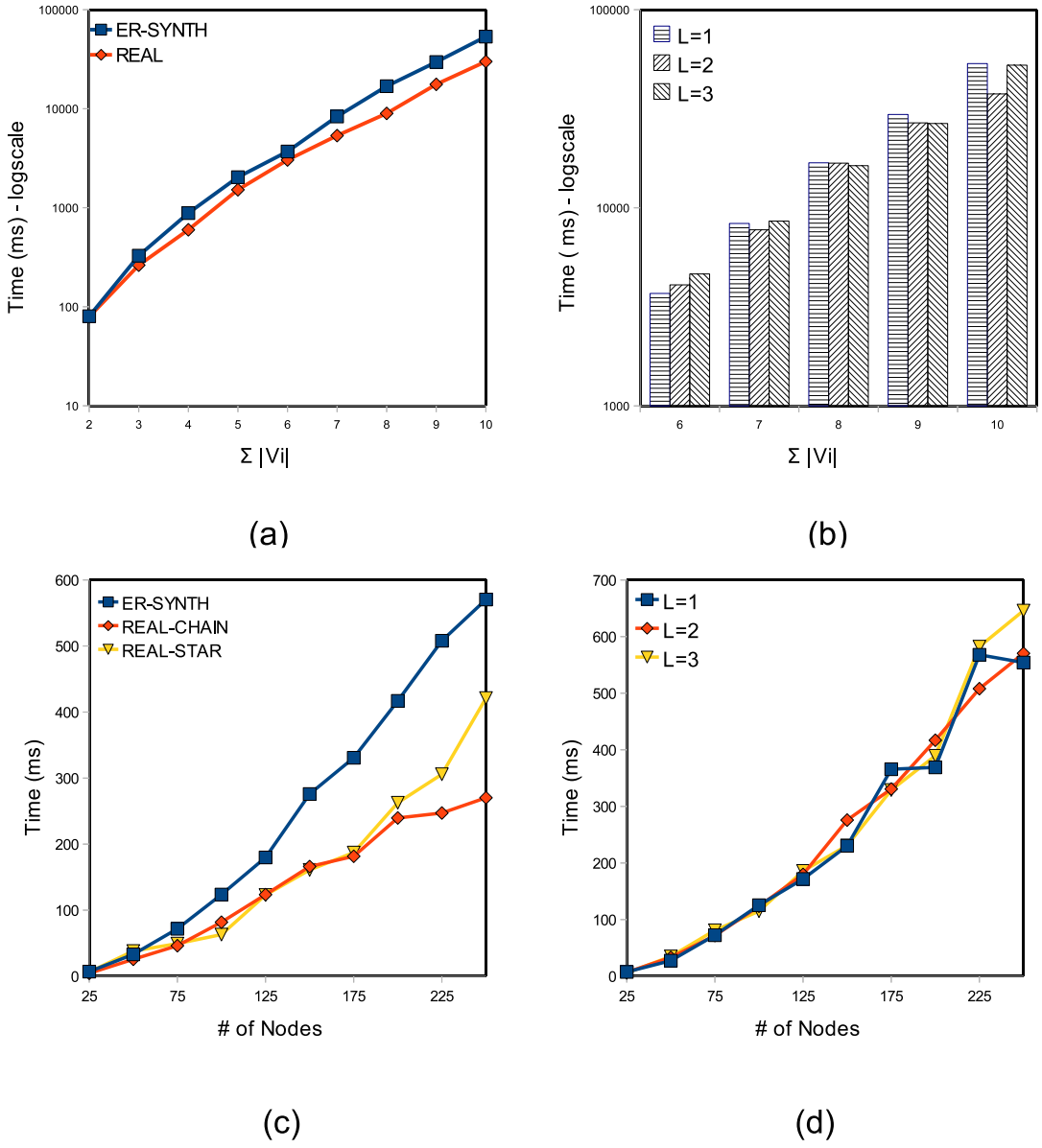


Figure 3.11: Steiner forest discovery performance

equal to $|V_1|$) from 2 to 10. For each size the average evaluation time of 25 random queries was recorded. The queries were evaluated both on synthetic and on real data. The synthetic graph was obtained using the Erds-Rnyi method and had $n = 57$ nodes and $m = 65$ edges. The real dataset graph was the one described previously. The results of this experiment are presented in Figure 3.11(a). The exponential grows in time (note that the time is presented on a logarithmic scale) with respect to a query size is consistent with the theoretical complexity of the Steiner forest algorithm.

To study how the parameter L affects the query execution time we kept the $\Sigma_{i=1}^{L}|V_i|$ constant but modified the number of the sets L from 1 to 3, and then we repeated the experiment for

values of $\sum_{i=1}^{L} |V_i|$ from 6 to 10 (we assumed that a minimal set size was 2). The results of the average of 25 random query execution times are reported in Figure 3.11(b). The characteristics of the graph were the same as those in the previous experiment. The current experiment showed that the execution time depends fully on the $\sum_{i=1}^{L} |V_i|$ and not on L itself. This means that within a reasonable range of query sizes the number of forest branches does not have any influence on the performance.

Scaling the Graph. In this experiment we studied the Steiner forest discovery time with respect to the size of the graph. We used three kinds of graph data: *ER-SYNTH*, *REAL-CHAIN* and *REAL-STAR*, with sizes from 25 to 250 nodes with a step of 25.

For the synthetic dataset the number of edges and nodes was almost the same. We generated 25 random inputs to the Steiner forest problem with $L = 2$ and $|V_1|$=3, and $|V_2|$=3. The results of this experiment are presented in Figure 3.11(c). The query evaluation time has a linear trend as expected, and it was interesting that the execution time was always less than a second.

We also studied the scalability of the algorithm in terms of the parameter L. For three queries with $\sum |V_i| = 6$ and L=1, 2 and 3 we varied the evolution graph size from 25 to 250 with step 25. The graph we used was the *ER-SYNTH* with the same number of nodes and edges as before. The results are shown in Figure 3.11(d), where it can be observed that the scalability of the algorithm depends on the total number of elements in the sets in the input set $\mathcal{V}$, i.e., the $\sum_{i=1}^{L} |V_i|$, and not on the number of forest branches, i.e., the number L, at least for values of $\sum_{i=1}^{L} |V_i|$ up to 10.

3.7.2 Query Evaluation

Apart from experimenting with the Steiner forest algorithm in isolation, we ran a number of experiments to evaluate the query answering mechanism we have developed for evolution databases. The query evaluation time depends not only on the evolution graph size and structure but also on the size and structure of the whole evolution database. First, we analyzed the behaviour of the system with respect to the query size. The query size is determined by the number of distinct variables, and their number of occurrences in the query. We started with a 1-variable query and we observe its behavior as size increases. Then, we tested the scalability of the query evaluation mechanism as a function of the evolution graph only. Finally, we studied the scalability as a function of the data (i.e., attributes and associations) and we found that their distribution (but not their size) can dramatically affect the performance of the system.

In the synthetic data generation, we generated values that were following the Zipfian distribution for the attributes/associations. We controlled the generation of the *ER-SYNTH* dataset through four parameters, and in particular, the *pool of concepts*, the *exponent* that is used to adjust the steepness of the Zipfian distribution, the *number of elements* that describes the maximum frequency of an attribute or association, and the *number of attributes*.The values of the parameters for the synthetic data are chosen to coincide with those of the real corpora.

Scaling the Query. We considered a number of 1-variable queries with a body of the form:

$$\$x(attr_1{:}value_1), \ldots, \$x(attr_N{:}value_N)$$

and we performed a number of experiments for different values of N, i.e., the number of atoms

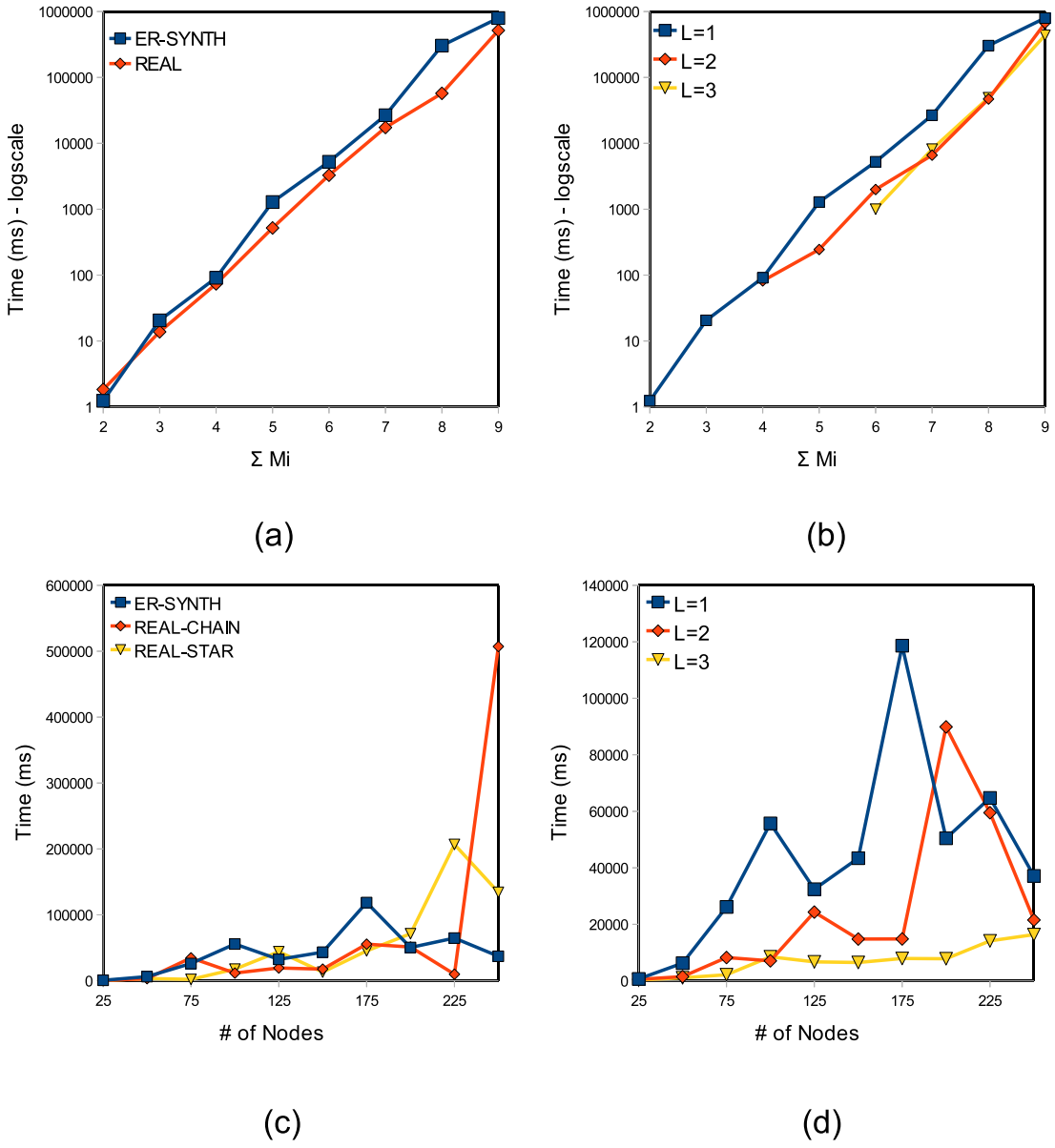


Figure 3.12: Query evaluation performance

in the query. For every atom we randomly chose an attribute-value pair from a pool of available distinct attribute name/value pairs. The *ER-SYNTH* graph that was generated had 57 nodes and 65 edges. The results are shown in Figure 3.12(a). The same figure includes the results of the query evaluation on the real dataset that had a size similar to the synthetic. For the generation of their non-evolution data we had the exponent set to 3, the number of elements parameter set to 15 and their total number was 537. The results of the Figure 3.12(a) are in a logarithmic scale and confirm the expectation that the query evaluation time is growing exponentially as the number of variables in the query grows. If we compare these results with those of the Steiner forest algorithm for the respective case, it follows that the integrated system adds a notable

overhead on top of the Steiner forest algorithm execution time. This is was due to the number of coalescence candidates and the number of Steiner forests that needed to be computed in order to obtain the cost of the elements in the answer set. Although the parameters for the generation of the synthetic and real data coincided, their trends were different, as Figure 3.12(c) illustrates.

We further tested how the number of concept variables in the query affect the performance. Note that we are mainly interested in the concept-bound variables. Let M represent the number of distinct concept-bound variables in the query, and M_i the number of appearances of the i-th variable in the query. Note that the number of distinct variables will require to solve a Steiner forest problem in which the input $\mathcal{V}=\{V_1,\ldots,V_L\}$ will have $L=M$ and $|V_i|=M_i$, for each $i=1..M$. The total number of variable appearances in the query will naturally be $\sum_{i=1}^{M} M_i$.

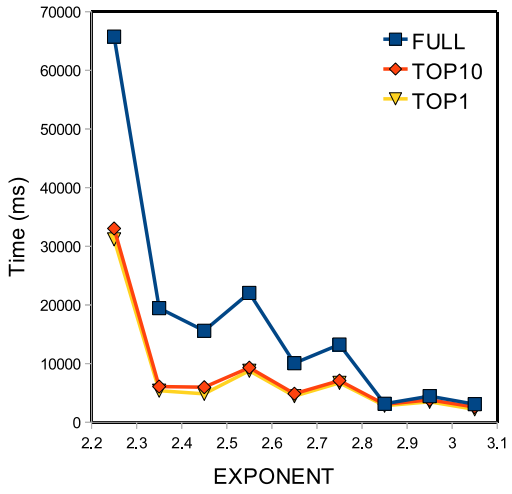
In the experiment, we chose a constant value for the $\sum_{i=1}^{M} M_i$ and we run queries for $M=1$, 2 or 3. As a dataset we used the *ER-SYNTH* with 57 nodes and 65 edges. 537 attributes were generated with the exponent parameter having the value 3 and the number of elements parameter to have the value 15.A total of 53 synthetic associations were also generated with the exponent parameter having the value 3, and the number of elements parameter to have the value 10. We used 25 randomly generated queries for each of the 3 M values, and took their average execution time. We did multiple runs of the above experiments for different values of $\sum_{i=1}^{M} M_i$ between 4 and 10. The outcome of the experiments is shown in Figure 3.12(b) in a logarithmic scale. Clearly, the number of branches in a forest did not affect the query evaluation time, i.e., queries with many variables showed the same increase in time as the 1-variable query for the same $\sum_{i=1}^{M} M_i$.

Scaling the Data. In this experiment we examined the query evaluation time with respect to the size of the evolution graph. As evolution data, we used both real and synthetic sources. Regarding the real data, we used a series of graphs (and their attributes as non-evolution data) with sizes from 25 to 250 with step 25. The number of edges was 110% of the number of nodes for all graphs. For the real dataset we used both the *REAL-CHAIN* and the *REAL-STAR* data. For each graph we generated 25 random queries with 3 distinct variables, i.e., $M=3$, and each variable had $M_1=2$, $M_2=2$ and $M_3=3$ appearances in the query, and we measured the average time required to evaluate them. As a synthetic dataset the *ER-SYNTH* was used, generated to be the same size as before but with the following Zipfian distribution parameters: exponent 3, number of elements 15 and number attributes 10 times more than the number of nodes. Note that we did not generate associations because the trademark dataset did not have any association that we could use as a guide. Figure 3.12(c) depicts the results of this experiment. It shows that there is a linear growth of time which is is accompanied with an increasing oscillations which can be explained by the growing exponent of non-evolution data, i.e. the number of coalescence candidates may become too large for evolution graphs with considerable size.

Furthermore, we studied how the query evaluation time scales for different values of M, i.e. for different distinct variables but with the same total number of variable appearances in the query (i.e., the $\sum_{i=1}^{M} M_i$). We used the *ER-SYNTH* dataset again with sizes from 25 to 250, using a step 25. The number of evolution relationships was 110% of the number of concepts. For each case, we generated 25 random queries with $M=1$ having $M_1=6$, $M=2$ having $M_1=3$, and $M_2=3$ and finally, $M=3$ having $M_1=3$, $M_2=2$, and $M_3=2$. We executed these queries and measured the average evaluation time. The non-evolution data was following

# of Branches	s=2.5	s=3.0	s=3.0
1	93,845	44,098	35,382
2	2,400	1,414	4,686
3	374	637	485
4	485	20	91
5	124	260	16

(a)



(b)

Figure 3.13: Efficiency for varying # of connected components (a) and exponent of data distribution (b)

the Zipfian distribution with exponent 3, the number of elements was 15 and the total number of attributes was 10 times more that the number of nodes (concepts). For the associations, the exponent was 3, the number of elements was 10 and their total number was 5 times more that the respective number for nodes. The results are presented in Figure 3.12(d). Similarly to the previous experiment, we observed a linear growth with increasing oscillations.

Evolution scalability for different forest structures. We further examined how the number of evolution graph components influence the query evaluation time. For this purpose, we generated data using *ER-SYNTH*, and in particular 5 datasets of evolution graphs with a total size of 300 nodes and 330 edges. The sets had 1, 2, 3, 4 and 5 evolution graphs, respectively. For each set we run 25 random queries with two distinct variables ($L = 2$) that were appearing in the query 3 times each, i.e., M_1=3 and M_2=3 and measured their average execution time. As non-evolution data, we generated attributes and associations with varying exponent parameter, 2.5, 3 and 3.5. The total number of elements and attributes/associations were 15 and 1000 in one case, while it was 10 and 100 in the other. Figure 3.13(a) contains a table with the query evaluation time for each number of branches and exponent values. From the result, we could observe the dramatic decrease in time with respect to the number of evolution graph components. This can be explained by the fact that the query evaluation spread over a number of evolution graph components where each evaluation time becomes considerably small.

Data distribution dependency. Finally, we studied the properties of the system in relation to the data distribution parameter, namely the exponent of Zip's distribution. The query optimizer described in Section 3.4.3 was taken into consideration here and we analyzed how the non-evolution data were affecting the top-k query answering. For this experiment we used the following input parameters: 25 random queries with M=2 distinct variables, and M_1=3 and

M_2=3 respective appearances of each distinct variable in the query. We used an *ER-SYNTH* dataset, the evolution graph of which had $n = 57$ nodes and $m = 65$ evolution edges. We also had 10000 attributes distributed over 30 concepts, and 1000 associations distributed over 15 concepts. The exponent we used varied from 2.25 to 3.05 with a step of 0.1. The results of the specific experiment are presented in Figure 3.13(b). For small exponents the difference between regular query answering and the top-10 or top-1 was significant. To justify this, recall that the number of pruned candidates depends on how different are the input sets in the Steiner forest algorithm input (ref. Section 3.4.3), thus, when the exponent is small the input sets share many concepts.

Chapter 4

Data Quality in Evolving User-Generated Contents

The continuously evolving nature of the data hinders the ability of determining the quality of the data that is observed at a specific moment, since there is a great deal of uncertainty on whether this information will remain as is. To obtain better quality articles in evolving user generated contents (e.g. Wikipedia) it is important to address the problem of automatically identifying controversies. In this chapter, we introduce the data quality problem (Section 4.1) and the concepts we use (Section 4.2). The controversy detection solution is presented in Section 4.3. Finally, we present our experimental setting (Section 4.4) and results (Section 4.5).

The work presented in this chapter is prepared for the submission to the VLDB 2013 conference ([Bykau et al., 2013a]).

4.1 Data Quality Problem

The advent of Web 2.0 and related social apps have transformed the average Internet user from a passive data consumer into an active data producer. This was a catalyst for the birth of collaborative content creation. Documents such as news blogs, computer source code and informational manuals increasingly rely on users to contribute content. The value proposition of massive open collaboration, apart from the significant reduction in financial costs, is that it enables information to be kept up-to-date, sometimes within seconds of occuring events, and content to be more complete, informed and pluralistic. Of course, allowing a broad base including non-professionals to edit content opens the door to unreliable information. Fortunately, recent studies have shown that the "wisdom of crowds" can lead not to chaos but to surprisingly high quality data [Giles]. This occurs when competent and well-intentioned users outnumber ill-intentioned, careless, or misinformed ones, by correcting [Chin et al., 2010] and completing [Gunningham and Leuf, 2001] information.

Wikipedia is heralded as a success story of collaboratively edited content. Nonetheless, situations can arise where there are incompatible viewpoints of the same issue, as often happens in real life. Ideally, a dispute will eventually converge to a community-accepted consensus after undergoing discussion between the disagreeing authors. However, disputes can sometimes degenerate into so-called *edit wars* which disrupt progress towards improving the article.

Towards the goal of obtaining better quality articles in Wikipedia, we study the problem of automatically identifying controversies, based on the revision histories maintained as part of the Wikipedia database. Semantically, a *controversy* is defined as follows: *a prolonged dispute by a number of different people on the same subject*.[1] A controversy may not appear in the current version of a page but may be hidden among its edit history. For example, consider the Wikipedia entry for "Caesar salad". At one time, one group of authors argued that the name derives from the Roman emperor Julius Caesar, while another group argued that it was from an Italian-American restauranteur named Caesar Cardini. The controversy resulted in back-and-forth arguments between a large number of different authors until a consensus was reached to attribute the creation to Caesar Cardini.

Identifying controversies is important for several reasons: to distinguish them from vandalism, giving disputes a chance to be aired out rather than squelched; to set appropriate edit policies and raise the bar for supporting evidence when needed; and for enabling the reader a more nuanced understanding of content. For Wikipedia specifically, controversy detection plays a significant role. It helps maintaining Wikipedia's neutral point of view policy, which strives for objectivity, meaning that articles should be fair to all sides (though not necessarily equal, in the case of minority views), and that controversial content requires more rigorous documentation. It also helps identify and protect sensitive pages, for instance by allowing only accounts that have been active for some duration to make changes, as well as tightly restricting the frequency of reverts blocking any editor for some time period who attempts to revert more than once in a 24-hour period.

Controversies in Wikipedia come in all shapes and sizes. Sensitive entries such as "Arab-Israeli Conflict" have, not surprisingly, been rife with controversial content involving the number of casualties, the proper calculation of Israeli per-capita GDP, etc. However, controversies may often arise within articles not known to be sensitive, such as whether the word "the" should be uppercased in (the music band) "The Beatles" [Chaudhuri, 2012]. In the Caesar salad example above, the controversy was explicitly discussed by authors on the so-called "talk" (discussion) tab for that page. However, in many if not most instances, controversies are not reported and sometimes the authors are not even aware of them. Thus, an automated detection mechanism is needed.

In contrast to prior work on detecting controversial pages in Wikipedia at the page level, we focus on identifying *fine-grained* controversies pinpointing the specific controversial topics, locations within a page and occurrences in time. We observe that controversies often arise from conflicting points of view or opinions on the same topic. Unfortunately, segmenting a page into distinct topics is a very difficult problem in the natural language processing and machine learning literature. Moreover, tracking a topic over time is particularly challenging, even for a minor rewording, since its location within a page can change (say, from the introduction to the biography section). We formalize a syntactic notion of controversy based on the assumption that the associated dispute is expressed as back-and-forth substitutions of content embedded within a similar context (rather than deletions, which was the basis of previous work). We have investigated two different models for that purpose: one sees the document as (space-separated) "words"; the other sees the document as a sequence of the hyperlinks (to other Wikipedia pages) contained within a document, ignoring the text, multimedia and links to external Web sites.

[1] http://en.wikipedia.org/wiki/Controversy

While the former uses a superset of the latter, and therefore has access to more information, the latter exploits the labeling (and, therefore, normalization) of semantic concepts implicit in links, since most real-world entities and concepts now have a Wikipedia page and Wikipedia pages often contain a high density of hyperlinks, as is encouraged in the Wikipedia style guide.[2]

Controversy identification is also a challenging task due to the large number of versions in the history of a document. For example, at the time of writing, Wikipedia contained 4,105,700 entries and was growing at a rate of 25,000 new entries per month; the total number of versions among these entries was approximately 570M and growing at a rate of 5M edits per month.

4.2 Modeling Collaborative Data

Collaboratively generated content in the real world is a collection of entries created and edited by many editors is some application. These applications are typically keeping also the version history of the entries throughout the edits applied on them. The entries can be text, html pages, wiki pages, office documents, etc. To process these entries, we need to convert them to documents in our model. We call this process *tokenization*. A *token* is a string that is considered undivided. A *document* D is a non-empty sequence of tokens. There are different types of tokens that can be used. Each token type determines a *model*. The choice of the model is based on what best fits the task at hand. In this work, since our focus is mainly wikipedia, we focus on two specific models: the *text* and the *link* model. The text model considers as tokens single words, meaning that every content entry is turned into a sequence of individual words. As we will show in the experiments, the text model has the advantage of capturing every change that may occur in the content entries, since it does not disregard any part of them, however, it increases ambiguity. Note that even if the model considers single work, our algorithm is able to capture modifications involving more than one word, i.e., the expression "New York" is replaced by the expression "Connecticut".

The *link model* considers only the links (i.e., references) to other entries, pages or entities, meaning that during tokenization the original entry is stripped of everything that is not a link. In contract to previous studies on wikipedia content have considered only the text model[Vuong et al., 2008, Yasseri et al., 2012], we also consider the link model, a fundamental novelty of our approach. The link model is a more semantic approach compared to the text model since the referenced entries are typically semantic concepts. In the case of wikipedia, every entry is describing unambiguously a concept or an entity, thus, the meaning of every reference[3] (link) in wikipedia has clear semantics. At the same time, since wikipedia has become one of the largest sets publicly available nowadays, almost every semantic concept has its respective entry in it, and with the wikipedia language highly facilitating the creation of links, the density of links is such that allows them to capture most of the page semantics in just a sequence of links.

According to the Manual of Style and Linking of Wikipedia[4] links should be used in the following cases:

- connection to another article that will help readers understand the article more fully;
- articles with relevant information;

[2] http://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style

[3] Note that we consider only internal links

[4] http://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking#What_generally_should_be_linked

- articles explaining technical terms;
- proper names that are likely to be unfamiliar to readers.

Moreover, one of the criteria of Wikipedia article quality assessment[5] is "supported by inline citations where appropriate".

Example 4.2.1. Consider the following Wikipedia content where the underline represents links. *"A Caesar salad is a salad of romaine lettuce and croutons dressed with parmesan cheese, lemon juice, olive oil, egg, Worcestershire sauce, garlic, and black pepper. It is often prepared tableside. It is generally attributed to restauranteur Caesar Cardini, an Italian immigrant who operated restaurants in Mexico and the United States. Cardini was living in San Diego but also working in Tijuana where he avoided the restrictions of Prohibition."* When tokenized according to the text module will become the sequence of words as seen in the previous sentence. When tokenized according to the link model will become the sequence:⟨*salad, romaine lettuce, croutons, parmesan cheese, olive oil, Worcestershire sauce, Caesar Cardini, Italian San Diego Tijuana Prohibition*⟩.

The *position* of a token in a document is an index number identifying its place in the sequence of tokens that compose the document. Documents can in general be modified through a series of *primitive changes* on tokens including insertions, deletions, substitutions, reorderings, etc. We focus only on substitutions, where existing token(s) are replaced by others. This is predicated on the assumption that controversies are often expressed via substitutions, that is, authors having opposing viewpoints tend to argue back-and-forth over alternative content, rather than inserting new content or deleting old content. Indeed, our data analysis in Section 4.4 supports this.

Definition 4.2.2. *Given a document* $D{=}w_1w_2\ldots w_n$*, a* substitution *of its sub-sequence of tokens* $E_b{=}w_pw_{p+1}\ldots w_{p+k}$ *with the sequence of tokens* $E_a{=}w'_1w'_2\ldots w'_q$ *is an action that convents the document* D *to the document* $D'{=}w_1w_2\ldots w_{p-1}w'_1w'_2\ldots w'_qw_{p+k+1}\ldots w_n$*. The* subject *of the substitution is the pair* $\langle E_b, E_a\rangle$*, and its* position *is the number* p*.* ∎

Changes are performed by authors at specific times. We assume a set $\mathcal{A}$ of authors, and a time domain $\mathcal{T}$, and we define an edit on a document to be the triple of the substitution change alongside the author that performed it and the time s/he did so.

Definition 4.2.3. *An* edit e *is a tuple* $\langle c_e, a_e, t_e\rangle$*, where* c_e *is a substitution and* $a_e{\in}\mathcal{A}$*,* $t_e{\in}\mathcal{T}$ *are the* author *and* time *of the edit, respectively. The* subject *of the edit* e*,* $subj(e)$*, and its position* $pos(e)$ *are the subject and position of the substitution* c_e*, respectively.* ∎

We make the natural assumption that no more than one author can perform changes on a document at the same time, but we do allow multiple changes to happen at the same time from the same author. The rationale behind this is that authors are typically performing a number

[5] http://en.wikipedia.org/wiki/Wikipedia:Featured_article_criteria

of changes in the document and then submit them all together (i.e., by clicking on the save button). Through a sequence of edits, a document changes form. We denote by $D_b \xrightarrow{e} D_a$ the fact that an edit e applied to a document D_b results in the document D_a. For a sequence of edits $E{=}e_1, e_2, \ldots, e_m$, having $time(e_1){=}time(e_2){=}\ldots{=}time(e_m){=}t$, the notation $D_p \xrightarrow{E} D_a$ is a shorthand for $D_p \xrightarrow{e_1} D_1 \xrightarrow{e_2} D_2 \xrightarrow{e_3} \ldots \xrightarrow{e_m} D_a$. By abuse of notation, $time(D_a)$ and $time(E)$ will be referring to the time t.

Definition 4.2.4. *A* version history h *is the sequence of documents* D_0, D_1, …, D_m *such that* $D_0 \xrightarrow{E_1} D_1 \xrightarrow{E_2} D_2 \xrightarrow{E_3} \ldots \xrightarrow{E_m} D_m$. E_1, E_2, …, E_m *is an inferred progression of edit sequences such that: (i) for every two edits* $e, e' {\in} E_i$, $time(e){=}time(e')$*; and (ii) for every two edits* $e {\in} E_j$ *and* $e' {\in} E_i$*, with* $j{<}i$, $time(e){<}time(e')$*, for* $i{=}1..m$ *and* $j{=}1..(m-1)$. *The* edits of the page history *is the set* $E_1 {\cup} E_2 \ldots {\cup} \ldots {\cup} E_m$ ∎

Definition 4.2.5. *A* collaboratively edited database*, or* database *for short, is a set of version histories* $\{h_1, h_2, \ldots, h_k\}$. *Each version history* h_i *is referred to as a* page P_i*, and the documents in* h_i *are the* versions *of the page* P_i. ∎

Note that given a page P, there may be more than one set of sequences that can serve as edits of the page history. For many practical applications, the exact edits may not be important, or may not be possible to know. For instance, in Wikipedia, the authors are editing the pages and committing to the system the edited version without any extra information. It is up to the system to compare the previous version with the one committed by the author and generate a set of edits, if that is needed, that when applied to the former generates the latter.

There are many definitions and interpretations of what a controversy is. A widely accepted definition is the one provided by Wikipedia itself which states that a *controversy* is a prolonged dispute by a number of different people on the same subject. [6] Since this is a semantic definition, to discover controversies there is a need for a syntactic interpretation. In collaboratively generated databases like Wikipedia, this form of disputes can be observed through the edits that the different authors have performed on the pages. Thus, the problem of identifying controversies boils down to the identification of groups of edits that represent such disputes.

Identifying edits on the same subject matter is a challenging task since there is no way to uniquely identity topics in a document that will allow them to be tracked over time. One idea is to use the positions edits as a reference; unfortunately, positions from one version to another may significantly change. A second idea is to use only the subject of the edit, that is, the change that actually took place. Unfortunately again, neither this is a viable solution, since the same content may appear in different places in a document with very different meanings. For example, the fact that "Polish" was replaced by "French" in two different versions of a page does not mean that the two users that performed these two edits were referring to the same thing. The first may refer to, say, the nationality of a person while the second may refer to the language a manuscript is written. The only way to understand this is to also consider surrounding content to take the right semantics into account. We refer to this surrounding content as the *context* of

[6] Source: http://en.wikipedia.org/wiki/Controversy

the edit. We posit that two edits with the same or very similar context likely refer to the same topic. An important issue related to context is to decide how much surrounding content should be considered. Naturally, the more tokens considered as context the better. However, too many tokens increase the risk of including tokens modified by other edits, reducing the similarity of context of the edit to others on the same subject matter. Therefore, we introduce the notion of the *radius* of a context.

Definition 4.2.6. *The context of an edit e of radius r on a document D with tokens $w_1 w_2 \dots w_n$, $ctx(e)$, is the sequence $w_{p-r} w_{p-(r-1)} \dots w_{p-1} w_{p+1} \dots w_{p+(r-1)} w_{p+r}$ of tokens, where $p{=}pos(e)$.* ∎

Note that since the context does not contain the subject of the edit, it can successfully identify edits about the same subject matter even in cases in which traditional semantic similarity techniques may fail. For instance, a substitution of the expression "his birthday is unknown" with the "born in March 1821" and of the expression "born in circa 1800" with the "lived from March of 1821" are essentially saying the same thing. A direct matching of the two edits will most likely conclude incorrectly that they are not related,

For measuring the similarity of the contexts of edits we use the well-known Jaccard distance [Jaccard, 1901], and we use this distance as a metric of similarity between edits.

Definition 4.2.7. *The distance between two edits e_1 and e_2 is $\frac{|S_{ctx(e_1)} \cap S_{ctx(e_2)}|}{|S_{ctx(e_1)} \cup S_{ctx(e_2)}|}$, where S_σ denotes the elements of sequence σ represented as a set.* ∎

Example 4.2.8. Suppose that on the 7th of May of 2008, the editor Mogism replaced in the Wikipedia entry from Example 4.2.1 the text *Caesar Cardini* with *Julius Caesar*. This modification is modeled by the edit
$e{=}\langle\langle CaesarCardini, JuliusCaesar\rangle, Mogism, 2008/05/07\rangle$. The subject of the edit is the pair $\langle CaesarCardini, JuliusCaesar\rangle$. Assuming context radius is equal to 2, for the text model the context of the above edit is
$\langle to, restauranteur, an, Italian\rangle$ while for the link model it will be
$\langle$*olive_oil, Worcestershire_sauce, Italian, San_Diego*$\rangle$.

To quantify the time span of a set of edits E, we define the *duration* of the set to be the time $time(e_m) - time(e_1)$, where e_1 and e_m are the earliest and latest edits in E, respectively. Similarly, we introduce the notion of *plurality* of a set of edits as the number of distinct users that have performed these edits, and the *cardinality* as the number of edits in E.

4.3 Detecting Controversies

To identify controversies, we need to look for edits in the history of a page that are about the same subject matter, have taken place in a period of some certain duration and have been performed by at least a certain number of users. Each group of edits that has these characteristics is an indication of a controversy. Thus, our controversy detection algorithm returns a set of sets of edits.

Algorithm 3 Controversy Detection Algorithm (CDA)

Input: h: A page history
Output: Set $\mathcal{C}$ of controversies

(1) // Edit extraction
(2) $E \leftarrow \emptyset$
(3) **foreach** i=1..($|h| - 1$)
(4) $t \leftarrow$ Time $h[i+1]$ was created
(5) $u \leftarrow$ User that created $h[i+1]$
(6) $\mathcal{E} \leftarrow$ MYERSALG($h[i]$,$h[i+1]$)
(7) **foreach** j=1..($|\mathcal{E}| - 3$)
(8) **if** $\mathcal{E}[j]$ is an $eq \wedge \mathcal{E}[j+1]$ is a $del \wedge$
(9) $\mathcal{E}[j+2]$ is an $ins \wedge \mathcal{E}[j+3]$ is an eq
(10) *Let* $del(E_d)$ be the $\mathcal{E}[j+1]$
(11) *Let* $ins(E_i)$ be the $\mathcal{E}[j+2]$
(12) **if** $|E_i| < n_{thrshd}$ and $|E_d| < n_{thrshd}$
(13) $E \leftarrow E \cup \langle\langle E_d, E_i\rangle, t, u\rangle$

(15) // Eliminate edits with low user support
(16) **foreach** $e \in E$
(17) $E_{eq} \leftarrow \{e' | e' \in E \wedge subj(e) = subj(e')\}$
(18) $U_e \leftarrow \{user(e') | e' \in E_{eq}\}$
(19) $U_e \leftarrow$ ELIMINATEDUPLICATES(U_e)
(20) **if** $|U_e| < k_{thrshld}$
(21) $E \leftarrow E$ - E_{eq}

(23) // Cluster edits based on context and using as distance
(24) // metric among the edits the distance of their contexts
(25) $\mathcal{E} \leftarrow$ CLUSTER(E, "$context$", $r_{thrshld}, cutoff^{ctx}_{thrshld}$)

(27) // Cluster & merge the sets of edits based on the
(28) // subject of their edits
(29) $\mathcal{C} \leftarrow \emptyset$
(30) $\mathcal{M} \leftarrow$ CLUSTER($\mathcal{E}$, "$subject$", $cutoff^{sub}_{thrshld}$)
(31) **foreach** cluster $M \in \mathcal{M}$
(32) $\mathcal{C} \leftarrow \mathcal{C} \cup flatten(M)$
(33) $\mathcal{C} \leftarrow top_k(\mathcal{C})$

Although one can look at the pool of all the edits of the pages of a database, in this work we focus on controversies identified within a specific document. Thus, we restrict our analysis on the sets of edits in the history of each specific page independently. Algorithm 3 (CDA for short) provides an overview of the steps we follow. The variables in the algorithm with the subscript

thrshld are configuration constants the values of which are specified offline.

The first step that needs to be done is to identify the edits that have taken place in the history of a page. Recall, that we typically have the documents in the history and not the edits. The edits are discovered by comparing consecutive documents in the history. For each pair of such consecutive documents (i.e., versions) the Myers' algorithm [Myers, 1986] is executed. The output of the algorithm is an edit script. The elements in the script can be of three different types: (i) $eq(E)$ indicating that a set of tokens E remained unchanged between the two page versions that were compared; (ii) $del(E)$ indicating that the sequence of tokens E of the first version was deleted; and (iii) $ins(E)$ indicating that a sequence of tokens E was inserted in the second version. In the generated edit script we are looking for patterns of a del followed by an ins that are located between two eq. This pattern indicates a substitution which is what is of interest to us. From the discovered patterns of this type, we do not consider those that the number of tokens in the ins or the del operation is more than a specific threshold $n_{thrshld}$. The intuition behind this which is based on a number of experiments that we have performed is that edits involving large pieces of text are not indicating controversies but are most of the time vandalisms. The remaining patterns are turned into substitution edits. (lines 1-13)

From the edits generated in the previous step, we eliminate those that have not been repeated by at least $k_{thrshld}$ users. The idea is that if an edit does not enjoy any broad support, i.e., has been made by only one (or extremely few) users, it is some personal opinion and not a point of view reflected by a considerable set of users. In counting the number of times that an edit has been repeated by a user, we consider only the subject of the edit and not its context, since the same user may perform the same edit in different places and in different versions. A typical value we consider for $k_{thrshld}$ is 2, i.e., eliminating single-user edits. (lines 15-21)

The remained edits are going through a process that tries to group together edits that are about the same subject matter. For this task a clustering algorithm is employed. The choice of the clustering algorithm is an orthogonal issue. CDA uses it as a black-box. However, for our implementation we have chosen the well-known DBSCAN [Sander et al., 1998] algorithm. The algorithm requires a metric for the similarity among the elements it needs to cluster. This similarity is based on the context of the edits and is measured using the Jaccard distance as described in the previous section. The step is configurable using the context radius $r_{thrshld}$ and the threshold level $cutoff^{ctx}_{thrshld}$ that the DBSCAN algorithm uses decide on whether two elements are similar enough to be in the same cluster. The outcome of this step is a set of groups of edits. Each one of these groups is considered to represent a controversy. (line 25)

Since the clustering performed in the previous step was based on the context of the edits and not on their subject, it may be the case that a controversy about a topic that appears in different parts of the document may end up been recognized more than one times, i.e., we may end up with some different groups of edits that talk about the same subject matter. To reconcile these groups into one, so that every controversy is represented by one group of edits, the clustering algorithm is executed once again, but this time on the sets of edits and not on the edits themselves. The similarity function used among the groups is again the Jaccard similarity but this time applied on the sets of edits that each group contain. The similarity function finds the edits that are common to two different groups by a comparison on their subject. (lines 29-30)

Before returned to the user, the found sets of edits are ranked based on the level of controversy they describe. This level of controversy can be determined by many factors. Our goal is not to define the right metric for that purpose. The CDA is treating it as a black box. However, in our implementation we have used the cardinality, duration and plurality as such a metric with the assumption that a controversy is stronger, the more users it involves, the more it last and the more for the power of the found controversies. (line 31)

4.4 Experimental Evaluation Setting

For the experimental evaluation we use the Wikipedia dump from March 2012. It consists of a set of entries in Wikitext (or Wiki-markup) format each one with all its historical versions. For each version the information on the time and the author that submitted the version is also contained. We extract a fraction of these entries that are related to religion, since this is a topic known to have many controversial issues. We did so by recursively extracted all the pages within up to 10 nested subcategories of the *Religion* category page[7] (July 2012). This resulted to $204K$ pages. We refer to this dataset as the *large dataset*.

We have implemented CDA (see Algorithm 3) in Java and we conduct the experiments on a 4GHz CPU PC with 4G memory running Ubuntu 12.04. For the discovery of links in the Wikimedia content of the Wikipedia entries we use the JWPL Wikimedia parser[8]. The CDA parameter configuration is summarized in Table 4.1.

For the experimental evaluation we use Wikipedia. The Wikipedia database consists of a set of articles ("pages") in Wikitext (or Wiki-markup) format as well as a changelog for each article charting its page history. The date and time of each edit is recorded, along with the IP address associated with the editor and also an account name, if the editor was logged in at the time of the edit.

The edit policy of a page is intended to protect against vandalism and edit wars, offering multiple levels of protection. Some pages allow anyone any editor including anonymous ones. Others, typically on more sensitive topics, require a trusted account to edit them. In general, there is a *three revert rule* (3RR), allowing no more than three reverts on any single page within a 24-hour period. Naturally, there are exceptions to this; for example, reverting vandalism or one's own content does not count towards 3RR. A *request for mediation* (RFM) can be filed on the administrators' noticeboard have the level of protection for an article increased, after which it may become 1RR.

In addition to articles, Wikipedia contains corresponding "talk pages" where editors can any discuss issues that arise in editing a page and which can be accessed by clicking the "Talk" tab at the top of any article. As has been noted before, talk pages are not always useful for identifying controversies ince they serve different utilities in different languages and understanding them would require natural language processing.

[7] http://en.wikipedia.org/wiki/Category:Religion
[8] http://code.google.com/p/jwpl/

parameter	range	default value
model	`link`, `text`	`link`
$n_{threshld}$	1,2,3,4,5	5
$r_{thrshld}$	2,4,6,8	8
$cutoff^{ctx}_{thrshld}$	[0 ... 1]	.75
$cutoff^{sub}_{thrshld}$	[0 ... 1]	.8
$k_{thrshld}$	1,2	2

Table 4.1: CDA Configurations parameters

4.4.1 Sources of Controversy Information

To study the effectiveness of CDA, it is required to know the ground truth, i.e. what Wikipedia entries, or what part of the entries are actually controversial. To obtain this information we used three different sources: (i) the list of textual description of controversies provided directly by Wikipedia; (ii) the controversy related templates as inserted by Wikipedia authors wherever it is believed that a controversy exists; and (iii) the controversies identified by users in the user evaluation experiments we have conducted.

[Wikipedia Provided Controversies (WPC)] Wikipedia provides a list[9] of controversies from which we manually extracted 263 Wikipedia entries with controversies. As a result we produced a list consisting of pairs ⟨entry,description⟩, with the former being the Wikipedia entry and the latter being the textual description of the controversy in the specific entry. As an example, one such pair is about the Wikipedia entry of `Freddie Mercury` and the respective description explains that there is a controversy about the ancestry of the famous singer on whether he is the most famous Iranian, Indian, Parsi or Azeri rock star.

The Wikipedia provided controversies is that they are well-known and broadly accepted controversies, nevertheless the list is by no means complete.

[Template-indicated] Another source of controversies that has been used in other works [Vuong et al., 2008] is based on the dispute template messages[10] that editors of the Wikipedia entries are leaving in the text to notify other editors and readers about an issue related to the page, a paragraphs or some words. There are many types of templates. Among them, we consider only those directly related to disputes, i.e., the `contradict`, `contradict-other`, `disputed`, `dispute about`, `disputed-category`, `pov`, `pov-check`, `need-consensus`, `disputed-section`, `pov-section`, `pov-intro` and `pov-statement`. (see Table 4.2).

For a Wikipedia entry that contains controversy templates, we need to be quantify how controversial the page is. To do so, we measure the number of controversial templates in the all the versions in the history of the entry. In particular, we use the *Article Tag Count*(ATC) [Vuong et al., 2008] metric:

$$ATC = \sum_{i=1}^{n} c_i \tag{4.1}$$

where n is the number of versions of the entry and c_i is the number of controversy related tags in the version i.

[9] http://en.wikipedia.org/wiki/Wikipedia:Lamest_edit_wars

[10] http://en.wikipedia.org/wiki/Wikipedia:Template_messages/Disputes

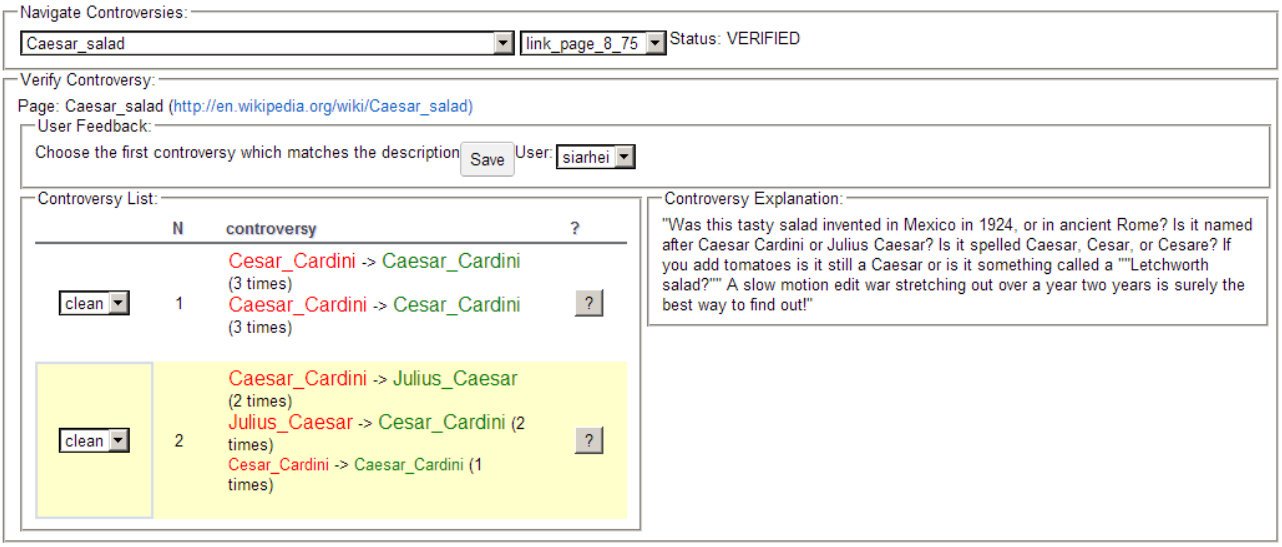


Figure 4.1: GUI for user-based evaluation

Templates offer many more controversies that the list of known Wikipedia controversies since they are flexible and easy to use. Nevertheless, despite their flexibility, a recent study has shown [Vuong et al., 2008] that they are not extensively used, thus, the information they provide on the controversies is neither in this case complete.

[User-specified Controversies] As a third form of controversy source is the set of users employed to conduct the user-based evaluation of CDA. We refer to the set of controversies provided in that way as the user specified controversies. The user-specified controversies have the advantage that they can provide more details on whether and why a wikipedia entry (or part of it) is controversial, and on whether the entry is more controversial than another. Of course, the set of pages which the users used in our evaluation can judge is considerably smaller then the set of pages of Wikipedia. Practically it is difficult to run a large scale evaluation since there are thousands of controversies in Wikipedia entries. Furthermore, the users have to be domain experts in order to provide a well-founded and trustful opinion on whether a piece of a Wikipedia entry that seems to have been edited extensively is controversial or not.

For the user-based evaluation the GUI illustrated in Figure 4.4.1 is built. On the left-hand side, it appears the list of sets of edits (i.e., the suspected controversies) as discovered by the CDA. On the right-hand side it appears textual descriptions of the known controversies. The user is asked to match the textual description with the right set of edits on the left. This allows the evaluation of CDA on how well it can identify known controversies.

The users are also used to create the ground truth by the results that the CDA has returned, i.e., the sets of sets of edits C. In particular, for every edit in every set that CDA has returned, the users are asked to provide a tag (topic) indicating a what they believe is the controversy that the edit is about, or provide the tag *unspecified* if they do not find any. Those tagged with the tag *unspecified* are discarded and the rest are re-grouped in a way that all the edits with the same tag end up in the same group and not two edits with different tags end up in the same group. The formed groups are then playing the role of the ground truth since they contain all the edits

Template	Meaning
`contradict`, `contradict-other`	This article or section appears to contradict itself or other articles.
`disputed`, `disputed-category`, `dispute about`, `disputed-section`	This article's factual accuracy is disputed (about a specific topic, category or in a section).
`pov`, `pov-check`, `pov-section`, `pov-intro`, `pov-statement`	The neutrality of this article is disputed (in the introduction, section or about a specific statement).
`need-consensus`	This article or section needs consensus.

Table 4.2: Template descriptions

that the CDA was also containing (apart of the *unspecified* of course) but groups in groups that the user specified. We refer to this ground truth as $ideal(C)$.

4.4.2 Metrics

To evaluate the results of CDA, we need to compare them with the ground truth. For this we use a number of different metrics to measure its different quality aspects.

Given a controversy, i.e., a set of edits, c found by CDA, and the set of tags that the user has provided to the sets of edits in all the found controversies, we define the noise of c, denoted as $noise(c)$, to be the set of edits taged with the *unspecified* tag. On the other hand, $clean(c)$ denotes the set of edits tagged with some topic specific tag, i.e. $clean(c) = c \setminus noise(c)$. The controversy c is said to be *clean* if $noise(c) = \emptyset$ and *noisy* otherwise.

[Noise/signal ratio] To realize the amount of edits among those returned by CDA that were not categorized to any topic, i.e., those tagged with the *unspecified* tag, we use the noise/signal ratio:

$$noise/signal = \frac{\sum_{i=1}^{k} \mid noise(c_i) \mid}{\sum_{i=1}^{k} \mid c_i \mid} \tag{4.2}$$

where k indicates the number of controversies returned by CDA. A noise/signal ratio of 1 indicates that there are no edits in topK results that could be successfully tagged with a tag describing a topic.

[AvePr] To measure the precision of controversies found by CDA, we use the average precision [Hawking et al., 2001]. In this experiment, a controversy is said to be *relevant* if it has at least one edit which is annotated by a topic tag. Note that the average precision accounts both for the number of relevant controversies and their order, i.e. the higher the relevant controversies in the result list, higher the average precision.

$$AvePr = \frac{\sum_{i=1}^{k} P(i)rel(i)}{\#\ of\ relevant\ controversies} \tag{4.3}$$

where k is the number of retrieved controversies, $P(i)$ is the precision at position i and $rel(i)$ is a function which takes 1 if the i-results is a relevant controversy and 0 otherwise. Note, that the

denominator is the number of relevant controversies in topK results and not the total number of relevant controversies in the dataset.

[Rand index] To compute the distance between the retrieved controversy C and the ideal $ideal(C)$ we employ a clustering evaluation metric, namely the Rand index [Rand, 1971]. Informally, the Rand index measures the blurriness of the retrieved controversies according the user provided ground truth.

$$Rand = \frac{a+b}{\binom{\sum_{i=1}^{k} \mid clean(c_i) \mid}{2}} \tag{4.4}$$

where $\sum_{i=1}^{k} \mid clean(c_i) \mid$ is the total number of topic edits in the top k controversies, a is the number of edit pairs that are in the same group in $clean(C)$ and in the same group in $ideal(C)$ and b is the number of edit pairs that are in different groups in $clean(C)$ and in different groups in $ideal(C)$.

[# of distinct controversies] For a set of controversies, C, we compute the total number of topics which can be found in it, i.e.

$$\#\ of\ distinct\ controversies = \mid ideal(C) \mid \tag{4.5}$$

The next two metrics are computed using the user feedback on a controversy and not edits. Such, the users either specify whether the controversy matches the given textual description (*Recall on WPC*) or the controversy corresponds to some controversial topic (*Precision on a set of pages*).

[Recall on WPC] To compute the recall when we know one controversy per page, i.e. the case of the list of WPC, for a given number of retrieved controversies k we count the number of pages which have the corresponding controversy in its first k controversies. Hence, the recall is computed as follows:

$$Re = \frac{\#\ of\ pages\ with\ controversy\ in\ topk\ results}{n} \tag{4.6}$$

where n is the number of pages we know controversies in.

[Precision on a set of pages] For a given number of retrieved controversies k and a ranking function f (for possible ranking functions see Section 4.3) we measure the precision on a set of n pages as follows. First we rank the controversies within a page using f and pick the top1 controversy. Second we rank pages using f of top1 controversies of pages. Finally, the precision is the ratio between the number of relevant controversies for topK pages and k.

$$Pr = \frac{\#\ of\ pages\ which\ top1\ is\ relevant}{k} \tag{4.7}$$

Even though we have both precision and recall we do not report F-measure since those metrics use different ground truth sources, i.e. for recall we use the list of WPC and for precision we conduct the user study. Moreover, the recall is computed with the assumption that for every page we know only one controversy (its textual description) what makes recall completeness bounded to WPC.

4.5 Experiments

In this section we present our experimental findings. We start with the discussion of the importance of links in Wikipedia (Section 4.5.1). Then, we present some detailed experiments which show how the links can be effectively used to identify the trustful content (Section 4.5.2). In Section 4.5.3 we present detailed experiments and discussion on the the `link` and `text` data models. Then, we study the recall of the CDA algorithm on the set of pages with WPC (Section 4.5.4). The precision of our algorithm is measured both on the set of pages with WPC (Section 4.5.5) and on a large set of Religion Wikipedia pages (Section 4.5.6). We present the results of controversy detection at the page level (Section 4.5.7), i.e. how the edits are correlated with the page controversiality in comparison to the start-of-art approaches. Section 4.5.8 describes the visualization technique we use to facilitate the process of controversy identification. Finally, in Section 4.5.9 we present distributed controversies, i.e. the controversies which span across a number of different Wikipedia pages.

4.5.1 Importance of Links

To study the importance of links to a page content we run the following experiment. For all revision of the Coca-Cola Wikipedia page we measured the amount of change by computing the Levenshtein distance [Levenshtein, 1966] between the revision of interest and its immediate descendant. Than, varying the threshold of Levenshtein delta between consecutive revision we measured the percentage of them which had link changes (both addition/removals and combinations). The results are presented in Figure 4.2 which shows that the percentage of edits which involves link changes grows logarithmically (starting from 43%) with the increasing size of edit expressed by the Levenshtein distance. That in turn means that links are an essential part of small (every second edit has a link update) and large edits.

4.5.2 Trust and Links

In this experiment we investigate how links can improve the methods of trust analysis in Wikipedia. More specifically, we focus on the methods of the WikiTrust system [Adler and de Alfaro, 2007, Adler et al., 2008] which computes the reputation of Wikipedia authors and the trust values of the text of a page. To represent the content of a page, WikiTrust uses a sequence of words. In our research we claim that links can be seen as an alternative data model with some important properties (see Section 4.2). Therefore, we conduct an experiment where we compare the results of the Wikitrust system for two data models, i.e. when a page is represented as a sequence of words (`text`) and when a page is a sequence of its internal Wikipedia links (`link`).

The details of the following experiment can be found in [Adler and de Alfaro, 2007]. The obtained results are shown in Table 4.3. The first column (WikiTrust all revision, text) represents the results from the original Wikitrust system where the underline data model is `text`. The effectiveness is reported as the precision and recall for the cases with and without anonymous authors as well as for the text (Text) and edit (Edit) longevities. Moreover, we also report the efficiency as the time which the system needs to compute the reputation/trust values (Time (min)). The second column (WikiTrust all revision, unique links) presents the precision/recall results where a sequence of unique links (i.e. links which appear only one time in the sequence)

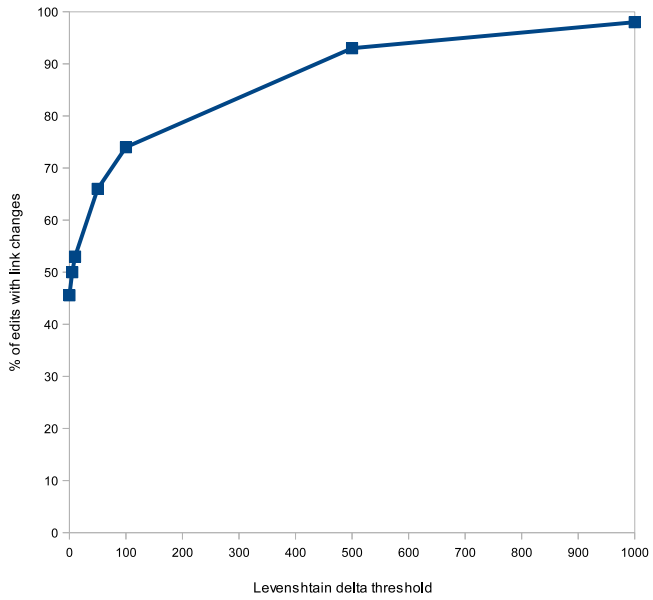


Figure 4.2: Coca-Cola edits which involve links

is used to represent the content of a Wikipedia page. In the third column (WikiTrust only link altered revision, unique links) we report the results for the link case where we consider only those revisions where some links where altered.

			WikiTrust: all revisions, text	WikiTrust: all revisions, unique links	WikiTrust: only link altered revisions, unique links
with anonymous	Text	Precision	11.91	32.15	14.22
		Recall	79.13	87.02	54.38
	Edit	Precision	24.85	14.22	32.15
		Recall	85.04	54.38	87.02
without anonymous	Text	Precision	3.00	5.23	3.01
		Recall	19.48	16.37	16.87
	Edit	Precision	9.01	3.06	5.27
		Recall	25.78	16.87	16.37
Time (min)			176	86	62

Figure 4.3: Wikitrust efficiency and effectiveness on the `text` and `link` data models

From the above table we conclude that the `link` data model leads to big benefits in the efficiency (from 176min to 62min), without any appreciable difference in the precision/recall.

4.5.3 Detailed Data Model Experiments

In the following experiment, we manually inspect the results of CDA for a predefined set of controversial Wikipedia pages. As a result, we report some observations which the following experiments are based on (Sections 4.5.4, 4.5.5 and 4.5.6). As the dataset we choose the Frederic Chopin, Barack Obama, and UEFA pages because those topics are expected to be controversial due to their high visibility. We test the following parameters: `text` and `link` data models; $n_{thrshld}$ takes the values from 1 to 5, $r_{thrshld}$ takes 3 and 10, $cutoff^{ctx}_{thrshld}$ takes 0.1 and 0.9 and $k_{thrshld}$ takes 1 and 2. In Figure 4.4 you can see the controversies from the Frederic Chopin page under the `link` data model with $r_{thrshld} = 3$, $cutoff^{ctx}_{thrshld} = 0.9$ and $k_{thrshld} = 2$.

A group of edits (between the lines of hyphens) represents a found controversy where every line is a single edit. In an edit, the first and second words represent the subject (e.g. `feb_22` is substituted for `March_1`). In addition, we show the time stamp, the author identifier and the revision number of that edit (the words delimited by tabs). For example, the first group is about the date of birth of Chopin where two links `March_1` and `Feb_22` were substituted many times. Note, that there are different kinds of links in the subjects which represent the same concept (e.g. `feb22`, `feb_22`). Using the clustering of edits by context we are able to group edits which have different subjects but refer to the same topic. Also we can see the controversy about Chopin birthday is split into two controversies # 1 and # 4. Manually inspecting the revisions of the controversies # 2 and # 3 we classified them as the controversies about the main painting of Chopin and his name spelling, respectively.

In the following, we report some observations from the CDA results on the above pages.
Text doesn't subsume links. The `text` model which uses $n_{thrshld}$ up to 5 cannot identify many controversies which are found by the `link` model. The main reason is that an edit of link can be translated to the `text` model as a very complex change. For example, in the Chopin page the link edit is a substitution of 9 words for 1 word.
$k_{thrshld} = 2$ *induces false positives.* For $k_{thrshld} = 2$ we may prune useful edits, e.g. in Chopin page we lost edits about Chopin's nationality because they are not repeated by a different user.
The precision of `text` *is low.* The precision of the `text` model is low because there are large clusters of edits which are not semantically meaningful controversies whereas the meaningful controversies can have a smaller number of edits.
Subtle Controversies. We found some controversies which are about some subtle changes in a content. Such, in the Obama page the controversy "a leading candidate for the Democratic Party nomination" vs "the leading candidate for the Democratic Party nomination" has attracted a lot of attention and, moreover, is accompanied with the POV template.
The recall of `text` *is high.* The UEFA page is not controversial from the `link` data model point of view, however the `text` model detects some factual controversies along with many noisy controversies.
$cutoff^{ctx}_{thrshld}$ *and* $r_{thrshld}$ *Trade-offs.* The `link` model with a small $r_{thrshld}$ and a high $cutoff^{ctx}_{thrshld}$ produces semantically more precise controversies than the `text` model with the same thresholds. Increasing $r_{thrshld}$ and lower $cutoff^{ctx}_{thrshld}$ for the `text` model allows one to find missing controversies but at the same time the results have more noise. Thus we can conclude that the `link` model is more suitable to capture semantic controversies.

```
------------ 1--------------
March_1_222     March_1 2006-03-25_21:29:21.607 603177  45468304
feb_22  March_1 2006-05-13_19:55:08.24  44727   53041322
March_1_2_22    March_1 2006-03-20_20:39:32.420 220765  44706611
feb_22  March_1 2006-05-04_22:42:49.760 44727   51592359
feb_22  March_1 2006-05-13_19:58:10.195 44727   53041761
March_1_222     March_1 2006-03-29_00:34:49.825 225832  45933589
feb_22  March_1 2006-04-24_22:18:59.136 57658   49986873
March_1_2_22    March_1 2006-03-16_23:17:28.342 220765  44125817
feb22   March_1 2006-05-07_02:06:29.869 521704  51918425
feb_22  March_1 2006-05-13_19:43:32.634 44727   53039784
feb22   March_1 2006-05-11_00:35:11.259 57658   52582845
feb_22  March_1 2006-05-13_20:02:03.242 44727   53042287
feb_22  March_1 2006-05-04_20:59:51.682 521704  51578140
feb_22  March_1 2006-05-13_19:45:43.680 44727   53040075
March_1222      March_1 2006-03-28_23:24:56.779 265063  45924187
March_1_222     March_1 2006-03-28_21:49:31.747 265063  45910895
March_1222      March_1 2006-03-22_17:24:18.513 57658   44970677
March_1_222     March_1 2006-03-21_22:52:30.467 57658   44863857
feb_22  March_1 2006-05-13_19:12:22.556 44727   53035885
feb_22  March_1 2006-05-13_19:50:29.899 44727   53040682
March_1_222     March_1 2006-03-27_21:19:01.638 44727   45755820
feb_22  March_1 2006-05-13_19:27:40.602 44727   53037765
March_1_222     March_1 2006-03-22_00:44:24.513 44727   44877308
feb_22  March_1 2006-05-13_19:48:54.805 44727   53040479
feb_22  March_1 2006-05-13_19:46:39.727 241337  53040192
feb_22  March_1 2006-05-04_22:22:41.713 57658   51589575
feb_22  March_1 2006-05-13_19:54:11.977 44727   53041170
feb22   March_1 2006-05-09_21:12:26.947 44727   52376086
feb_22  March_1 2006-05-13_20:08:35.367 44727   53043223
Feb_22  March_1 2006-04-28_20:35:14.292 173640  50638522
Feb_22  March_1 2006-05-10_22:02:13.212 44727   52560030
feb_22  March_1 2006-05-13_20:06:16.320 44727   53042862
March_1_2_22    March_1 2006-03-18_19:30:19.389 44727   44393641
feb_22  March_1 2006-05-13_19:57:04.148 44727   53041601
March_1_222     March_1 2006-03-30_22:13:27.919 57658   46228059
March_1222      March_1 2006-03-29_20:57:14.872 675573  46062321
feb22   March_1 2006-05-13_20:03:25.273 44727   53042473
feb_22  March_1 2006-05-06_20:43:27.838 44727   51878961
feb22   March_1 2006-05-09_23:57:59.994 44727   52401280
------------ 2--------------
Frédéric_Chopin Eugène_Delacroix        2008-09-20_09:03:28.388 4062893 239755239
Eugène_Delacroix        Frédéric_Chopin 2008-09-19_07:39:51.13  7826095 239497766
Eugène_Delacroix        Frédéric_Chopin 2008-09-21_16:20:48.746 1229061 240023394
Eugène_Delacroix        Frédéric_Chopin 2008-09-21_10:55:27.512 7826095 239976182
Frédéric_Chopin Eugène_Delacroix        2008-09-19_04:32:43.888 4062893 239470252
Frédéric_Chopin Eugène_Delacroix        2008-09-21_22:16:31.996 4062893 240096436
Frédéric_Chopin Eugène_Delacroix        2008-09-20_01:46:04.138 4062893 239704158
Eugène_Delacroix        Frédéric_Chopin 2008-09-22_01:37:48.105 716485  240134394
Frédéric_Chopin Eugène_Delacroix        2008-09-21_15:29:34.622 5877666 240013578
Eugène_Delacroix        Frédéric_Chopin 2008-09-20_07:56:33.263 7826095 239748594
Eugène_Delacroix        Frédéric_Chopin 2008-09-18_22:40:57.654 7826095 239394424
------------ 3--------------
Fryderyk_Chopin's_illness       Frédéric_Chopin's_illness       2011-01-27_13:04:05.529 33566   410359968
Fryderyk_Chopin's_illness       Frédéric_Chopin's_illness       2010-10-26_22:59:41.565 13324505        393088972
Frédéric_Chopin's_illness       Fryderyk_Chopin's_illness       2011-01-27_11:01:42.310 13878188        410347758
Frédéric_Chopin's_illness       Fryderyk_Chopin's_illness       2010-10-26_22:46:06.367 0       393086869
------------ 4--------------
February_22     March_1 2005-10-13_05:43:48.587 41799   25411707
February_22     March_1 2003-03-02_05:28:37.171 62      774883
```

Figure 4.4: Chopin controversies for the `link` data model

4.5.4 Recall of CDA

In this experiment we aim at measuring the recall of the CDA, i.e. what fraction of WPC the algorithm is able to retrieve.

As ground truth we use the list of WPC where we select 25 pages for which our algorithm produces at least one match in top 10 results under at least some parameter setting (the details on the parameter settings see below). We run the algorithm on those 25 pages and retrieve top

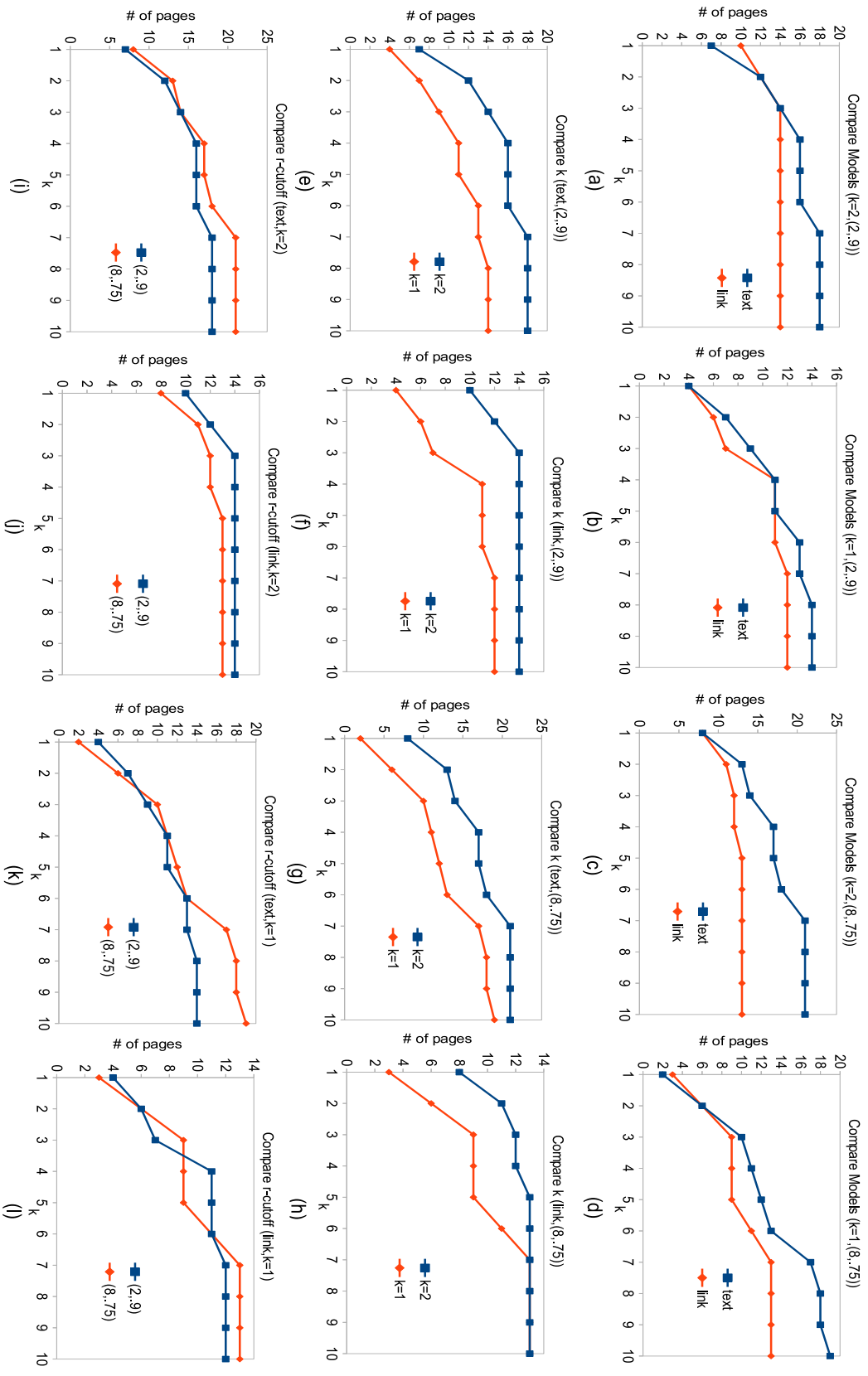


Figure 4.5: The recall experiment on WPC where the first row (a,b,c,d) compares models, the second row (e,f,g,h) compares $k_{thrshld} = 1$ and $k_{thrshld} = 2$, and the third row (i,j,k,l) compares the $(2, .9)$ and $(8, .75)$ $(r_{thrshld}, cutoff^{ctx}_{thrshld})$ parameter pairs.

10 controversies per page ranked by the cardinality of controversy. Then every controversy of a page is validated by the user who checks whether the corresponding textual description of the page matches the retrieved controversy.

For the recall measurement we test both the text (`text`) and link (`link`) models, $k_{thrshld}$ of 1 and 2, $r_{thrshld}$ of 2 and 8, and $cutoff^{ctx}_{thrshld}$ of .9 and .75. The above choices of $r_{thrshld}$ and $cutoff^{ctx}_{thrshld}$ are based on the following empirical observations. By varying the context $r_{thrshld}$ from 2 to 8 and $cutoff^{ctx}_{thrshld}$ from .1 to 1 we observe that the highest recall detected at two pairs of values. First, when $r_{thrshld}$ is very small and the $cutoff^{ctx}_{thrshld}$ is large `(2,.9)`. That is the case when the edits are clustered by considering only a few neighboring tokens but requiring them to be almost the same. Second, when $r_{thrshld}$ is large but $cutoff^{ctx}_{thrshld}$ is relaxed `(8,.75)` which means that we use more tokens around but we are not very strict to have the tokens exactly the same.

The recalls of CDA for varying k under different parameter values are shown in Figure 4.5. Each column represents the comparison of two values of one parameter. Such, the first row (Figure 4.5 (a,b,c,d)) compares the `text` and `link` models under all combinations of the $k_{thrshld}$ and $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ parameters. In the second row (Figure 4.5 (e,f,g,h)) we report the comparison of the recall values with two $k_{thrshld}$: 1 and 2. Finally, the comparison of `(2,.9)` and `(8,.75)` $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ is in Figure 4.5 (i,j,k,l).

In the above experiments, the `text` model outperforms the `link` model which means that `text` is able to detect a bigger fraction of WPC. Clearly, we observe that $k_{thrshld} = 2$ improves the recall for any model and $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ pair. Hence, $k_{thrshld}$ allows CDA to eliminate many noisy edits. Regarding the $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ pair, `(8,.75)` produces a slightly better recall (for the `(link,k=2)` the difference is only 1 page).

In the next experiment we measure the recall on the entire set of pages with WPC (263 pages). We vary the number of retrieved controversies k from 1 to 10 and measure the recall as we did it in the previous experiment. As the parameter values, we use the ones which show the highest recall values (see Figure 4.5), i.e. the `text` model , $k_{thrshld} = 2$ and `(8,.75)` $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ pair. The results are shown in Figure 4.6(a) where we report the recall for both the clean and noisy controversies.

From the full recall experiment on WPC, we conclude that CDA is able to retrieve a large portion of WPC (e.g. 62 out of 263 pages have the corresponding controversy as the top1 result). Surprisingly, the results show a small constant difference between the clean and noisy recalls which means that the algorithm mainly retrieves the controversies which have topic edits.

4.5.5 Accuracy of CDA

In this experiment we aim at measuring the accuracy of CDA. As a dataset we use a set of 25 pages with WPC for which our algorithm produces at least one match in top 10 results under at least some parameter setting (the same set of pages which is used in the Section 4.5.4). As the source of controversies we conducted the user study where every edit is annotated with a topic or the unspecified tag (see Section 4.4.1 for details).

For the parameters we compare the effectiveness of the `text` and `link` models, where $k_{thrshld}$ and $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ parameters are assigned the values which maximize the recall in the experiment in Section 4.5.4, namely the $k_{thrshld} = 2$ and `(8,.75)`, respectively.

In this experiment, for every page we measure 4 metrics which address different aspect of

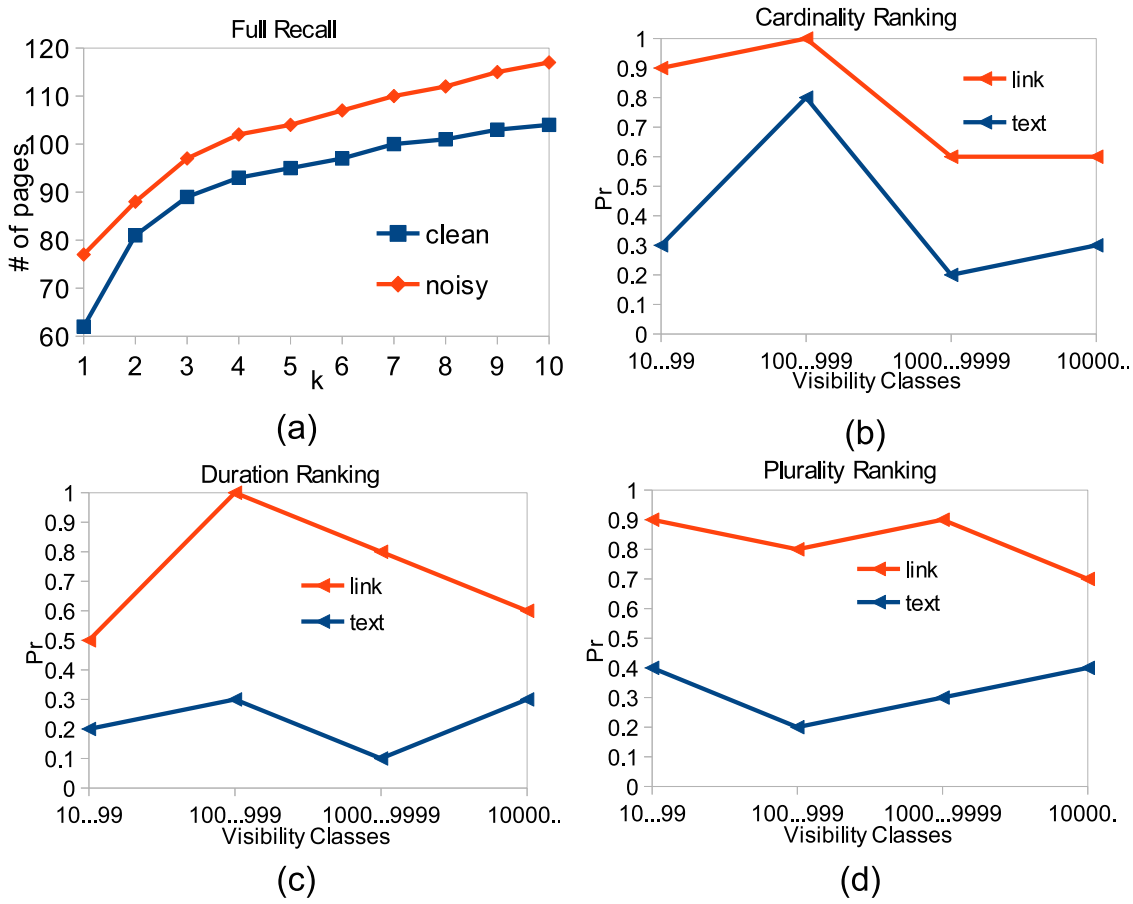


Figure 4.6: (a) the recall on the entire set of WPC of clean and noisy controversies; the cardinality (b), duration (c) and plurality (d) ranking for 4 levels of visibility of `link` and `text`.

the effectiveness of CDA: the noise/signal ratio (noise/signal), the average precision (AvePr), the Rand index (Rand) and the number of distinct controversies (# of distinct controversies). In Table 4.3 we report the average values across 25 pages of the noise/signal ratio, AvePr and Rand index and for # of distinct controversies we show the sum of all distinct controversies of those pages.

As a result of the experiment, we observe that the `text` model is able to retrieve more distinct controversies (56 vs 42). However, the `text` model is almost 2 times noisy (0.22 vs 0.39 noise/signal ration). The AvePr of `link` model is noticeably higher than the one of the `text` model (0.68 vs 0.51). Finally, the `text` model is less blurred (0.66 vs 0.49 Rand index).

4.5.6 Accuracy of CDA at Large Scale

In this experiment we study the effectiveness of CDA at the scale of thousands of Wikipedia pages. For that purpose we use the dataset of Religion pages (204K) which is known to have many controversies.

For the algorithm parameters, we conduct experiments with the `text` and `link` models

metric	link	text
noise/signal	0.22	**0.39**
AvePr	**0.68**	0.51
Rand	0.49	**0.66**
# of distinct controversies	42	**56**

Table 4.3: noise/signal, AvePr, Rand and # of distinct controversies for the `text` and `link` data models on a set 25 pages with WPC

and $k_{thrshld}$ and $r_{thrshld}$,$cutoff^{ctx}_{thrshld}$ parameters fixed to the values which maximize the recall in the experiment in Section 4.5.4, namely $k_{thrshld} = 2$ and `(8,.75)`, respectively.

The obtained edit and controversy statistics are shown in Table 4.4.

parameter	link model	text model
# of pages with at least one edit	138K	193K
# of pages with at least one controversy	8K	31K
# of edits which are in controversies	115K	2.2M
average # of edits per controversy	5.5	6.2

Table 4.4: Religion pages statistics.

The statistics indicate that the `text` model provides 40% more edits which results in a large portion of pages (31K vs 8K) which can be potentially analyzed by CDA. Not surprisingly, the `text` model average size of a controversy is larger than the one of the `link` model (6.2 vs 5.5).

During our experiments, we observed that pages which are more popular (have a large number of revision) are likely to be attacked by vandals. To address this fact, we introduce the notion of visibility which is determined by the number of revisions of a page. In the next experiments, we study the precision of CDA for each class of visibility individually. For that purpose, we bin the pages based on the number of revision on exponential scale, i.e. 10-99, 100-999, 1000-9999, and 10000-....

According to the definition of controversy we discuss in Section 4.2, there are three properties which characterize controversial content, namely the number of edits of a controversy is large, the edits duration and the number of distinct authors (plurality). In the precision experiment, we aim at testing all that three properties and therefore we experiment with the following ranking functions: the cardinality of controversy (`cardinality`), the duration of controversy (`duration`), the plurality of controversy (`plurality`). In addition, in order to see whether our ranking functions are truly correlated with controversies we also use a random ranking (`rand`). A ranking function f is applied as follows: first we rank the controversies within a page using f and pick the top1 controversy, second we rank pages within the same visibility class using f of top1 controversies of pages.

In Figure 4.6(b,c,d) we report the precision which is computed on the set of Religion pages for top 10 retrieved controversies per page. Moreover, we do it for 4 classes of visibility individually, i.e. only pages from the same visibility class are ranked. Three ranking function are used: the `cardinality` (b), `duration` (c) and `plurality` (d).

The results show that the `link` model results in higher precision than using the `text` one, for each of `cardinality`, `duration` and `plurality`.

In the next experiment, using the same setting we study which ranking function leads to higher precision and also compare the proposed ranking functions with a random ordering. The results for the `link` and `text` models are shown in Figure 4.7. Both for the `link` and

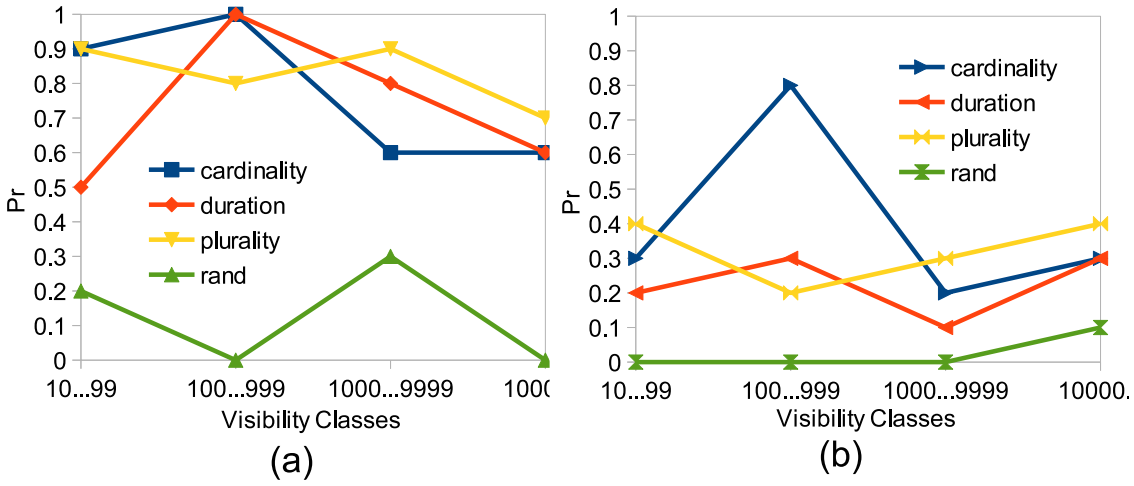


Figure 4.7: `Cardinality`, `duration` and `plurality` for the `link` (a) and `text` (b)

`text` the proposed ranking functions are able to detect controversial content because they lead to significantly higher precision in comparison with the random ordering. Interestingly, we observe a decline in precision for the `link` data model with the increasing level of visibility (Figure 4.7(a)). That confirms the fact that more visible pages are likely to be attacked by vandals. However, that is not generally true for the `text` model.

4.5.7 Effectiveness at Level of Pages

In this experiment we compare our approach with the existing solutions of controversy detection. CDA proposed in this paper focuses on finding individual controversies whereas the previous approaches mainly concentrate on detecting controversial pages. In order to run a comparison experiment we bring the output of our algorithm to the page level. For that purpose we aggregate all controversies from a page and produce a single value which characterizes the level of controversiality of that page. We use two values which are produced by the `text` and `link` models, namely the number of `text` and `link` edits of a page. The reason is that those numbers are highly correlated with the controversies our algorithm is able to detect (these edits are the input for CDA).

The effectiveness at the page level is measured by precision, recall and Normalized Discounted Cumulative Gain (NDCG) (see Jarvelin and Kekalainen [2000] for details) of retrieved controversial pages. As a dataset for that experiment we use the large dataset ($204K$ pages related to the Religion topic). As the ground truth of the level of controversy of a page we use ATC, i.e. the number of controversy template in the revision history of a page. For the details of experiment see Vuong et al. [2008].

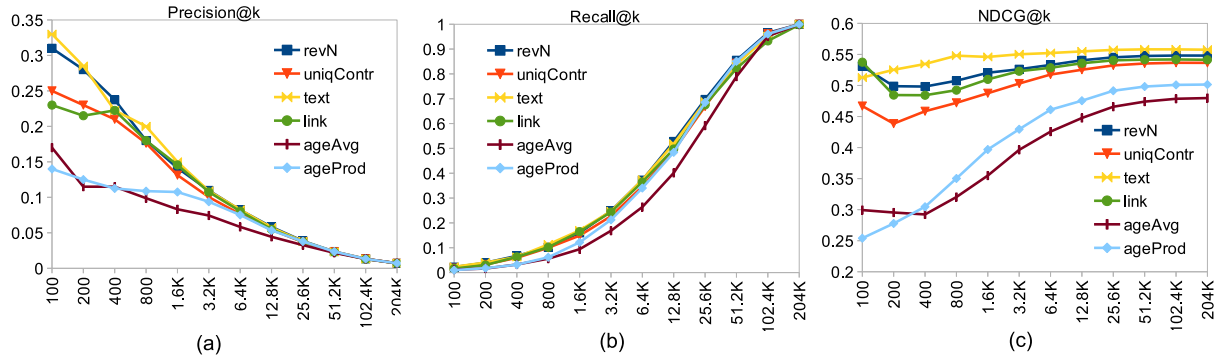


Figure 4.8: Precision (a), Recall (b) and NDCG (c) of the approaches based on # of revision (revN), # of unique author (uniqContr), ageAvg and ageProd from [Vuong et al., 2008] and the statistics based on the `text` and `link` edits on the set of 204K Religion pages.

As competing algorithms we use the number of revisions of a page (*revN*), the number of unique contributors of a page (*uniqContr*), the age-aware CR Average (*ageAvg*) and the age-aware CR Product (*ageProd*) from Vuong et al. [2008] and the number of repeated edits from the text (*text*) and link (*link*) data models. We also compared with the less advanced techniques from Vuong et al. [2008] (the basic model, CR Average, CR Product, the age-aware basic model) but for the sake of presentation we report only the results of age-aware CR Average (*ageAvg*) and age-aware CR Product which showed the highest effectiveness.

The results for the varying number of retrieved pages, $k = 100$ to $k = 204K$ on exponential scale, are shown in Figure 4.8. The measured accuracy metrics indicate that both the `text` and `link` edits are able to detect controversial pages with the similar or higher precision, recall and NDCG. In turn, it means that the input we use for CDA is highly correlated with the known controversial page level metrics (the revision number, the number of unique authors and the approaches from Vuong et al. [2008]).

4.5.8 Controversy Visualization

In this section we discuss our controversy visualization technique. The visualization method of controversy detection was widely used in the past[Viégas et al., 2004, Adler et al., 2008, Brandes and Lerner, 2008]. Its goal is to build a visual representation of the data such that the user can easily detect the patterns of interest (e.g. vandalism, controversies). In our case, for a given controversy, we aim to plot the edits in a such way that the users can understand who is involved into the controversy, what are the opposing opinions, what is the intensity of the controversy, etc.

Recall that a controversy is a cluster of edits (see Section 4.2 for details) where every edit has the author and subject. Usually, there are two groups of authors who disagree on some specific argument and make a lot of back and forth substitutions where they insist on the content which reflects their opinion. To visualize the opposing groups of authors and the respective subjects of a controversy, we proceed as follows. First, we draw ovals and rectangles where the user ids and subjects are depicted in, respectively. By parsing every edit of the controversy, we extract the positive and negative attitudes of the author towards the subject. Such, if the author substituted

link l_1 for link l_2 we interpret that as a positive attitude towards l_2 and a negative attitude towards l_1. During the visualization, a positive or negative attitude is depicted as a green or red line from the user's oval to the subject's rectangle, respectively.

In Figure 4.9 we show the controversy from the Digimon Adventure[11] page. Interestingly, the controversy consists of a number of opposing subjects (`CBS`, `Fox_Kids`, `CJON-TV` and `Japanese_language`). Clearly, many authors argued whether it should be `CBS`, `CJON-TV` or `Fox_Kids` whereas `Japanese_language` has both positive and negative attitudes without some alternative link. That can be explained by the fact that `Japanese_language` was vandalized and every time it was replace with some non-repeated link (and eventually filtered by the repetitiveness filter).

Throughout our experiments (Section 4.4) we found the above visualization very helpful in understanding the details of a controversy.

4.5.9 Distributed Controversy

In this experiment we aim at identifying controversies which span across a number of page. In particular, we found that there can be a group of authors who have a particular bias and this results in controversies in different pages with that group. We refer to that kind of controversies as *distributed controversies*.

In this section we present some preliminary experiments which allows us to find some examples of distributed controversies. As our data we use a list of 6.5K pages related to the religion topic. After the extraction of edits we obtained the list of 2K pages which have at least one edit. We eliminated all the edits whose authors altered only one page because they cannot take part in a distributed controversy. As a result, we obtained 672 pages with 4757 unique authors and 24K tuples which relate authors and the respective pages. By enumerating pairs of author who jointly edited more than 2 pages we can detect distributed controversies.

For example, two authors with the ids 3537067 and 7925541 disagreed on the `Kabbalah` reference, either it should be `Jewish Kabbalah` or simply `Kabbalah`. This distributed controversy is appeared in the following Wikipedia pages: Essenes, Hasidic Judaism, Judaism, Orthodox Judaism, Reconstructionist Judaism.

Using the above data set we also conducted a number of experiments aiming at finding *synchronous edits*, i.e. controversies which are about the same topic and occur in the same period of time. Unfortunately, we've not found any noticeable example of that kind.

Another distributed controversy hypothesis we tested is that the same controversy may appear in different languages of the same page (i.e. *multilanguage controversies*). After studying some pages across different languages we didn't find much evidence that there are such kind of controversies which appear in other languages.

4.5.10 Experiment Conclusions

The main takeaways from the experimental evaluation are as follows.

Comparing the models, we discovered that the `text` model is able to retrieve more WPC than the `link` one. However, the `text` model is more noisy and at a large scale it shows a

[11] http://en.wikipedia.org/wiki/Digimon_Adventure

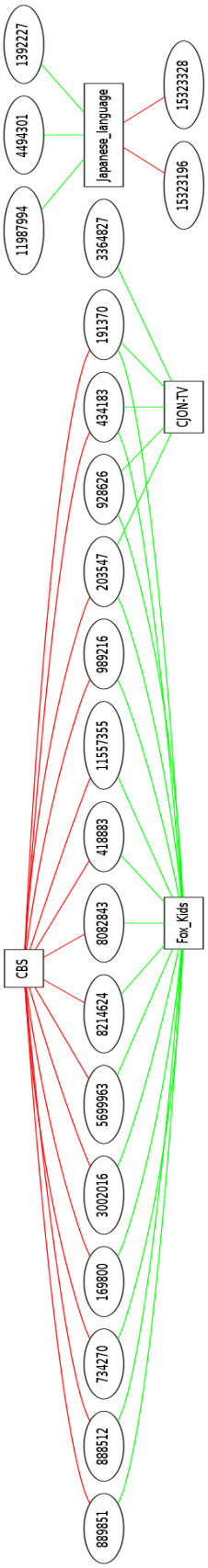


Figure 4.9: Digimon Adventure Controversy Bipolarity Author Graph

lower precision. This behavior can be explained by the fact that `text` has a large number of edits (it is more easy to update a word than a link) but at the same time it is more likely that `text` edits are more noisy or vandalised.

In our experiments, we clearly observe that eliminating edits which are not repeated by at least two different authors (i.e. using the $k_{thrshld} = 2$) significantly improves the quality of results by reducing the amount of noisy edits.

Regarding the controversial properties, we found that the `cardinality`, `duration`, `plurality` are good indicators of content controversiality which experimentally confirms the semantic definition of a controversy as "prolonged public disagreements or heated discussions".

Finally, we show that both `text` and `link` edits itself can serve as statistics to detect controversial pages.

Chapter 5

Coping with Evolving Items and Users in Recommendation Systems

A difficult problem commonly faced by recommender systems where items and users are continuously evolved is the cold-start problem. In that problem, recommendations are required for new items (new-item cold-start) or users for whom little or no information has yet been acquired (new-user cold-start). In that context, most recommendation algorithms fail because they cannot build the user's profile or identify users with similar preferences. In this chapter, we focus on the case when both the users and items lack their historical information, i.e. the new-user/new-item cold start problem. First, we discuss the problem of cold-start in Section 5.1, then we present a motivating example which is used to explain the problem, expectations from a solution and ideas throughout the chapter in Section 5.2. Then, we propose recommendation methods that can be applied to the new-user/new-item cold-start problem in Section5.3. Section 5.4 describes how we can deal with differences in the distribution characteristics of the ratings in the data sets. Finally, detailed information about the experimental evaluation and our findings of this evaluation for all the proposed solutions and their related aspects can be found in Section 5.5.

The work of this chapter was submitted to the KDD 2013 conference ([Bykau et al., 2013b]).

5.1 Cold-start Problem

Recommender systems help users select items of interest by predicting items that the user will most likely like. There are two main categories of recommender systems: the *content-based* systems that exploit selections that a user has made in the past [Mladenic, 1999] and the *collaborative filtering* systems that recommend items selected by users with similar preferences [Su and Khoshgoftaar, 2009]. Some times, there is no previous information about a user or an item in order to allow a recommendation to be made. This is known as the *cold-start problem*. Depending on whether the missing information is about user's previous choices, item's previous selections by users, or both, the cold start problem is found under the names *new-user*, *new-item* or *new-user/new-item*, respectively.

The cold start problem has already been studied in the scientific literature. Solutions vary, but are mainly based on the idea of using other sources of information in order to make a

recommendation. For instance, one way is to assume that the most popular item in general will also be popular to a user for which we have no history. An initial interview [Rashid et al., 2002] or a social network [Guy et al., 2009, Jamali and Ester, 2010] can provide a first idea of the user preferences and guide the selection of items to recommend. For new items, ontologies or other forms of external information can provide a better picture of the nature of the object and guide the selection of the user to which the item is to be recommended [Middleton et al., 2002]. As an alternative, the features of users and items can be used (if this is possible) to capture the similarities between users and items and then recommend each item to the users to which it is most similar [Agarwal and Chen, 2010]. Finally, since no solution works perfect all the time, hybrid approaches may be followed [Burke, 2002].

Most of the existing works on the cold start problem focus on the new-user or the new-item situation. However, there are many practical scenarios in which there is a continuous lack of historical information both for items and users. Examples include cases in which the items have a short life span, or lose very fast their value, while at the same time there is a continuous influx of new users, practically, the user population follows the power law distribution with respect to participation frequency, with most users lying on the long-tail part [Zhou et al., 2011]. One such application are news sites like CNN or NYTimes, where new or unknown users appear constantly in the system with the majority being interested in new content [Liu et al., 2010]. At the same time, an article can become important and get recommended only when it receives a significant amount of visitors, but by the time this happens, its content has become "old" news, and the article is by default less important. Another example are course recommendations. In such system, the users are either freshmen (i.e., without history) or have too short history (max 4/5 years and on completely diverse courses) [Parameswaran et al., 2010] to allow for a good user profiling. At the same time, the characteristics of the courses are changing frequently in terms of content, instructor, exam, teaching methodology, etc, to a point that every new offering of a course can be considered as a new course.

Dealing with applications like those just mentioned where the the new-user/new-item cold start problem is permanently present is a challenging task. Neither collaborative filtering, nor content-based solutions can be applied since there is no historical information about the new users or the new items, respectively. Randomly making some suggestions until a significant amount of information has been collected and informative recommendations can be made, a solution to which many applications often resort [Park and Chu, 2009], will not work either due to the continuous appearance of new items and users. To the best of our knowledge, no other work has studied this problem per se, but have all considered it as a small side-problem of a recommendation system. We believe that the applications in which this type of cold-start problem is present are too many and important to ignore. Our focus in this work is exactly on this.

We study three different methods for dealing with the problem. Each method leverages on the item features to produce recommendations. The first method, that we refer to as the *Similarity-based*, sees each item as a whole and is based on the assumption that the more similar a new item is to an already popular item, the more popular the new item would also be. The second method, referred to as the *Feature-based*, sees the items as a set of independent features. It measures the popularity of each feature individually, and then generates a popularity rating based on the collective popularity of the individual features. For instance, if courses by pro-

cid	year	area	instructor	trimester	exam	student	rating
cs343	2011	DB	Fox	1	written	s5	*avg*
cs343	2010	DB	Fox	1	written	s6	*low*
cs343	2011	DB	Fox	3	written	s7	*avg*
cs241	2010	PL	Smith	2	oral	s5	*avg*
cs241	2010	PL	Smith	2	oral	s9	*low*
cs241	2011	PL	Smith	2	oral	s5	*high*
cs241	2011	PL	Smith	2	oral	s1	*avg*
cs241	2010	PL	Smith	2	oral	s2	*low*
cs241	2011	PL	Smith	2	oral	s4	*high*
cs120	2008	OS	Fox	1	oral	s4	*low*
cs120	2009	OS	Fox	1	oral	s10	*high*
cs120	2010	OS	Fox	1	oral	s11	*high*
cs120	2011	OS	Fox	1	oral	s15	*high*
cs100	2011	HW	Bell	3	oral	s2	*high*
cs100	2011	HW	Bell	3	oral	s3	*high*
cs100	2011	HW	Bell	3	oral	s4	*avg*
cs100	2010	HW	Bell	3	oral	s5	*high*
cs100	2010	HW	Bell	3	oral	s6	*high*
cs400	2011	DB	Newton	3	oral	s10	*high*
cs400	2010	DB	Newton	3	oral	s20	*high*
cs400	2011	DB	Newton	3	oral	s18	*high*
cs123	2011	TH	Megan	2	project	s6	*low*
cs124	2011	GR	Megan	4	project	s12	*low*
cs123	2010	TH	Thomson	4	project	s14	*low*

Figure 5.1: A Course Evaluation System Data

fessor 'Ullman' have been popular in the past, then new courses taught by 'Ullman' are very likely popular courses for new students. The third method, referred to as *Maximum Entropy-based*, takes into consideration popularity of combinations of features. By using the maximum entropy principle [Jaynes, 1957] it predicts the popularity of the combination of features in new items. For instance, the method may identify that courses taught by 'Ullman' are popular only when they are related to data management, thus, a course on VLSI taught by Ullman will not be highly recommended. We evaluate the performance of these methods on the new-user/new-item cold-start problem based on: (i) the correctness of ranking, using the normalized Discounted Cumulative Gain (nDCG), (ii) the prediction accuracy, using the Root Mean Square Error (RMSE), and (iii) the recommendation time. We also compare them to an adapted version of the Pairwise preference regression method [Park and Chu, 2009] that is work closest to ours also dealing with the new-user/new-item cold start problem. Since we do not assume the existence of any auxiliary sources, e.g., social data, we cannot directly compare to other approaches that operate on such assumptions.

5.2 Course Evaluation Scenario

Consider a system used by students in a university for evaluating courses that they have taken. Figure 5.1 illustrates a portion of the information kept in the system. It is Sept 2013 and a new student has just arrived and is wondering whether to take the course `cs421` offered in the 3rd trimester by professor Fox in the area of DB, with an oral final exam. This combination of professor, type of exam, area, etc., has not been seen before, so it is not clear whether this course would be liked by the students or not. Furthermore, since the student is new, there is no knowledge about her preferences. Thus, it is not clear how strongly this course should be recommended to the students. Of course, one may use the history of the courses that a student has already taken to build a profile of preferences for the student. However, since the average study time is four years, and very often course selections may not be driven by preference, building such a profile may not be a functional solution. Thus, we can essentially assume *every user as a new user* and base our decision simply on the evaluations of the courses in the past.

One can look at the Figure 5.1 and observe that the courses `cs123` and `cs124`, which differ substantially from the `cs421`, have received very low evaluations, while the course offering of `cs343` in 2011 that, similarly to the new course `cs421`, is also in DB, taught by Fox and in the 3rd trimester (with a different type of exam, though), has received an avg evaluation. This means that very likely `cs421` will also receive an avg evaluation if taken.

Another way to reason is by looking at the characteristics of the course individually. In particularly, it can be seen that in average, when an oral examination is in place, the course receives an above avg evaluation. The same applies for the DB topic, and to some degree with the 3rd semester. Thus, even if courses with oral examination are typically rated low or avg, the new offering of the `cs421` will be more likely to receive an avg or high evaluation.

Nevertheless, one can see that of the three times that Fox taught a course with an oral exam, the course received a high evaluation. The similar apply for every time an oral examination was done in the 3rd trimester, while whenever Fox taught DB, the satisfaction was not very high. As a result, `cs421` should most likely satisfy the student and recommended to her.

5.3 Predicting Evaluations

Assume the existence of an infinite set $\mathcal{U}$ of users, $\mathcal{F}$ of features and a domain $\mathcal{D}$ of values. An item i is a tuple of features $\langle f_1, \ldots, f_n \rangle$. Let $\mathcal{I}$ represent the set of all possible items. An *evaluation* is a tuple $\langle i, u, v \rangle$ that denotes the assignment of a value $v \in \mathcal{D}$, referred to as rating, to an item i from a user u. Often, we will use the notation $r_{u,i}$ to denote the value v, and F_i to denote the set of features $\{f_1, f_2, \ldots, f_n\}$ of the item i.

Given a finite set of evaluations $R \subset \mathcal{I} \times \mathcal{U} \times \mathcal{D}$, a new item i_o is essentially an item for which there is no evaluation in R. Similarly, a new user u_o is a user who has provided no evaluations in R.

NEW-USER/NEW-ITEM RECOMMENDATION PROBLEM. Given a set of evaluations $R \subset \mathcal{I} \times \mathcal{U} \times \mathcal{D}$, a set $\mathcal{I}_o$ of new items, and a new user u_o, our goal is to: (i) compute a rating r_{i_o} for every item $i_o \in \mathcal{I}_o$ according to the evaluation each one is expected to receive if taken by user u_o, and (ii) produce a ranking of the items in $\mathcal{I}_o$ using their computed ratings.

5.3.1 Linear Regression

As a baseline approach we use the Pairwise Preference Regression algorithm [Park and Chu, 2009] which is the closest state-of-the-art solution which can address the new-user/new-item problem in the environment similar to ours. However, we cannot directly apply the method since it requires the users to have some features (e.g. user demographic, age information). To adapt the algorithm, we assume that all users have only one feature which is the same. After applying this assumption, i.e., the user feature vector has only one feature, we reduce the Pairwise Preference Regression algorithm to the *linear regression* problem, i.e., every rating is a weighted sum of the features of the corresponding item.

5.3.2 Similarity-based Recommendation

Given a new item i_o and a new user u_o, one approach to predicting how much u_o would like i_o is to see how other users have rated similar items in the past. The more similar i_o is to popular past items, the more likely it is that i_o will be a popular item itself, and hence the more likely it is that i_o will be liked by u_o, i.e., the higher its recommendation score should be.

In order to measure the similarity $sim(i, i_o)$ between two items, i and i_o, different similarity functions can be used, but a prevalent one is the Jaccard similarity [Jaccard, 1901] that compares the similarity of two sets. In our case, this will translate into the similarity of the set of features of the two items, in particular,

$$sim(i, i_o) = Jaccard(i, i_o) = \frac{\mid F_i \cap F_{i_o} \mid}{\mid F_i \cup F_{i_o} \mid}$$

Then, the expected rating of the new item i_o can be computed as the average of the existing ratings, weighted according to the similarity of the respective items with i_o. More specifically,

$$r_{i_o} = \frac{\sum_{\langle i,u,r\rangle \in R} sim(i, i_o) r}{\sum_{\langle i,u,r\rangle \in R} sim(i, i_o)}$$

The formula makes sure that the ratings of the items that are more similar to i_o contribute more to the final rating for i_o. Considering all the existing ratings in R may not only be computationally expensive, it may also create a lot of noise since many items with low similarity will be actively affecting the results. To avoid this issue we consider only the ratings in R of the top k nearest neighbors [Cover and Hart, 1967].

Furthermore, the above formula is based on the assumption that all the existing ratings have the same importance. There are many cases that this may not hold. For instance, the rating of an experienced student, as this is dictated by the courses that the student has already taken, may count more than the evaluation provided by a student for a course that is outside of the kind of courses that the student typically follows. To take this factor into consideration, the above formula can be extended with a weight factor $w(i, u)$ based on the item i and the user u that provided the evaluation, as follows,

$$r_{i_o} = \frac{\sum_{\langle i,u,r\rangle \in R} sim(i, i_o)\ w(i, u)\ r}{\sum_{\langle i,u,r\rangle \in R} sim(i, i_o)\ w(i, u)}$$

Note that the use of weights does not depend on its actual semantics. The weight may represent some preference to ratings of specific "privileged" items, ratings from specific users, e.g., friends or people of trust, etc. In our experimental evaluation (Section 4.4), we show how we capture the user expertise in the weight $w(i, u)$.

5.3.3 Feature-based Recommendation

The Similarity-based Recommendation considers each item as a monolithic structure on which each user rating is referring to. However, an item consists of a number of features, and a rating assigned to an item as a whole is actually a reflection of how the individual features of the item are evaluated. Hence, if courses with professor Fox have been rated highly in the past, then a new course by professor Fox will probably be a good recommendation, since professor Fox has been a 'success ingredient' in the past.

With this in mind, one can compute a rating for an individual feature based on the ratings of the items with this particular feature. Then, a rating prediction for a new item can be computed by combining the ratings of its individual features. Of course, this approach assumes that features are independent. This idea is similar to the Naïve Bayes classifier which is also based on the feature independence assumption.

Assuming that a rating of an item translates equally to ratings of its features, the received rating $rt(f)$ of an individual feature f is the average rating of all the items that have that feature, i.e.,

$$rt(f) = \frac{\sum_{\langle i,u,r\rangle \in R^f} r}{|R^f|}$$

with R^f denoting the set of evaluations on items that have the feature f, i.e., $R^f = \{\langle i, u, r\rangle | \langle i, u, r\rangle \in R \wedge f \in F_i\}$.

As in the Similarity-based Recommendation, the above formula can be extended to include some weights on the existing ratings.

$$rt(f) = \frac{\sum_{\langle i,u,r\rangle \in R^f} w(i, u)\ r}{\sum_{\langle i,u,r\rangle \in R^f} w(i, u)}$$

The expected rating for a new item i_o can then be computed as the average of the received ratings of the features that the item has.

$$r_{i_o} = \frac{\sum_{f \in F_{i_o}} rt(f)}{|F_{i_o}|}$$

5.3.4 Max Entropy-based Recommendation

The Similarity-based and the Feature-based Recommendation approaches represent two extremes on how to interpret existing user evaluations. The former assumes that the evaluation is for the maximum number of features that are common with the item for which a prediction needs to be made, while the latter assumes that it is for each feature independently. The former fails to acknowledge the fact that it may be for fewer features, the latter that it may be for more

and recognize the joint relationships between features. An approach that lays somewhere between the two is to associate the rating to every possible combination of features that also exist in the new item for which the prediction needs to be made.

Of course considering all the possible combinations of features is computationally expensive due to their exponential number. Furthermore, it has the risk of not well weighting the various alternatives. Ideally, we want to recognize *preference patterns*, i.e., combinations of features, that capture user preferences as expressed in the existing evaluations made by users on different items. For this purpose, a more principled approach is to use the notion of max entropy [Jaynes, 1957] which tries to maximize the amount of information that the features and any combination of them are providing.

Intuitively, the idea is to treat the (non-)existence of a feature and the rating that an item receives as random variables, and find a joint probability distribution that satisfies the existing evaluations and introduces the minimum possible additional information, i.e., that is the closest to the distribution with the maximum entropy.

More specifically, given n features, f_1, ..., f_n, we define n+1 random variables x_i, with i=1..(n+1). The variable x_i (for i=1..n) corresponds to the feature f_i and gets values 1 or 0 meaning that the feature f_i is present or not. The n-(th+1) variable, denoted as r, can take any of the values that can be provided by a user as an evaluation to an item. Having these n+1 random variables, their joint distribution, denoted as $p(r, \vec{x})$ can be defined, where $\vec{x}$ is a vector of the random variables $x_1 \dots x_n$. Each of the evaluations that previous items have received can be represented as a vector $\vec{c}$ of n binary values, indicating the existence or not of the respective features and a value v representing the evaluation received. The vector $\vec{c}$ and the value v can be interpreted as an observation of the $n + 1$ random variables.

PREFERENCE PATTERN. We define a preference pattern as a set of features and a rating. Given the set of n features of the data, and the domain of the ratings, we can create a list $\mathcal{P}$ of all the patterns.

Viewing the existing evaluations as observations of the $n+1$ variables, the goal is to find the joint probability distribution $p(r, \vec{x})$ that satisfies the constraints set by the observations, i.e., is consistent with them, and at the same time is as close as possible to the probability distribution $\pi(r, \vec{x})$ that has the maximum entropy. To do the former, for each pattern q_j in $\mathcal{P}$, with j=1..$|\mathcal{P}|$, a rule is created of the form:

$$\sum_{r,\vec{x}} p(r, \vec{x}) k(r, \vec{x} \mid q_j) = m_j/m \tag{5.1}$$

where $k(r, \vec{x} \mid q_j)$ is a function that returns 1 if r and $\vec{x}$ satisfies the pattern q_j and 0 if not. Satisfying the pattern q_j means that the features of the vector $\vec{x}$ and the variable r contain those of the pattern. The number m_j is the number of evaluations in the dataset that satisfy that pattern. The number m is the total number of evaluations in the dataset. The rules that are generated in practice do not really have to be for all the possible patterns but only for those for which m_j is above a specific cut-off value.

Example 5.3.1. *Assume a pattern q' that has the features (PL, $Smith$) and the evaluation $high$. In the dataset of Figure 5.1, out of the total 24 evaluations, there are two that contain these two features and have the rating $high$. As such, we require from the joint distribution function we*

Algorithm 4 Generalized Iterative Scaling

1: Choose an initial approximation of $p(r, \vec{x})$ by choosing an initial set of $\lambda_0 \dots \lambda_{|\mathcal{P}|}$
2: **while** $\exists$ costraint not satisfied **do**
3: **for** $j = 0$ to m **do**
4: $\lambda_j \leftarrow \lambda_j \Big(\frac{n_j/n}{S_j}\Big)^{\frac{1}{d_{max}}}$;
5: **end for**
6: **end while**
7: return $\lambda_0 \dots \lambda_{|\mathcal{P}|}$

are looking to find, to return the value $\frac{2}{24}$ *when its arguments are the vector* $\vec{x}$=$(PL,\ Smith)$ *and the value* r=$high$*, by creating the rule*

$$\sum_{r,\vec{x}} p(r, \vec{x}) k(r, \vec{x} \mid q') = 2/24$$

Finding the joint distribution function that is as close as possible to the one with the maximum entropy, requires first the introduction of a distance function between the distributions. For that we use the Kullback-Leibler (KL for short) divergence, which measures the expected number of extra bits required to code samples from $p(r, \vec{x})$ when using a code based on $\pi(r, \vec{x})$, rather than using a code based on $p(r, \vec{x})$. For two discrete probability distributions the KL divergence is computed with the following formula:

$$D_{KL}(p \mid\mid \pi) = \sum_{r,\vec{x}} p(r, \vec{x}) log \frac{p(r, \vec{x})}{\pi(r, \vec{x})} \tag{5.2}$$

The problem of finding $p(r, \vec{x})$ is formulated as finding the function that minimizes $D_{KL}(p \mid\mid \pi)$ and satisfies a set of constraints of the form of Equation (5.1). To find the $p(r, \vec{x})$ we employ the method of Lagrange multipliers. More specifically, the minimum of $D_{KL}(p \mid\mid \pi)$ with the h constraints of the type shown in Formula (5.1) is found as the stationary points of the Lagrange function for some $\lambda_1, \dots, \lambda_{|\mathcal{P}|}$, i.e.

$$\nabla \Big[\sum_{r,\vec{x}} p(r, \vec{x}) log \frac{p(r, \vec{x})}{\pi(r, \vec{x})} \Big] + \sum_{j=1} \lambda_j \nabla \Big[\sum_{r,\vec{x}} p(r, \vec{x}) k(r, \vec{x} \mid q_j) - m_j/m \Big] = 0 \tag{5.3}$$

The solution for this equation is of the form [Darroch and Ratcliff, 1972]:

$$p(r, \vec{x}) = \pi(r, \vec{x}) \prod_{j=0}^{|\mathcal{P}|} \lambda_j^{k(r,\vec{x}|q_j)} \tag{5.4}$$

Thus, in order to find the probability distribution $p(r, \vec{x})$ we need to find such $\lambda_0, \dots, \lambda_m$ that every constraint of the form

$$\sum_{r,\vec{x}} \Big(\pi(r, \vec{x}) \prod_{h=0}^{|\mathcal{P}|} \lambda_j^{k(r,\vec{x}|q_h)} \Big)\ k(r, \vec{x} \mid q_j) = m_j/m \tag{5.5}$$

is satisfied. To achieve this, we utilize the method of Generalized Iterative Scaling algorithm (GIS) [Darroch and Ratcliff, 1972]. GIS can be applied only if the following condition is true:

$$\forall r, \vec{x} \sum_{j} k(r, \vec{x} \mid q_j) = const \tag{5.6}$$

and in order to guarantee this we add one additional constraint:

$$k(r, \vec{x} \mid 0) = d_{max} - \sum_{j=1}^{|\mathcal{P}|} k(r, \vec{x} \mid q_j) \tag{5.7}$$

where $d_{max} = max\{k(r, \vec{x} \mid q_j) \mid j = 1..|\mathcal{P}|\}$

The idea of the GIS algorithm is to start with an approximation of the $p(r, \vec{x})$ function that uses some initial values of the parameters λ_j, and then iteratively update these parameters generating a better approximation and heading towards the satisfaction of all the constraints set by the existing evaluations in the dataset. At the moment all the constraints are found to be satisfied, the process stops and the values of the λ_js are used to generate the solution. In every iteration, the values of the λ_js (denoted as $\lambda_{j,t}$) are updated to the new values $\lambda_{j,t+1}$ according to the following formula:

$$\lambda_{j,t+1} = \lambda_{j,t} \Big(\frac{m_j/m}{S_{j,t}} \Big)^{\frac{1}{d_{max}}} \tag{5.8}$$

where $p_t(r, \vec{x})$ is an estimation of $p(r, \vec{x})$ at iteration t and

$$S_{j,t} = \sum_{r,\vec{x}} p_t(r, \vec{x}) k(r, \vec{x} \mid q_j)$$

To compute a prediction for an item with a set of features $\vec{c}$, we need to make a prediction for the vector of features $\vec{c}$. For this we use the conditional probability of ratings with known set of features, i.e. $p(r \mid \vec{x} = \vec{c})$, and consider as prediction the mean of that distribution:

$$r_{i_o} = r_{\vec{c}} = mean_r(p(r \mid \vec{x} = \vec{c}))$$

5.4 Average Rating Confidence

The average of a set of values is typically used as a representative of the set. An important dimension that needs to be taken into consideration is the level of confidence that can be placed upon a computed average as the true representative of the set. This confidence is affected by two factors, namely the level of *support* and the *deviation*.

The support is intuitively describing the size of the set of evidences, i.e., evaluations that were used in the computation of the average value. Naturally, an average computed on a large data set is more likely to actually represent the true value of the item since too many different users have evaluated it, and the chances of been mistaken are low. Instead, an average coming from a very small user group may not be accurate since it may simply reflect the opinion of those few people. To include the notion of support in our computations we adopt the idea of *shrinkage*, a solution that is prevalent in the area of statistics, and has already been used in popularity computation techniques [Park and Chu, 2009] and in parameter estimation [Bell

et al., 2007] for recommender systems. The intuition behind shrinkage is that the actual average rating of any item is by default the average of all the ratings of all the items that exist in the database. However, the more evidence, i.e., evaluations, exists for an item, the more likely it is that its actual value is shifted towards the average that these evaluations are producing. As such, the value of an item i_o is considered to be:

$$r^s_{i_o} = \frac{\bar{r}_{i_o} * s_{i_o} + \bar{r} * \alpha}{s_{i_o} + \alpha} \tag{5.9}$$

where α is a parameter specifying how fast the value shifts from the global average $\bar{r}$ towards the average $\bar{r}_{i_o}$ of the evaluations for the item i_o, and s_{i_o} is the number of evaluations for that item.

In the case of the max entropy, where we have a probability distribution, s_{i_o} is actually the probability of the vector $\vec{c}$ of the features of i_o, i.e., $p(\vec{x} = \vec{c})$. $\bar{r}$ is the average of all the mean values of all the items in the database, and $\bar{r}_{i_o}$ is the $mean_r(p(r \mid \vec{x} = \vec{c}))$, where $\vec{c}$ is the vector of the features of the item i_o.

The second factor, i.e., the deviation, intuitively measures the level of disagreement that is observed among the users that have provided evaluations for a specific item. Given the average of a given set of evaluations, the further the provided ratings are from the average, the less the confidence on the computed average should be, since it indicates a higher disagreement among those that provided the ratings. The idea of shrinkage can be used also here for the same problem. Instead of computing the average, the value $r^d_{i_o}$ of a specific item i_o is computed as:

$$r^d_{i_o} = \frac{\bar{r}_{i_o} * \sigma_{i_o} + \bar{r} * \beta}{\sigma_{i_o} + \beta} \tag{5.10}$$

where β is a parameter of shrinkage for the deviation and σ_{i_o} is the standard deviation of the evaluations for item i_o. In the case of max entropy where we have a probability distribution the σ_{i_o} will be the standard deviation of the distribution.

To take into account both the support and the deviation we use the composition of the two Equations (5.9) and (5.10):

$$r_{i_o} = r^s_{i_o} \circ r^d_{i_o} \tag{5.11}$$

This equation generates a rating prediction based on the max entropy method and taking into account the confidence in each rating.

5.5 Experimental Evaluation

To evaluate our solutions we have conducted a number of experiments on two real datasets on a 4GHz machine with 4G of memory. The techniques were all implemented in Java. The Maximum-entropy approach was done by adapting accordingly to our requirements parts from the Apache OpenNLP project[1].

5.5.1 Metrics

We quantitatively measured the effectiveness of our proposals from two aspects: *predictability* and *coverage*. The former aims at measuring the accuracy of our recommendations with respect

[1] http://opennlp.apache.org/

to the available data. The latter measures the number of items and users for which we can successfully produce a recommendation.

For the *predictability* we use two metrics. We use the *Root Mean Square Error (RMSE)* for measuring the effectiveness of the individual rating predictions, i.e., how well we can predict a rating for a new item. Given a list of predicted ratings $(\tilde{r}_1, \ldots, \tilde{r}_n)$ as generated by our methods and a list of actual ratings $(r_1, \ldots, r_n)$ provided by a user, the RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\tilde{r}_i - r_i)^2}{n}} \tag{5.12}$$

The higher the RMSE is, the further the predicted ratings are from the actual ratings.

The second metric measures the effectiveness of the order of the recommended items based on their predicted rating. This metric uses the Normalized Discounted Cumulative Gain (NDCG) [Jarvelin and Kekalainen, 2000], which intuitively measures how a list differs from another based on the assumptions that (i) highly relevant items are more useful when appearing earlier and that (ii) highly relevant items are more useful than marginally relevant items. The Discounted Cumulative Gain (DCG) at a particular position m is

$$DCG_m = \sum_{i=1}^{m} \frac{2^{rel_i} - 1}{log_2(1+i)} \tag{5.13}$$

where rel_i is the score (i.e., rating) of item at position i. The DCG_m measured at the perfect ranking, i.e., when the items are ordered according to the ground truth, is referred to as $IDCG_m$. Using the DCG_m and $IDCG_m$ of a position m, the $NDCG_m$ of that position can be defined as

$$NDCG_m = \frac{DCG_m}{IDCG_m} \tag{5.14}$$

which provides a normalized value of the effectiveness of the order.

For the *coverage*, we simply need to measure the ratio between the number of items for which the algorithm is able to produce recommendations and the total number of items, i.e.,

$$coverage = \frac{\#\ of\ items\ with\ a\ recommendation}{\#\ of\ all\ items} \tag{5.15}$$

5.5.2 Datasets

We have run our experiments on two real applications datasets.

Course Rating. We used a dataset from the Stanford course evaluation system [Bercovitz et al., 2009] containing ratings across different departments between the years 1997 and 2008. Since students of one department generally do not take courses from other departments we divided the datasets to fragments of different departments. We ran all our experiments on two or three different departments and we experienced similar behavior, so we are reporting here the results only on the Computer Science department. For this specific department we had 9799 course ratings for 675 individual courses with 193 instructors in 17 terms within the period mentioned above.

Each course offering is described by a number of basic features like the id of the course, the instructor, the term it was taught, the title and a description of the course contents. Assuming

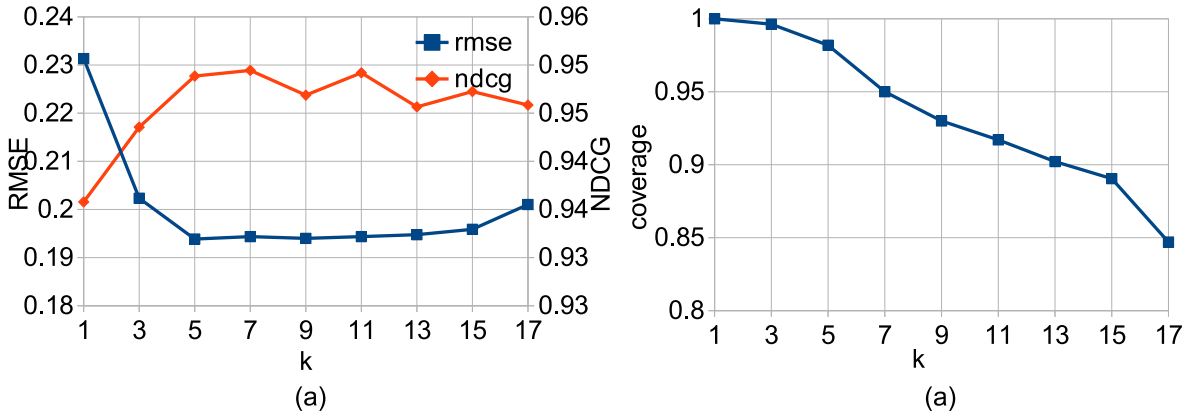


Figure 5.2: Parameter k effect in the Similarity-based approach.

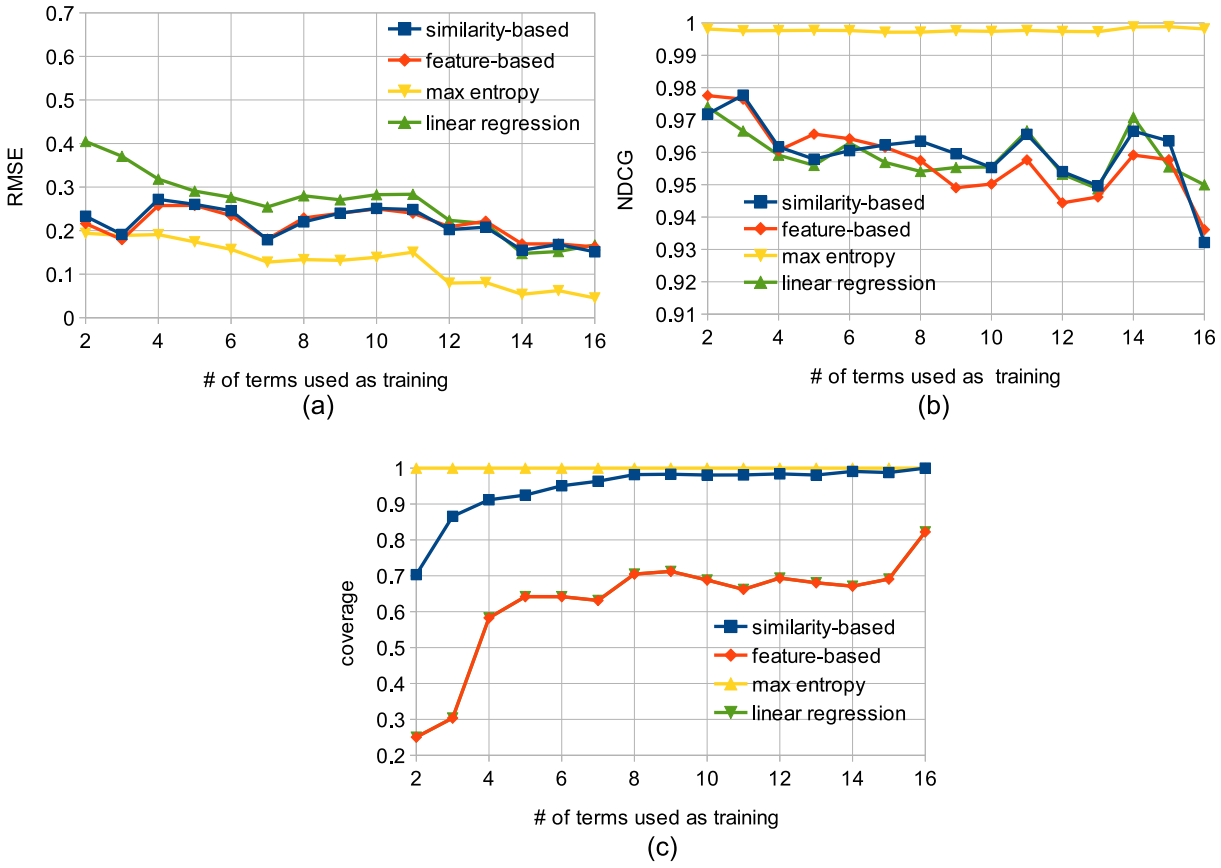


Figure 5.3: Accuracy and coverage for different numbers of terms (n_{train}) used as training on the Course Rating dataset

that the title is reflecting accurately what the course is about, as it is typically happening in reality, we extracted keywords from the titles and we converted these keywords into features. To exploit the information in the description we have used the Latent Dirichlet Allocation [Blei et al., 2003] to identify topics that the description contains, and then we converted them into

features. We set the algorithm to produce 20 features for each course given its description. We have experimented with different numbers of topics but too few topics did not provide enough information while too many topics were too detailed with not so satisfactory results. 20 is a good choice for this dataset.

To measure the accuracy and the coverage we consider a part of the data as the given and another part for testing. A pair of given-test data is created as follows. We consider a specific point of time t which coincides with the end of some specific term. We consider any evaluation that has been made before that time t as given (i.e., part of the training data), and the evaluation of the 3 quarters after time t as the ground truth (i.e., test data). This partition scheme reflects very accurately the reality since the only information that the users taking and evaluating these courses in the 3 terms after time t are the evaluations that have been received before the time t, and the courses in these 3 terms have never been evaluated before, so we are actually in a real cold-start situation. The reason we can safely consider 3 terms after time t (and not only one) is that courses are very rarely offered more than once in the same academic year, thus, all the offerings in these 3 terms are unique. At the same time, using 3 terms gives the opportunity to have more test data. Since we wanted to investigate the version of the cold-start problem in which we had new users and new items, we are also removing from the testing set all the evaluations made by users that have also evaluated some other course in the given data set.

Note that every course offering is by default a new item. Even if the same course has been offered in previous years, since we operate at the offering level, two offerings, even if they agree in all features but the year, are considered different items.

Furthermore, for a given point of time t we have decided not to consider all the evaluations that have been made in terms before time t because this will include very old evaluations that may be of small use for predicting evaluations for new items. As such, we consider only the n_{train} terms that immediately precede time t, and we have run experiments for different values of this number.

We also repeat the experiments for every possible value of t generating different train/test datasets and we report the average and deviation of the measured parameters. By doing so, we avoid peculiarities that may have been observed in specific years due to external factors and make our experimental findings more robust.

MovieLens. The second dataset we have used is from the MovieLens[2] web site where the users evaluate movies. The particular dataset we used consists of 100K ratings made by 1000 users for around 1700 movies with 42000 unique features and on average 39 features per movie. As features we used characteristics like title, year, genre, etc. For each movie we extracted from IMDB the director and the actors and used them as features, which explains the many features that we have for the movies.

For the MovieLens data we cannot be sure that at the moment of an evaluation generated by a user, no other evaluation has been seen before, i.e., making it a little uncertain whether the user is rating a new item. To alleviate this problem, we have employed a cross-validation approach that has typically been used in cold-start problem experiments [Park and Chu, 2009]. In particular, we partition the data into 5 equal datasets and then use one dataset for testing and the rest for training, giving a ratio of testing to training data 1:4, and we repeat the experiment for each of the partitions (5 fold cross-validation). What we report at the end is the mean value

[2]http://www.grouplens.org/node/73/

and deviation of measured parameters.

5.5.3 Experiment Planning and Tuning

We have implemented our proposed approaches, i.e., the Similarity-based approach, referred to as `similarity-based` which uses the k most similar items in the training set to make a prediction for the expected rating of a new item, the Feature-based approach, referred to as `feature-based`, that assumes the independence among the features, and the maximum entropy approach, referred to as `max entropy`, which exploits the maximum entropy principle for generating predictions, plus the linear regression (`linear regression`) which is an adapted version of the Pairwise Preference Regression [Park and Chu, 2009].

Since the `similarity-based` and the `feature-based` approaches may not be able to produce recommendations for every item in the testing data (their coverage is less than 1 as we will see), we are reporting the RMSE and NDCG values only for the items that both approaches can predict a (non-zero) rating.

Before using the `similarity-based` algorithm we have to decide the right value for the parameter k. We ran the algorithm on the Course Ratings dataset and we varied the value of k from 1 to 17, measuring at the same time the RMSE, the NDCG and the coverage. The results of this experiment are shown in Figure 5.2. From Figure 5.2(a) we observe that there is a specific point after which any increase in the value k has little effect on the observed RMSE and NDCG values. At the same time, it can be observed in Figure 5.2(b) that the coverage linearly deteriorates. We get some similar observations when repeating the same experiment on our movie dataset. Thus, we have fixed k to the value 5 in our experimental evaluation.

For `max entropy`, we have to fix the values for the shrinkage parameters, i.e., α and β. (ref. Section 5.4). Similar experiments as above showed that the values $\alpha = 2$ and $\beta = 2$ lead to the highest accuracy of `max entropy` on the Course rating dataset.

5.5.4 Effectiveness

Effect of the number of existing evaluations. To investigate the importance of the number of available evaluations in the recommendation process, we run our algorithms multiple times, while we gradually increase the number of terms used as training from 2 to 16. Figure 5.3 shows the results for each experiment, where chart (a) shows the individual rating predictability (RMSE), (b) the effectiveness of course ordering (NDCG) and (c) the coverage.

It can be noticed that RMSE is improving with the growing number of terms used for training for `similarity-based`, `feature-based` and `max entropy`, because the more information they have on features the better their predictions are. All three algorithms outperform the adapted version of the method in [Park and Chu, 2009] (`linear regression`) for the individual rating prediction accuracy. (Recall that the higher the RMSE is, the further the produced results are from the ground truth, meaning that the lower the RMSE is the better.)

On the other hand, the NDCG (Figure 5.3(b)) remains almost stable which means that increasing the number of ratings in the training dataset doesn't lead to much improvement in the predictability of the right order in which the items are recommended. The interesting observation is that `max entropy` always gives a high NDCG value, which means that even if the

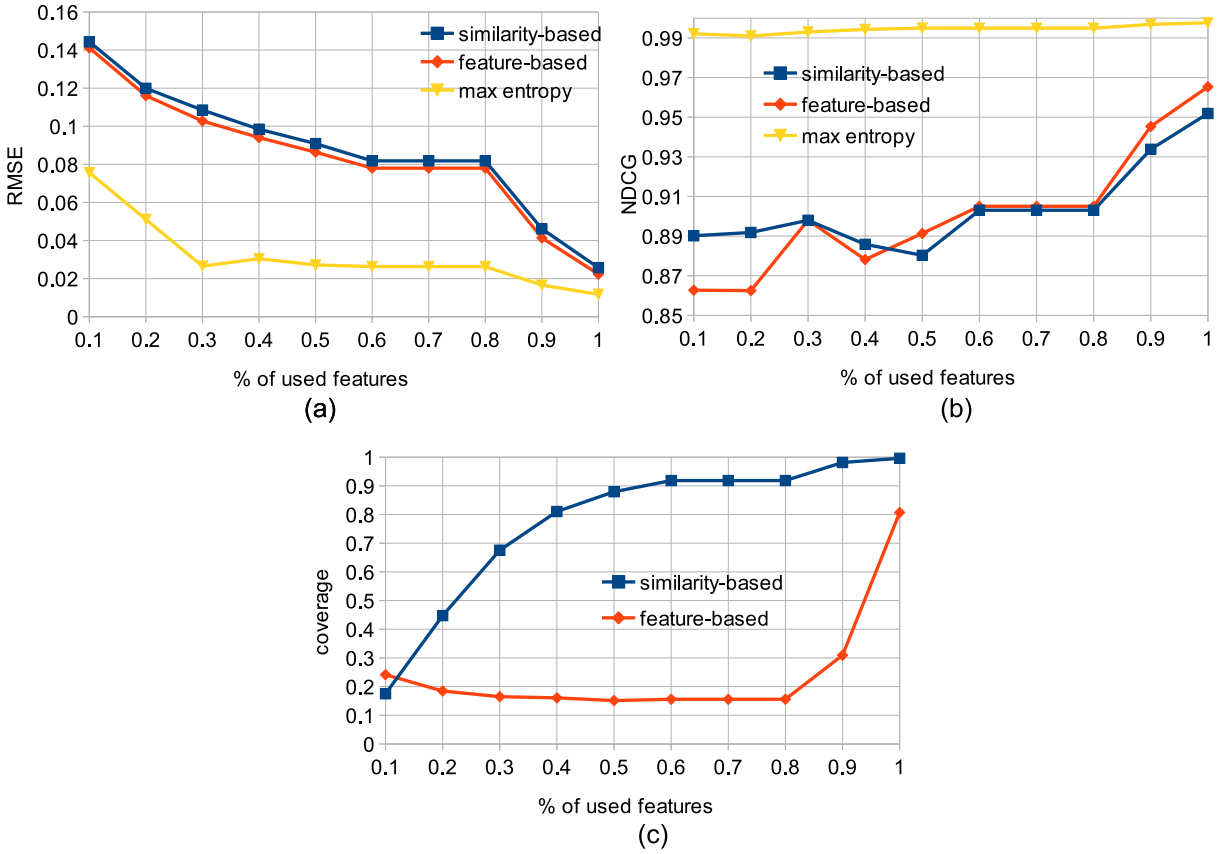


Figure 5.4: Effect of the number of features in the various approaches.

method fails to predict the exact evaluation that a new item will receive from the users, the relative values of the evaluations it predicts are similar to the relative values the items will actually receive from the users, leading to the same ordered list of recommendations. A slight decrease in NDCG that can be observed in almost all the approaches may be due to the number of times the experiments are repeated given the number of terms considered in the training data. Given that the course dataset spans a fixed time period, when we train on periods spanning more terms, the fewer such periods we can find in our data, and hence, the fewer times we can repeat the same experiment. Therefore, the observed minor decrease is most probably due to some fluctuations that naturally exist in the data, and the small number of repetitions of the experiment is not enough to offset. Another explanation may be that if the number of recent terms we consider as training data increases too much, it will contain too old courses for which their evaluation is not so important in more modern times.

Regarding the coverage (Figure 5.3(c)), both `similarity-based` and `feature-based` increase the coverage with the number of training terms, although `feature-based` has always a lower value. The latter is due to how `feature-based` works: if an item has even one feature that does not appear in any evaluation used for training, the approach will make no prediction. The `similarity-based` approach will not be able to make a prediction for an item if all the item features are missing from the items in the training set, which is less likely

to happen. On the other hand, since the decision of the max entropy approach is based on a probability distribution, even in the case in which the new item has features that do not appear in the existing evaluations, a prediction can still be made. Regarding linear regression, since it makes a recommendation on the basis of the features in the item of interest, the method fails if some of the features were not present in the training data. This coincides with the recommendation failure condition of feature-based and therefore they both have exactly the same coverage.

In conclusion, max entropy is more accurate than similarity-based and feature-based both at the prediction of individual ratings and the order under any number of training terms.

Effect of the number of features. To study the effect of the number of features on the accuracy and coverage we use the MovieLens dataset (that has a larger number of features than the Course Rating dataset). We derive sets of new items from the original dataset by omitting from the items a percentage of their features. We do this for values from 10% to 100% (100% means that all features are considered) with the step of 10%. For each dataset created, we conducted a 5-fold cross-validation and recorded the average values for RMSE, NDCG and coverage shown in Figure 5.4. Note that we do not show the results of the linear regression algorithm in the following experiments since the solution proposed in [Park and Chu, 2009] fails for a large matrix of features (since the operation of finding inverse matrix is computationally expensive) and thus we conducted the comparison only for the course dataset.

Figure 5.4(a) shows that the predictability of individual ratings (RMSE) is improving with the number of features, and all algorithms illustrate a similar behavior, with max entropy having the best.

In contrast to the experiment on the training dataset size, here NDCG is improving with a growing number of features since more information is becoming available for making the right predictions (Figure 5.4(b)). max entropy is again showing a very robust behavior. Even with few features, its success rate is very high. It is able to effectively compose information provided by ratings on items with overlapping features and make the right predictions.

The coverage dynamics of similarity- and feature-based are different. The former is increasing fast but the rate of increase is getting smaller as the number of features gets larger, since the algorithm starts having enough features to make the predictions, and some more new features do not actually add new items. Asymptotically, the coverage grows to the full coverage. The feature-based coverage, on the other hand, remains for many different numbers of features around 0.2 and only when the algorithm starts considering the 80% of all the available features the coverage gets sharply to 0.8. This can be explained by the fact that the feature-based fails to produce a recommendation for an item if any of its features is missing from the training data. (Note that the number of distinct features in the MovieLens dataset is really high.) For the same reasons that we explained in the case of Figure 5.3, the coverage for the max entropy is always 1 and not depicted on the Figure 5.4(c).

As a final note on coverage, max entropy is naturally expected to have higher coverage compared to the other approaches. The fact that it achieves coverage equal to 1 in both datasets is due to the nature of these domains. Course offerings (or movies for that matter) will most likely share some feature with previous ones. For example, a new movie will most probably fit

	RMSE		NDCG		coverage	
	Mean	STD	Mean	STD	Mean	STD
similarity-based	0.0257	0.0011	0.9518	0.0039	0.9960	0.0011
feature-based	0.0223	0.0009	0.9653	0.0051	0.8070	0.0060
max entropy	**0.0117**	0.0009	**0.9976**	0.0051	**1.000**	0.0000

Table 5.1: 5-fold cross-validation results

into an existing category.

Effect of the weights. To investigate how the rating weights are affecting the recommendation process, we compute a weight $w(i, u)$ for every rating r on an item i by a user u. The weight is computed in such a way that it reflects the expertise of the user u. In particular, it is the fraction of the number of times that the features of the item i appear in items rated by the user u up to the point that the rating r is made, over the total number of features in these items. The weight, $w(i, u)$, is embedded into the computation process as described in Sections 5.3.2 and 5.3.3. When applied on the Course ranking dataset, the meaning of the expertise is that of putting more trust on ratings provided by students that in the past have taken courses with similar characteristics.

The experiments conducted with the expertise have not shown a significant improvement, which is a strong indication that in the domain of university course rankings, the cold start problem is clearly a (new-user/new-item) problem. This is to be expected since the four years that the students remain on average in the university are not enough to produce a significant dataset that can accurately characterize the expertise of a student.

Differences among the approaches. To study the effectiveness of the different approaches on the MovieLens dataset, we use a 5-fold cross-validation experiment. The results are summarized in Table 5.1. The `similarity-based`, `feature-based`, and `max entropy` approaches are very accurate in predicting individual ratings (RMSE) with `max entropy` being the best. The deviation (STD) of the obtained errors is small and in the range of $[0.0009, 0.0021]$. Concerning the effectiveness of movie ordering (NDCG), our methods show good absolute results (more than 0.95) as well. Similarly to RMSE, the results of NDCG have a small standard deviation. Regarding the coverage, we observe that `feature-based` has a comparatively smaller coverage (0.8 for `feature-based` versus 0.99 for `similarity-based`).

As a conclusion, all our proposed methods accurately predict both the individual ratings and the order of the movies. In particular, `max entropy` is noticeably better both in RMSE and NDCG. Note that `max entropy` has the coverage of 1 because it can deal with new movies in any case very well.

5.5.5 Efficiency

Since recommender systems may be used in highly interactive scenarios, an important factor of the proposed approaches that also needs to be studied is the efficiency.

The `max entropy` approach depends heavily on the number of unique features in the dataset. For this reason, we use the approximation explained in [Lau et al., 1993] which allows to run `max entropy` in $O(NPA)$ where N is the training dataset's size, P is the number of

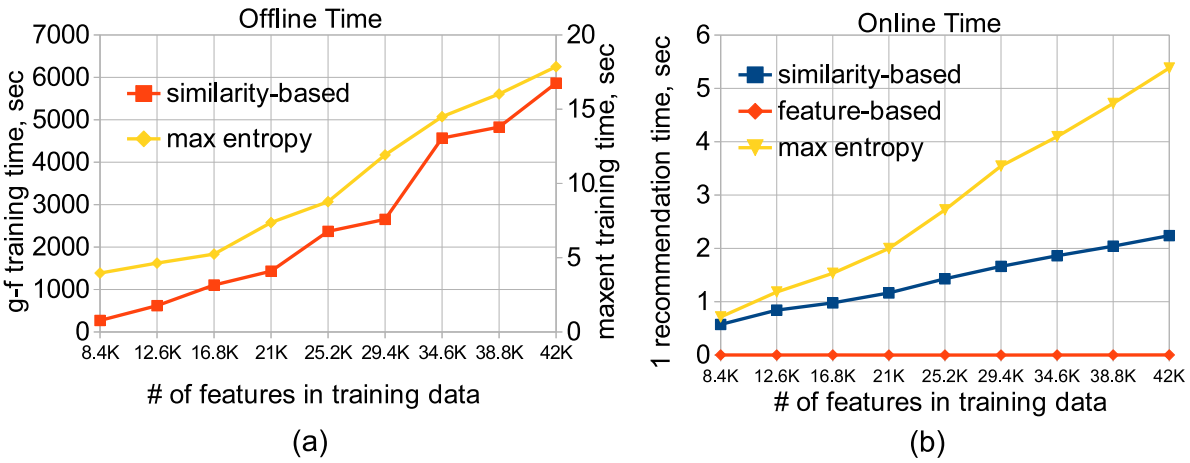


Figure 5.5: The efficiency of the three methods.

predictions, and A is the number of features of the item whose rating is predicted.

Each method has two parts: an *offline* phase needed to train the model and an *online* phase that produces the recommendation for an item. `similarity-based` has only the online mode where for a given item it needs to find the similar items. For `feature-based`, we pre-compute the feature ratings of training data in advance, and then a recommendation is produced with the help of the cache. `max entropy` requires some time to train the model, and then it can produce individual predictions. Figure 5.5(a) shows the offline time for `feature-based` and `max entropy` varying the number of features in the training data of the MovieLens dataset (from 8.4K to 40K features). Both the `feature-based` and `max entropy` times grow linearly with the training data size. `max entropy` builds a model two orders of magnitude faster than the `feature-based`.

We also studied the algorithms' online time. Using the MovieLens dataset and varying the total number of features from 8.4K to 40K, we ran a 5-fold cross-validation for the three approaches. Figure 5.5(b) reports the average time per recommendation. The time required by `similarity-based` and `max entropy` grows linearly. The time of `feature-based` is negligible since it works with the cache obtained at the training phase.

Chapter 6

Conclusion

Modeling, querying and exploiting the volatile nature of data in evolving repositories and systems with user generated contents are challenging tasks nowadays. The problems of creating the right models that will capture not only the changes that have taken place on the values of the data but also the semantic evolution of the concepts, determining the quality of evolving data in user-generated repositories and matching a specific piece of information which is recently updated (or added) in a repository to a user or query at hand are the problems which the dissertation addresses. In this chapter we recap the key contributions of the work and discuss the possible future directions of research.

6.1 Key Contributions

The contribution of the dissertation consists of three parts: the modeling and querying of data evolution [Rizzolo et al., 2009][Katifori et al., 2011][Bykau et al., 2011][Bykau et al., 2012] [Bykau et al., 2013c], the data quality in evolving user-generated repositories[Bykau et al., 2013a] and the personalization with evolving items and users[Bykau et al., 2013b].

6.1.1 Modeling and Querying of Data Evolution

We presented a framework for modeling evolution as an extension of temporal RDF with mereology and causal properties. These properties are expressed with a set of evolution terms and its query language is an extension of nSPARQL that allows navigation over the history of the concepts. Furthermore, we have designed and implemented an evaluation technique that allows query answering over databases where the evolution model that the user has in mind is of different granularity than the one used in the database. The solution required the computation of a Steiner forest. For the later we have presented a novel algorithm for computing its optimal solution. Finally, we have studied two real use cases where we show the applicability of the proposed framework. In addition, we have performed a number of an extensive experimental evaluation to determine the efficiency of the evaluation technique in the case for evolution-unaware queries.

6.1.2 Data Quality in Evolving User-Generated Contents

We studied the problem of controversy detection in user generated evolving repositories, and more specifically in Wikipedia. In contrast to previous works in controversy detection in Wikipedia which addressed the problem at the page level, we have developed an algorithm that considers the individual edits and can accurately identify not only the exact controversial content within a page, but also what the controversy is about. Furthermore, apart from analyzing the text as is traditionally done in similar approaches, we have additionally developed a novel model that is based on links and we have shown that it generates more semantically meaningful controversies than the text-based model. Our experimental evaluation has studied the different aspects of the problem and has illustrated the effectiveness of our proposed approach.

6.1.3 Coping with Evolving Items and Users in Recommender Systems

We investigated different solutions, one based on the similarity, one on the features and one on the maximum entropy principle for the new-user/new-item cold-start problem with the assumption that there is absolutely no form of external auxiliary information. We have described the methods and then presented the results of an extensive set of experimentation. We have performed the evaluation of the efficiency and effectiveness of these methods, not only between them but also with other related approaches. Furthermore, we have identified various application scenarios in which the cold start situation is permanently present, thus, traditional solutions that call for a learning period at the beginning are inapplicable.

6.2 Future Work

The problems of evolving data are of primary importance for theoreticians and practitioners due to its ubiquity. With the recent advances of new fields such as big data, social networks, mobile platforms and their dynamic nature, new challenges in exploiting data evolution have arisen. We intend to concentrate on the above challenges in future.

6.2.1 Social Data Evolution

One of the challenging directions is the study of the general problem of evolution in the context of social networks. Social networks have attracted a particular attention in the last decade. It is enough to look at the proliferation of platforms like Facebook, LinkedIn, and Twitter. The data in such systems is unstructured, heterogeneous and distributed. Since users are the main authors in those systems, the generated data is highly subjective, personalized and frequently modified. Exciting challenges are coming from the exploitation of the volatility of social data, i.e. analyze how the social connections evolve over time, what are the temporal patterns of user communications, how to predict future behavior, what kind of trends are in the data, and others. Moreover, due to the subjective nature of social data (the users post their opinions and not facts) the data quality becomes an important issue in that context as well. In particular, the problems of content stability, controversy and vandalism detection in the social application context haven't been fully addressed to date.

6.2.2 Big Data Evolution

Having the concept of evolution well understood, it is interesting to study its form in the context of Big Data. In recent years, we have been observing that data repositories (especially user generated) accumulate huge volumes of data. This scale calls for new ways of dealing with the data since traditional systems fall short of doing that. Discovering hidden evolution patterns and trends, uncover new data insights for such big repositories are challenging and interesting problems.

Bibliography

B. T. Adler and L. de Alfaro. A content-driven reputation system for the wikipedia. In *WWW*, pages 261–270, 2007.

B. T. Adler, K. Chatterjee, L. de Alfaro, M. Faella, I. Pye, and V. Raman. Assigning trust to wikipedia content. In *WikiSym*, pages 26:1–26:12, 2008.

D. Agarwal and B.-C. Chen. fLDA: matrix factorization through latent dirichlet allocation. In *WSDM*, pages 91–100, 2010.

E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *WSDM*, pages 183–194, 2008.

J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, pages 832–843, 1983.

A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, pages 171–210, 2001.

A. Artale, C. Parent, and S. Spaccapietra. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence*, pages 5–38, 2007.

A. Artale, N. Guarino, and C. M. Keet. Formalising temporal constraints on part-whole relations. In *KR*, pages 3–14, 2008.

R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD*, pages 95–104, 2007.

K. Berberich, S. J. Bedathur, M. Sozio, and G. Weikum. Bridging the terminology gap in web archive search. In *WebDB*, 2009.

B. Bercovitz, F. Kaliszan, G. Koutrika, H. Liou, Z. M. Zadeh, and H. Garcia-Molina. Courserank: a social system for course planning. In *SIGMOD Conference*, pages 1107–1110, 2009.

G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

J. Blakeley, P. A. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. In *SIGMOD*, pages 61–71, 1986.

D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *J. Machine Learn. Res.*, pages 993–1022, 2003.

U. Brandes and J. Lerner. Visual analysis of controversy in user-generated encyclopedias. *Information Visualization*, pages 34–48, 2008.

P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. In *SIGMOD*, pages 1–12, 2002.

R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, pages 331–370, 2002.

S. Bykau, N. Kiyavitskaya, C. Tsinaraki, and Y. Velegrakis. Bridging the gap between heterogeneous and semantically diverse content of different disciplines. In *DEXA Workshops*, pages 305–309, 2010.

S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis. Supporting queries spanning across phases of evolving artifacts using steiner forests. In *CIKM*, pages 1649–1658, 2011.

S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis. On modeling and querying concept evolution. *J. Data Semantics*, pages 31–55, 2012.

S. Bykau, F. Corn, D. Srivastava, and Y. Velegrakis. Controversy detection in wikipedia. In *VLDB*, 2013a.

S. Bykau, G. Koutrika, and Y. Velegrakis. On dealing with the permanent coping with the persistent cold-start problem. In *KDD*, 2013b.

S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis. A query answering system for data with evolution relationships. *SIGMOD*, 2013c.

S. Chaudhuri. Editors Won't Let It Be When It Comes to 'the' or 'The'. *The Wall Street Journal*, page A1, Oct 2012.

S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. In *Theory and Practice of Object Systems, Vol 5(3)*, pages 143–162, 1999.

S. Chien, V. Tsotras, and C. Zaniolo. Efficient management of multiversion documents by object referencing. In *VLDB*, pages 291–300, 2001.

S.-C. Chin, W. N. Street, P. Srinivasan, and D. Eichmann. Detecting wikipedia vandalism with active learning and statistical language models. In *WICOW*, pages 3–10, 2010.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, pages 21–27, 1967.

N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, pages 523–544, 2007.

N. N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu. A web of concepts. In *PODS*, pages 1–12, 2009.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 1972.

B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding Top-k Min-Cost Connected Trees in Databases. *ICDE*, pages 836–845, 2007.

X. Dong and A. Y. Halevy. Indexing dataspaces. In *SIGMOD*, pages 43–54, 2007.

X. Dong, A. Y. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*, pages 85–96, 2005.

S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, pages 195–207, 1972.

G. Druck, G. Miklau, and A. McCallum. Learning to Predict the Quality of Contributions to Wikipedia. *AAAI*, pages 983–1001, 2008.

C. E. Dyreson, W. S. Evans, H. Lin, and R. T. Snodgrass. Efficiently supported temporal granularities. *IEEE Trans. Knowl. Data Eng.*, pages 568–587, 2000.

D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green. Automatic generation of social tags for music recommendation. In *NIPS*, 2007.

G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: classification and survey. *Knowledge Eng. Review*, pages 117–152, 2008.

E. Gassner. The Steiner Forest Problem revisited. *Journal of Discrete Algorithms*, pages 154–163, 2010.

J. Giles. *Nature*, (7070):900–901, Dec. . ISSN 0028-0836.

N. Golbandi, Y. Koren, and R. Lempel. On bootstrapping recommender systems. In *CIKM*, pages 1805–1808, 2010.

H. Gregersen and C. S. Jensen. Temporal Entity-Relationship models - a survey. *IEEE Trans. Knowl. Data Eng.*, 11(3):464–497, 1999.

N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *Int. J. Hum.-Comput. Stud.*, pages 625–640, 1995.

W. Gunningham and B. Leuf. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley, 2001.

R. Gupta and G. Hall. An abstraction mechanism for modeling generation. In *ICDE*, pages 650–658, 1992.

C. Gutiérrez, C. A. Hurtado, and A. A. Vaisman. Temporal RDF. In *ESWC*, pages 93–107, 2005.

I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *RecSys*, pages 53–60, 2009.

D. Hawking, N. Craswell, P. Bailey, and K. Griffihs. Measuring search engine quality. *Inf. Retr.*, pages 33–59, 2001.

H. He, H. Wang, J. Y. 0001, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*, pages 455–468, 1990.

E. Ioannou, W. Nejdl, C. Niederee, and Y. Velegrakis. OntheFly Entity-Aware Query Processing in the Presence of Linkage. *PVLDB*, 3:429–438, 2010.

P. Jaccard. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, pages 241–272, 1901.

M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*, pages 135–142, 2010.

K. Jarvelin and J. Kekalainen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

E. T. Jaynes. Information theory and statistical mechanics. *Physical Review Series II*, pages 620–630, 1957.

X. Jin, Y. Zhou, and B. Mobasher. A maximum entropy web recommendation system: combining collaborative and content features. In *KDD*, pages 612–617, 2005.

R. Johnsonbaugh and M. Kalin. A graph generation software package. In *SIGCSE*, pages 151–154, 1991.

V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

A. Katifori, C. Nikolaou, M. Platakis, Y. E. Ioannidis, A. Tympas, M. Koubarakis, N. Sarris, V. Tountopoulos, E. Tzoannos, S. Bykau, N. Kiyavitskaya, C. Tsinaraki, and Y. Velegrakis. The papyrus digital library: Discovering history in the news. In *TPDL*, pages 465–468, 2011.

T. Kauppinen and E. Hyvonen. Modeling and reasoning about changes in ontology time series. In *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*, pages 319–338, 2007.

C. M. Keet and A. Artale. Representing and reasoning over a taxonomy of part-whole relations. *Applied Ontology*, pages 91–110, 2008.

B. Kimelfeld and Y. Sagiv. New algorithms for computing steiner trees for a fixed number of terminals. http://www.cs.huji.ac.il/ bennyk/papers/steiner06.pdf, 2006.

A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi. He says, she says: conflict and coordination in wikipedia. In *CHI*, pages 453–462, 2007.

M. C. A. Klein and D. Fensel. Ontology versioning on the semantic web. In *SWWS*, pages 75–91, 2001.

H. Kondylakis and D. Plexousakis. Enabling ontology evolution in data integration. In *EDBT*, pages 38:1–38:7, 2010.

G. Konstantinidis, G. Flouris, G. Antoniou, and V. Christophides. On RDF/S ontology evolution. In *SWDB-ODBIS*, pages 21–42, 2007.

Y. Kotidis, D. Srivastava, and Y. Velegrakis. Updates through views: A new hope. In *ICDE*, page 2, april 2006.

R. Lau, R. Rosenfeld, and S. Roukos. Adaptive language modeling using the maximum entropy principle. In *HLT*, pages 108–113, 1993.

M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

B. S. Lerner. A Model for Compound Type Changes Encountered in Schema Evolution. *ACM Trans. Database Syst.*, 25:83–127, 2000.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical report, 1966.

P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, pages 956–967, 2011.

J. Liu and S. Ram. Who does what: Collaboration patterns in the wikipedia and their impact on article quality. *ACM Trans. Manage. Inf. Syst.*, pages 11:1–11:23, 2011.

J. Liu, P. Dolan, and E. R. Pedersen. Personalized news recommendation based on click behavior. In *IUI*, pages 31–40, 2010.

C. Lutz, F. Wolter, and M. Zakharyashev. Temporal description logics: A survey. In *TIME*, pages 3 –14, 2008.

S. E. Middleton, H. Alani, and D. C. D. Roure. Exploiting synergy between ontologies and recommender systems. In *WWW*, 2002.

D. Mladenic. Text-learning and related intelligent agents: A survey. *IEEE Intelligent Systems*, pages 44–54, 1999.

D. Molle, S. Richter, P. Rossmanith, and M. G. Anvertraut. A faster algorithm for the steiner tree problem. In *STACS*, pages 561–570, 2005.

E. W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, pages 251–266, 1986.

J. Neetil, E. Milkov, and H. Neetilov. Otakar borvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233:3 – 36, 2001.

T. Palpanas, J. Chaudhry, P. Andritsos, and Y. Velegrakis. Entity Data Management in OKKAM. In *SWAP*, pages 729–733, 2008.

G. Papadakis, G. Giannakopoulos, C. Niederée, T. Palpanas, and W. Nejdl. Detecting and exploiting stability in evolving heterogeneous information spaces. In *JCDL*, pages 95–104, 2011.

A. G. Parameswaran, G. Koutrika, B. Bercovitz, and H. Garcia-Molina. Recsplorer: recommendation algorithms based on precedence mining. In *SIGMOD*, pages 87–98, 2010.

C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag New York, Inc., 2006.

S.-T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *RecSys*, pages 21–28, 2009.

D. Pavlov, E. Manavoglu, D. M. Pennock, and C. L. Giles. Collaborative filtering with maximum entropy. *IEEE Intelligent Systems*, pages 40–48, 2004.

J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. In *ISWC*, pages 66–81, 2008.

A. Presa, Y. Velegrakis, F. Rizzolo, and S. Bykau. Modeling associations through intensional attributes. In *ER*, pages 315–330, 2009.

W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, pages pp. 846–850, 1971.

A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI*, pages 127–134, 2002.

F. Rizzolo and A. A. Vaisman. Temporal XML: modeling, indexing, and query processing. *VLDB J.*, 17:1179–1212, 2008.

F. Rizzolo, Y. Velegrakis, J. Mylopoulos, and S. Bykau. Modeling Concept Evolution: A Historical Perspective. In *ER*, pages 331–345, 2009.

J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, pages 169–194, 1998.

A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, pages 253–260, 2002.

H. Sepehri Rad and D. Barbosa. Towards identifying arguments in wikipedia pages. In *WWW*, pages 117–118, 2011.

H. Sepehri Rad, A. Makazhanov, D. Rafiei, and D. Barbosa. Leveraging editor collaboration patterns in wikipedia. In *HT*, pages 13–22, 2012.

M. D. Soo. Bibliography on Temporal Databases. *SIGMOD*, pages 14–23, 1991.

X. Su and T. M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, pages 1–20, 2009.

N. Tahmasebi, T. Iofciu, T. Risse, C. Niederee, and W. Siberski. Terminology evolution in web archiving: Open issues. In *International Web Archiving Workshop*, 2008.

Y. Velegrakis, R. J. Miller, and L. Popa. Preserving mapping consistency under schema changes. *VLDB J.*, pages 274–293, 2004.

F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *CHI*, pages 575–582, 2004.

B.-Q. Vuong, E.-P. Lim, A. Sun, M.-T. Le, and H. W. Lauw. On ranking controversies in wikipedia: models and evaluation. In *WSDM*, pages 171–182, 2008.

W3C. RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004.

T. Yasseri, R. Sumi, A. Rung, A. Kornai, and J. Kertéz. Dynamics of conflicts in wikipedia. *PLoS ONE*, 2012.

K. Zhou, S.-H. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. In *SIGIR*, 2011.

Index