INSTITUT FÜR INFORMATIK

der Ludwig-Maximilians-Universität München

Diplomarbeit

Who's Who in the World Wide Web: Approaches to Name Disambiguation

Vanessa C. Klaas

Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Prof. Dr. François Bry, Benedikt Linse
Abgabetermin:	31. Januar 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 31. Januar 2007 Vanessa C. Klaas

Abstract

Personal names are fundamental in our civilization. Names serve to refer to individuals, but in contrast to e.g. social assurance numbers that are unique for each citizen, names are not treated that strictly. Names do not identify persons in a non-ambiguous way. There are people that share the same name. Furthermore, the fact that names are not treated as carefully as numbers often leads to misspellings and confusion.

The increasing importance of the Web in our lives has as consequence that people are more frequently confronted with names of things, places and persons. On the one hand, the Web provides access to more information sources and by this to more information, on the other hand the search for relevant information is getting more difficult. A particular problem occurs in (digital) libraries. They are expected to catalog publications in a convenient way that facilitates and supports literature research. It is necessary to distinguish authors that are referred to by their names. The problem of homonymous authors arises, i.e. there may be several authors sharing a name. Inversely, authors may publish under different names or name variations, deliberately or unintendedly. Digital libraries spend much effort on the disambiguation of author names.

This work reports the results of a literature research focusing on disambiguation of homonymous authors and proposes a different perception of the data that is processed during name disambiguation. Different pieces of information are integrated into a graph based approach. An implementation of some ideas of the presented approach serves as a proof of concept and gives further insights in the nature of name disambiguation.

Zusammenfassung

Personennamen sind in unserer Zivilisation grundlegend. Namen dienen dazu, sich auf Personen zu beziehen, aber im Gegensatz zu z. B. Sozialversicherungsnummern, die für jeden Bürger eindeutig sind, werden Namen nicht so streng gehandhabt. Sie identifizieren Personen nicht in eindeutiger Weise. Es gibt Menschen, die sich den gleichen Namen teilen. Außerdem führt eine weniger sorgfältige Behandlung von Namen gegenüber Nummern oft zu Schreibfehlern und Verwirrung.

Die steigende Bedeutung des Web in unserem Leben hat zur Folge, daß Menschen immer öfter mit Namen von Sachen, Orten und Personen konfrontiert werden. Einerseits bietet das Web Zugang zu mehr Informationsquellen und damit zu mehr Informationen, andererseits wird die Suche nach relevanten Informationen immer schwieriger. Ein besonderes Problem taucht in digitalen Bibliotheken auf. Ihre Aufgabe ist es, Publikationen so zu katalogisieren, so daß Literaturrecherche ermöglicht und unterstützt wird. Dabei werden Autoren anhand ihrer Namen unterschieden. Das Problem von Homonymen taucht auf, d. h. es kann mehrere Autoren zu einem Namen geben. Andersherum können Autoren auch unter verschiedenen Namen oder Namensvarianten absichtlich oder unbeabsichtigt veröffentlichen. Digitale Bibliotheken betreiben Namensdisambiguierung mit viel Aufwand.

Diese Arbeit berichtet die Ergebnisse einer Literaturrecherche, die sich auf die Disambiguierung von homonymen Autorennamen und schlägt eine andere Sicht auf die Daten, die während der Namensdisambiguierung verarbeitet werden, vor. Verschiedene Informationensarten werden in einem graphbasierten Ansatz vereinigt. Die Implementierung einiger Ideen, die im Rahmen des Ansatzes vorgestellt werden, dient zur Bestätigung der Ideen und erlaubt weitere Einblicke in das Wesen von Namensdisambiguierung.

Acknowledgments

First of all I would like to thank my supervisor *Prof. Dr. François Bry* for offering the research topic of this thesis and my tutor *Benedikt Linse* for the intensive support. I am deeply grateful to both of them for inspiring and motivating discussions and all-embracing mentoring. I am grateful for the agreeable research environment that was provided by the entire team of the teaching and research unit *Programmier- und Modellierungssprachen*. I would like to mention especially *Dr. Norbert Eisinger* for giving advise during my whole studies and *Dr. Uta Schwertel* for amicable support. I would also like to thank *Dr. Michael Ley* and *Patrick Reuther* at the *University of Trier* for their information on DBLP and valuable hints. Furthermore, I would like to thank my mother for fruitful discussions and for keeping me grounded and my whole family for their understanding. Last but not least, I am grateful to my friends which I got to know during and besides my studies who gave support and guidance.

CONTENTS

Part I

Introduction

CHAPTER

ONE

Introduction

1.1 Motivation

While searching the Web for information using search engines, one often gets irrelevant hits due to words having several meanings, e.g. a query for "jaguar" returns Web pages concerning the wild cat as well as the car company as results. The same happens if one searches for personal names, e.g. a search for "François Bry" results in Web pages concerning at least two different individuals, a French musician living in Toulouse and a professor at the Institute for Informatics at the University of Munich. In this case, one would probably not encounter difficulties to decide to which François Bry an occurrence on the Web refers, but if one searches for "Martin Hofmann", Google finds several scientists and even several professors, a TV film producer, a manager, a high school graduate etc., whereas DBLP[1], a digital library for computer science, proposes seven different authors having published in the domain of Computer Science. Apparently, the problem of ambiguities is more stringent for personal names: all over the world, there are people with identical names. With the growth of the Web, more and more persons' names occur on the Web which means that a single name may represent dozens of different persons. Therefore, it would be more than helpful if there was a "people search engine" that would distinguish different individuals not only based on purely textual comparisons, but also and primarily by semantical means, i.e. by analyzing the semantic context in which names occur. In this project, semantic context is understood as structured or unstructured text, though other kinds of data, e.g. pictures, provide room for interesting approaches to this problem. Anyway, this project focuses on text, more precisely on XML data extracted from DBLP, since other data formats belong to other research domains.

The basic underlying problem has two complementary facets: the first category of ambiguities consists of several persons having identical names are called *homonyms* in [Reu06], e.g. the above-mentioned Martin Hofmann. The complementary category is called *synonyms* in [Reu06] and results from the fact that a single person may have several names or rather be represented by several *"name labels"*, so-called "pseudonyms". This may happen accidently through typos and inconsistent use of names or deliberately, e.g. last names might be changed because of marriage as the professor Claudia Linnhoff-Popien[2] who started her career as "Claudia Popien". Frequently, different spellings for a name are induced by transliteration from e.g. an Asian language or Cyrillic alphabet etc. Transliteration actually depends on the target language, e.g. the Russian-American pianist, composer and conductor Сергей Васильевич Рахманинов is in Germany (mostly) written as Sergei Wassiljewitsch Rachmaninow (see the German Encyclopedia Brock-

[1] Database & Library Project: http://www.informatik.uni-trier.de/~ley/db/

[2] http://www.mobile.ifi.lmu.de/~linnhoff/linnhoffPub.php

haus[3]), in English language mostly as Sergey (or Sergej) Vassilyevich Rachmaninoff (see Encyclopædia Britannica Online[4]).

For solving the task of name disambiguation, two main obstacles are the diversity and the inhomogeneity of the Web. The former requires a semantical categorization of Web pages which is still not realized, from the latter follows that one can neither make any assumptions about the structure of visited Web pages, nor easily get categorical informations such as year of birth, working domain, social relationships to others etc. The Semantic Web is expected to categorize the diversity of contexts. Information Retrieval and computational linguistics search for techniques for solving the problem of inhomogeneous structure. To partially avoid the diversity and inhomogeneity of the Web in this project, the field of research is narrowed to a domain that is well-understood and provides better structured data in a more homogeneous way. The benefit of such a restriction is that this project focuses the task of name disambiguation in a given context. Furthermore, it is a common practice to solve a (kind of) simplified problem first and use the results to solve the complex problem version in a second step: if one does not manage to fulfill the small task, how could one possibly find a solution for the big problem? Nevertheless, the adverse conditions coming with the Web are an interesting and fascinating challenge and after all, this decision should not be seen purely as a restriction, but rather as the beginning, a first step, to manage eventually the big problem.

Digital Libraries of scientific publications are a familiar domain satisfying the proposed criteria. In addition, ambiguities among author names is an urgent issue in digital libraries. Some of them like DBLP provide well and uniformly structured data which is publicly accessible. For practical reasons, we do not consider digital libraries in general, but rather concentrate on the scientific field of Computer Science: it is easier to handle names and subjects of the own field of interest than it is in some other scientific domain like e.g. Chemistry. Since we have more contact to computer scientists as to chemists, it will be easier to get information about some computer scientist through colleagues than about some chemist. Furthermore, related topics are more easily recognized as such, in contrast to research about e.g. “Diaryl ether peptoids from marine sponges”[5].

Keeping these aspects in mind, the digital library of choice for examining name disambiguation that will provide real-world data, is DBLP for several reasons. First, it provides public access to data covering indeed not completely, but sufficiently well, the publications of research in Computer Science. Second, the data is given in XML format, i.e. the data is provided uniformly ensuring quite easy data processing. Furthermore, we have close contact to its founder and maintainer Michael Ley, University of Trier, which already turned out to be very useful.

Since DBLP has an important role in this project, we give a more detailed description based on the article [LR06] of Michael Ley and Patrick Reuther. As already mentioned, the focus of DBLP lies on data quality. That is why DBLP is that popular among computer scientists. The goal of Michael Ley is to assure and improve data quality. This has impact on the process of inserting new data, i.e. publications. The sources of new data are diverse and heterogeneous: electronic documents, online portals (e.g. SpringerLink[6]) and less informative and reliable (due to typos) sources like title pages and tables of content, e.g. of volumes of journals. There is a policy of insertion which is strictly followed, the so-called *all or nothing policy* which instructs that insertions are made completely for a volume, i.e. all publications of a volume are inserted at once. Every author occurring in DBLP has its own author page, i.e. for each author an html page is generated containing a list of his publications and a list of his coauthors. The “coauthor relation” induces a social network, the so-called *coauthor graph*. The insertion of a new publication results in the insertion of new edges in the graph or in the increment of weights of edges. The correct location of insertion, i.e. the correct individuals, are searched manually with regular expressions.

[3] http://www.brockhaus.de
[4] http://www.britannica.com
[5] http://www.cup.uni-muenchen.de/oc/lindel/WS%20Lindel%20research.htm
[6] http://www.springerlink.com

Additionally, a search for errors is conducted. Errors are detected by examining the neighborhood of a person in the coauthor graph: in most cases errors are locally restricted. The used tool is the DBL-Browser described in [KLR+04b, KLR+04a]. Additionally, *data bashing* is conducted: suspicious data is compared with data entries of other sources. In the case of contradictions one has to decide which source provides the correct data. Furthermore, business rules are applied for data edits. The system warns if there are two similarly spelled authors with identical coauthors. The distance measure for author names is an edit-distance, a modified Levenstein-distance. See Section 4.4.1 for more detailed information on such distance measures. To cope with the huge amount of data, a blocking function is applied to pre-group the coauthor graph into blocks. Then only the nodes of the same block are compared pairwise.

DBLP is subject of a variety of researches, e.g. Elmacioglu and Lee provide a statistical analysis of a subset of DBLP in [EL05]. They analyze a sub-community in DBLP of authors researching in the area of databases in the period from 1968 to 2003. The focus of that analysis lies on the structure of the social network. The analysis reveals interesting numbers about "new authors", i.e. authors publishing for the first time, "active authors" having at least published once in the considered year, number of papers per author, number of coauthors per author and provides an insight into the structure of the coauthor graph of the community. Some interesting results are that in the database community, the number of new authors increased steadily as well as the number of active (old and new) authors. On average, an author publishes about 0.3 papers per year which is a quite constant number since the 80ies. The average number of coauthors increased more or less steadily to almost 3.5 coauthors per author in 2003, whereas less and less active authors publish alone. Based on their figures, Elmacioglu and Lee conclude that DBLP or rather the considered sub-community of DBLP is a "small world" network, i.e. a network in which all members can reach each other only by a small number of "hops". The term *small world* was created by S. Milgram in [Mil67].

As generally on the Web, the two afore-mentioned problems of homonyms and synonyms are encountered in digital libraries, also (even though less frequently) in DBLP:

homonyms there are several different authors (i.e. real persons) having the same name; abbreviated first names, which are avoided in DBLP, even increase frequency of this phenomenon.

synonyms one person has several different names, as described above due to typos or inconsistencies in using names, but there are also legitimate "semantic irregularities"[HOL04], an author changes his/her name because of e.g. marriage.

These problems are also referred to as "name authority control" coming from library and information science, "personal name matching" as described in [BS92], "mixed citation and split citation problems" as reported in [LOKP05] (for homonyms and synonyms) and generally as"name disambiguation". A solution is essential for ensuring data quality in digital libraries as difficulties (due to name ambiguities) in literature research and in maintaining digital libraries show. Thus, the restriction to names occurring on the Web to author names occurring in DBLP preserves the problem of ambiguities among names and at the same time focuses a domain where the problem really is acute: Michael Ley and his team spend much time on semi-automated name disambiguation to get consistent names and identify homonyms. The decision in favor of DBLP probably turn out to be useful and to be rather the starting point for the more general and complex problem than a restriction.

After a first study of relevant literature about name disambiguation, it seems that the problem of synonyms is more often examined than the one of homonyms, especially in the domain of digital libraries. But there are lots of other applications besides general Web search that would be improved by conducting name disambiguation in both directions. The examples range from e-commerce (an online shop might gather additional informations about its customers) over selective advertising to security issues. In [AMNR+06] the authors equally face the problem of ambiguous names while examining ways to detect conflicts of interests. Such use cases are described in detail in the next section.

This project concentrates on the problem of homonyms, i.e. the question to answer is, how and to which extent it is possible to determine, if occurrences of identical or similar author names in publication citations[7] refer to one unique or to several different persons.

1.2 Use Cases of Name Disambiguation

As already sketched, name disambiguation is important in lots of applications. In the following, some of these applications are described. Notice that these use cases occur in a diversity of situations in the real world. We just give some examples to illustrate the importance of name disambiguation. Statements about the potential of realization are not given.

1.2.1 Digital Libraries

The example in which name disambiguation is needed that is the closest to this project comes from the area of digital libraries. As already described, it is not easy to recognize homonymous names as well as synonyms. Such errors complicate literature researches. Moreover, digital libraries are analyzed to evaluate publication venues, authors, research groups etc. The standard but controversial measure for journals is the impact of citations: it is counted how much papers of a journal are cited in papers of other journals. Analogously, single persons and research groups count how much their articles are cited by others. A particular example of such a citation analysis is given in [RT05]. Citeseer, too, estimates the impact of journals[8]. But in general, erroneous data yields to misleading results. For example, citations of an author X are falsely contributed to another author Y. Then the author Y is considered to be "better" than he is, on the costs of X. Name disambiguation helps to assure a high quality of the data. In consequence, literature research is eased and injustices in valuations are eliminated.

1.2.2 Detection of Conflicts of Interests

There are many situations in which people get conflicts of interests. An example is given in [AMNR$^+$06]. Potential reviewers participating in a scientific peer review process may be in a conflict situation if there are social relations to an author of the subject of review. In [AMNR$^+$06] there is described an approach to the detection of such conflicts that integrates two different social networks to establish social relations between individuals. It is necessary to identify individuals that appear in both social networks by their names. To solve this task, name disambiguation is required.

1.2.3 (Specialized) Search Engines

A related example is of commercial interest. Assume the situation where a customer searches for offers of some product, e.g. flights, hotels or also cars. It would be useful if the customer could use some specialized search engine that "understands" what the customer is searching for. Such search engines would need to disambiguate names in general, personal names as well as names of products or geographic locations. Truly, some search engines comparing prices already exist, but, to put it mildly, they are not very intelligent. Such search engines require even more than name disambiguation, the Semantic Web promises further improvements.

[7] in terms of references to papers as e.g. appearing in the list of references in the appendix

[8] http://citeseer.ist.psu.edu/impact.html

In general, search engines conducting name disambiguation will return more accurate results than traditional search engines. They can group the retrieved results according to a cluster structure. An example is vivisímo[9]. Here, too, name disambiguation needs to be defined more generally so that not only personal names are handled. Clusty[10] is another example similar to vivisímo. Another search engine that is specialized on people and companies is zoominfo[11]: when querying for a name of a person or of a company, results are clustered according to such entities and additionally, summaries are generated. Persons and companies can also be combined, e.g. the query "Martin Hofmann" yields "Martin Hofmann at Munich", "Martin Hofmann at Volkswagen AG", "Martin Hofmann at Electronic Data Systems Corporation" amongst others. The search engine distinguishes 28 "Martin Hofmann"s, number 23 is "Martin Hofmann at Ludwig-Maximilians-University". Notice that the search engine is able to distinguish between the professor in Munich and the prosecutor "Martin Hofmann at Munich" who was involved in the case of al Masri described in the next example. Number 26 is a "Martin Hofmann at Edinburgh University", some Dr Martin Hofmann. We know that this is the same person like the professor who is living now in Munich, because we know that he was for some time at Edinburgh University. To eliminate last doubts, we could just ask him. This example shows that there is also still room left for improvements.

1.2.4 National Security

In the context of the terroristic attacks of September 11, 2001, the CIA mistook the German citizen of Lebanese origin Khaled al Masri for a member of the terroristic organization al-Qaeda and abducted him to Afghanistan in 2003. It seems to be a standard practice called "rendition" to abduct suspected people. Besides the terrifying experiences that al Masri underwent, a political scandal was aroused in Germany. The ethnic question if the end justifies the means shall not be discussed here. But we argue that it would be good for all (except the terrorists) if such cases could be avoided. Name disambiguation could serve here. If a person is characterized by comparable features, she could probably be eliminated from the list of suspects because she is identified to be a homonym, i.e. a name sister, of the real terrorist. It seems, that the criminal search[12] is a corresponding method but works the other way round: this method tries searches for suspects, but does not eliminate falsely suspected individuals from a given list of suspects. Maybe, al Masri is even a victim of a kind of criminals search.

1.2.5 Detection of Social Welfare Freeloaders

In 2005, the German media reported that the German government thinks about an application of an adjusted "Rasterfahndung" to detect social welfare freeloaders. Once again, ethnic and juristic questions are not discussed here. Just imagine that there is e.g. a Hans Meier being on social benefits and another occurrence of a Hans Meier who has just married a woman having a good job. Name disambiguation could decide if these two Hans Meier are the same person. It seems that some kind of name disambiguation is already done. This illustrates, that name disambiguation often happens, even if there does not exist any formal method.

[9]http://search.vivisimo.com/
[10]http://clusty.com/
[11]http://www.zoominfo.com/
[12]in German: Rasterfahndung

1.3 Intentions of this Work

1.3.1 Problem Definition

As we described earlier, the term “name disambiguation” comprises two different, but connected problems, namely the task of disambiguating homonyms and synonyms. The former is also called “mixed citation problem” which means that authors having the same name are mixed up and the goal is to assign these name occurrences to the correct individuals. The latter is also called “split citation problem”, describing the fact that citations of a single author are split up in several names.

In this work, we concentrate on disambiguating homonyms, which is defined in [HZG05] in a general way:

> *“Given a set of citations that have an ambiguous (e.g. identical) name label, how do we disambiguate authors if the name label refers to a single author, or different authors with ambiguous names?”*

Relevant literature formulates this quite general question in slightly different variants, but all these variants assume that only one name at a time needs to be disambiguated and occurrences of other names can be treated as already disambiguated. We call the problem resulting from this assumption *single name disambiguation*. In the following, different problem definitions are presented:

"k-way classification problem","mixed citation problem" Additionally, it is assumed that the number k of different authors hiding behind a given author name X is known. Then, the disambiguation problem consists in assigning each citation c containing name X of a set of citations C to one of the k distinct authors with name X, obviously to the person who authored the publication.

Definition in [HGZ$^+$04] *“Given a full citation with the query name implicitly omitted, our name disambiguation is to predict the most likely canonical name from the citation base.”*

1.3.2 Our Contributions

As already motivated, we give a comprising overview over different approaches to name disambiguation. The overview is restricted to the mixed citation problem. The approaches are described in detail. Additionally, experimental results are reported. This allows to draw some conclusions about room for improvement and natural limitations, i.e. limitations due to the nature of name disambiguation. Most of the approaches presented in Chapter 5 have in common that they represent citations as elements of a vector space. During the preparation of our experiments and the datasets, some interesting numbers concerning DBLP are revealed. These facts are also reported.

In our own approach, we understand the data that is processed in a different way, namely in terms of graphs. The *graph approach* is explored, its advantages and limitations are shown. For this purpose, several similarity measures, i.e. measures for quantitatively comparing citations, will be proposed and evaluated: we present different possibilities to measure similarity between papers with an author name in common (where a high similarity indicates, that the name in common stands probably for a single person). Furthermore, we present an algorithm that can easily be modified by just replacing the similarity measure. These different similarity measures are provided in terms of functions that can be used by the clustering algorithm. Additionally, an evaluation framework is provided that allows to compare the performance of different similarity measures.

Notice that the intention of this work is *not* to provide a perfect solution to the problem. It even seems that a perfect solution, i.e. some algorithm that clusters without any errors does not exist for several reasons.

The problem definition cited above includes one essential assumption: there is only one single name to disambiguate at once and the rest of DBLP is (more or less) error-free. Therefore, the problem definition as given above is called *single name disambiguation problem* in this work. On a first glance, this assumption seems to sabotage the project since DBLP is not error-free which leads to more ambiguities theoretically possible for every name. But there are several arguments which legitimate this assumption.

Remember that this approach is designed for the use on DBLP data. DBLP was chosen from the beginning amongst other reasons because it is maintained with very high efforts, even manually, to secure data quality so that one can argue that error quota is sufficiently low. In case of doubts, the data could be checked manually since only a small subset of DBLP is used. Such controlling of data quality would be useful for evaluation purposes.

We also argue that the probability that authors having ambiguous names publish together is sufficiently small. In consequence, we can neglect this fact.

However, we give hints how to eliminate this strong assumption leading to a real extension of the problem definition. The consequence will be a more complex algorithm consisting of some recursive fix point calculation based on matrix manipulations.

Intuitively, it seems possible to generalize the algorithm computing the *single name disambiguation problem* to one computing the *multi name disambiguation problem*, which contains the other extreme situation: no name is disambiguated; as well as all states in between: some names are disambiguated and some are not. The most important question seems then, how do similarity values of similarity measures propagate when some cluster assignment changes during clustering. To which computational complexity does this propagation of values lead and is it possible to guess some optimal processing order in advance? The last question can be important because it is possible that the processing order influences the result.

The main problem of multi name disambiguation is that "all depends on all" when using coauthor information which in past work proved as an information essential for good results. This fact "all depends on all" reminds of Google's PageRank which is "only" calculation of matrix powers.

Essentially, this work covers the following aspects:

Literature research An exhaustive inquiry of relevant literature will give a comprehensive review of existing methods and algorithms for name disambiguation concentrating on disambiguation of homonyms. This review introduces the subject of name disambiguation and categorize its current approaches so that the reader easily comprehends the presented methods and gets a good impression of the state-of-the-art. A survey comparing the performance of the different approaches will be included as far as possible on the basis of the published results.

Conception In dependency of the results of the literature research, we design another method and it is examined what else may improve name disambiguation, e.g. is it possible to *automatically* recognize more information of the semantic context in which names occur? Improvement means e.g. an algorithm with higher precision, recall or accuracy or a more scalable solution.

To respond to the above-mentioned question, one needs to solve the problem of *automatically* extracting and using more information as e.g. email-addresses, date and location, affiliation etc. It would be interesting to have an estimation of how useful such further information actually is.

This project will focus mostly on the question if it is possible and how to utilize more of the given semantical context and also Semantic Web technologies, e.g. FOAF[13] (friend-of-a-friend) and RDF[14] (Resource Description Framework) may be useful in the future. Actually, available data seems to be too sparse. Also worth answering is the question, how ontologies (e.g. of publication venues) can be constructed and used for name disambiguation. A generic approach to this task is another research topic that should be investigated in a separate project. In this case, further sources as WordNet[15] or Wikipedia[16] may be useful.

Implementation Since it will be necessary to retrieve and process Web pages and/or XML data which is well supported by Java, the implementation is done in Java. Web query languages like XQuery or especially Xcerpt could also be of use, but actually, there are no mature implementations available. The latter is especially interesting because Xcerpt is able to query both XML and RDF. Due to the huge amount of data to process a "real" database is useful.

Evaluation One or several test collections are needed. The datasets are extracted from DBLP. Publication lists gathered from the Web could come into play, but the amount of data in DBLP is sufficiently huge and manually extracting information from the Web consumes lots of time. There are some test collections (see also the report of Patrick Reuther [Reu06]), e.g. created by a working group at Pennsylvania State University[17]. Some of them are designed to evaluate the performance of algorithms computing the problem of synonyms, or are not any more publicly available. All of them have in common that they do not provide all or rather different information that is required by our approach. It would be good to implement some algorithm as benchmark but due to the lack of time, we compare our approach with the others presented in this work.

1.4 Outline

The next chapter continues the introduction with a wide overview of related work. This overview comprises tightly connected topics as well as a wider context, e.g. research areas providing techniques used in name disambiguation and research areas using name disambiguation.

The part about fundamentals provides in Chapter 3 the basic theory of graphs and knowledge of linear algebra that is fundamental for the rest of this work.

Chapter 4 introduces as generally as possible the clustering and classification methods that are applied to name disambiguation. Additionally, particular tools adjusted to name disambiguation, e.g. similarity measures, are described.

The main part of this work starts with Chapter 5. Clustering and name disambiguation are identified as related topics and it is described how in general clustering can be applied to name disambiguation. Then, a detailed overview of the existing approaches to name disambiguation in the sense of disambiguating homonyms is given. Besides a detailed description of the functioning of each method, the results of experiments are reported. The chapter concludes with a discussion of the found results.

In Chapter 6, we present our own approach to name disambiguation. The approach is motivated by facts concerning social networks and by conclusions of the previous chapter. This chapter presents the features that can be used as well as the way of representation of these features.

[13]http://www.foaf-project.org/
[14]http://www.w3.org/RDF/
[15]http://wordnet.princeton.edu
[16]http://en.wikipedia.org/
[17]http://clgiles.ist.psu.edu/data/nameset_author-disamb.tar.zip

The involvement of features in a clustering algorithm is presented in Chapter 7. The logical design of the algorithm is described. Then we give some details about implementation and finally, we evaluate our approach in some experiments. The chapter concludes with a comparison to other approaches.

Finally, Chapter 8 concludes the work with a discussion of the achieved results and an outlook to still open questions.

CHAPTER

TWO

Related Work

This chapter embeds the topic of name disambiguation in the environment of related topics and research areas. The main research areas are databases, information retrieval and research concerning the Web. All of them have in common to be interested in data, yet the sources and representations are different. This identification of related fields induces the structure of the chapter: there is a section devoted to each of it.

2.1 Related Topics in the Research Area of Databases

All of the following topics investigate the problem of name matching. The different problem names come from the fact that the problem of name matching occurs in several research areas and in different situations.

2.1.1 Record Linkage

Fellegi and Sunter published in 1969 a famous article [FS69] in the statistics community discussing a problem also fascinating the database community. *Record linkage* is the task to link records occurring in different files (or databases) referring to the same entity. Such techniques should be applied if data sets from different sources are to be joined. Record linkage is the equivalent of the merge/purge problem described in Section 2.1.3. The difference is just that the problem is addressed with one name by the statistics community, with the other name by the database community. Another name is *data deduplication*.

2.1.2 Duplicate Record Detection

The topic of *duplicate record detection* explores how to identify and eliminate duplicates in large databases. These duplicates were not introduced by merging data from different sources. Even without such merges, data gets "dirty" in the curse of time due to inconsistent abbreviations, errors in the data etc. Therefore, this task is also related to "data cleaning". Several work has been done, e.g. [BD83, ME97]. A knowledge-based approach is proposed in [LLL00]. An adaptive approach that adjust the way of measuring similarities between records is given in [BMC+03].

2.1.3 Merge/Purge

The task called *merge/purge* is to merge different sets of data, e.g. coming from multiple sources and then purge the obtained set of data from unwanted records, e.g. duplicates. For example, there are two tables (of two relational databases) with equivalent schemes. For some reason these two tables should be merged to one. Then it is often the case that the merged table will contain duplicate records. The problem is that they may be not recognized as such easily. For example, some names occurring as values in data records could be abbreviated in one table, but unabbreviated in the other. Some approach for large databases is given in [HS95].

2.2 On Natural Language Processing

Two subareas of *natural language processing (NLP)* are *Information Retrieval (IR)* and *Information Extraction (IE)*. IR provides techniques for retrieving information that seemed to be lost. With the accelerated growth of amounts of data it becomes more and more important to retrieve data relevant for queries. Such queries are not necessarily precise, but may be vague or given in natural text. In contrast to IR, IE provides techniques that extract information from natural text and presents it in a (semi-)structured form. IE has among others the tasks *named entity recognition* and *coreference resolution*. Both of them are involved in the following mentioned works, coreference resolutions corresponds more directly to name matching.

2.2.1 Cross-Document Coreferences

Cross-document coreference depends on "intra-document coreference": first, entity names have to be identified in documents and relations between them recognized that occur within documents. Then, relations across documents can be recognized. An approach to this task is given in [RK]. The automatic extraction and recognition of names is focused, too. An approach that is restricted to personal names within documents is given in [WRC97]. Disambiguating personal names in text is addressed in [MY03]. All these works are close to IE in the sense that the focus lies on the automatic extraction of relevant information from natural text. Then information extracted from a defined context of the occurrence of a name is used to disambiguate several occurrences of names. A good overview of IE is given in [Cun05].

2.2.2 Automatic Generation of Ontologies

Ontologies have the purpose to model some domain in terms of classes or entities that have attributes and relationships to other classes or entities. If such ontologies did not evolve, one could construct them manually. That is not an elegant way and depending on the domain it would require some time and human resources, but it would be feasible. Unfortunately, domains usually change and with them ontologies evolve. Therefore, a better solution would be to generate *automatically* ontologies, e.g. based on text sources coming from a particular domain. This is a challenging task and one subtask that is performed to reach the goal is name-matching. An actual subarea of IE explores methods to generate *automatically* ontologies based on some text corpus of a particular domain. Possible tasks are to populate ontologies, i.e. assigning entities occurring in text to concepts of the ontology, as well as just recognize instances of an ontology in text. The first task comprises *ontology enrichment*, the process of "updating" existing ontologies. It is easy to imagine that named entities may have several different names, so-called *synonyms*. That is where name matching enters the game: name matching is the task to recognize such synonyms. The awareness of the existence of synonyms allows to match affected entities and treat them accurately. Such an approach is presented in [VPKV04].

Inversely, ontologies can be involved to improve similarity measures. An example is given in [Y05] that demonstrates also the wide range of applications using a *structured cosine similarity* for document summarization of audio data, i.e. speech documents.

2.3 Name Disambiguation on the Web

Existing literature describes two categories of approaches for disambiguation of references to entities (e.g. persons). Such references appear on Web pages. If an approach does not make any use of specifics of the Web, e.g. of the hyperlink structure, this type of name disambiguation is equivalent to name disambiguation on documents in a general sense. Approaches of the second category are more specialized on Web data and propose remedies for inconveniences arising e.g. in Web searches.

2.3.1 Name Disambiguation on Web Data

The approach presented in [LBG05] concentrates on personal names occurring in Web pages. Analogously to approaches of IE and IR, the context of names is used to conduct disambiguation, i.e. Web pages are reduced to text snippets around name occurrences which provide features representing these occurrences. In the case that name disambiguation is based only on text, it does not matter what kind of documents is given since Web pages, too, can be considered to be text documents if other elements, e.g. hyperlinks, are ignored.

2.3.2 About Search Engines

Search Engines provide a more or less direct access to required information populated on the Web. Of course, it depends on the quality of a query and of a search engine if URIs to relevant pages are returned. Search engines *crawl* the Web to get a knowledge about existing pages. *Index structures* are used to speed up the access. The difference makes the *ranking algorithm* that ranks Web pages according to their importance which has impact on the order in which results are presented. The most famous of such ranking algorithms is probably *PageRank* of Google. A detailed explanation is given in [BL06]. The algorithm performs a *random walk* to distribute *"importance scores"* to Web pages. This score depends on the ingoing links to a page, so-called *backlinks*. Very simplified spoken, the idea is that a link from one page to another means that the first page considers the second page to be important. Recursively, the score of a page has impact on the degree in that the page influences scores of other pages. The random walk is described by means of *Markov Chains*, a stochastic model that assumes a stochastic process consisting of several states related by randomized transitions, i.e. each transition is performed with a particular probability. In the case of PageRank, the process is the crawling, states correspond to Web pages and transitions to links between Web pages with probabilities in form of importance scores. This structure is represented by a matrix called "link matrix", here denoted by A. Then, the importance scores are computed by solving the linear equation system $Ax = x$, i.e. finding an *eigenvector* with *eigenvalue* 1: as explained in Chapter 3.2.3, an eigenvector x is a solution to the equation $Ax = \lambda x$, where λ is a scalar called eigenvalue. This eigenvalue problem is approximated by a *fixed point calculation* that consists of matrix multiplications, more exactly of computing matrix powers.

2.3.3 Extensions for Search Engines

Nowadays, existing search engines are insufficient by means that they do not please the users anymore. Due to the amount of Web pages it is hard to find the pages that are relevant to a query in the user's opinion. For example, a user is already thinking about a particular object of the real world, an entity, e.g. a book, a person, a company etc. Now, the user is interested in retrieving Web pages concerning that entity. To satisfy the user, some name disambiguation that relate names occurring in Web pages with real-world entities is needed. An approach is given in [HySK04].

Another approach is proposed in [GG04]. The idea is to extend search engines by a sophisticated sort algorithm that ranks query results in two steps. First, the "normal" list of results is displayed and the user can follow a link that is given for each hit. The link leads to a reordered list of results that rank all hits that ranks result higher with tighter relation to the selected hit.

Furthermore, search engines should return enhanced results that consist not only of relevant links but also of summaries based on the result set of relevant pages. This would be applicable especially for persons and organizations which is approached in [YCLK02]. Such a summary for an entity is constructed by extracting all information from the Web pages that are clustered as being related to that entity.

In [BM05] the search for particular persons is extended to the search for several persons that are related in a social network. The approach differs from previous ones in the fact that the link structure of Web pages is involved.

2.4 Name Disambiguation in Digital Libraries

To conclude this chapter about related work, the most closely related works are presented. The task of disambiguating authors in digital libraries, particularly in DBLP, is addressed in the sense of detecting synonymous names.

The authors of [LHK04] take a "database point of view" and formulate the problem of name disambiguation as "cleaning the spurious links", i.e. eliminate multiple links between database records and entities of the real world. Such spurious links are equivalent to synonymous names. The approach is based on comparisons of the contexts in which author names appear, i.e. in citations.

A "(digital) library point of view" is taken in [SvR00]. Name disambiguation is identified as a problem occurring during name authority control which is the task to fix a so-called "authoritative" form for each name.

The approach presented [Fei04] is specialized to recognize synonyms that differ due to variations in first names. This is done without using other information than the names themselves.

The problem of synonymous names is also addressed in [LOKP05]. String-based and content-based approaches are compared. A blocking method is used to increase efficiency. A similar blocking-based approach to the disambiguation of synonyms is proposed in [ALLM05]. Furthermore, the presented techniques are involved to disambiguate homonymous names. The resulting approach is described in detail in Section 5.2.3.

Researchers at the home of DBLP, the University of Trier, too, focus on the problem of name disambiguation in the sense of detecting synonyms and involve social networks. In [Reu06, RW06] an overview on approaches and available test collections are given and a "social network based approach" is proposed.

Part II

Fundamentals

CHAPTER

THREE

Graph Theory

For later chapters, some fundamental terms of graph theory are needed. Therefore, this chapter gives a short introduction to graph theory. Notions and definitions are based on the book [Die05].

3.1 Basic Definitions and Notions Concerning Graphs

3.1.1 Types of Graphs

Definition 3.1 (Graphs.) *A* (undirected, unweighted) graph *is a pair $G = (V, E)$ of disjoint sets V and $E \subset [V]^2 (= \{\{v, w\} \mid v \in V and\ w \in W\}$, where $[V]^2$ denotes the set of all subsets of V having exactly two elements. The elements of V are called* vertices *or* nodes *and also denoted as $V(G)$. The elements of E are called* edges *and also denoted as $E(G)$.*

A graph $(\emptyset, \emptyset)$ is called *empty graph*, the graphs with 0 or only one vertex is called *trivial*. Notice that the definition of edges as two-elementary sets means that edges are undirected. Indeed sets are *unordered*, i.e. $\{v, w\} = \{w, v\}$. An alternative way of definition would be to describe edges as tuples $(v, w) \in V^2$, but this definition would lead to *directed* graphs as basic graph structure. The standard way of definitions is the one involving sets as structure for edges. Then, directed graphs need to be defined based on undirected graphs as done in the following definition.

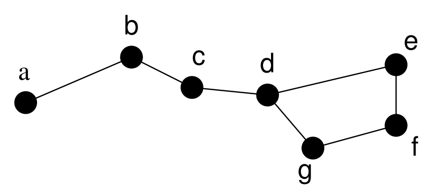


Figure 3.1: A graph $G = (V, E)$ with $V = \{a, b, c, d, e, f, g\}$ and $E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{d, e\}, \{e, f\}, \{f, g\}, \{g, d\}\}$

To shorten the following definitions, let $G = (V, E)$ be an (undirected) graph.

Definition 3.2 (Directed graphs.) *A* directed graph *or* digraph *is a graph $G = (V, E)$ together with two maps $anker, head : E \to V$ with the property that $anker(e), head(e) \in e$.*

If $anker(e) = head(e)$, the edge is called a *loop*. With V_{anker} and V_{head} we denote the the subsets of V that contain all nodes that are ankers, heads resp., i.e. $V_{anker} = \{v \in V \mid \exists e \in E : anker(e) = v\}$ and $V_{head} = \{v \in V \mid \exists e \in E : head(e) = v\}$.

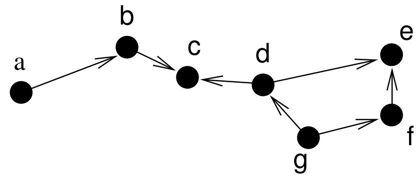


Figure 3.2: A directed graph

Often, it is desirable to *quantitatively* evaluate the connections, i.e. the edges, between vertices. A number w on an edge can mean e.g. the probability that the edge exists or the number of connections between two vertices.

Definition 3.3 (Weighted graphs.) *A* weighted graph *is a graph $G = (V, E)$ whose edges are* weighted, *i.e. the graph has an associated* weighting map $\omega_G : E \to \mathbb{R}^+$. *The value $\omega_G(e)$ for an edge e is called* weight. *If it is clear to which graph G the weighting map ω_G is associated, the index G can be omitted. A* weighted digraph *is a digraph together with a weighting map $\omega_G : E \times V_{head} \to \mathbb{R}^+$.*

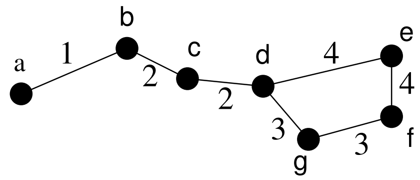


Figure 3.3: A weighted graph

3.1.2 Characteristics of Graphs

The main characteristics of graphs are the relations between vertices and edges in all combinations, i.e. the general structure of graphs. Notice that the definition of edges as sets of two elements allows the notion $v \in e$ for a vertex v and an edge e. The meaning is given in the next definition.

Definition 3.4 (Incidence, adjacency.) *A vertex $v \in V$ is* incident *with an edge $e \in E$, written as $v \in e$, if there exists a vertex $v' \in V$ such that $e = \{v, v'\}$. The set of all edges incident with v, is denoted by $E(v)$.*

Two vertices $v, w \in V$ are adjacent *or* neighbors *if $\{v, w\} \in E$. Two edges e, f are called adjacent if there exists a vertex $v \in V$ such that $v \in e$ and $v \in f$.*

Two vertices or edges that are not adjacent are called *independent*.

Example 3.1 *In the graph G of Figure 3.1 a is incident with edge $\{a, b\}$ and adjacent with b. The edge $\{a, b\}$ is adjacent with edge $\{b, c\}$.*

The above defined situations are also *quantitatively* described.

Definition 3.5 (Order, volume of a graph.) *The number of vertices, $|V|$, is called the* order *of the graph, written as ord(G), the number of edges, $|E|$, is the* volume *of the graph, written as vol(G).*

Graphs can be *finite*, i.e. $\mathrm{ord}(G) = n \in \mathbb{N}$, as well as *infinite*, depending on the number of vertices. In computer science, only finite structures are processed. Therefore, we assume finite graphs in the remaining chapters.

Definition 3.6 (Degree of a vertex.) *The degree of a vertex v is the number of edges with which v is incident,* $\deg(v) = |E(v)|$.

Example 3.2 *For the graph of Figure 3.1 it holds that ord$(G) = 7$, vol$(G) = 7$ and* $\deg(d) = 3$.

3.1.3 Graph Manipulation with Matrices

To handle graphs in a simple way, it is convenient to describe them in form of matrices. All important properties of graphs can be expressed by matrices. This mapping of graphs to matrices allows to process graphs using linear algebra. The main benefit is that results from the theory of linear algebra can be used and applied to graphs. Thus, elegant "tools" are given for free. With regard to a future implementation, matrices are convenient data structures to represent graphs.

Definition 3.7 (Adjacency matrix.) *The* adjacency matrix $\mathcal{A} \in \mathbb{N}^{|V|\times|V|}$ *of an unweighted graph is a matrix with*

$$a_{ij} := \begin{cases} 1 & \text{if there is an edge between the vertices } v_i \text{ and } v_j \\ 0 & \text{otherwise} \end{cases}$$

The adjacency matrix for a weighted graph is defined analogously except that $\mathcal{A} \in \mathbb{R}^{|V|\times|V|}$ where a_{ij} equals the weight of the edge between v_i and v_j, instead of 1, the weight of edge $\{v_i, v_j\}$ is the value of i_{ij}.

Notice that the adjacency matrix of an undirected graph is *symmetric*. Analogously, a matrix representing incidence is defined.

Definition 3.8 (Incidence matrix.) *An* incidence matrix $\mathcal{I} \in \mathbb{N}^{|V|\times|E|}$ *for an unweighted graph G is a matrix with*

$$i_{ij} := \begin{cases} 1 & \text{if vertex } v_i \in e_j \text{ for edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

The incidence matrix for a weighted graph is defined analogously except that $\mathcal{I} \in \mathbb{R}^{|V|\times|E|}$. Instead of 1, the weight of edge e_j is the value of i_{ij} for vertex v_i.

Definition 3.9 (Degree matrix.) *The* degree matrix $\mathcal{D} \in \mathbb{N}^{|V|\times|V|}$ *for an unweighted or weighted graph is a diagonal matrix with*

$$d_{ii} := \deg(v_i)$$

Example 3.3 *The matrices for graph G of Figure 3.1 are the following:*

$$\underbrace{\begin{pmatrix} 0&1&0&0&0&0&0\\ 1&0&1&0&0&0&0\\ 0&1&0&1&0&0&0\\ 0&0&1&0&1&0&1\\ 0&0&0&1&0&1&0\\ 0&0&0&0&1&0&1\\ 0&0&0&1&0&1&0 \end{pmatrix}}_{\mathcal{A}_G} \underbrace{\begin{pmatrix} 1&0&0&0&0&0&0\\ 1&1&0&0&0&0&0\\ 0&1&1&0&0&0&0\\ 0&0&1&1&0&0&1\\ 0&0&0&1&1&0&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&1&1 \end{pmatrix}}_{\mathcal{I}_G} \underbrace{\begin{pmatrix} 1&&&&&&\\ &2&&&0&&\\ &&2&&&&\\ &&&3&&&\\ &&&&2&&\\ &0&&&&2&\\ &&&&&&2 \end{pmatrix}}_{\mathcal{D}_G}$$

These matrices are the tools to describe the "inner-graph" structure, the general structure of a graph is captured. The next section defines particular substructures appearing in graphs.

3.1.4 Structures in Graphs

Different substructures are identified in graphs. The most general are subgraphs.

Definition 3.10 (Subgraph.) *Let $G' = (V', E')$ be a graph. G' is a subgraph of G if $V' \subset V$ and $E' \subset E$. This relation is written as $G' \subset G$.*

Notice that the "borders" of a subgraph to its supergraph G are "materialized" by vertices, not by edges. Therefore, the case that an edge $\{v, w\}$ is part of the subgraph, but either v or w is not part of the subgraph cannot happen.

Definition 3.11 (Walk, path, cycle.) *A* walk *P in a graph $G = (V, E)$ is an ordered list of vertices $P = [p_0, \ldots, p_k]$ where $p_i \in V$ for all $i \in P$ and $\{p_i, p_{i+1}\} \in E$ for all $0 \leq i < k$. The length of P is k. If $p_0 = p_k$ the walk is* closed. *A walk P of length k is called a* path *of length k if no vertex is contained twice in P, i.e. $p_i \neq p_j \forall i \neq j \leq k$. If $P = \{p_0, \ldots, p_k\}$ is a path, then $Pp_0 = \{p_0, \ldots, p_k, p_0\}$ is called a* cycle.

Walks, paths and cycles can also be defined in terms of subgraphs of G, i.e. walks, paths and cycles in G are considered to be subgraphs of G. It is said that there is a path P *from* p_0 *to* p_k and noted with $p_0 P p_k$. The following definitions are used to characterize graphs by means of connectivity, i.e. to answer the questions whether in a graph is a single "component" or there are subgraphs that are not connected by any path and how easy is it to split a graph by removing vertices and/or edges.

Definition 3.12 (Connected graph.) *A non-empty graph $G = (V, E)$ is called* connected *if all pairs of vertices in V are connected by a path in G, otherwise G is called* disconnected.

A fully connected graph or complete graph is a graph where all vertices are connected by a path of length 1.

Definition 3.13 (Connected component.) *For a graph $G = (V, E)$ a maximal connected subgraph of G is called a* connected component*.*

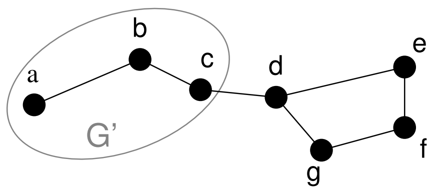


Figure 3.4: A subgraph

Example 3.4 *The graph of Figure 3.4 is connected, $G' = (\{a, b, c\}, \{\{a, b\}, \{b, c\})$ is a subgraph of G. The only connected component of G is G itself. $[a, b, c, b]$ is a walk, but not a path, $[b, c, d, e]$ is a path and $[d, e, f, g, d]$ is a cycle.*

"Maximal connected" means that all vertices that are connected to a vertex of the subgraph by a path are contained in the subgraph. Connected graphs can get disconnected by removing particular vertices or edges. These vertices and edges carry special names.

Definition 3.14 (Separator, cutvertex, bridge.) *Let $A, B \subset V$ and $X \subset V \cup E$ so that every path in G starting in A and ending in B contains a vertex or edge in X. Then X* separates *the sets A and B in G. X* separates *G if $G - X$ is disconnected. A separating set of vertices is called* separator*. A vertex that separates two vertices of the same component is a* cutvertex*, an edge with the same characteristics is called a* bridge*.*

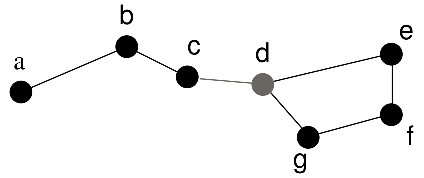


Figure 3.5: Separators, cutvertices and bridges

Intuitively, the meaning of $G - X$ for a graph $G = (V_1, E_1)$ and a subgraph $X = (V_2, E_2) \subset G$ is clear, the operation is formally defined as $G - X := (V_1 \setminus V_2, E_1 \setminus E_2)$.

Example 3.5 *For example, the vertex d is a cutvertex (as well as b and c), the edge $\{c, d\}$ is a bridge (as well as the two edges on the left hand side). The set consisting of the vertex d is a separator.*

These terms can be used to depict what happens during spectral clustering. The following operation is fundamental for the clustering algorithm presented in Section 6.

Definition 3.15 (Contraction.) *Let $G = (V, E)$ be a graph, $e = (v, w) \in E$ an edge of G. Then G/e is the graph that results from* contracting *the edge e into a new vertex v_e that "inherits" all edges incident to v, w, i.e. $G/e = (V', E')$ with*

$$V' := (V \setminus \{v, w\}) \cup \{v_e\}, v_e \notin V \cup E \text{ and}$$
$$E' := \{\{x, y\} \in E \mid \{x, y\} \cap \{v, w\} = \emptyset\} \cup \{v_e y \mid \{v, y\} \in E \setminus \{e\} \text{ or } \{w, y\} \in E \setminus \{e\}\}$$

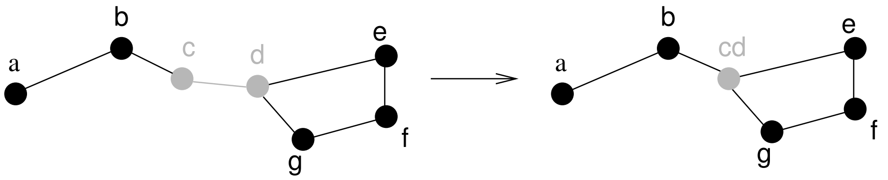


Figure 3.6: Contraction of edge $\{c, d\}$ yields the new vertex cd

3.1.5 Similarity Graphs

In the following, a particular semantics is given to graphs, i.e. it is defined what it means that there is an edge between two vertices. Here, a vertex stands for some data object. We are interested in comparing different data objects and express quantitatively how similar or different data objects are.

Definition 3.16 (Similarity graph.) *A* similarity graph *is a weighted connected graph $G = (V, E)$ where the weight of an edge $\{v_i, v_j\}$ expresses the similarity between the vertices v_i and v_j.*

According to different definitions, there may be some positive threshold for weights of edges that regulates the existence of edges: if the similarity value does not pass the threshold, the corresponding edge is not added to the similarity graph. Another possibility would be just to add only the edges between a vertex and its k most similar neighbor vertices or just to add all edges.

Such graph structures are fundamental for a class of clustering methods called *spectral clustering*: the goal is to partition the graph in such a way that vertices representing similar objects belong to the same connected component and vertices that represent unsimilar objects belong to different connected components.

3.2 Linear Algebra Related to Graphs

This section serves only to fix some notations and recall some structures and results from linear algebra that are related to graphs. Notice that the purpose is not to give a complete introduction to linear algebra.

3.2.1 Some Notions for Special Matrices

The following definitions recapitulate some special matrices and fix the notations used in the following chapters.

Definition 3.17 (Identity matrix.) *The* identity matrix *of dimension $k \in \mathbb{N}$ is denoted by I_k, i.e.*

$$I_k = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

If the dimension k is clear from the context, the index k can be omitted.

Definition 3.18 (Permutation matrix.) *A* permutation matrix *E is an identity matrix with permuted rows.*

If a matrix M is multiplied by the right hand side with a permutation matrix E, i.e. ME, the resulting matrix has the same but permuted columns of M. Is E applied to M by the left hand side, i.e. EM, the resulting matrix has the same but permuted rows of M.

Definition 3.19 (Gram matrix.) *The* Gram matrix *G for some matrix $V = [v_1, \ldots, v_k]$ is defined as*

$$G = (g_{ij})_{1 \leq i,j \leq k} := v_i^T v_j$$

where $v_i^T v_j$ is the inner product.

3.2.2 Norms and Metrics

Several different *(vector) norms* occur in the following chapters. Therefore, the exact definition is given here as well as the most important examples of norms.

Definition 3.20 (Norm.) *A* norm *over a vector space V is a function $|| \; || :\to \mathbb{R}^+$ onto the nonnegative real numbers with the following statements holding true for all vectors x, y and scalars α:*

- $x = 0 \Rightarrow ||x|| = 0$ *(definite)*
- $||\alpha \cdot x|| = \alpha \cdot ||x||$ *(homogeneity)*
- $||x + y|| \leq ||x|| + ||y||$ *(triangle inequality)*

Metrics that are induced by particular norms provide the ability to measure distances. Similarities can be seen as the counterpart to distances.

Definition 3.21 *$[L_1, L_2, L_\infty.]$ The following norms are examples of the so-called p-norms. Let $x = (x_1, \ldots, x_n) \in V$, V vector space.*

Manhattan norm. $L_1(x) := \sum_{i=1}^{n} x_i$

Euclidean norm. $L_2(x) := \sqrt{\sum_{i=1}^{n} x^2}$

Maximum norm. $L_\infty(x) := max_{i \leq n} x_i$

p-norm. $L_p := \sqrt[p]{\sum_{i=1}^{n} x^p}$

Definition 3.22 (Metric.) *Let X be an arbitrary set. Then a* metric *is a function $d : X \times X \rightarrow \mathbb{R}$ with the following properties: $\forall x, y, z \in X$ it holds*

- $d(x, x) = 0$ *(reflexivity)*
- $d(x, y) = 0 \rightarrow x = y$
- $d(x, y) = d(y, x)$ *(symmetry)*
- $d(x, y) \leq d(x, z) + d(z, y)$ *(triangle inequality)*

Popular metrics are the *Euclidean metric* and the *Manhattan distance* which are induced by the corresponding norms from Definition 3.21, e.g. for two points $x, y \in X, d_2(x, y) = ||x - y||_2$ is the Euclidean distance between x and y.

3.2.3 Tools for Spectral Clustering

Graph Laplacians

The term *graph Laplacians* is quite general. It merely denotes a particular matrix having some special properties that is useful for spectral clustering.

Definition 3.23 (Unnormalized Laplacian.) *An* unnormalized graph Laplacian matrix *for an undirected, weighted graph G is defined as*

$$L := \mathcal{D} - \mathcal{A},$$

where $\mathcal{D}$ is the degree matrix and $\mathcal{A}$ the adjacency matrix of G.

This Laplacian matrix of a graph G can be used for some reasoning on the corresponding graph G. For example, it can be shown that the multiplicity m of the eigenvalue 0 is exactly the number of connected components of the graph G.

Another possible graph Laplacian is the Gram matrix as shown later. The idea is to define similarities only implicitly in terms of minimizing a cost function that depends on the composition of groups of vertices. This optimization problem is equivalent to another optimization problem involving the Gram matrix. More on spectral clustering is given in Section 4.1.3.

Orthogonalization

Often, it is desirable to orthogonalize a set of vectors, i.e. to adjust them in such a way that they are pairwise orthogonal.

Definition 3.24 (Eigenvector, eigenvalue.) *Let A be a $n \times m$ matrix. The solutions of the equation $Ax = \lambda x$, where x is a m-dimensional vector and λ a scalar, are called* eigenvectors *and* eigenvalues.

Assume a set of vectors. After orthogonalizing them, the vectors are represented by vectors that are orthogonal to each other. One popular orthogonalization technique is the (pivoted) QR decomposition. This method is used in Section 5.2.2 to assign papers that are represented by a matrix to clusters. Since QR decomposition is mentioned only in Section 5.2.2 and is not needed for various approaches presented in this work, we do not explain it in detail, but refer to standard literature about linear algebra, e.g. [Fis00].

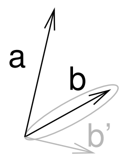


Figure 3.7: Vector b is orthogonalized against vector a to vector b'

CHAPTER

FOUR

Clustering Methods for Name Disambiguation

This chapter illuminates the relation between clustering and classification and name disambiguation and generally presents the methods that are applied to name disambiguation in the approaches of Chapter 5. After presenting how clustering and classification help to disambiguate names, the difference between clustering and classification is emphasized. Then the different methods are presented in a general way, whereas Section 4.3 illustrates a vector space model that is adjusted for the disambiguation of names occurring in citations. Finally, Section 4.4 presents similarity measures that allow to quantitatively compare strings and data objects represented as lists of tokens, i.e. as sets of strings.

Name disambiguation is relevant for many applications, as we argued in the introduction (cf. Section 1.2). It is essential not only for publication databases and digital libraries, but also interesting for applications referring people, or more generally named entities. Indeed the problem of name disambiguation can be generalized to disambiguate any kinds of names, e.g. of geographic locations as done in [ZWS+05, OMR06]. Information Retrieval tasks like gathering all available information concerning a person with name X from a set of documents or, even more challenging, from the Web, also involve name disambiguation: independently from the type of resource, e.g. text documents, Web pages or publication lists, there may exist homonymous persons, i.e. several individuals carrying the same name X.

On the one hand, relevant literature presents the task of name disambiguation in slightly different variants. The degree of difficulty varies, e.g. knowing *a priori* the number of individuals eases name disambiguation: a difficulty in data mining lies in guessing the correct number of clusters, so assuming to know it eliminates the first obstacle.

On the other hand, all definitions presented in Section 1.3 are equivalent with regard to the underlying problem: the task is to partition a set of citations into groups according to the individuals having authored these publications. More precisely, each group should contain only citations authored by a single person, i.e. all citations of a group have in common that they are authored by the same person standing behind an ambiguous name. More abstractly, each group should contain citations that are more similar to each other than to citations of other groups, where similarity means “authored by the same homonym”. As a consequence, clustering methods known from data mining are used to solve the problem.

Clustering is the task of *dividing* a set of data objects into (disjoint) subsets such that the similarity between data objects of the same group is high and inter-similarity between data objects belonging to different groups is low. These groups are called *clusters*. Their specific characteristics are usually not known a priori. Clustering methods are also called *unsupervised method*. Indeed clustering methods do not use any data for training.

Comparing data objects requires some comparable representation of data objects. Characteristic *features* represent data objects. Features can be "nominal", i.e. it is decidable if values are different but there does not exist any kind of distance or order, "ordinal", i.e. values are part of an order, but there does not exist any kind of distance between them, or metric, i.e. there is a well-defined distance between values that can be discrete or continuous. In the case of metric features, data objects are situated in a (metric) *feature space*, e.g. an Euclidean vector space. Then, similarity can be defined by a norm, in any case, some *similarity measure* or *distance measure* has to be defined. Some examples of norms are given in Section 3.2.2.

The quality of clustering strongly depends on the quality of features and of similarity measures. Therefore, features should be chosen wisely so that they are symptomatic for the different clusters.

Classification is the task of *assigning* data objects to *a priori* known classes according to some wisely chosen features. Classification is a kind of *supervised learning*. Such methods are also called *supervised methods*. Classifiers are *trained* using a data set containing objects that are already known to belong to the classes.

It is important to ensure that classifiers are not too specialized to the training data, since in such a case one risks that classifiers yield poor results when applied on real data. *Cross validation* as described in [SM00] is performed to avoid this so-called *overfitting*: the training data is divided into a *training set* used for training, and a *test set* used for checking the quality of the classifier and for verifying that the classifier is not overfitted. This (easiest) kind of cross validation, the *holdout method*, requires that the amount of available training data is large enough, since it holds that the smaller the amount of training data, the higher can be the variance when evaluating the classifier: results depend on the partition of the training data into training set and test set. This dependency should fade with enlargement of the data set used for training. Improvements of the holdout method overcome the problem of high variance in classifier evaluation. For example, according to the *k-fold cross validation*, training data is divided in k subsets. Then, k holdout cross validations are performed: each time one of the subsets serves as test set and the other $k-1$ subsets are used for training. In this way, k accuracy values are obtained. Taking the average yields in a balanced result that is more convenient for the evaluation of a classifier.

Clustering vs. classification. A comparison of clustering and classification reveals that classification promises higher accuracies than clustering. The advantage is that already known information can be exploited. At the same time, this can be a disadvantage of classification, since the required information is not always available. In [NM01] the authors showed in experiments that unsupervised methods can be as good as supervised methods. Admittedly, this is not a general result since only one particular supervised method, namely decision trees that are not further described here, is compared with one particular unsupervised method, namely an extended version of k-means. Both types of methods are equally scalable. Methods of both types can be used in (or adjusted to) situations where efficiency and scalability are important.

The approaches presented in the Chapters 5 and 6 justify the following sections whose purpose is to provide a general description and introduction of the methods that are applied in name disambiguation. The selection of unsupervised and supervised methods conforms with Chapter 5 presenting approaches to name disambiguation that are most often based on a vector space model. Therefore, this vector space model is also presented in detail. As already described, clustering without similarity measures, distance measures resp., is impossible, wherefore this chapter closes with a section devoted to measuring similarity.

4.1 Unsupervised Methods

4.1.1 K-means

Maybe the most popular clustering algorithm is k-means. Despite its simplicity, it yields quite good results in most cases. It was originally presented in [Mac67] by J. B. MacQueen in 1967. The fact that k-means is still widely used is due to its effectiveness and its simplicity. As a partitioning method, k-means searches iteratively an optimal partition of the data points starting from some (random) initial partition. This implies that the number k of clusters is fixed and known in advance. A partition is determined by k *cluster representatives*. Notice that the cluster representatives do not unambiguously define partitions, since a data point can be equally similar to two (or more) different cluster representatives. As the denomination implies, cluster representatives should *represent* their clusters. K-means uses *centroids* for this task. A centroid is the calculated central point of a cluster. These points are iteratively optimized by k-means to obtain an optimal partition of the data points. "Optimal" means that a *cost function* is minimized, more exactly the *sum-of-squares* cost function that calculates the sum of the square distances of each point to the (nearest) centroid for each cluster $\sum_{p \in C} dist(p, \mu_C)^2$, where C is a cluster, μ_C its centroid and $dist$ is a distance function, e.g. the Euclidean distance. Notice that due to the definition of the cost function, k-means finds only a local optimum.
For applying k-means, it is necessary that data objects are represented as data points in an Euclidean vector space. Furthermore, a *metric* to measure distance between points, e.g. Euclidean distance, is defined. Then, the following fixed point calculation is executed:

1. *Initialization.* Generate an initial partition consisting of k clusters and calculate the k centroids.

2. Reassign consecutively data points to their nearest centroid. If a data point changes its centroid, recalculate the affected centroids.

3. Repeat step 2 until centroids do not change any more.

In step 2., there are two possibilities of cluster assignment. *Hard clustering* follows the policy that a data point belongs to exactly one cluster at every point in time, whereas *soft clustering* assigns data points to multiple clusters (as defined in [HZG03]). In practice, k-means already has proved its usefulness. The main properties of k-means are:

- Advantages:
  - Computation of iterations is fast:$O(n)$
  - K-means converges fast: a small number (about 5 – 10) of iterations is sufficient.
- Disadvantages:
  - Results depend both on the initial partition and on the processing order of the data points.
  - K-means does not always converge to a global optimum, but returns locally optimal partitions.
  - The resulting clusters are quite compact (i.e. with low costs), so that less cohesive clusters are divided up in two or more. In general, clusters have a convex form. Hierarchies of clusters (as described in the next section) are not recognized.

4.1.2 Hierarchical Agglomerative Clustering (HAC)

Hierarchical agglomerative clustering is a well established clustering method which was first proposed by S. C. Johnson in 1967 [Joh67]. It tries to capture the *cluster hierarchy* and therefore overcomes a weakness of k-means. Figure 4.1 illustrates how clusters can be composed hierarchically due to different degrees of cohesion within clusters: the density of points in the data space varies.

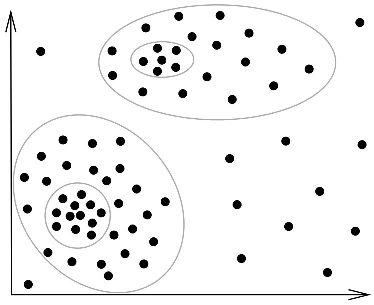


Figure 4.1: A hierarchical cluster structure

Given a set of data points $\mathcal{D}$, HAC computes a *hierarchy* of clusters, i.e. it allows to reconstruct different degrees of compactness in clusters: the denser a cluster is, i.e. the more similar its data points are, the earlier they are grouped together in a cluster. Agglomerative means that the clusters are constructed in a bottom-up way starting from each data point being the single element of its own cluster. The algorithm consists of the following steps:

1. Compute for all clusters pairwise a similarity value based on a similarity measure (distance function, resp.) $\delta(x, y),\ x, y \in \mathcal{D}$.
2. Merge the most similar pair of clusters to one bigger cluster.
3. Compute the similarity values between the new cluster and each of the remaining clusters.
4. Steps 2 and 3 are repeated until only one single cluster remains or another termination condition is fulfilled. Such a condition could be a threshold value that similarities have to exceed to allow that two clusters are merged.

The result of such a computation can be presented in form of a so-called *dendrogram* as shown in Figure 4.2.

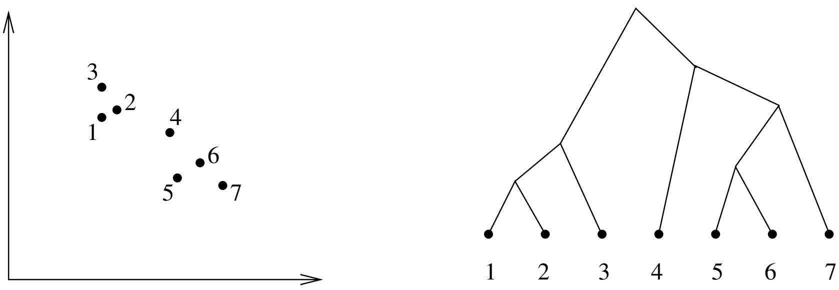


Figure 4.2: A dendrogram

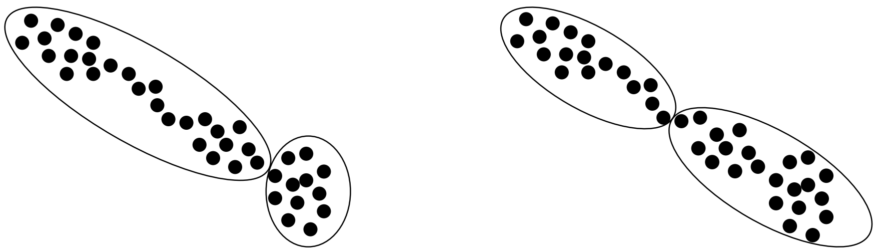
HAC requires that the similarity measure, by default a function having two data points as input, is extended to a function comparing clusters of data points. There are three possibilities for computing similarity values between clusters:

single link The similarity value between two clusters is equal to the maximal similarity value (i.e. minimal distance value) between a data point of the first cluster and a data point of the second cluster.

average link The average of similarity values between the data points of the first cluster and the data points of the second cluster is computed.

complete link The similarity value between two clusters is equal to the minimal similarity value (i.e. maximal distance value) between a data point of the first cluster and a data point of the second cluster.

Single link, complete link resp., have efficient implementations ($O(n^2)$), but lead to the single link effect, resp. complete link effect. These undesirable effects are illustrated by Figure 4.3.



(a) Single link effect (b) Complete link effect

Figure 4.3: Link effects

The problem of single link lies in the fact that two distinct clusters are "early" merged to one cluster if there is a chain of data points in between. Figure 4.3(a) shows a long cluster which should be two separate clusters. But they are connected by a chain of data points that are close together so that the two clusters are merged to one cluster.

The problem of complete link is also called "migration effect" in literature. As shown in Figure 4.3(b), it may happen that the data points are not clustered as it should be. Intuitively, borders of clusters are regions with very few or without any points. Using complete link facilitates the migration effect especially in "higher hierarchy levels" of clusters, i.e. near the root of a dendrogram: unwished partitions occur due to e.g. different shapes of clusters. Complete link does not measure only the distance between clusters, but also comprises the size and shape of clusters.

4.1.3 Spectral Clustering with QR Decomposition

Spectral clustering is a graph based approach, where a *similarity graph* (see Chapter 3), i.e. a graph should be clustered in which nodes represent data objects and weighted edges denote similarities between these data objects. Graph clustering means that the vertices are grouped to clusters. A good clustering maximizes weights of edges between nodes of the same cluster and minimizes weights of inter-cluster edges. The graph is represented by a so-called *graph Laplacian* during the computation. This matrix can be considered as a similarity matrix. An example of a similarity graph and a Laplacian is given in Chapter 3.1.5. Clustering is achieved by using the spectrum of the similarity matrix. Simple methods just use k-means as a part of the spectral clustering algorithm. Methods then differ from each other due to miscellaneous constructions of similarity graphs and graph Laplacians. A good introduction to such methods is given in [vL06].

The spectral clustering method applied to name disambiguation in [HZG05] is differing in the fact that k-means is not used. The Gram matrix is used as Laplacian. The cluster assignment is realized by a pivoted QR decomposition. Since the method does not yield the best results in name disambiguation and seems to be a quite special version, spectral clustering is not further described in this chapter. The clustering method presented in [HZG05] is described in more details in the next chapter.

4.2 Supervised Methods

Supervised clustering methods have in common that they need some training data, i.e. data points of which we know the class memberships. As a consequence, we know about the number and kind of existing classes. Intuitively, training of a classifier builds the borders between the classes. The classifier then assigns data objects to classes according to the learned borders. The in the following presented methods are selected for description here because they are the supervised methods applied to name disambiguation in Chapter 5. They belong to two different categories of supervised methods. The first type is based on stochastics. Such classifiers use probability distributions on training data to depict data objects leading to a stochastic model allowing to predict class memberships. In contrast, the second method considers data objects as elements in a vector space and tries to divide them into (two) classes by constructing a separating hyperplane between the classes.

4.2.1 Naive Bayes Model (NBM)

Naive Bayes is a quite simple but effective stochastic classifier. As the name suggests, such classifiers exploit Bayes' rule to construct a model that is used to predict classes of "new", yet unknown data objects..

Bayes' Theorem

First, some basic definitions and notions of stochastics are given. We restrict this introduction to stochastics to the indispensable part of basics. In the following, let $O = o_1, \ldots, o_N$ be a set of data objects having different attributes $A_1, \ldots, A_M$ consisting of different possible features, e.g. $A_i = \{a_1^{(i)}, \ldots, a_{m(i)}^{(i)}\}$, where $m(i)$ is the number of possible values for the attribute A_i. Then $X = \cup_{i=1}^{M} A_i$ can be considered as a set of possible results. The basic structure that is needed is the *probability space*. It is necessary to define *probabilities*.

Definition 4.1 (Probability space.) *A triple* $(\Omega, \mathcal{A}, P)$ *with*

- $\Omega \neq \emptyset$ *a set of results*
- $\mathcal{A} \subset \mathcal{P}(\Omega)$ *set of events with*
  $\emptyset \in \mathcal{A}$
  $A \in \mathcal{A} \Rightarrow \overline{A} \in \mathcal{A}$
  $A_k \in \mathcal{A}, k \in \mathbb{N} \Rightarrow \cup_{k=1}^{\infty} A_k \in \mathcal{A}$
- $P : \mathcal{A} \to [0,1]$ *probability, where*
  $P(\mathcal{A}) = 1$
  $P(\cup_k A_k^{\infty}) = \sum_k^{\infty} P(A_k) \forall A_k$ *pairwise disjoint*

is called a probability space.

Definition 4.2 (Stochastic event.) *A* stochastic event S *is a subset of the set of possible results, more exactly* $S \in \mathcal{A}$. *Here,*

$$S \subset \cup_i A_i$$

.

Definition 4.3 (Probability.) *Given a set X of possible results, the probability $P(S)$ of an event S is defined as*

$$\frac{|S|}{|X|}$$

In the situation of the real world that one wants to model with probabilities one encounters *conditional probabilities*.

Definition 4.4 (Conditional probability.) *Let A, B be arbitrary distinct events with associated probabilities $P(A)$ and $P(B)$. Then $P(A|B)$ denotes the probability of event A, given that event B occurs.*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Thomas Bayes published 1763 in [Bay63] the following theorem that is very important for practical applications.

Theorem 4.5 (Bayes' theorem.) *For events A, B it holds*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Proof.

$$P(A) \cdot P(B|A) = P(B \cap A) = P(A|B) \cdot P(B)$$

□

Another important term for calculating with probabilities is the *statistical independency*.

Definition 4.6 (Statistical independency.) *Two events A, B are* (statistically) independent *iff*

$$P(A \cap B) = P(A) \cdot P(B)$$

Conclusion 4.7 *For two statistically independent events A, B it holds*

$$P(A \wedge B) \equiv P(A, B) = P(A) \cdot P(B)$$

Naive Bayes Classifier

Assume a set of data objects that are represented in form of feature vectors. A *Bayes classifier* decides about belongings to classes based on the probabilities that data objects belong to classes. For a given feature vector $F = (f_1, \ldots, f_m), m \in \mathbb{N}$ and a class $C_i \in C = \{C_1, \ldots, C_k\}$, the probability that F belongs to C_i is computed by

$$P(C_i|F) = \frac{P(F|C_i) \cdot P(C_i)}{P(F)} = \frac{P(F|C_i) \cdot P(C_i)}{\sum_{C_j \in C} P(C_j) \cdot P(F|C_j)}$$

The most probable class is then chosen as class of F.

Naivety. For a large m, i.e. for high-dimensional feature spaces, it is difficult to estimate the probabilities $P(F|C_i) = P(f_1, \ldots, f_m|C_i)$ and $P(C_i|F) = P(C_i|f_1, \ldots, f_m)$. The *assumption of conditional independency* ignores statistical dependency of features. This eases the estimation of probabilities and at the same time, it does not disable the classifier. Finally, the posterior probability can be estimated as

$$P(C_i|f_1, \ldots, f_m) = \frac{P(C_i) \cdot \Pi_l(P(f_l|C_i)}{\sum_j \Pi_l P(f_l|C_j)}$$

where the denominator is constant for all classes. Therefore, after the maximum-likelihood principle the class with the maximal numerator is chosen as the class of F.

Mixture Models

In a *mixture model*, several (independent) variables are measured and composed to an ensemble. In combination with the expectation maximation algorithm (see Section 5.2.5, it is more convenient to consider mixture models to be a sum of parameterized functions with the purpose to model a probability density function. Intuitively, the term "mixture model" just emphasizes that a probabilistic model is composed by several probabilities.

4.2.2 Support Vector Machines (SVM)

The idea of *support vector machines* was first published by Vladimir N. Vapnik et al. in [VL63] in 1963, a training algorithm is given in [BGV92]. Let $\{(x_1, c_1), \ldots, (x_n, c_n)\}$ be a training set, where $x_i \in \mathbb{R}^n$ are data points and $c_i \in \{-1, 1\}$ indicates to which of two possible classes the point x_i belongs. Based on this training set, one can construct a $(n-1)$-hyperplane in $\mathbb{R}^n$ that separates the n data points such that points x_i with $c_i = 1$ are situated on one side of the hyperplane and the other data points are located on the other side of the hyperplane which is then called a *linear classifier*. Generally, there exist many such hyperplanes, the one of interest separates the data points with a margin as large as possible and is called *maximum margin hyperplane*.

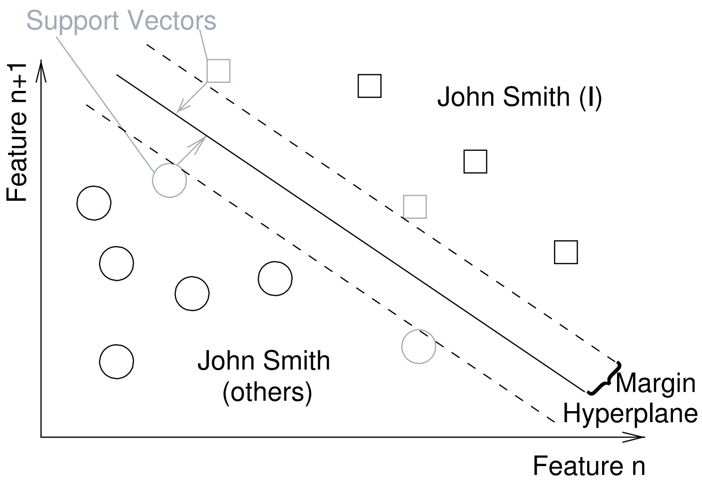


Figure 4.4: A maximum margin hyperplane

Obviously, such a maximum margin hyperplane does not always exist, since data points are not always linearly separable. In such cases or to enlarge the margin, one can extend maximum margins to soft margins that allow falsely labeled training data in the limits of the so called soft margin (entailing penalties). The task of finding a soft margin hyperplane is then an optimization problem which is solved with *Lagrange multipliers*.

Another approach to separate non-linearly separable data points is to apply the *kernel trick* first proposed by M. Aizerman et al. in [ABR64] (1964) to maximum margin hyperplanes as proposed by Boser, Guyon and Vapnik in [BGV92] in 1992: so called *kernel functions* non-linearly map the data space to some higher-dimensional space in which the data points are linearly separable, so that a linear classifier can be applied.

4.3 A Vector Space Model for Name Disambiguation

The vector space model that often is used as basis in name disambiguation originates from information retrieval. Salton et al. established a so-called vector space model that allows to compute similarities between documents as well as between documents and queries (which should be answered by retrieving relevant documents). It became the standard model for this task and allows the comparison of different documents, queries, summaries etc., more generally, the comparison of pieces of text. This model is based on mathematics and formally defined. It was developed for a specific purpose: given a query, retrieve all documents related to that query. Although having this concrete application in mind, the inventors of the vector space model created a model that can be applied for solving various tasks. A good explanation of this original version can be found in [Sal89].

In [BB98, BB00], Bagga et al. transfered the vector space model from information retrieval to the problem of cross-document coreferencing. Here, the goal is to identify identical entities occurring in different documents. The vector space model is also used the other way round, to disambiguate homonyms (i.e. people having the same name) occurring in several documents. Since this problem is far related to the name disambiguation problem examined in this work (see Chapter 2), many approaches to name disambiguation also are based on the vector space model. Therefore, it is presented here in detail in the version for name disambiguation as defined in Chapter 1.

The main difference is that we do not process documents and queries or text summaries, but citations. The term *citation* denotes here the information describing a scientific publication, given when citing publications, i.e. author names, title, publication venue title of the publication and maybe other information as year, editors etc. Plenty of examples for citations can be found in the bibliography given in B.2. For a set C of citations , we first need to build a *term set* consisting of terms used to characterize citations (e.g. keywords of paper titles). Assuming that the term set consists of t terms, citations are represented as *term vectors* of the form $c_i = (a_{i1}, \ldots, a_{it})$, where a_{ik} is the value of term k in citation c_i, e.g. one could set

$$a_{ik} = \begin{cases} 1 & \text{if term } k \text{ appears within citation } c_i \\ 0 & \text{else} \end{cases}$$

In information retrieval, one uses numeric values depending on the importance of the term in the document. It also makes sense to weight terms according to their importance in citations. It is then useful to use normalized factors, i.e. to take values in the interval $[0, 1]$.

Each term of the t terms of the term set can be represented by some vector $T_j, 1 \leq j \leq t$. If all T_j are linearly independent, they span a t-vector space. Therefore, each citation c_i can be written as the linear combination

$$c_r = \sum_{i=1}^{t} a_{ri} T_i$$

The important thing which is still missing and which is our actual goal, is a similarity measure that compares two citations. An example for a similarity measure is presented here, more are presented in the next section (Section 4.4.1).

The similarity between two vectors x, y can be measured by the *dot product*

$$x \cdot y = |x||y| \cos \alpha,$$

where $|x|$ is the length of x and α is the angle between x and y.

Therefore, we compute the similarity for two citations c_r, c_s as follows:

$$c_r \cdot c_s = \sum_{i,j=1}^{t} a_{ri} a_{sj} \underbrace{T_i \cdot T_j}_{\text{term correlations}}$$

Since term correlations often are not available, the standard solution is to set

$$T_i \cdot T_j = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

which means that different terms are not correlated. Although this assumption does not hold in reality in most cases, it has not a too strong influence on the accuracy of clustering. The computation of the similarity between two citations is then simplified to

$$c_r \cdot c_s = \sum_{i,j=1}^{t} a_{ri} a_{sj}$$

4.4 Comparing Citations: Similarity Measures and Weighting Factors

4.4.1 Typical Vector Similarity Measures

The following list of similarity measures is extracted from [Sal89], p. 318.

Every similarity measure on vectors can be represented by two different formulas, depending on the values of the vectors. If term vectors have only unweighted values (i.e. values are either 1 or 0), one can compute similarities in terms of operations on binary numbers, otherwise the common operations on real numbers are used. Since the latter is more widely used and the more general one, the following similarity measures are formulated for weighted term vectors only.

Assume a set $\mathcal{T}$ of t terms as in the previous section and a weighting function $\omega : \mathcal{T} \to \mathbb{R}$. Let $x, y \in \mathbb{R}^t$ be weighted term vectors. Then, the following similarity measures are defined:

Inner Product

$$\sum_{i=1}^{t} x_i y_i$$

Dice Coefficient

$$\frac{2 \sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2}$$

Cosine Coefficient

$$\frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} x_i^2 \cdot \sum_{i=1}^{t} y_i^2}}$$

Jaccard Coefficient

$$\frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i y_i}$$

Notice that the inner product is unbounded, whereas the other similarity measures are normalized so that similarity values range in $[0, 1]$. Normalization allows to use thresholds according to the chosen similarity measure and independently of the datasets.

4.4.2 About String Distance Metrics

The following distance measures are not used in this work, but are important for name matching tasks. They rather appear in the disambiguation of synonyms than in the disambiguation of homonyms. They are compared in [CRF03]. Cohen et al. classify them into four groups: *edit-distances*, *token-based distance functions*, *hybrid distance functions* and *blocking methods*. For example, let the *Levenstein distance* be mentioned as a representative of edit-distances. Token-based distances measure the distances between sets of terms. Hybrid distances combine recursively two (or more) distance functions. Blocking methods try to quickly group terms so that fewer terms have to be compared to each other. Since these functions are not needed explicitly by the following approaches, we refer to the paper [CRF03] for detailed descriptions.

4.4.3 Weighting Factors

The idea is that features discriminate citations differently well, i.e. some features are very characteristic for few citations, other features are not very characteristic since they appear very frequently. In such a case, the missing of a feature would be more characteristic than the existence of a feature. Weighting factors, so-called *feature weights* are designed to give very characteristic features higher weights, i.e. more influence, and less characteristic features lower weights.

Term Frequency.Inverse Document Frequency

We describe the weight tf.idf generally for weighting terms, not only in the sense of features. The dot in the name symbolizes the multiplication that combines the two parts *term frequency* and *inverse document frequency*. Let D be the set of documents. In the case of name disambiguation in citations, the considered documents are the citations. Let t_i be a term of the set T of terms that contains N terms. Let n_i be the number of occurrences of t_i. Then

$$\text{tf}_{d_j}(t_i) = \frac{n_i}{\sum_{j=1}^{N} n_j}$$

is the *term frequency* of term t_i. The *inverse document frequency* is defined as

$$\text{idf}_T(t_i) = \log \frac{|D|}{\text{df}_{d_i}(t_i)},$$

where $\text{df}_{d_i}(t_i)$ is the frequency of t_i in document d_i. Then, tf.idf is the product

$$\text{tf.idf}_{d_i}(t_i) = \text{tf} \cdot \text{idf}$$

This weight proved to be robust. The intuition is that a high "inner-document" frequency of a term expresses that the term is very relevant for the particular document, whereas a high document frequency of a term expresses that a term is not characteristic for a particular document. That is why the inverse document frequency is taken. The logarithm assures that the idf part does not dominate the tf part. More explanation and a theoretical background especially for idf is given in [Rob04].

Normalized Term Frequency

With the same notations as above, ntf is defined as

$$\text{ntf}_{d_j}(t_i) = \frac{\text{tf}_{d_j}(t_i)}{\max_k \text{tf}_{d_j}(t_k)}$$

Log Entropy

This weight is not tested in any approach presented in this work, but some authors propose to use it since it might function better than tf.idf. Therefore, we present it here. Once again, the above notations are used. Then the *log entropy* of a term t_i is

$$\mathrm{le}(t_i) = \sum_{j=1}^{N} \frac{\frac{\mathrm{tf}_{d_j}(t_i)}{\mathrm{df}_T(t_i)} \cdot \log \frac{\mathrm{tf}_{d_j}(t_i)}{\mathrm{df}_T(t_i)}}{\log N}$$

More on weighting factors is written in [CK99]. It is difficult to evaluate weights theoretically. Most insights in the quality of weights are obtained by conducting experiments that compare different weights on some sample data. It can be said that there is not one weighting function that is the best for all purposes. One has to decide from case to case which weight should be used. The approaches of the next chapter mostly use tf.idfsince it is a very common (if not the most common) and well-established weighting measure.

Part III

Approaches to Name Disambiguation

CHAPTER

FIVE

Overview of Approaches to Name Disambiguation

The goal of this chapter is to provide a comprehensive overview of the approaches to name disambiguation described in literature. The main focus lies on disambiguating homonyms in citations occurring in scientific publications. In the following, the term *citation* and *publication* are used equivalently since during the task of name disambiguation as defined in Section 1.3 only the information as given in citations is available. In addition to to detailed descriptions of different methods, the results achieved in experiments are reported. Most of the presented approaches are evaluated by using test collections extracted from DBLP[1] as well as by using datasets collected from the Web (e.g. from publication lists found on personal homepages). Such test collections are datasets that usually have the form of sets of strings, where each string represents one citation. These datasets are provided in files where each line contains one string, i.e. one citation. Each dataset contains citations with one ambiguous name, e.g. all citations in a file are published by some homonymous author called "A Gupta". Therefore, these datasets are also called "name datasets". Each string representation of a citation consists of the ground truth, i.e. a cluster id that tag the citation as originating from the same author as all other citations with the same cluster id. A citation id (i.e. a "serial number") and finally the, possibly preprocessed, information used for clustering follow. For example, a working group at Pennsylvania State University provides name datasets that contain citations of the following form[2]:

```
10_1 A Gupta; Bjorn Kvande ; Irwin B Levinstein ; Kurt Maly ; Margrethe H
    Olson ; Ravi Mukkamala ; Rita Chambers ; Roy Whitney ; S Nanjangud<>PMES:
    privilege mangagement and enforcement system for secure distributed
    resource sharing <>IFIP International Federation for Information
    Processing World Conference IT Tools
```

The first section of this chapter recalls the connection between name disambiguation and clustering methods. Furthermore, it recapitulates the aspects to consider when designing a clustering method for name disambiguation, i.e. the related facets and tools that reappear in the following sections.

Clustering methods are commonly divided into *unsupervised method* and *supervised clustering methods*. We consider this common classification of clustering methods to be the most natural. Therefore, this classification is reflected by the structure of the remaining chapter. Each method is described in detail in the Sections 4.1 and 4.2. The experiments carried out and the obtained evaluation results are included. A final discussion of the depicted approaches concludes this chapter.

[1]http://www.informatik.uni-trier.de/ ley/db/
[2]http://clgiles.ist.psu.edu/data/nameset_ author-disamb.tar.zip

5.1 Applying Clustering to Name Disambiguation

We first remember that name disambiguation is a clustering problem. Independently from use cases that need name disambiguation, the matter is the following. It is desired to identify the individuals standing behind different occurrences of names, so-called *entity names*, in some kind of documents or text fragments, e.g. citations, web pages etc. Such ambiguous entity names are called *homonyms* in contrast to *synonyms* that are different names representing a single entity. Use cases are e.g. the search in digital libraries, evaluation of researchers by means of publication impacts or price comparisons of the same products available in many different online shops as sketched in the introduction in Chapter 1. Abstraction from such use cases leads to a reformulation of the problem. The task is to group occurrences of (same) entity names such that each group contains only (and all) occurrences that refer to a single entity.

Additionally, it can be useful for an application to extract relevant information about individuals and output it group-wise for each named entity leading to a kind of summaries. Notice that named entities are not restricted to persons, but this case is very popular in research and economically relevant as was described in the introduction (see Chapter 1).

Equally important is the inverse phenomenon that a single entity is referred to by several names or variations of names. For example, to guarantee that *all* relevant information that is available for some entity is retrieved from a given set of resources, it is necessary to assure first that the particular entity is represented consistently by a single form and name, or to have the knowledge about the different name variations. Furthermore, these so-called *synonyms* may induce “new” cases of homonyms. Therefore and since name disambiguation is indeed an interesting research topic, several groups of researchers already developed a diversity of approaches.

This thesis concentrates on the problem of disambiguating homonyms in scientific publication databases, so that the given overview comprises approaches concerning homonymous author names. Approaches to disambiguation of synonyms and other, more distant topics, are sketched as related work in Chapter 2.

Each approach to name disambiguation comprises several aspects. First of all, an established clustering algorithm is chosen on which the new method is based. Maybe the most important decision is to select some *attributes* that discriminate the name occurrences in a way, that occurrences referring to the same individual have similar *representations* when reduced to the *features* extracted from the chosen attributes. Notice that in literature, the notions “attributes” and “features” are used interchangeably. According to the chosen clustering algorithm, an accurate representation of name occurrences is designed. Then, a concrete representation can be constructed by using the features extracted from the attributes. The third decision affecting the quality of the approach determines the *similarity measure*, *distance function* resp., that allows a quantitative comparison of distinct name occurrences, i.e. defines numerically what it means that two given occurrences are similar to each other.

The remaining part of this section is devoted to commonly used feature types, similarity measures and also clustering evaluation methods.

5.1.1 Commonly Used Feature Types

There are different kinds of features that promise to improve clustering quality. Their usefulness depends on the quality in which they are provided. Notice that features occur in unstructured or semistructured text, e.g. keywords occurring in the title. As a consequence, it is necessary to carefully extract such features. Otherwise, used features may be enriched with errors. Since comparison of features is text-based, it is important that the words are extracted correctly. Furthermore, it is necessary that feature extraction, too, is automated. Obviously, due to the huge amount of real-world data, it is impossible to extract features manually.

Methods for Extracting Citation Features

Several feature types require techniques from information retrieval to extract the features from a given set of resources, citations here. For example, a desired feature type can be keywords occurring in publication titles. The expectation is that such keywords describe the author's research interests, at least partially.

However, such kind of feature types require some pre-processing of words. Stop words like "a", "the", "and" etc. are removed since they usually do not carry any semantic information and are frequently used, i.e. their occurrence is not a characteristic phenomenon.

Furthermore, the remaining words are stemmed. Word stems are the parts of words that remain when removing the changeable endings, e.g. to transform a verb into a noun or due to conjugation (even in English, there still exists the "s" of the third person singular) and tenses. For example, the word "estimation" is reduced to the stem "estimat" by a *stemmer*, in contrary to a *lemmatizer* that reduces it to the linguistically correct root "estimate". After all, the purpose of stems is to consider words as equivalent "modulo" *morphological variants*. A second advantage of stemming is that the amount of different words that have to be considered as possible features decreases, reducing the size of a dictionary or, as in the case of some of the following approaches, the dimension of the vector space in which clustering is performed. Popular stemmers are the Krovetz stemmer([Kro93]) and the Porter stemmer([vRRP80]).

A third pre-processing step can be a semantical clustering of words. Researchers already observed that authors usually do not use the same words in titles several times. We expect that this property also depends on research topics. At least, the overlapping is assumed to be relatively small. However, experts recognize titles that are related to each other. This is achieved thanks to additional knowledge about the domain. For instance, a philosopher probably would not recognize a connection between "NP-completeness" and "computational complexity theory". A clustering method without such additional knowledge cannot better associate titles with research topics or research areas than a philosopher, i.e. only by comparing keywords by means of string matching. The given example would require an ontology about computer science where all keywords appear in. Such an ontology does not exist at the moment. To find a generic way to produce one may be an interesting task of further research. An approach that does not go that far is to cluster title keywords according to their meaning to induce some additional knowledge. This is realized by different methods as described in Chapter 2, e.g. using WordNet[3].

In the following, the features used for name disambiguation are presented. In general, relevant literature focuses on three citation attributes that might serve well as feature types, namely coauthors in citations, title keywords and publication venue titles. Since this combination of attributes is the most common, we introduce the shorthand CoTiVe to refer to them more easily. A wise choice of features is essential for a good algorithm. The idea is to choose features that are characteristic to authors and their publication habits. For instance, the page numbers in the publication journal of a particular article surely is not influenced by the author. Therefore, page numbers are not indicators for the individual hiding behind an author name.

Coauthor Information

To determine if two citations are authored by one single author or two distinct authors, it is a straightforward approach to consider the coauthor names occurring together with the ambiguous name in the two citations. The idea is that a researcher is likely to collaborate together with a group of researchers that remains relatively stable for some time. More exactly, the fact that a researcher stops publishing with a particular group of researchers, probably means that the researcher's field of interests has changed rather strongly and/or that the researcher switched the working group, maybe even the affiliation, city, country and/or continent. Furthermore,

[3] http://wordnet.princeton.edu/

it is quite unlikely that two (or more) homonymous authors not knowing each other or even publishing together collaborate with the same authors. Notice that the case where two authors having the same name publish together is more than rare. Furthermore, such citations already indicate that the name is homonymous since it is occurring twice in each such citation. Less rare, but still not very probable is the situation that two homonymous authors operate on the same research field and work in the same community (in the sense of a group of people collaborating closely and publishing rather frequently together). Reformulating the situation the other way round, it is not very probable that a researcher collaborates with two distinct authors having the same name.

Single-author publications and also authors that published only once obviously represent an invincible problem to name disambiguation based on coauthor information. Another natural limitation is the fact that authors often collaborate with several disjunct or loosely related communities, e.g. due to personal development in the course of time. As a consequence, a single author may be (virtually) split into several persons.

Paper Titles

The idea is that it is possible to extract keywords from paper titles of citations that characterize the research area of the authors of citations. As described above, this feature type (as well as the one of publication venue titles) require some pre-processing to work well. First, it is better to use only word stems instead of the whole words. Second, a semantical clustering of words may improve the clustering quality. Obviously, syntactically different words that could not be brought into a syntactical relation, are then considered as belonging to the same class (containing words with similar meanings) and therefore are considered to be the same. On the one hand, this might help recognize similarities. On the other hand, words might be identified falsely as being equivalent which might introduce new errors during clustering. Another problem of titles is that they do not necessarily contain any keywords at all indicating the research area, e.g. "Is Hillary Rodham Clinton the President? Disambiguating Names across Documents" ([RK]) does not contain really specific keywords, and the word "president" even could indicate a topic related to politics. Furthermore, researchers' interests evolve in the course of time as well as collaborators.

Publication Venue Titles

Publication venues denote places of publications, e.g. conferences, proceedings etc. The titles of these events usually contain some keywords that allow experts to classify them according to research areas. Intuitively, publication venue titles will perform better than paper titles because they seem more stable (as argued in [HZG03]). Having semantical clustering in mind, it seems that publication venue titles constitute a kind of abstraction from research topics occurring in paper titles to research areas. However, the reported experiments do not support this. Especially, publication venue titles as feature type do not perform very well. Possible reasons for that unexpected phenomenon are given together with the relevant experiments.

In Chapter 6, additional promising feature types are presented and already mentioned feature types are examined in a different way.

5.1.2 Expressing Similarity

Based on the selection of feature types and the way of involving them to represent citations, the term of *similarity* in form of *similarity measures* is defined. The general purpose is to compare different citations and to be able to make statements about the similarity. The fact that two citations are similar to each other should express that they probably originate from the same person in question. Here, the term "person in question" denotes an author having the name that should be disambiguated. Often, it is useful to normalize similarities to values in the range of $[0, .., 1]$. In this case, similarities can also be considered as probabilities that two citations are generated by the same person. Depending on the approach, similarities are modelled differently. A detailed description of this topic is given in Chapter 4.

Cosine Similarity

A common way is to represent citations as vectors according to the vector space model (see Section 4.3) where each element of a vector is associated with a feature occurring in a citation of the given citation set. Then, similarity between two citation vectors is represented by the angle between these citation vectors.

Stochastic Models

An alternative approach directly models the probability that an author is the creator of a particular citation. Doing this, an author is represented by some prior probabilities based on a set of citations which is already accredited to that author. The precise proceeding is explained in the sections devoted to approaches based on Naive Bayes.

5.1.3 Interpreting Results

Assume a set of citations $c = \{c_1, \ldots, c_N\}$ produced by k distinct authors $A_1, \ldots, A_k$ having the ambiguous name n and a clustering C providing k clusters $C = \{C_1, \ldots, C_k\}$ representing k authors . For evaluation purposes, it is a good idea to construct the so-called *confusion matrix*.

Originally, the confusion matrix is used for evaluating classifiers, but if ground truth is available, it can be used to evaluate clusterings, too. Intuitively, the confusion matrix for a clustering indicates the number of times an author is assigned to a cluster. The rows of the matrix A are devoted to the real authors, whereas the columns represent the clusters, i.e. the "predicted" or "clustered" authors . Using the confusion matrix, it is easy to define different values that measure the quality of the clustering.

Definition 5.1 (Confusion matrix) *A* confusion matrix $A_c = (a_{ij})_{1 \leq i,j \leq k} \in \mathbb{N}^{k \times k}$ *on a citation set c is a matrix, where k is the total number of clusters and an element a_{ij} contains the absolute number of predictions of "Author i" as "Author j":*

$$a_{ij} = |\{c_l \in c \mid c_l \textit{ was produced by } A_i \textit{ and } \ c_l \in C_j\}|$$

Notice that the diagonal elements a_{ii} contain the number of correctly *clustered citations.*

Intuitively, *recall* specifies how many relevant citations out of all citations produced by a *particular author* were assigned to the corresponding cluster. Naturally, recall is given relative to an *author's number of publications*. The *total recall* of a clustering is then computed by taking the average of the recall values of the clusters.

Definition 5.2 (Recall) *Given a confusion matrix $A_c \in \mathbb{N}^{k \times k}$, the* recall *of class C_i, cluster C_i resp., produced by a classifier C, clustering C resp., is defined as the proportion of the number of citations correctly identified as belonging to C_i to the total number of citations assigned to C_i (i.e. the sum of the i^{th} row of A_c).*

$$\textit{recall}_c(C, i) = \frac{a_{ii}}{\sum_{j=1}^{k} a_{ij}}$$

The total recall of C is then obtained by taking the average

$$recall_c(C) = \frac{1}{k} \sum_{i=1}^{k} recall_c(C, i)$$

Intuitively, *precision* specifies how many relevant citations out of all citations contained in a *cluster* were really produced by the author associated with the cluster. Naturally, precision is given relative to the *size of a cluster*. The *total precision* of a clustering is then computed by taking the average of the precision values of the clusters.

Definition 5.3 (Precision) *Given a confusion matrix $A_c \in \mathbb{N}^{k \times k}$, the* precision *of class C_j, cluster C_j resp., produced by a classifier C, clustering C resp., is defined as the proportion of the number of citations correctly identified as belonging to C_j to the total number of citations assigned to C_j (i.e. the sum of the j^{th} column of A_c).*

$$prec_c(C, j) = \frac{a_{jj}}{\sum_{i=1}^{k} a_{ij}}$$

The total precision of C is then obtained by taking the average

$$prec_c(C) = \sum_{j=1}^{k} prec_c(C, j)$$

Intuitively, *(disambiguation) accuracy* expresses the total number of *correctly* assigned citations relative to the total number (N) of citations. These favorable assignments are situated on the trace of the confusion matrix, i.e. in the diagonal entries.

Definition 5.4 (Disambiguation accuracy (definition 1)) Disambiguation accuracy *is defined as the sum of the values of the diagonal elements divided by the sum of all elements in the matrix, i.e.*

$$acc_c(C) = \frac{\sum_{i=1}^{k} a_{ii}}{\sum_{i,j=1}^{k} a_{ij}}$$

The following definition of disambiguation accuracy is not based on the confusion matrix and was given in [TKL06].

Definition 5.5 (Disambiguation accuracy (definition 2)) *Suppose the classes are labeled from 1 to k. Let a_i and p_i be the actual and predicted classes of the ith citation. Let $s(x, y)$ be 1 if $x = y$, 0 otherwise. Then,*

$$acc = \max_{\pi \in S_k} \sum_{i=1}^{k} s(\pi(p_i), a_i),$$

where S_k is the set of permutations on the classes 1 to k.

In the rest of this chapter, we describe several approaches to name disambiguation based on unsupervised and supervised clustering.

5.2 Applying Unsupervised Clustering

5.2.1 Model-Based K-means

The authors Han et al. start their particular research on the topic of name disambiguation in scientific publication databases with the following approach published in [HZG03]. Their idea is to build a *stochastic model* of a set of citations to be able to compare citations. To conduct clustering based on this *naive Bayes model*, they pick up a variation of k*-means* as described by Hartigan and Wong in [HW79].

Usually, naive Bayes is deployed as a *classification* method (as described in Section 4.2, i.e. a supervised method requiring a dataset with ground truth available to train a classifier). Anyway, naive Bayes is here just used to *estimate probabilities* needed in a single step of an *iterative algorithm*, not to classify data objects (here: citations). Therefore, instead of training data with ground truth, the *actual partitioning* (i.e. the actual clusters), is used to (re)estimate probabilities in each iteration.

Problem definition

In the introduction in Chapter 1, several more or less different definitions of the problem of name disambiguation are given. This approach considers name disambiguation to be the task to cluster a set of citations published by authors having an ambiguous name so that each cluster represents one individual, and to output identity information such as research interests and coauthors. The number k of clusters is assumed to be known *a priori* since the authors prefer to concentrate on the evaluation of the model-based clustering algorithm. Nevertheless, guessing the correct number of clusters is also an interesting topic to investigate as the authors state.

Approach

The authors design a variation of k*-means* in which the distance function computing the distance between a cluster C and a citation c is defined to be the *probability*, that cluster C (that can be considered as a *"virtual entity"* since its citations are considered as belonging to one author) produces c. In this case, the distance function rather describes a similarity, since a high probability means a low distance.

The authors decide to model the probabilities with Naive Bayes since it is a simple but effective probability model and easy to extend.

Feature Types and Representation of Citations in Model-Based k-Means

The authors chose CoTiVe as set of feature types, i.e. coauthor names, publication titles and publication venue (i.e. journal, proceedings resp.) titles.

Citations are represented as probabilities. In the model, citations are not considered isolated from each other, but rather group-wise, where each group of citations is assumed to belong to a single homonymous author. As a consequence, citations rather contribute to depictions of authors than being depicted themselves. Representations are reduced to some few prior probabilities. By applying the Naive Bayes rule, the desired posterior probability of clusters standing for authors having produced particular citations can be calculated which is done in step 3 of the k-means algorithm.

Clustering Algorithm

In the following, first the model is described and then its application in a variant of the k-means algorithm.

Naive Bayes model (NBM). We already described the general method of Naive Bayes classification in Section 4.2. As already mentioned, this approach does not use naive Bayes as a *supervised method*, but for *unsupervised clustering*.

For estimation of the model parameters for each author, Han et al. use an author's past citations, i.e. the citations that are actually contained in the cluster representing an author. In order to determine the needed probabilities, all N citations $c_j \in \{c_1, \ldots, c_N\}$ are freed of the author name that should be disambiguated since it appears in every citation c_j. The model is used to calculate the function that finds the cluster $C_i \in \{C_1, \ldots, C_k\}$ whose corresponding author is most likely to have authored the citation c_j:

$$\max_i P(C_i|c_j)$$

This conditional probability is reformulated by application of the Bayes rule:

$$\max_i \frac{P(c_j|C_i) \cdot P(C_i)}{P(c_j)},$$

where $P(C_i) = \frac{|C_i|}{N}$ is the prior probability of cluster C_i considered as virtual entity producing papers, estimated by the ratio between the number of papers in C_i, $|C_i|$, and the total number of papers in the citation set, N. $P(c_j) = \frac{1}{N}$ denotes the probability of citation c_j. Since this probability is independent of the cluster C_i and it matters only for which cluster C_i the posterior probability $P(C_i|c_j)$ is maximized, $P(c_j)$ can be omitted to reduce the formula to

$$\max_i(P(c_j|C_i) \cdot P(C_i))$$

It remains to estimate $P(c_j|C_i)$. In contrast to the reality, Han et al. assume (as widely done) that the feature types are *statistically independent* and their elements (e.g. different coauthors, title keywords etc.) are not correlated, but independent from each other. This assumption is more exactly illuminated at the end of the paragraph. At this stage of the model, it is essential since it allows the estimation of $P(c_j|C_i)$ as

$$P(c_j|C_i) = \prod_{m=1}^{3} P(F_m^{(j)}|C_i) = \prod_{m=1}^{3} \prod_{n=1}^{j(m)} P(F_{mn}^{(j)}|C_i),$$

where $F_1^{(j)}$ denotes the feature of coauthor names in c_j, $F_2^{(j)}$ the feature of paper titles in c_j and $F_3^{(j)}$ the feature of venue titles in c_j. Each feature $F_m^{(j)}$ is split into the probabilities of the (assumed to be independent) values $F_{mn}^{(j)}, 1 \leq n \leq j(m)$ occurring in citation c_j. For example, a citation c_j may have $j(1)$ coauthors $F_{11}^{(j)}, \ldots, F_{1j(1)}^{(j)}$.

To resume the discussion of statistically independent feature types and values, notice that intuitively, in reality it is quite probable that an author publishes several times with the same coauthors (finally, that's what makes the coauthor information an effective feature type for name disambiguation) and since the research topic probably does not vary strongly, title keywords cannot be considered as independent from coauthors. Remember that the independence of feature types is a common assumption (seldomly holding true) utilized by naive Bayes since this violation often does not disturb the good performance of naive Bayes. For feature values, Han et al. admit that a violation of the assumption of independent feature values indeed may affect the classification accuracy as shown in [DP96, Fri97] by Friedman, Domingos and Pazzani.

k-means algorithm. The authors describe k-means more or less in the same way as we have done in Section 4.1.1, but with the already mentioned, more complicated, probability based distance function. Notice that the alternated distance function (probabilities instead of e.g. Euclidean distance) eliminates the requirement of a (Euclidean) vector space and replaces centroids by prior probabilities as cluster representatives. These important variations from the basic k-means algorithm presented in Section 4.1.1 justify a detailed description of the model-based version of the algorithm. The following presentation is a summary of the algorithm as presented by the authors.

Let $\{c_1, \ldots, c_N\}$ be the dataset of N citations produced by k distinct authors.

1. *Initialization.* Generate a random initial partition consisting of k initial clusters.

2. *Computation of prior probabilities for "virtual entities", i.e. recalculation of cluster representatives.* Estimate the *prior* probability of each cluster (aka "virtual entity") and the probability that a certain feature type is produced by the cluster, as well as the probabilities that the virtual entity collaborates with a researcher, uses certain keywords in the paper title, etc.

3. *Computation of distances (= posterior probabilities), i.e. cluster reassignment.* Compute the pairwise distances between each cluster $C_1, \ldots, C_k$ and each citation $\in \{c_1, \ldots, c_N\}$, i.e. the *posterior* probabilities that C_i produced $c_j, 1 \leq i \leq k, 1 \leq j \leq N$. Then assign c_j to the cluster C_{i_1} with the highest posterior probability $P(C_{i_1}|c_j)$ and also to the cluster C_{i_2} with the second-highest posterior probability if C_{i_2} has a probability $P(C_{i_2}|c_j) \leq 75\%$, i.e. the authors decided to use *soft clustering* (cf. Section 4.1.1) during iterative computation steps.

4. *Convergence.* The algorithm converges, if fewer than 1% of the citations change cluster assignments. Then, citations are finally reassigned to clusters following *hard clustering* (cf. 4.1.1) and for each cluster C_i a ranking of top publication title keywords, publication title keywords and coauthors of citations is reported (ranked by the probabilities that they were generated by C_i).
   Otherwise go to step 2 starting the next iteration.

Notice that in step 3, there is another departure from the basic k-means algorithm presented in Section 4.1.1, since the authors' algorithm performs soft clustering in iterations and does not recalculate the cluster representatives after each changing of a citation's cluster assignment, but first reassign all citations to the actual cluster representatives. The decision in favor of soft clustering and it's realization is not justified by the authors. We guess that this decision is based on empirical experiences.

The initial clusters can be considered as virtual entities that are "contaminated" with noise. The task of the algorithm is then to clean the clusters by repartitioning of the set of citations, so that after convergence clusters remain that are assumed to correspond to real entities.

Semantic clustering on keywords. A possible improvement of used features is to cluster title words (both from paper titles and from venue titles) *semantically*. The purpose of title words is to capture an author's research interests as well as to give hints on the author's writing style (in terms of preferred words for usage in titles). However, Han et al. observe that paper and venue title keywords are not completely available information, so that a semantic clustering of them promises improvement since authors may not reuse some keywords in the paper title, but use *semantically similar* words. Therefore, using semantically clustered keywords as features can identify different words belonging to the same word cluster as indicating the same or similar research fields. For this reason *recall* should be increased. But it is also possible, that two citations are erroneously considered to belong to the same author because their title keywords were made equal mistakenly. In this case, *precision* is reduced. At the beginning of this chapter (see 5.1.1) different methods for this kind of clustering were mentioned. Han et al. apply the CBC (Clustering By Committee) algorithm.

Details of Proceeding

Datasets used for testing. Han et al. collect two (name) datasets containing citations of 6 distinct "J Anderson"s and 9 distinct "J Smith"s, by using their EbizSearch system [PTH$^+$03] and querying Google with the cited names plus the keyword "publication". All author names are simplified to the first name initial and last name, e.g. "Robert L. Winkler" becomes "R Winkler". Title words are stemmed with Krovetz stemmer [Kro93] after stop words, e.g. "a", "the" etc., were removed.

Conducted experiments. The two datasets are processed twice with the model-based k-means algorithm, namely before and after clustering title words with the CBC algorithm. Since the initial partitioning is randomized and the algorithm depends on the initial partition, the authors conduct the experiments 10 times to get an average accuracy.

Evaluation and Discussion

Disambiguation accuracy. The evaluation is based on the *confusion matrix* $A \in \mathbb{N}^{k \times k}$. The *disambiguation accuracy* is exactly defined as described in Section 5.1.3. To recall the definition of the confusion matrix, its elements a_{ij} contain the number of how often "Author i" was predicted as "Author j", so that the diagonal entries contain the number of correctly clustered citations. Therefore, *disambiguation accuracy* is defined as the sum of the values of the diagonal elements divided by the sum of all elements in the matrix

$$\frac{\sum_{i=1}^{k} a_{ii}}{\sum_{i,j=1}^{k} a_{ij}}$$

Results. Since the accuracy depends on the random initial partitions, accuracies after the initialization step are given together with the final accuracies.

Initial accuracies are above 20% in average, the best result without CBC is given with 64.7%, with CBC about 73,6%. The average values are about 50% and 62,1% respectively.

Discussion. Despite of its simplicity, the approach yields promising results taking into account that the Naive Bayes model leaves room for extensions, e.g. by adding further feature types. The achieved accuracies are not fully comparable to the ones of the following approaches since the datasets used for tests are not the same, in particular they contain fewer different individuals. In comparison to datasets extracted from DBLP, they are more diverse, i.e. the entities are less similar as in the case of DBLP. We assume that the sample datasets are subsets of some datasets used in other approaches.

5.2.2 K-way Spectral Clustering with QR Decomposition

The authors of [HZG05] propose a *spectral clustering* algorithm. The term "k-way" highlights that the algorithm fixes the number k of clusters in advance, i.e. the algorithm gets k as input. Spectral clustering algorithms use a matrix representation of the similarities between data objects (citations here). Eigenvectors and Eigenvalues, i.e. the spectrum of the *similarity matrix* is considered to provide information for clustering. A detailed overview of spectral clustering is given in Section 4.1.3. The particular algorithm proposed by Han et al. performs a pivoted QR decomposition to assign citations to clusters. The relation between QR decomposition and cluster assignment is explained below in this section. The algorithm is designed to solve the following problem of name disambiguation:

Problem definition

The aim is to partition sets of citations into clusters, such that a cluster contains only citations written by the same author. The number k of different clusters is known in advance.

Approach

The authors study not only the quality of spectral clustering applied to name disambiguation, but also compare two different feature weight functions, namely tf.idf and ntf (as defined in Section 4.4.3). The former one stands for "term frequency inverse document frequency" and considers the term frequency of a term in a particular document or citation in relation to the inverse frequency of the term in all documents, citations resp. Remember the idea that a term that is not very common in the set of documents, citations resp., should be awarded with a high weight, whereas a term occurring in almost every document, citation resp., should be penalized with a low weight since it does not contribute much to the selection of the relevant documents, citations resp. The latter weight is called "normalized term frequency". The pairwise similarities of citations are represented by a *similarity graph* (see Chapter 3) which itself is represented by a *similarity matrix*.

Features and Representation of Citations in K-Way Spectral Clustering

A citation c is represented by a feature vector consisting of coauthor names, paper titles and publication venue titles. Since this combination of feature types is quite essential and widely used, we abbreviate it by *CoTiVe* as introduced in Section 5.1.1. Assuming that there exist m features in the whole set of citations, citation c_i is represented by a m-dimensional vector $c_i = (\alpha_{i1}, \ldots, \alpha_{im})$, where α_{ij} is the weight of the jth feature or 0 if the jth feature does not appear in citation c_i. The Gram matrix of the citation vectors contains the pairwise cosine similarities between the citations.

Clustering Algorithm

The clustering algorithm is based on slightly complicated linear algebra which is described in detail in the Chapter 3 and Section 4.1.3.

Spectral Relaxation. A set of m-dimensional citation vectors c_i, $1 \leq i \leq n$ leads to the m-by-n citation matrix $C = [c_1, \ldots, c_n]$.

The citation matrix C shall be partitioned into k clusters C_i, $1 \leq i \leq k$, expressed by: $CE = [C_1, \ldots, C_k]$, where $C_i \in \mathbb{R}^{m \times s_i}, C_i = [c_1^{(s)}, \ldots, c_{s_i}^{(s)}]$ with E a permutation matrix and s_i the number of citations in cluster i. For such a partition Π (which is what the clustering algorithm tries to find) of the citation vectors, the associated *sum-of-squares cost function* is:

$$ss(\Pi) = \sum_{i=1}^{k} \sum_{s=1}^{s_i} ||c_s^{(i)} - m_i||^2, \text{with } m_i = \sum_{s=1}^{s_i} \frac{c_s^{(i)}}{s_i},$$

where m_i is the mean vector, i.e. the center of cluster C_i according to a norm $|| \; ||$. The clustering algorithm should minimize $ss(\Pi)$, since intuitively this means that the cohesion of clusters is maximized.

This optimization problem is equivalent to a relaxed maximization problem

$$\max[\text{trace}(X^T \underbrace{C^T C}_{\text{Gram matrix}} X)],$$

where $X^T X = I_k$ is the k-dimensional identity matrix and X is an arbitrary orthonormal matrix. Remember that $X^T X = I$ holds true for every arbitrary *orthonormal* matrix. The relaxed version of the problem has a *closed-form solution*, i.e. the solution can be given in terms of a formula consisting of computable functions and operators.

From a theorem of Ky Fan it follows that it suffices to compute the largest k eigenvectors, more precisely the eigenvectors of the largest k eigenvalues, of the Gram matrix C^TC. Combining these eigenvectors to a matrix $X_k \in \mathbb{R}^{n\times k}$ transforms the n m-dimensional citation vectors to n k-dimensional citation vectors, since each row of X_k corresponds to a citation vector.

However, the focus is not on reducing the dimension of the feature space, but on identifying the clusters, which is achieved with QR decomposition.

Pivoted QR Decomposition for Cluster Assignment. The bridge between pivoted QR decomposition and cluster assignment is built based on some mathematical background involving the k largest eigenvectors and corresponding eigenvalues of the Gram matrix. For a more detailed description, we refer to [HZG05].

The theory is implemented as follows: Let $x_1, \dots, x_k$ be the eigenvectors of C^TC and $\lambda_1 \geq \dots \geq \lambda_n$ be the corresponding eigenvalues, i.e. $C^TCx_i = \lambda_i x_i, 1 \leq i \leq n$. Then define $X_k^T := [x_1, \dots, x_k]$ as the matrix consisting of the k largest eigenvectors

$$X_k^T = \begin{pmatrix} x_{11} & \dots & x_{k1} \\ \vdots & & \vdots \\ x_{1n} & \dots & x_{kn} \end{pmatrix}$$

where a row represents a citation.

1. Take a column x_i of X_k^T which has the largest norm, i.e. $i = max_j ||x_j||$.
2. Orthogonalize the other columns x_j of X_k^T against x_i. For the other columns belonging to the same cluster, the norm is rather small, whereas for columns belonging to other clusters, the norm is rather large.
3. Repeat the step for the residual vectors $k-1$ times, i.e. take the residual vector with the largest norm and orthogonalize the other $k-2$ columns against this vector.

This procedure corresponds to a pivoted QR decomposition: a permutation matrix P is found with

$$X_k^T P = QR = Q[R_{11}, R_{12}]$$

where Q is an orthogonal $k \times k$ matrix and R_{11} is a $k \times k$ upper triangular matrix, i.e. the elements below the diagonal are 0. The final computation yields

$$\hat{R} = R_{11}^{-1}[R_{11}, R_{12}]P^T = [I_k, R_{11}^{-1}R_{12}]P^T.$$

Now, each citation c_j is assigned to (hopefully the correct) cluster i where i is the row index of the largest absolute value of the jth column of $\hat{R}$.

Details of Proceeding

Preparation of datasets for testing. The authors prepare two different types of data sets. The first type are data sets containing citation data extracted from DBLP. These data sets are constructed by clustering according to a reduced name representation which is obtained by reducing names to their first name initial and last name, e.g. "Martin Hofmann" would be reduced to "M Hofmann". The underlying idea is that the reduced format is commonly used and increases the ambiguity of names. The authors then order these clusters of citations by their size and use the four most ambiguous names (all of them coming from Asia: "J Lee", "S Lee", "Y Chen", "C Chen") plus 10 other name sets. Furthermore, the name datasets are enriched with citations of relevant authors found on publication lists in the Web.

The second type of datasets consists of two name datasets for the names "J Anderson" and "J Smith". They are constructed manually by searching for scientists' homepages or rather publication lists in the Web. The resulting name sets consist of 15 "J Anderson" with 229 citations and 11 "J Smiths" with 338 citations. In contrast to the DBLP datasets, both data sets contain citations not only concerning computer science or computer science related subjects, but contain citations from various research areas.

For evaluation purposes, the ground truth consisting in *canonical name labels* is verified and added manually. In this context, ground truth means the true belongings of citations to clusters and a *canonical name* as defined in [HZG03] is a name that is the minimal invariant and complete name entity for disambiguation. Notice, that an unambiguous name may contain more than just the author's personal name. The features are processed as follows. Occurring coauthor names are also reduced to an abbreviated form in the same way as described above. Abbreviated titles of publication venues are extended to their original form, processed with Porter's stemmer and freed from stop words like "a", "the" etc.

Experiments. The conducted experiments are designed to reveal several properties of name disambiguation.

The effect of *dataset size* on clustering accuracy is examined by two different kinds of experiments. The first one selects from each name dataset only the authors having authored at least a certain minimal number $i, 2 \leq i \leq 10$ of publications. The second one considers only authors having 10 or more citations. It randomly selects a certain percentage p of citations of each author, starting with $p = 10\%$ and augmenting p stepwise by 10% up to 100%. To avoid falsified results due to these random selections, this kind of experiment is conducted 10 times and the average accuracy is taken.

To compare the *feature weights tf.idfand ntf*, each of these experiments is conducted twice: the first time weighting features with tf.idf, the second time using ntfto weight features.

The *power of the feature types* is examined by using each feature type alone and comparing the achieved accuracies.

The effect of the *amount of name information* is studied by running experiments both with fully reduced names (i.e. first name initial and last name) and with partially reduced names (i.e. first three letters of first name and last name, e.g. "Mar Hofmann").

As stated in the problem definition, the correct number of clusters is always given and not subject of examinations here.

Evaluation and Discussion

The authors evaluate their approach based on the *confusion matrix* $A \in \mathbb{N}^{k \times k}$ in the same way as described at the beginning of this chapter in Section 5.1.3. To give a short recall, an element a_{ij} of the confusion matrix contains the value of how often "Author i" was predicted as "Author j" and for $i = j$ the number of correct predictions. Therefore, *disambiguation accuracy* is defined as the sum of the values of the diagonal elements divided by the sum of all elements in the matrix

$$\frac{\sum_{i=1}^{k} a_{ii}}{\sum_{i,j=1}^{k} a_{ij}}$$

The following results are produced by experiments conducted on DBLP datasets only.

Effect of the size of datasets. The subsets extracted by minimum number of citations per author from the DBLP name datasets contain between 4 and 100 authors and about 60 to 1500 citations. In general, the experiments reveal that with increasing minimal number of citations attributable to a single author, disambiguation accuracy increases, too. The counterexample where two authors always stick together shows that in particular cases, increasing the number of citations per author can lead to a decreased disambiguation accuracy. Han et al. explain this case as follows. Both of the "misbehaving" authors of DBLP mainly published on the same topic "databases", so intuitively, adding more citations on "databases" strengthens the bonds between them, e.g. some citation may even introduce shared coauthors. Since most of the name datasets are better clustered than the datasets of the four Asian names, there is a tendency that increasing ambiguity makes name disambiguation more difficult. To avoid cases as depicted in the counterexample, other feature types must be added.

tf.idf vs. ntf. Independently of the size of datasets, tf.idf in general performs better than ntf. This can be explained as follows. Looking at the definitions of tf.idf, ntf resp., it is obvious that tf.idf considers not only the frequency of the feature in the citation like ntf does, but also the frequency of the feature in the whole dataset. The fact that citations are only small pieces of text induces that words are rather seldomly repeated within a single citation. As a consequence, the "intra-citation" frequency of features as measured by ntf is nearly meritless, i.e. as good as using no feature weight at all. It remains to study if and to which amount better feature weights, e.g. Log Entropy (see Section 4.4.3), can improve the disambiguation method.

Power of feature types. For the examination of the power of feature types, only authors having at least 10 publications are considered. The authors differentiate for the coauthor feature type, whether publications authored by a single author are counted as incorrectly clustered or ignored. The authors claim that coauthor information outperforms one of the other two feature types in most name datasets. We argue that the results of the experiments published in [HZG05] do not prove a *significant* predominance of the coauthor feature: the mean of accuracies of clustering using only coauthor information is even poorer than the one of using only paper titles in the case of counting single author citations and nearly the same in the case of ignoring single author citations. Referring to the latter case of treating coauthor feature types, the standard deviation, too, is quite similar to the one of paper titles.

Han et al. hypothesized that venue title information is more stable than paper title information because it is unlikely that authors use the same words in titles several times and often, the title does not contain any words revealing an author's research area. They give several reasons for the fact that their experiments disprove their hypothesis. First, the latter information is less available and more often parsed incorrectly. Second, some publication venue titles, e.g. "Ph.D. Dissertation", do not contribute to the identification of the author's research area. Third, several publication venue titles may share the same abbreviations. We argue that this case should not occur when disambiguating DBLP data. When extracting data directly from the DBLP.xml file, it should be possible to get unambiguous venue title information, since publication venues are also disambiguated in DBLP. However, it seems useful to add further information such as the topic of the paper as the authors propose. Fourth, the full title information is not always available as the authors observe. Fifth, since DBLP is restricted to topics relevant to computer science, most authors from DBLP datasets belong to the computer science community. As a consequence, different authors are likely to have similar research interests and meet each other at the same publication events, so that this feature type is disengaged. The authors of [HZG05] propose that this analysis of feature types can be used to automatically tune weighting factors for different attributes to achieve further improvement.

Effect of the amount of name information. In these experiments still carried out by Han et al., only the coauthor feature is used and all citations coming from the Web and not from DBLP as well as citations authored by a single person were eliminated in the DBLP name datasets. The former type of citations is eliminated because names are often given only by first name initial and last name in publication lists in the Web, so that there is no three letter representation available. As a consequence inconsistent name formats lead to several features that represent an affected (co)author. The latter type of citations is eliminated because citations that do not contain coauthor information obviously cannot be clustered using only coauthor information.

In general, adding name information improves the disambiguation accuracy, even though in most cases, it does not make a big difference. In few cases (2 out of 14), accuracy even decreases. The authors explain this phenomenon as follows. DBLP still contains synonyms due to inconsistent name representations, e.g. "W. Tsai" and "Wen Tsai" (this particular inconsistency may have been eliminated by now) and name misspellings, e.g. "Kohji Zettsu" is misspelled as "Koji Zettsu" in some citations. Keeping the first three characters or more of the first name may therefore introduce "new" ambiguity, so that the names of the given examples are misleadingly treated as being different. The authors propose to apply techniques of duplicate string detection as an improvement.

Effect of the diversity in research areas of authors. As stated in the previous paragraphs, the tight connections of research areas in the DBLP name datasets constitutes a problem for clustering with only three feature types. The authors considered each author as a class and then measured the *within-class* and the *cross-class similarity distributions* based on similarity according to publication venue titles. Similarity values range between 0 and 1. The within-class similarity distribution indicates the cohesion of a cluster. The argumentation is that citations of one author are more similar to each other than publications authored by different individuals. Therefore, a good clustering requires within-class similarities distributed near 1, since high similarity means a very similar research area (which is determined based on common words in publication venue titles). In contrast, cross-class similarities should be distributed near 0. So, distributions that are inverse to this ideal cases may be an explanation for low disambiguation accuracies. To avoid this problem, the authors propose to introduce additional feature types that enlarge differences between citations of different authors, i.e. that reduce cross-class similarities.

Disambiguation accuracy for Web collected name datasets. The achieved higher accuracies for the two Web collected name datasets confirm the remarks of the last paragraph. The authors further state, that the variety of family names which is different for different nationalities, influences the degree of difficulty of name disambiguation: it seems that a low variety of most frequently occurring nationalities makes disambiguation more difficult.

Summary

Due to the variety of conducted experiments, the following conclusions can be drawn. Notice that also in computer science it holds that *the exception proves the rule*.

- In general, the more citations in the dataset, the higher the disambiguation accuracy.
- The weighting function tf.idf performs better than the weighting function ntf.
- Coauthor information performs slightly better or as well as paper title information and both perform better than publication venue title information.
- The more name information is available, the better is the clustering under the condition that this name information is correct.

- The more diverse the authors' research areas, the better is the clustering.
- The more various family names of frequently appearing nationalities, the easier is the clustering.

The authors achieved an average accuracy between 61.5% and 64.7% for DBLP name datasets and 84.3% and 71.2% for the Web collected name datasets with a standard deviation of about 10%.

5.2.3 A Blocking-Based Solution

Lee et al. have a slightly different view on name disambiguation. They define the problem as the "mixed citation" problem as follows:

Problem Definition

Given a collection of citations, C, produced by an author, a_i, can we quickly and accurately identify false citations produced by another author a_j when a_i and a_j have the identical name spelling?

Notice that the studied problem remains the same: name disambiguation. However, this definition is different from the others appearing in this chapter by the following characteristics. Other problem definitions consider a set of citations C that contains citations of several authors and the task is to divide this set into subsets according to the authors. Contrarily, the idea of the problem task now is that there is the set of citations of a single author, but it is "contaminated" with citations of another (homonymous) author. In a general sense, these definitions are equivalent, since the set of citations could be "contaminated" by citations of even more homonymous authors. So, the citation sets can be the same for both cases. But the way of approaching the task is different. Instead of having different groups of citations right from the beginning or at least having the goal to get different groups, the task lying in the above definition is to sort out "false" citations. Just imagine a nest with eggs and the goal is to kick out all cuckoos' eggs.

Approach

The authors Lee et al. introduce in [LOKP05] an approach that applies a sampling technique to name disambiguation. The idea of sampling is to pre-select a set of promising candidates applying a fast, but accurate enough approximating algorithm. This reduces the size of the dataset that then requires further, more exact processing. Thus, the purpose of sampling is not to increase accuracy, but only to speed up the method. The benefit is obvious: an inefficient method may get efficient and therefore scalable to large sizes of datasets.

In particular, name disambiguation is achieved by the following method. Assume, that in a citation c_j one author a_i name is missing. Then, a function that labels correctly c_j with a_i is called *citation labeling function*. Assume, there is a "good" citation labeling function $f_{cl} : c_j \mapsto a_i$ available, then it can be used to construct a *citation labeling algorithm* in the following way:

Let $C = \{c_1, \ldots, c_N\}$ be the set of citations of author a_1, let a_2 be another author name. For each citation $c_j \in C$: compute $a_2 := f_{cl}(c_j)$. If $a_1 \neq a_2$ the remove c_j from C since a_1 does not occur in c_j.

Features and Representation of Citations in the Sampling-Based Approach

A citation is described by the common feature types, i.e. coauthor information, paper title keywords and the venue title (CoTiVe). Due to the chosen disambiguation algorithm, citations are represented by a triple of vectors, one vector for each feature type. Each vector contains term weights associated to the occurring features of a particular citation. The terms are weighted with tf.idf, i.e. the weight depends on the frequency of terms both in the particular citation and in the whole set of citations. Notice that in contrast to the vector space model described in Section 4.3 where one vector per citation is constructed, this approach uses three vectors, for each feature type one: v_c, v_p, v_v where v_c stands for a vector containing coauthor information, v_p for paper title information and v_v for publication venue information. The advantage becomes apparent in the next step. For name disambiguation, a *similarity measure* must be defined to compute the similarity between citations and authors: the below defined similarity measure is composed by three "similarity components" that can be weighted according to the power of each feature type. The similarity between an author $a = (a_c, a_p, a_v)$ and a citation $c = (c_c, c_p, c_v)$ is then computed as

$$\mathrm{SIM}(a, c) = \alpha\ \mathrm{sim}_c(a_c, c_c) + \beta\ \mathrm{sim}_p(a_p, c_p) + \gamma\ \mathrm{sim}_v(a_v, c_v),$$

where $\alpha + \beta + \gamma = 1$. The vector-based similarity is defined as the *cosine similarity* (see also Section 4.4), i.e. for vectors v, w the cosine of the angle θ between v, w

$$\cos(\theta) = \frac{v \cdot w}{||v|| \cdot ||w||}$$

expresses the similarity. On the one hand, the factors weighting the similarity components require some experience to find optimal values. On the other hand, there already is prior work based on which the factors can be set.

Citation Labeling Algorithm

Baseline approach. The authors of [LOKP05] present the following "baseline" approach. Assume the same set of citations as above, C, as well as a set of authors $A = \{a_1, \ldots, a_k\}$.

For each $c_j \in C$: return $a_i \in A$ with

$$\max_{1 \le i \le k} (\mathrm{SIM}(a_i, c_j))$$

Notice that the performance of this approach extremely depends on the quality of the similarity measure. The complexity of this naive approach is $O(|A||C|)$, i.e. quadratic. That is why the authors combine it with sampling which reduces complexity as shown later on.

Sampling approach. The authors adopt the *sampling-based join approximation method* developed by Gravano et al. in [GIKS03]. *Sampling* is a technique that allows to reduce quickly the size of a set that needs to be processed. The idea is to extract a set of *candidates* being most probably desired objects. For each citation c_j only the authors a_i that are *suspected* to be highly similar are extracted and put into the sample set of "wannabe authors". The goal is to draw a sample set S of size s from the set of authors A in such a way that the frequency of $a_i \in A$, freq(a_i), can be used to approximate SIM$(a_i, c_j) = \sigma_i$ which can be approximated by

$$\frac{\mathrm{freq}(a_i)}{s} \cdot T_V(c_j), \text{where } T_V(c_j) = \sum_{i=1}^{|A|} \sigma_i = \sum_{i=1}^{|A|} \mathrm{SIM}(a_i, c_j)$$

The "entrance ticket" for a_i into the sample S is that $\frac{\mathrm{freq}(a_i)}{s} \cdot T_V(c_j) \geq \theta$ for some pre-defined threshold θ. The baseline approach is then extended to the following algorithm.

For each $c_j \in C$: extract a sample S from A and return $a_i \in S$ with

$$\max_{a_i \in S}(\mathrm{SIM}(a_i, c_j))$$

The complexity of this sampling approach is $O(|A| + |S||C|)$. Since $|S| \ll |A|$, e.g. $|S| = 64$ is chosen and fixed at the beginning in contrast to $|A| = O(10^5)$ assuming that A is the set of authors appearing in DBLP.

Details of Proceeding

Preparation of datasets. The authors use two different digital libraries as test data, DBLP and EconPapers[4]. Since the purpose of their studies is to develop a *scalable* solution, they consider the whole databases into the testing procedure and do not extract subsets. The authors choose some homonymous names from the DBLP dataset. Since the authors know DBLP well, they also know homonymous authors and can therefore test their method with real-world examples. In contrast, the EconPaper dataset is not well known to the authors since it is for economics instead of computer science. Therefore, "false" citations are introduced. Obviously, then it is also known which citations are *surely* false.

Conducted Experiments. The authors use the sampling technique as implemented in [GIKS03] with a sample size $s = 64$ and a threshold $\theta = 0.1$. The feature vectors are constructed twice, in different ways. The model called "Bag" counts each occurrence of multiple occurrences, the "Set" model counts a group of multiple occurrences of equal tokens only once. The factors weighting the feature types are set to $\alpha = 0.5$, $\beta = 0.3$ and $\gamma = 0.2$.

Evaluation and Discussion

Measuring quality. One aspect of quality of the developed method is computational complexity since the authors' intention is to design a *scalable* solution. The *computation time* is the measured indicator for scalability.

The implementation of the algorithm returns a ranking of all citations as result, i.e. a false citation c_f has a low rank, e.g. in the bottom 10% of all ranked citations. The accuracy is given in form of a *percentage/rank ratio*. This kind of accuracy seems to be similar to a kind of *recall*.

Results on efficiency. Independently from the model type "Bag" or "Set", the experiments show a significant speed-up: the sampling approach is 3 to 4 times faster than the baseline approach. Furthermore, the computational complexity of the sampling approach is *bounded* by the size s of the sample S which is fixed at the beginning, e.g. $s = 64$.

Results on accuracy. Assume that false citations occur with a frequency of 1 to 100, e.g. there is one false citation c_f hidden in 100 citations. Then, the sampling approach using the "Bag" model detects c_f with an accuracy of 60% (i.e. 64%, rank = 0.1).

The authors observe that the difficulty of the name disambiguation problem grows with the number of different individuals accountable for the citations of a particular citation set. Their experiments show that for a citation set produced by only two homonymous authors, all false citations are ranked lower than 30%, but in the case of more than three different authors, false citations already get ranked higher. Notice that the percentages do not denote the amount of false citations but denote the upper limit of the ranking of false citations. For example a ranking of

[4] http://econpapers.repec.org/

30% for all citations means that all false citations are ranked in the lowest part of the size of 30% of all citations.

5.2.4 Search Engine Driven Approach

The authors of [TKL06] show how to involve yet another type of features which has not been described so far. The idea is to use web search results (gained from queries by citation titles) for unsupervised clustering. To evaluate this idea, the authors define the problem equivalently to the "mixed citation problem" in [LOKP05] which restricts the original name disambiguation problem by assuming that the number of persons having the examined name is known before the clustering process.

Problem definition

The given information consists of an author name X, a set C of citations c having X as author and the number k of different authors hiding behind the name X.

The task is to cluster the list of citations into k groups so that each group contains citations of a single author with name X.

Approach

The underlying idea is that each person can be characterized by the set of related web pages found by a search engine. Since the title of a paper is related to the authors who have published this paper, a query by the title returns search results that are also related to the authors. For example, authors often list their publications on their personal homepages. This leads to the following approach:

For all citations $c \in C$, query the Web by the title of c using a search engine (e.g. Google). The result is a set of relevant URIs which is used to construct feature vectors in a similar way as in previous approaches. For weighting the features, a so called "inverse host factor" (IHF) is used, which weights features (i.e. URIs) according to the (inverse) frequency of hosts in a set of URIs: URIs with hosts occurring often in the set of URIs get small weights. The idea is that URIs with rare hosts probably point to more significant pages (e.g. personal homepages, homepages of groups etc.) than a URI containing a common hostname (e.g. CiteSeer, DBLP, amazon, etc.).

Features and Representation of Citations in the Search Engine Driven Approach

The set of features consists of all URIs resulting from the queries mentioned above. A citation is therefore represented by a vector (following the vector space model as described in Section 4.3) whose fields correspond to the elements of the set of result URIs. The values are the computed IHF weights. An improved version uses domains instead of URIs and resolves hostnames to IP addresses to avoid distortions due to hosts having several names.

Clustering Algorithm

The clustering method is hierarchical agglomerative clustering (HAC) as described in Section 4.1.2. The algorithm stops when the right number k of clusters is reached, which yields best results in combination with single link (see Section 4.1.2). Cosine similarity as described in Section 4.4 is used as similarity measure.

Details of Proceeding

The calculation of the IHF values is based on a representative Web corpus constructed from the citations belonging to top 100 authors of DBLP. The testbed consists of a manually-disambiguated set of citations containing 24 ambiguous names each representing 2 or 3 persons and 30 citations per name in average. Each citation is represented by 10 relevant URIs (if possible; the authors did not always find 10 or more relevant URIs).

Evaluation and Discussion

In tests carried out by the authors, an accuracy (defined as the ratio between the number of correctly predicted classes and the number of citations) of 0.836 was achieved. The algorithm is quite efficient (see 4.1.2 for details), but collecting the features may involve some time, e.g. if network connection is slow or the search engine's capacity is exhausted.

Advantages of this method are that it is easy to combine with other methods using different features (e.g. HAC with the common feature set CoTiVe (definition in Section 5.1.1). It uses a clustering algorithm which is easy to implement and well established and doesn't need complex mathematics. On the other hand, this approach depends on the search engine (e.g. Google's ranking algorithm) and results are not reproducible (since the order of search results may vary). Finally, efficiency depends on the Internet connection and on the search engine.

5.2.5 A Hierarchical Naive Bayes Mixture Model for Name Disambiguation in Author Citations

The authors Han et al. improve the naive Bayes model of [HZG03] and use it in two different ways in [HGZ$^+$04] and another time in [HXZG05]. The former is described later in Section 5.3.1. The stochastical model is in both approaches more or less the same, but the involvement is different. In the former approach, the naive Bayes model is part of a "standard" naive Bayes classifier, whereas this section describes another unsupervised usage of naive Bayes. Notice that normally, training is needed for parameter estimation of the model. The following approach replaces training with the *Expectation-Maximation algorithm* (EM).

Problem definition

The approach requires the knowledge of the correct number of clusters in advance.

Approach

The authors refine the naive Bayes model developed in [HZG03]. Model parameter estimation and cluster assignment is done by the EM algorithm. The stochastic model is hierarchically structured, which is described in detail below.

Features and Representation of Citations in the Naive Bayes Mixture Model

Similar to Section 5.2.1, the feature types to represent citations consist of the set CoTiVe, i.e. coauthor information, paper titles and publication venue titles.

The idea is that the set of citations is partitioned into groups where each group stands for one author person. The citations of each group are used to construct a stochastical model. The purpose of this model is to estimate the probabilities that an author that is described by her/his prior citations, i.e. the citations belonging to the accordant group, writes a particular paper. The clustering can be considered to happen in two steps. First, (virtual) authors are depicted by estimating the model parameters. The second step consists of a kind of query: for a given citation, the group representing the author that is most likely having produced the given citation is determined. The used algorithm determines these groups iteratively starting from a random initial partition. This proceeding reminds of the model-based k-means algorithm that is described in Section 5.2.1.

Composition of the Naive Bayes Model

The purpose of the model is to illustrate the history of authors, i.e. the model should reflect the authors' habits of coauthoring with certain other researchers, of using keywords in titles, of publishing on certain conferences. More generally spoken, the model simulates the context or environment of author's that can be extracted from citations.

The posterior probability and the mixture model. For a given set of citations $c = \{c_1, \ldots, c_N\}$ and k authors having the canonical (see Section 5.2.2), i.e. non-ambiguous, names $C_1, \ldots, C_k$, the most probable author being responsible for a citation c_i is determined by finding the author having the maximal posterior probability of authoring c_i:

$$\max_i P(C_i|c_j)$$

The authors assume that a citation $c_j \in \{c_1, \ldots, c_N\}$ is generated by a mixture of several authors having canonical names $C_1, \ldots, C_k$. These authors are considered to be the "ingredients" of the mixture model (see Section 4.2.1). The probability of c_j therefore is composed by the weighted probabilities of each author having produced the citation, where the weights are the prior probabilities of authors having published

$$P(c_j) = \sum_{i=1}^{k} (P(C_i) \cdot P(c_j|C_i))$$

The EM-algorithm estimates these conditional probabilities by maximizing the expectation of the citation set $\{c_1, \ldots, c_N\}$, i.e. the sum of the probabilities $P(c_j), 1 \leq j \leq N$. The parameter estimation therefore is conducted by solving the target

$$\max(\sum_{j=1}^{N} P(c_j))$$

Then, the naive Bayes theorem is used in the same way as in Section 5.2.1: the naive Bayes theorem is applied and the denominator of the resulting fraction is omitted since it is constant and therefore irrelevant for determining the maximum leading to

$$\max_i P(c_j|C_i) \cdot P(C_i)$$

Assuming statistically independent feature types and features, the conditional probability $P(c_j|C_i)$ is decomposed according to the involved feature types to

$$P(c_j|C_i) = \prod_{m=1}^{3} P(F_m^{(j)}|C_i) = \prod_{m=1}^{3} \prod_{n=1}^{j(m)} P(F_{mn}^{(j)}|C_i),$$

where $F_1^{(j)}$ denotes the feature type of coauthor names in c_j, $F_2^{(j)}$ the feature type of paper titles in c_j and $F_3^{(j)}$ the feature type of venue titles in c_j. Notice, that citations may have several features of each feature type. In the case of coauthor information, there may be also 0 features, expressing that the considered citation was published by a single author. A citation c_j therefore may be represented by the following tuple of features:

$$c_j = (\underbrace{F_{11}^{(j)}, \ldots, F_{1j(1)}^{(j)},}_{j(1) \text{ coauthors}} \underbrace{F_{21}^{(j)}, \ldots, F_{2j(2)}^{(j)}}_{j(2) \text{ paper title keywords}}, \underbrace{F_{31}^{(j)}, \ldots, F_{3j(3)}^{(j)}}_{j(3) \text{ venue title keywords}})$$

Notice that the comments of Section 5.2.1 about statistical independency of feature types and feature values still hold.

Hierarchical composition of model parameters. The model is constructed by involving two pairs of complement probabilities about single authoring and collaborating with others, and about coauthoring several times with the same people and coauthoring with "new" coauthors. These conditional probabilities are used to make statements about collaborating with a particular coauthor in the two cases of already "seen" and new, i.e. "unseen", coauthors.

Han et al. justify the model hierarchy described in detail below by the following hypotheses. First, authors have different probabilities of authoring papers alone. For example, a student writing her/his master thesis will not publish a paper alone, but professors may publish alone, albeit the more frequent case is that they, too, collaborate, e.g. with doctoral and master students. Second, the distribution of coauthors discriminates each author since it is quite unique, almost like a fingerprint. Similar circumstances are observed with usage of paper title keywords and publication venues. Nevertheless, the parameter estimation is exercised for the coauthor feature type only, the parameters for the other feature types are determined analogously.

In the following, the term "training data" stands for the citation data actually assigned to a cluster C_i.

Modeling the behavior concerning the habit to collaboration.

- $P(\text{No}|C_i)$ denotes the probability that the author with the canonical name C_i writes a paper alone given the event that C_i publishes. Using the training data this probability is estimated as the ratio of the number of single-authored papers of C_i to the total number of papers of C_i.
- $P(\text{Co}|C_i) = 1 - P(\text{No}|C_i)$ denotes the probability that the author with the canonical name C_i writes a paper in collaboration with at least one coauthor given the event that C_i publishes.

Modeling the behavior concerning the habit to collaboration with "new acquaintances".

- $P(\text{Seen}|\text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper *with previously seen* coauthors given the event that C_i publishes in collaboration with someone. A coauthor is considered to be "seen" if C_i has collaborated with him at least twice. This probability is estimated using the training data as the proportion of the number of papers of C_i authored in collaboration with "seen coauthors" to the total number of papers of C_i authored in collaboration with someone.
- $P(\text{Unseen}|\text{Co}, C_i) = 1 - P(\text{Seen}|\text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper *with previously unseen* coauthors given the event that C_i publishes in collaboration with someone. A coauthor is considered to be (previously) "unseen" if C_i has collaborated with him only once.

Estimating the habit to collaboration with a particular coauthor.

- $P(F_{1n}|\text{Seen}, \text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper with the particular coauthor F_{1n} given the event that C_i publishes in collaboration with previously "seen coauthors". This probability is estimated using the training data as the proportion of the number of papers of C_i coauthored with F_{1n} to the total number of papers of C_i.
- $P(F_{1n}|\text{Unseen}, \text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper with the particular coauthor F_{1n} given the event that C_i publishes in collaboration with previously "unseen coauthors". This probability is estimated using the training data as

$$\frac{1}{\text{"total number of (co)author names in the training set"} - \text{"number of coauthors of } C_i\text{"}}$$

  This estimation is based on the assumption that C_i has equal probability to collaborate with an unseen author.

Putting probabilities together.

$$P(F_1|C_i) = \begin{cases} P(\text{No}|X_i) & \text{if } j(1) = 0 \\ \prod_{n=1}^{j(1)} P(F_{1n}|C_i) & \text{if } j(1) > 0, \text{where} \end{cases}$$

$$\begin{aligned} P(F_{1n}|C_i) = {} & P(F_{1n}|\text{Seen}, \text{Co}, C_i) \cdot P(\text{Seen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i) + \\ & P(F_{1n}|\text{Unseen}, \text{Co}, C_i) \cdot P(\text{Unseen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i) \end{aligned}$$

This composition can be explained as follows. It holds:
$P(F_{1n}|C_i) = P(F_{1n}, \text{No}|C_i) + P(F_{1n}, \text{Co}|C_i) = 0 + P(F_{1n}, \text{Co}|C_i)$. The first equality obviously holds, the second equality holds because the probability that C_i authors together with F_{1n} given the event that C_i does not coauthor with anyone is obviously 0. Furthermore, it holds:
$P(F_{1n}, \text{Co}|C_i) = P(F_{1n}, \text{Seen}, \text{Co}|C_i) + P(F_{1n}, \text{Unseen}, \text{Co}|C_i)$. Finally, the two probabilities on the right hand side are composed as described above:
$P(F_{1n}, \text{Seen}, \text{Co}|C_i) = P(F_{1n}|\text{Seen}, \text{Co}, C_i) \cdot P(\text{Seen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i)$ and
$P(F_{1n}, \text{Unseen}, \text{Co}|C_i) = P(F_{1n}|\text{Unseen}, \text{Co}, C_i) \cdot P(\text{Unseen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i)$

The authors similarly estimate the probabilities that an author C_i writes a paper title ($P(F_2|C_i)$) and that an author C_i publishes in a particular journal ($P(F_3|C_i)$).

Clustering Algorithm: EM-algorithm

The expectation-maximation algorithm iteratively estimates the model parameters of the naive Bayes mixture model and according to this reassigns all citations to the approbate clusters. Assume the citation set consists of N citations $c_1, \ldots, c_N$ and is produced by k canonical authors $C_1, \ldots, C_k$.

1. *Initialization.* The set of citations is equally and randomly partitioned into k initial clusters, the prior probabilities estimated based on this initial partition, $P(C_i) = \frac{1}{k}$ and the conditional probabilities of the hierarchical model as given above calculated.
2. *E-step.* Each citation c_j is reassigned according to the posterior probabilities of each cluster representing an author of producing c_j.

$$P(C_i|c_j) = \frac{P(c_j|C_i) \cdot P(C_i)}{\sum_i (P(c_j|C_i) \cdot P(C_i)}$$

3. *M-step.* For each cluster C_i, $P(C_i)$, the conditional probabilities appearing in the model hierarchy and $P(c_j|C_i)$ are computed

$$P(C_i) = \frac{\sum_j P(C_j|c_i)}{N}$$

4. *Convergence.* Let $\sum_j'(P(c_j))$ be the total probability of the citation set. Then, the algorithm converges, if $|\sum_j(P(c_j)) - \sum_j'(P(c_j))| < 0.1$, where

$$P(c_j) = \sum_i P(c_j|C_i) \cdot P(C_i)$$

In this case, each citation c_j is assigned according to the maximum-likelihood principal, i.e. c_j is assigned to the cluster C_i that maximizes $P(C_i|c_j)$. Otherwise go to step 2 starting the next iteration.

Details of Proceeding

Preparation of datasets for testing. The authors use the datasets as prepared in [HZG05]. Section 5.2.2 describes the preparation exactly. The datasets containing 15 different "J Anderson"s and 11 "J Smith"s are collected from the Web in the same way as in [HZG05]. From DBLP 14 large datasets are chosen similarly to the way in [HZG05]. The correct name labels were checked manually. "Correct name labels" denote some unique name labeling each homonymous author. The datasets are preprocessed in the already known way by removing stop words and applying Krovetz's stemmer [Kro93]. Once more, journal abbreviations are replaced by their full names and author names are represented by the first name initial and last name. Furthermore, the authors prepare each dataset twice. In the second variant, they also semantically cluster keywords appearing in paper and venue titles. For this purpose, the CBC algorithm is applied. A more precise description of semantical clustering is given in Section 5.1.1, a comment on its application in Section 5.2.1.

Benchmark algorithm. The authors compare their EM-algorithm to a standard implementation of k-means, where citations are represented by feature vectors of the vector space model (see Section 4.3) using tf.idf as feature weights and the Euclidean distance as distance function.

Conducted experiments. The authors apply each algorithm 10 times since both depend on the initial partitions. Furthermore, each round of 10 experiments is applied to both variants of the datasets, with and without semantical clustering of keywords.

Evaluation and Discussion

Disambiguation accuracy. Once again, the *confusion matrix* $A \in \mathbb{N}^{k \times k}$ is constructed to determine *disambiguation accuracy* which is exactly defined as described in Section 5.1.3. An element a_{ij} of A contains the number of how often "Author i" was predicted as "Author j", so that the diagonal entries contain the number of correctly clustered citations. Therefore, *disambiguation accuracy* is defined as the sum of the values of the diagonal elements divided by the sum of all elements in the matrix

$$\frac{\sum_{i=1}^{k} a_{ii}}{\sum_{i,j=1}^{k} a_{ij}}$$

Results. The authors observe that their approach outperforms the k-means algorithm chosen to be the benchmark for almost all datasets and in average. The Web collected datasets are easier to cluster due to higher diversity of research areas. An average accuracy of about 54% with a standard deviation of almost 10% is achieved for the DBLP datasets, in contrast to 58% with a standard deviation of only 1.56% for the Web collected datasets.

The semantical clustering is shown to be and not to be useful. On the one hand, for some datasets the clustering is improved due to the fact that the semantical clustering introduces new information by connecting semantically similar words. On the other hand, some information about the different usage of words is lost. The authors state that the size of the word clusters regulates this balance of information gain and loss. It seems that large clusters can gain more information, but also lose more information.

5.3 Applying Supervised Clustering

The following approaches are distinct to the ones previously described in one main aspect. They belong to the category of supervised methods and are considered as *classifiers*. Classifiers need to be trained with sample data. This training set consists of citations of which the ground truth, i.e. the correct author entities, is known. Then, each author is considered as a class of which the characteristics are described by the sample data. This additional information promises higher accuracies. Unfortunately, such sample data is not always available in the real world.

5.3.1 Naive Bayes Model (NBM)

The authors Han et al. of [HGZ$^+$04] use the improved naive Bayes model of [HZG05] first given in [HZG03]. They transform their naive Bayes approach to a *supervised* method based on the same stochastic model as in [HXZG05]. Therefore, the following definition of name disambiguation first seems more general since it does not explicitly require the a priori knowledge of the number of clusters, but a precise consideration reveals that this is not the case. Truly, they do not explicitly require to know the number of clusters or rather classes in advance, but since they use even more information for training of their (supervised) classifier, this information is implicitly given in the training data.

Problem definition

Given a full citation with the query name omitted, the authors' name disambiguation is to predict the most likely canonical name from the citation base.

The definition of *canonical names* of [HZG03] is reported in Section 5.2.2 as the minimal name that is clear without ambiguity in the set of all occurring names in the dataset, e.g. in DBLP. Minimal name denotes a name with minimal length.

Approach

The authors refine the naive Bayes model developed in [HZG03] in the same way as in [HXZG05], but this time, they deploy it as a supervised method. In combination with *maximum likelihood estimation* (see Section 4.2.1), it can be used as a classifier. The model is hierarchically structured identically to Section 5.2.5.

Features and Representation of Citations in the Naive Bayes Model

Once again, the feature types to represent citations consist of the set CoTiVe, i.e. coauthor information, paper titles and publication venue titles.

Then, citations are grouped in such a way that each group represents a single author. Starting from these sets of citations, authors can be depicted by a stochastical model. The purpose of this model is to estimate the probabilities that an author that is described by her/his prior citations writes a future paper. "Prior citations" denote the citations already belonging to the set of citations that is associated with the author.

In contrast to the model-based k-means algorithm and the hierarchical naive Bayes model combined with the EM-algorithm, the "pure" naive Bayes approach needs training data to estimate the model parameters. The dataset used for training consists of citations together with the ground truth, i.e. citations in combination with the canonical names (see Section 5.2.2 unambiguously denoting the authors responsible for the citations. Consequently, both the nature and the number of possible classes is known in advance. Notice that the classifier performs well only if the training data covers all classes.

Composition of the Naive Bayes Model

The model "learns" about the history concerning collaboration habits, research topics and areas and preferred conferences of authors. When a (new) citation should be assigned to an author or rather to a class representing an author, the model is "queried" to find the author to which the citation fits best, i.e. to find the author who is most probable to have produced the citation.

The posterior probability. Assume that an ambiguous name stands for k distinct authors having the canonical names $C_1, \ldots, C_k$. The most probable author being responsible for a citation c_i is determined by finding the author having the maximal posterior probability of authoring c_i:

$$\max_i P(C_i|c_j)$$

Applying the naive Bayes theorem leads to a fraction whose denominator is constant and therefore irrelevant for determining the maximum. By omitting the denominator, it remains

$$\max_i P(c_j|C_i) \cdot P(C_i)$$

Once again, it is necessary to assume statistically independent feature types and features, so that the conditional probability $P(c_j|C_i)$ can be decomposed into probability components according to the involved feature types and therefore be computed as the product of the probability components $F_1^{(j)}$ denoting the feature type of coauthor names in c_j, $F_2^{(j)}$ denoting the feature type of paper titles in c_j and $F_3^{(j)}$ denoting the feature type of venue titles in c_j:

$$P(c_j|C_i) = \prod_{m=1}^{3} P(F_m^{(j)}|C_i) = \prod_{m=1}^{3} \prod_{n=1}^{j(m)} P(F_{mn}^{(j)}|C_i),$$

Notice, that citations may have several features of each feature type. In the case of coauthor information, there may be also 0 features, expressing that the considered citation was published by a single author. A citation c_j therefore may be represented by the following tuple of features:

$$c_j = (\underbrace{F_{11}^{(j)}, \ldots, F_{1j(1)}^{(j)}}_{j(1)\text{ coauthors}}, \underbrace{F_{21}^{(j)}, \ldots, F_{2j(2)}^{(j)}}_{j(2)\text{ paper title keywords}}, \underbrace{F_{31}^{(j)}, \ldots, F_{3j(3)}^{(j)}}_{j(3)\text{ venue title keywords}})$$

Notice that the comments of Section 5.2.1 about statistical independency still hold.

Hierarchical composition of model parameters. The model is constructed by involving two pairs of complement probabilities about single authoring and collaborating with others, and about coauthoring several times with the same persons and coauthoring with "new" coauthors for the first time. These conditional probabilities are used to make statements about collaborating with a particular coauthor in the two cases of already "seen" and new, i.e. "unseen", coauthors.

Han et al. motivate this construction of their model in the same way as in [HXZG05] which is described in detail in the previous section. Here, just a short recall of the underlying hypotheses is given. First, authors are more or less likely to write papers alone. Second, the profile of coauthors is characteristic for each author. This discriminating profile can be extended to comprise also the usage of paper title keywords and publication venues. Nevertheless, the parameter estimation is exercised for the coauthor feature type only, the parameters for the other feature types are determined analogously. Furthermore, a detailed description of the model parameters is already given in the previous section. Therefore, the following paragraphs just give a short overview of the model composition.

Modeling the behavior concerning the addiction to collaboration.

- $P(\text{No}|C_i)$ denotes the probability that the author with the canonical name C_i writes a paper alone given the event that C_i publishes.
- $P(\text{Co}|C_i) = 1 - P(\text{No}|C_i)$ denotes the probability that the author with the canonical name C_i writes a paper in collaboration with at least one coauthor given the event that the author with the canonical name C_i publishes.

Modeling the behavior concerning the addiction to collaboration with "new acquaintances". The notion of "seen" and "unseen coauthors" describes the fact that a coauthor already occurs in the citation set, i.e. occurs several times, respectively that a coauthor does not already occur in the citation set, i.e. occurs only once.

- $P(\text{Seen}|\text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper *with previously seen* coauthors given the event that C_i publishes in collaboration with someone.
- $P(\text{Unseen}|\text{Co}, C_i) = 1 - P(\text{Seen}|\text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper *with previously unseen* coauthors given the event that C_i publishes in collaboration with someone.

Estimating the addiction to collaboration with a particular coauthor.

- $P(F_{1n}|\text{Seen}, \text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper with the particular coauthor F_{1n} given the event that C_i publishes in collaboration with previously "seen coauthors".
- $P(F_{1n}|\text{Unseen}, \text{Co}, C_i)$ denotes the probability that the author with the canonical name C_i writes a paper with the particular coauthor F_{1n} given the event that C_i publishes in collaboration with previously "unseen coauthors".

Putting probabilities together. Now, the above probability components are put together to compute the total probability of the coauthor profile of the author with the canonical name C_i.

$$P(F_1|C_i) = \begin{cases} P(\text{No}|X_i) & \text{if } j(1) = 0 \\ \prod_{n=1}^{j(1)} P(F_{1n}|C_i) & \text{if } j(1) > 0, \text{where} \end{cases}$$

$$P(F_{1n}|C_i) = P(F_{1n}|\text{Seen}, \text{Co}, C_i) \cdot P(\text{Seen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i) + \\ P(F_{1n}|\text{Unseen}, \text{Co}, C_i) \cdot P(\text{Unseen}|\text{Co}, C_i) \cdot P(\text{Co}|C_i)$$

The justification for this composition is given in Section 5.3.1. Han et al. similarly estimate the probabilities that an author C_i writes a paper title ($P(F_2|C_i)$) and that an author C_i publishes in a particular journal ($P(F_3|C_i)$).

Details of Proceeding

Preparation of datasets for testing. The authors use the datasets as prepared in [HZG05]. Section 5.2.2 describes the preparation exactly. The datasets containing 15 different "J Anderson"s and 11 "J Smith"s are collected from the Web in the same way as in [HZG05]. From DBLP 9 large datasets are chosen similarly to the way in [HZG05]. Only the authors having a complete name in DBLP are used. In this manner, the citations are already disambiguated as long as there are no errors in the relevant part of DBLP. Another condition to the authors is that they have more than 5 citations. The datasets are preprocessed in the already known way by removing stop words and applying Krovetz's stemmer [Kro93]. Once more, journal abbreviations are replaced by their full names.

Conducted experiments. The classifier is evaluated according to cross validation: a dataset for a particular ambiguous name is randomly split into two parts, so that one part can be taken for training of the classifier and the other one for testing the classifier. The size of training sets varies between 2 to 27 citations per author for the "J Anderson"s, but for the datasets extracted from DBLP, the number of citations per author used for training ranges between 63 and 244. The size of the datasets used for testing is the same or slightly smaller. Since randomness is involved, each experiment is conducted 10 times.

Purpose of the conducted experiments. The results of the experiments are expected to allow drawing statements about the power of different feature types. Therefore, name disambiguation is executed using each feature type alone and additionally trying two kinds of hybrid models. "Hybrid I" computes the *equal joint probability of different features*, "Hybrid II" uses the coauthor information alone if a coauthor occurring in the test citation collaborated with an author having a canonical name. Otherwise, all three of the feature types are used. Secondly, the authors study the influence of the size of training datasets on the accuracy of name disambiguation. This is done by testing with different sizes of the training datasets. The authors start experiments with using 10% of available citations in a dataset as training data and increase stepwise by 10% until reaching 90%.

Evaluation and Discussion

The authors observe the following results.

Power of feature types. In the Web collected datasets, the classifier using only paper title information performs the best, whereas in the datasets extracted from DBLP, the classifier using only coauthor information performs best compared to the classifiers using another single feature type. This can be explained with the higher diversity of research areas in the Web collected datasets as follows. The smaller diversity of keywords in DBLP due to more similar and overlapping research areas disables a little bit keywords to be characteristic features that distinguish distinct authors.

In both types of datasets, the combination of different feature types according to "Hybrid II" is the best selection of feature types in both kinds of datasets. The following values are given for this best selection. In contrast to the datasets extracted from DBLP for which an average accuracy of 73.3% with a standard deviation of 5.4% is achieved, the 15 "J Anderson"s are quite well disambiguated achieving an average accuracy of 93.5% with a standard deviation of 1.8%. The 9 "J Smith"s are almost that well disambiguated with an average accuracy of 93.0% with a standard deviation of 2.1%.

Effect of the size of training datasets. Increasing the size of training datasets improves accuracy. The authors observe that using only 2% of the available data as training data already yield in acceptable results. This is important, since it is tedious to generate training data and the less is needed, the better for the approach in terms of usability.

About complexity. For k different authors having averagely N training citations with averagely J features, the authors report a computational complexity for training is $O(NJk))$. The computational complexity for the application of the model using only coauthor information is $O(Jk)$. Therefore, the approach scales to real-world applications.

To be continued. The described experiments are also conducted with another approach using support vector machines (SVM). This approach is presented in detail in the next Section. Therefore, the results comparing the two approaches, NBM and SVM, are described at the end of the next Section.

5.3.2 Support Vector Machines (SVM)

Together with the supervised naive Bayes approach of the last section, Han et al. study in [HGZ$^+$04] another supervised method. A support vector machine is trained and applied. The principle of support vector machines is explained in Section 4.2.2.

Problem definition

Given a full citation with the query name implicitly omitted, the authors' name disambiguation is to predict the most likely canonical name from the citation base.

Approach

Remember that support vector machines are classifiers that work for two classes only. This is due to their way of functioning. To apply a SVM to name disambiguation, it is necessary to extend it to a multi-class classifier. This happens by the "1-vs-all" approach (see Section 4.2.2), i.e. for k classes, k SVMs are combined where each SVM distinguishes between belonging to one particular class which is the positive of the two classes, and not belonging to this particular class (but to one of the others) which is indicated when a citation is classified as negative.

Features and Representation of Citations with Support Vector Machines

Similar to the spectral clustering approach described in Section 5.2.2, features are represented as weighted elements in feature vectors, where the weight of a feature in a vector is the frequency of the feature in that citation. Assume that the dimension of each feature vector is $m > 0$, i.e. in the whole set of citations occur m different features. Each feature vector $c_j \in \mathbb{R}^m, 1 \leq j \leq N$, is normalized by $|c_j|_\infty$ which promises improvement of classification as shown in [HGM$^+$03].

Clustering Algorithm

The idea of support vector machines is to separate data points in a vector space that belong to two different classes by placing a hyperplane in between them. This hyperplane should be optimal in the sense that it is as far as possible from the data points. In the case of name disambiguation, these data points consist of the feature vectors that represent citations.

Training Similar to the approach described in the previous section, the datasets are randomly partitioned in two parts of equal sizes, where the first one is used for training of the SVM, the second is used for testing. This principle is called *cross-validation* and described more exactly in Section 4.2.2. Assume that the training set consists of $\{(c_1, y_1), \ldots, (c_n, y_n)\}$ where n is the size of the training set and $y_j \in \{-1, +1\}, 1 \leq j \leq n$ denotes the correct class of c_j, i.e. y_j expresses on which side of the hyperplane c_j lies. In the case that the training data is *linearly separable*, this hyperplane is obtained by minimizing $||w||$ such that

$$y_j(w_j \cdot c_j + w_0) - 1 > 0 \; \forall i \leq n$$

Then, classification corresponds to applying the following *linear decision function*

$$f(x) = \mathrm{sgn}\{(w \cdot x + w_0\} = \mathrm{sgn}\{\sum_{i=1}^{s} \alpha_i^* y_i (x_i \cdot x) + w_0\},$$

where s is the number of support vectors. Remember that the support vectors are the (feature) vectors lying the most closely to the hyperplane. In the decision function, this is expressed by the fact that the coefficients α_i^* of support vectors are unequal to 0. The decision function than works as follows. Is the value of $f(c_j)$ positive, than c_j belongs to the class $+1$, otherwise to the class -1. Obviously, different features are differently strongly discriminating citations. This fact can be used for further improvement by favoring such features that contribute the most for name disambiguation. The authors implement that idea by introducing an additional factor x_{ij} which is the weight of feature j in support vector i

$$\sum_{i}^{s} \alpha_i^* y_i x_{ij}$$

Details of Proceeding

Datasets used for testing. The experiments were conducted together with the experiments evaluating the naive Bayes classifier. Therefore, these experiments are already described in detail in Section 5.3.1. There are 11 datasets in total, two Web collected datasets for the names “J Anderson” and “J Smith” and 9 datasets extracted from DBLP.

Conducted experiments. As already mentioned in Section 5.3.1, the evaluation principle is cross-validation. To get meaningful results, each experiment is conducted 10 times.

Purpose of the conducted experiments. The experiments evaluating the SVM approach support the statements drawn from the results of the NBM approach in Section 5.3.1. A difference lies in the definition of the “Hybrid I” scheme of composed feature types. Different feature types are combined in the same vector space. The scheme “Hybrid II” is not defined for SVMs.

Evaluation and Discussion (Continuation)

The following results also concern the NBM approach of Section 5.3.1 and constitute the second part of results obtained in [HGZ+04].

NBM better exploits the coauthor information. When using coauthor information alone, the NBM approach outperforms the SVM approach. Han et al. give as reason that the support vector machine is not able to involve information about "unseen coauthors" (see Paragraph 5.3.1), whereas the naive Bayes model captures both, probabilities of "seen" and "unseen coauthors".

SVM vs. NBM. Using other features or combinations makes the SVM doing slightly better than NBM in the Web collected datasets for the two names "J Anderson" and "J Smith". Han et al. give as a possible reason that SVMs look at *all* data points used for training "at once" to construct an optimal separating hyperplane between the two classes. In contrast, the naive Bayes model considers only the citations belonging to *one* class. Therefore, the naive Bayes model does not rank the features according to the power of the feature. In contrast, the SVM is able to recognize features that are unique to one class and make these features more important in the decision function.

Interestingly, the experiments on the DBLP datasets prove the naive Bayes model as the "winner" in all categories of used features, except for the category of using paper titles alone. Han et al. give as a reason that in the DBLP datasets, the portion of "unseen" features, i.e. "unseen" coauthors and keywords, is even bigger than in the Web collected datasets. Therefore, the NBM can make even more use of its advantage. Additionally, research areas in DBLP are close which leads to bigger overlappings of keywords between different authors. In consequence, the SVM probably ranks these features too low and does not make enough use of them.

Worse classification on DBLP data. Han et al. explain the poorer results on DBLP datasets as follows. First, by abbreviating first names of author names, they introduce ambiguity. Tests using a three letter abbreviation yield better results which shows that the approaches, both NBM and SVM, still have reserves to perform better when being fed with information of higher quality. The authors state that the data quality of the DBLP datasets is worse than the one of the Web collected datasets, which is the main reason that the approaches have more difficulties in classifying the DBLP datasets.

The second problem is that the research areas occurring in the DBLP datasets are more similar to each other and even may overlap. All authors occurring in DBLP are members to the computer science community, or at least closely related to. Therefore, it is more probable that they have common or similar keywords and meet each other on the same conferences than in the case of the authors occurring in the Web collected datasets. Han et al. concentrate on different scientific disciplines, have quite non-overlapping sets of conferences to participate in and surely use more diverse keywords. For example, a biologist would probably never write a paper having "P-NP problem" in its title. This lack of diversity makes classification more difficult.

Another problem is that DBLP is incomplete in the sense that there are not necessarily all citations of a person listed. Michael Ley, the founder and maintainer of DBLP, describes this fact as necessary and unavoidable. His insertion policy is to insert new data in groups of whole conferences and journal volumes, i.e. all papers published at one event are inserted completely and at once, requiring a huge amount of manual work and verification to assure data quality. Notice that citations are not "author-wise" inserted. Additionally, Han et al. take only citations in which the full names occur and leave out citations with abbreviated author names to avoid additional name ambiguity. The bad side effect is that citation data is even more incomplete. This incompleteness makes classification more difficult.

Furthermore, information on full publication venue titles is only incompletely available, i.e. it is not possible to reconstruct full titles from all abbreviated titles. Therefore, publication venue information cannot be fully exploited.

Finally, DBLP is good, but not without errors. Incorrectly assigned publications introduce misleading information that is used for classification.

5.4 Discussion

This section serves as a short summary of the previously described approaches. After presenting a comparison of the most important facts of the approaches, the focus lies on results concerning the character of name disambiguation and of disambiguation methods. The chapter is concluded with proposals for possible improvements.

5.4.1 Comparison of Approaches

Table 5.1 resumes the above presented approaches and provides a short comparison. Values that are not stated in literature are not given here ("–"). Accuracies are presented in combination with the number of homonymous authors that is given in parentheses.

5.4.2 Recapitulation of Results

Before our own approach is presented in the next chapter, a short summary of the properties of name disambiguation is given. These characteristics are revealed by the previously described methods.

Effects on Disambiguation Accuracy

Size of datasets. Disambiguation increases with the size of datasets.

Number of homonymous authors. Obviously, a large number of homonymous authors increases the difficulty of the disambiguation task.

Feature weight. tf.idf performs better than ntf. Finding other (even better) feature weights might further improve disambiguation accuracy.

Feature types. In general, the combination of different feature types yields in best accuracy results. Coauthor information seems to be the most useful feature type of CoTiVe, but the power of feature types depends on the kind of datasets, e.g. on the diversity of research areas of which the citations are taken. Therefore, it is important to have lots of different feature types that are "stable" in different situations. Intuitively, if one feature type is "knocked out" by a particular constellation of citations in a dataset, there should be other feature types involved that counterbalance the "knock out".

Quality of features. Obviously, the data quality in which features are provided has impact on the disambiguation accuracy: erroneous features may provide misleading information. Notice that data quality means also the completeness, i.e. unabbreviated versions, of features.

Diversity of datasets. The more similar the homonymous authors are, the more difficult becomes name disambiguation. Intuitively, very similar objects are not that easy to distinguish as very different objects. Here, especially the diversity of research fields is crucial. If

Properties	Unsupervised				Supervised	
Approach	NBM	Spectral Clustering	Blocking-based	Search Engine Driven	NBM	SVM
k known in advance	yes					
needed training data	no				yes	
underlying method	k-means/ EM-algorithm	QR-decomposition	sampling	HAC	Naive Bayes	support vector machine
used features	CoTiVe	CoTiVe	CoTiVe	URI's	CoTiVe	CoTiVe
semantical clustering of features	CBC/CBC	no	no	no	no	no
computational complexity	–	–	$O(\|A\| + \|S\|\|C\|)$	–	$O(Jk)$	–
avg. accuracy (%) computed with confusion matrix						
DBLP datasets (avg. k)	–/$\sim$ 54 (4–38)	61.5–64.7 (4–100)	–	86.3 (2–3) (datasets of computer science)	73.3 (11–35)	65.4 (11–35)
Web collected datasets (k)	$\sim$ 62 (6;9)/ $\sim$ 58 (11;15)	$\sim$ 88 (11;15)	–	–	$\sim$ 93 (11;15)	$\sim$ 95 (11;15)
accuracy (percentage/rank)	–	–	64 / 0.1 (DBLP & EconPaper) (2; $>$ 3)	–	–	–

Table 5.1: Overview of Approaches. J = #features, k = #clusters or classes, A = set of homonymous authors, S = sampling, C = set of citations

different authors have similar research interests, it is more probable that they share coauthors and keywords in titles as well as that they meet on conferences; a situation that might "knock out" three feature types at once.

Distribution of personal names. It is a fact that each nationality has a characteristic distribution of names. If frequent nationalities of researchers have a high variety of family names, disambiguation becomes easier.

Aptitude of Clustering Methods

The different experiments show that each approach has advantages and disadvantages. Some do not show any peaks in either direction of disambiguation accuracy, but perform constantly quite well. Others perform better in most of the cases, but yield quite poor results in few cases.

Furthermore, some scenarios can be handled by only few approaches. For example, naive Bayes approaches are the only ones considered in this paper that can exploit coauthor information of single authored papers, i.e. the coauthor information "no coauthors". All the other approaches fix the problem by using only information of other feature types.

Another aspect is the ability of a method to weight different feature types. For example, constructing a separate feature vector for each feature type instead of one for all feature types allows to compute the similarity measure as the weighted sum of "sub-similarities", one for each feature type. Otherwise, the feature types are only implicitly weighted by means of the number of features and by means of feature weights. Another example is observed by the authors of [HGZ$^+$04]: support vector machines better rank feature types according to their usefulness than the Naive Bayes model.

5.4.3 Concluding Remarks

The chapter is concluded by some remarks about possible improvements proposed by different authors end this chapter.

- Other feature types can be used additionally, e.g. authors' homepages, affiliations, email addresses, text analysis, e.g. bigrams,
- Other weights for measuring the importance of features, e.g. LogEntropy as described in [BYRN99] can be evaluated so that each feature can be exploited optimally.
- Already found results about the power of feature types can be used for weighting feature types.
- Techniques from Information Retrieval, e.g. string-based distance measures for use in duplicate string detection like the Levenstein distance and other edit distance (see Chapter 2), can improve the quality of features.

CHAPTER

SIX

A Graph Based Approach

In this chapter we present another approach to name disambiguation. Most of the approaches presented so far are based on the vector space model that we explained in Section 4.3, so we give only a short summary here. Remember that the purpose of the vector space model is to define quantitatively the term of “similarity” between citations. This goal is achieved in two steps. First, citations are represented in form of vectors, where each (weighted) element is associated with a particular feature occurring in at least one citation of the citation set. Second, we need to define a vector similarity, e.g. cosine similarity.

Our goal is to leave this vector space model for several reasons. We miss the *semantic* aspect of the given information and *semantic relations* between different pieces of information. We focus on semantics by introducing graph structures, e.g. the so-called *coauthor graph* for coauthor information.

First, we give a small excursion to *social networks* to motivate our approach. Then we critically check where room is left for improvements. In the second section we discuss promising features, how to combine them and how to integrate them into a graph-based approach. Two similarity measures involving coauthor information are integrated into a clustering algorithm in Section 6.3. This algorithm is constructed in a way that allows to easily integrate further similarity measures.

6.1 Motivation for a Graph based Approach

6.1.1 Social Networks

In the last years, research on social networks boomed. A good introduction and overview is given by Duncan Watts in his book [Wat03]. The term “social network” denotes graphs in which individuals are connected by some defined relation: the vertices represent individuals, edges stand for the considered relationship.

Analysis of social networks is applied to communities in the environment of computer science, too. An example is given in [HH04]. The authors use data from DBLP to construct a coauthor graph representing a community that is interested in reengineering, maintenance of software and software engineering in general. Using this coauthor graph, the authors are able to analyze the structure of the community, their habits to collaborate and compare their results with other research fields inside and outside of computer science. Their analysis reveals that the community of Software Reverse Engineering, too, is a small world. It is interesting to analyze the largest component. It starts growing slowly and then accelerate the growth. This is explained as follows. Researchers need some time to get integrated in the community and start to collaborate with others. The community is compared to the one of MedLine for medical science and SPIRES for physics. It is interesting that each field has its own characteristics. For example, the average number of coauthors is significantly higher for medics and physicists than for computer scientists. Other results concerning DBLP were already given in the introduction in Section 1.1.

Thus, DBLP is reflecting a part of a bigger community since DBLP does not contain all publications that are produced by the computer science community, which is perhaps impossible because the borders of this community are fluent. This community aspect is getting more and more conscious to the community itself and its members, having a graph as underlying basic structure. Therefore, the motto of the approach is to consider all aspects embedded in graph structures.

6.1.2 Working Points in Prior Work

To resume the previous chapter, the main approach to name disambiguation is as follows: choose some features to represent publications as vectors (see Section 5.1 for exact description). The features normally are weighted by some factor, e.g. tf.idf, finally the similarity of two publications is computed as the cosine similarity of the two vectors representing the publications. This similarity is used as similarity measure for clustering. There are four aspects which need to be examined to see if and how much space is left for improvements.

Clustering algorithm. Each clustering algorithm has advantages and disadvantages and the decision which algorithm should be used has to be made for each use case.

Representation of objects. Data objects can be represented by features in different ways using different structures as, e.g. vector spaces, graphs etc. depending on the way in which the designer understands them. Room for choices is also provided by weighting functions for features, e.g. features can be weighted depending on their frequency in the whole set of objects or depending on local occurrences.

Feature selection. Clustering accuracy depends on the selection of features. One has to think about, which features allow to cluster or classify objects, i.e. which features are characteristic with regard to the structure of groups into which the data objects should be clustered.

Similarity measure. The construction of similarity measures depends on the way in which features are involved to represent data objects. Obviously, a similarity measure is defined in terms of the structure in which features are given. The similarity measure should mathematically formalize the intuition of similar objects. Instead of similarity measures, one could also use a distance function that computes the distance between data objects.

In the following it is examined if and to which extent the common approaches should be transferred unchanged to our own approach and where room is left for improvements or other perceptions which hopefully lead to some improvement, e.g. higher accuracy.

Clustering Algorithm

The chosen clustering algorithm does not seem to be the main problem. Clustering algorithms can only work as well as is the choice of features and similarity measures. In most cases, one could easily replace one algorithm with another one, as long as the new algorithm uses a similar representation of features (e.g. metric feature spaces). It would be a problem, if the algorithm uses categorical features which cannot be measured in the sense of e.g. Euclidean distance and which are not elements in an (Euclidean) feature space, whereas the replacement algorithm works on such a feature space. In such cases, either the representation of objects or the algorithm needs to be transformed if possible. Furthermore, research on knowledge discovery in databases already provided lots of clustering methods which can be easily applied. We decided to apply a modified hierarchical agglomerative clustering algorithm for reasons described later.

Representation of Objects

As already described, objects are represented by a set of so-called features that are significant in terms of making one object different to another one. As already described, features are commonly provided as elements in form of feature vectors, most often they are weighted in dependency of their frequency, the most common weight is tf.idf (see Section 4.4.3). Consequently, citations are considered to be vectors in a feature space. This representation of features and citations will be called "vector approach". This short nomination also emphasizes the contrast to the "graph approach" described later in this chapter.

In the vector approach, features indeed are weighted to account their power, but connectivity between possible feature values is not considered. In this work, a so called "graph approach" is examined. The idea is demonstrated by the following example: Figure 6.1 shows a "coauthor graph" that semantically combines coauthor features of different data objects. More exactly, the vertices of the graph represent the authors which are connected if and only if they have published together. The graph approach functions most naturally for this feature type, but there also exist meaningful graph structures for features of other types. This approach, too, allows weights, e.g. in form of weighted edges.

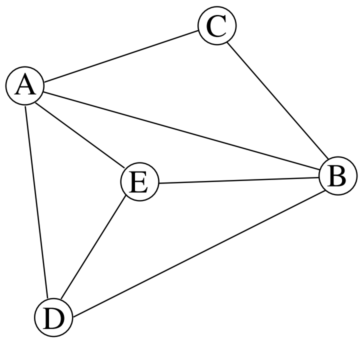


Figure 6.1: A coauthor graph

The idea is that by using graphs instead of vectors there is a gain of information in the sense of using not only the frequencies of occurrences of features, but also connections between features.

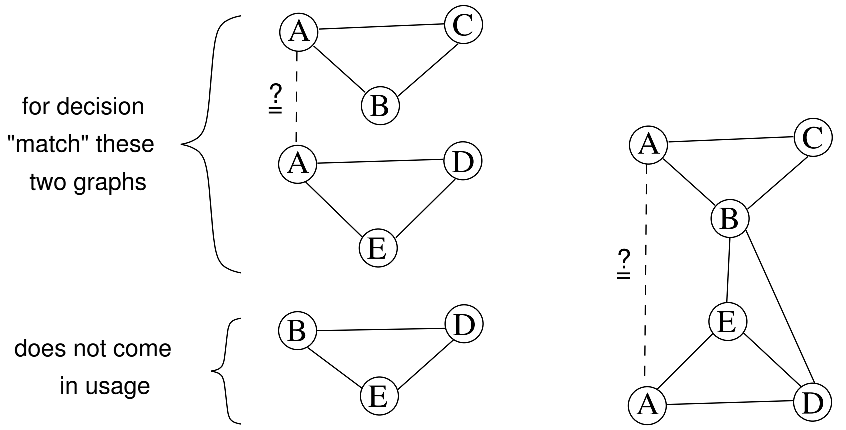
Example 6.1 *Consider the following simple example consisting of three papers.*

$$\begin{array}{lcl} A, B, C & . & P1 \\ A, D, E & . & P2 \\ B, D, E & . & P3 \end{array}$$

A is the author that is suspected to be homonymous, i.e. we do not know if the two occurrences of A belong to one author or represent two distinct persons. In Figure 6.1 it is assumed that A stands for a single person. For the sake of simplicity, we omit weights since they do not influence the main idea.

$$\begin{pmatrix}1\\1\\1\\0\\0\end{pmatrix} \quad \begin{pmatrix}1\\0\\0\\1\\1\end{pmatrix} \quad \begin{pmatrix}0\\1\\0\\1\\1\end{pmatrix}$$

Figure 6.2 illustrates how the graph derived of a vector representation looks like and compares it to the graph used in the graph-based approach. Notice that Figure 6.2(b) differs from the "normal" coauthor graph in Figure 6.1 in the representation of the suspicious author. In the graph-based approach, such authors are split into authorships and each authorship is represented by a vertex.



(a) A vector-based translation

(b) A graph-based translation

Figure 6.2: Translation of a vector-based representation into a graph-based representation

Feature Selection

Considering the structure of DBLP and its data, one notices that all information which is available for each entry is already used in existing approaches except the publication year. Notice that for several authors the URI of homepages are available, but not for all and it seems that this kind of information is less reliable. So it remains to explore, why the publication date is not used yet and in which way it could be involved.

Unfortunately, there is no other kind of information (e.g. email addresses) available for all persons in DBLP. For profiting of such information, one needs to involve other sources of information, e.g. the Web, ACM etc. As already stated, in this project usage of the Web in general should be avoided for reasons of complexity and diversity of the Web and limitations as the schedule of this project. Section 6.2 also includes promising features which are not used in this work for several reasons. In the first phase of this project, we explored the availability of data on the Web. Besides the difficulty of automatic extraction of data it can be said that it is even hard to retrieve relevant Web pages like personal homepages using a search engine, e.g. Google. After retrieving such relevant pages it is necessary to process natural text. This task is a wide research field of computational linguistics involving a diversity of techniques. Nowadays it still lacks of ready-to-use tools.

Similarity Measure

Generally, similarity measures depend first of all on feature representations. The goal is to quantify how similar two objects are. Often similarity measures formalize an intuitive understanding of similarity between some objects. It is a common way to represent objects as data points in a vector space, where each dimension contains the values of a single feature type. If features are given as vectors, a similarity measure is naturally expressed as a function on vectors, e.g. cosine similarity (see Section 4.4). Since we follow a different approach for representing features and citations, a completely different similarity measure must be defined, in terms of functions on graphs, e.g. involving the length of paths between vertices.

Therefore, the rest of this chapter is structured in a top-down way. First, a general way of using features to construct similarity measures is shown. A kind of framework is needed in which it is possible to combine different similarity measures.

The next step is to design concrete similarity measures. Since these measures are based on different types of features as e.g. coauthors, these categories on the level of features are also applicable on the level of measures to divide them in groups.

6.2 Promising Features

6.2.1 Combining Features

As already stated, it is difficult to gather more information from inhomogeneous sources as Web pages. This task is another interesting problem to solve but is not the object of this project. Since there is not more information available as in previous approaches, at least not covering whole DBLP, this section examines how to combine the given informations in another way than in the "vector approach".

As already argued, available information can be divided in three categories, namely coauthor information, keywords appearing in the paper title and keywords appearing in the venue title. This aspect can be formalized in a similarity function that combines these features.

$$\text{SIM} = \alpha \, \text{sim}_{coauthors} + \beta \, \text{sim}_{title} + \gamma \, \text{sim}_{venue},$$

$$\text{where } 0 \leq \alpha, \beta, \gamma \leq 1 \text{ are factors with } \alpha + \beta + \gamma = 1$$

The advantage of such a composed structure of the similarity is that the components of the similarity can be defined *optimized* for the particular feature type. The factors can be determined in an empirical way, i.e. by testing and finding optimal values for a given training set. One could argue that the requirement of such factors is unfavorable because it is difficult to find out values that are globally optimal, but on the other hand, this way of combining similarities (and therefore features) allows to easily substitute similarity measures and to examine them isolated and in different combinations. Furthermore, already existing results that evaluated the contribution of several features can be used to assign automatically factors regulating their influence.

6.2.2 Involving Coauthor Information

This section develops a graph based method to involve coauthors as features for clustering. After recalling the differences between vector and graph representation, different possibilities to exploit the coauthor graph are presented.

Vectors vs. Graphs

Recall how the common way of using coauthor information in clustering of publications looks like: the first step is to build a list of (canonical) names of all occurring authors in the relevant set of citations. Indeed it is a list (and not only a set), because the names are in some fixed order. A citation is now represented by a (large) vector, each element standing for one name of the author list. An element has value 0 if and only if the corresponding author does not appear as author in the citation (the vector stands for). An element has value 1, a weighted value (e.g. by tf.idf) resp., if the author is listed as coauthor of the paper in the citation. Considering only coauthor information as feature type, a vector v representing a citation c is element of $\mathbb{R}^n$ if the author list contains n elements, i.e. $c \in \mathbb{R}^n$. For the sake of simplicity weights are omitted for the next consideration since they do not influence basic considerations. As previously mentioned, this method is called "vector approach".

As sketched in the previous section, another perception on coauthor information (as well as in the following sections on other types of features) should be applied. It is quite natural to consider authors not as isolated objects, but as participants in a social network who cultivate relationships. The relationship of interest for this work is coauthorship, i.e. author A *"has published with"* author B. The social network of authors can be represented as a graph, where the vertices represent authors and edges represent the "has-published-with" relationship, shortly "coauthor relationship" or "coauthorship". In a more elaborated version, the edges can be annotated with weights (e.g. how often have two authors published together), objects representing the common publications etc.

Representation of Occurrences of a Name

There are two different possibilities to represent several occurrences of a name in a graph. Assume that there are some papers written by an author A, maybe together with other authors. Each such combination of an author A and a paper published by A is called an *"authorship of A"*. To be precise, we do not cluster coauthor graphs, but a mixture graph containing vertices that represent coauthors as well as authorships. In the case of *single name disambiguation*, a coauthor vertex represents several authorships in general: the authorships of that coauthor. In the case of *multi name disambiguation* coauthors are dealt with in the same way as the authors that need to be disambiguated: the differentiation between authors and coauthors does not exist anymore, only authorships occur. Indeed, there are two possibilities to represent the occurrences of a name.

The first possibility is to assume that all occurrences of A are related to a single person, i.e. construct a single vertex A possessing all edges resulting from the citations. This vertex therefore represents all authorships of all authors with name A. Then, the task is to split this node as long as there is a discrepancy between the graph and the real world / ground truth, i.e. as long as there are authorships of different authors represented in one vertex.

The second possibility is to process occurrences of A i.e. authorships of A in the inverse direction: construct a new node for each authorship of A, i.e. n publications published by author A will lead to n vertices labeled $A_1, \ldots, A_n$. The task then consists of merging vertices until as many vertices for A as real persons sharing the name A exist. This procedure reminds of agglomerative clustering presented in Section 4.1.2. That is why we decided to apply HAC.

It does not matter, what kind of representation is chosen. Both representations leave room for extending the algorithm of disambiguating one person to disambiguating several persons at once which was introduced as multi name disambiguation in the introduction. There is no significant difference in terms of complexity. The second way generates too many nodes of small size at the beginning whereas the first way constructs a more complex instance of the data structure implementing nodes. The structure of the later presented algorithm relies on the decision which representation is used. Since generating a vertex for each authorship followed by merging vertices is more intuitive, the algorithm will follow this direction and similarity measures will be defined in terms of separated authorships.

Indirect Coauthors and Coauthors of n^{th} Degree

As argued above, one can augment the amount of used information by involving indirect coauthorship. This idea can be generalized to coauthorships of n^{th} degree.

Augmenting the Degree of Coauthorship. One easily notices, that the graph representation uses more information. Naturally, the costs for providing this information must be paid, i.e. to disambiguate a name A, one needs to search not only for all citations containing this name, but also for all citations containing a coauthor of A. This leads to an extended search in DBLP and requires a larger amount of memory and time. The benefit is a gain of information. Intuitively, it should improve the similarity measure in such a way, that clustering accuracy will be augmented. Our experiments described in Chapter 7 show that our algorithm that uses solely coauthor information has the problem that authorships cannot be recognized as belonging to the same author because the coauthors are not connected closely enough. Indirect coauthorships are illustrated in Figure 6.3

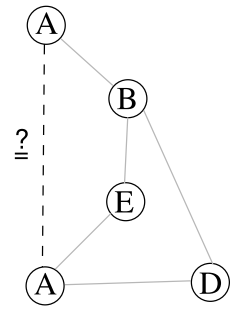


Figure 6.3: The "indirect coauthorship" relation

The network analysis performed in [EL05] supports the assumption that the close environment of an author, i.e. an author's working group, is left quickly with only a few hops. In consequence, an arbitrarily large number n of "intermediate coauthors" leads to the result that all authorships are connected with each other. Furthermore, the costs of computation will augment exponentially since the considered subgraph of the coauthor graph will grow exponentially in the number of nodes, whereas information gain will diminish and even errors will be introduced.

Intuitively, it seems useful to involve "indirect coauthors", i.e. intermediate authors of "first degree" (see Figure 6.3), but how far it would be useful to involve higher degrees is not that obvious. The figure illustrates this trade-off. It is only a sketch not resulting from measurements, therefore the axes are not annotated with any metrics.

Deriving a feature weight. To derive a feature weight, the number of direct and indirect coauthors can be used. The edges in the coauthor graph are annotated with weights expressing the number of coauthorships between the vertices that are incident with the edge. A similarity measure can take into account that the less tight two authorships are connected via coauthors, the less characteristic is the coauthorship. Values should be normalized to the interval $[0, 1]$ in the case of weighted features.

In our implementation we do not use weights for features. As explained in Chapter 7, indirect coauthorships probably do not introduce any errors. Therefore, there is no need to reduce their influence. However, weights may be useful for coauthorships of higher degrees. But such weights

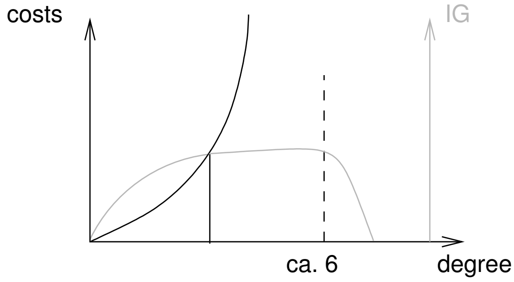


Figure 6.4: The trade-off between information gain and computation costs

would require an elaborated fine tuning to assure good results. Due to the limited schedule of this work, we had not the time to explore this further.

An Extension: Involving the Environment of Paths

It may prove to be useful to consider the *environment* of the considered paths (resulting from the hopping via coauthors), i.e. the subgraph in which the path is located. First of all, one has to determine an optimal size of the environment, secondly it can be defined in terms of vertices as well as in terms of edges. Here, the definition in terms of vertices is used. An environment of a vertex v is defined as the set of vertices that are reached by v by following a path of maximal length k for some $k \in \mathbb{N}$.

Definition 6.1 (Environment of a vertex.) *Let v be a vertex in a graph $G = (V, E)$. Then the k-environment for some $k \in \mathbb{N}$ is defined as the k-neighborhood of v, i.e.*

$$env_k(v) = \{w \in V \mid \text{there exists a path } P \text{ between } v \text{ and } w \text{ of maximal length } k\}$$

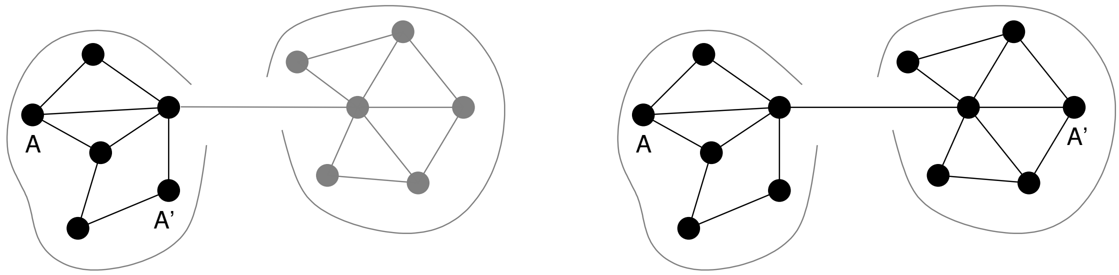
The definition of environments of *vertices* is extended to environments of *edges* as the union of the environments of all vertices belonging to the path.

Intuitively, the environment of a path influences the relevance of the path as follows: a longer path can be more relevant than a shorter path if the environment of the path is quite dense whereas the environment of the second path leads through a “desert”. The first situation in the real world would be a working group where two scientists are located at two extreme ends of the working group. Their connection passes a big part of the group. The second situation reminds more of a connection between two working groups which augments the probability that the two nodes, one in each group, represent two different scientists. Such “deserts” can be detected by finding bridges (as defined in Chapter 3) in the subgraph consisting only of the environments of all vertices on the path.

Now the question arises, how to measure the degree of cohesion of an environment so that finally, feature weights can be constructed. There are different possibilities, but most of them make computation complexity exponential or use approximations which leads to additional errors and lower accuracy. Due to the lack of time and therefore a missing *efficient* implementation of the basic coauthor similarities, we did not evaluate this extension in experiments.

Overlapping Environments

Another idea is to measure the degree of overlapping of environments. Imagine the environments of two authorships. Then consider what happens if one environment is overlaid with the other. The a huge amount of matching vertices implies that the two authorships probably belong



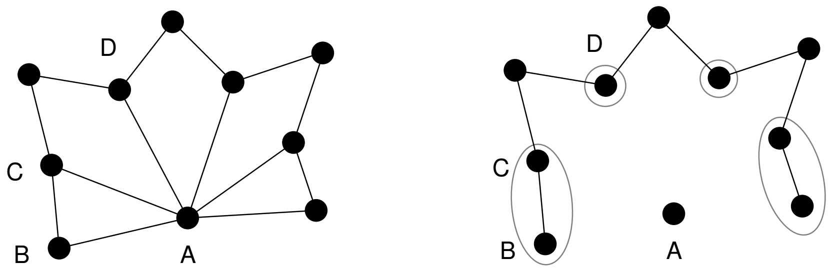
(a) Inside a working group (b) A desert

Figure 6.5: Two environments

to the same author. This approach reminds of the approach of the n^{th} coauthorship in combination with consideration of the environment of paths. We even argue that "overlapping environments" just is a colorful illustration of the extended coauthorship approach. There is a different point of view, since the coauthorship approach focuses the paths whereas overlapping environments focus the whole environment. However, formalizing overlapping environments leads to measures that are equivalent to measures resulting to the extended coauthorship approach. Due to the lack of time, we did not further investigate this approach and delay it including an implementation to the future. However, we argue that the environment does not help to improve our algorithm. The reason is discussed in detail at the end of this section about coauthor information.

Connected Components

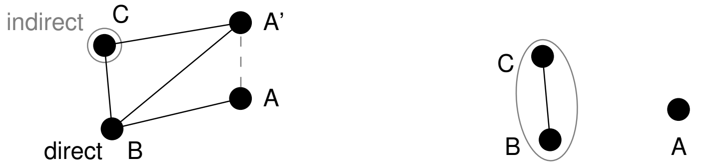
We realized that the following approach is applied in DBLP. The results are even explicitly given by the colors of the coauthors on an author's DBLP Web page. For this idea, it is easier to imagine authorships (that may belong to several authors) as a single vertex. Then the goal is to split this vertex so that for each author one vertex exists.



(a) An author and his coauthors (b) Connected component

Figure 6.6: Forgetting edges

Let's consider an author "consisting" of several authorships that is suspected to be homonymous. Figure 6.6(a) gives an example. Then, temporarily forget all edges in the coauthor graph that are incident with the author as illustrated in Figure 6.6(b). If only the subgraph consisting of the direct coauthors is considered, the subgraph may collapse into several connected components. In Figure 6.6(b) these connected components are marked by the grey ellipses. Notice that all vertices of a connected component have collaborated together at least once. This could be happened with a single paper, e.g. a paper having the authors A, B, C. The other case comprises several papers. Just imagine a set of three articles. In two of them the name of the suspicious author occurs together with one of the coauthors' names in each case, the third article was created by the two coauthors of the suspect without collaboration with the suspect, e.g. a paper by A, B, one authored by A, C and one by B, C. Such a situation was also given in Example 6.1. Even components consisting of only a single vertex are possible. For example a professor publishes together with his master student within the scope of the masters thesis and the student leaves the scientific environment after finishing his studies. The easiest way would be to consider all authorships of one connected component to belong to one distinct author. The example of the professor and his student gives already a counterexample in which this idea does not work.



(a) Coauthorship

(b) Connected components

Figure 6.7: Coauthorships and connected components

However, we argue that the approach of connected components is equal to the approach of direct coauthorships and indirect coauthorships. Just the point of view is different: in the approach of "direct coauthorships", all authorships of "suspicious" authors are treated as single vertices, one in each case and the goal is to merge vertices of authorhips belonging to the same person, whereas in the approach of "connected components", all authorships of "suspicious" authors are treated as a single vertex that has to be split. These different points of view make a comparison more difficult. The two figures in Figure 6.7 illustrate the equivalence of the two approaches.

In the case that the edge between the coauthors B and D are results from a collaboration without involving an authorship A corresponds to the case of indirect coauthorships (of authorships). The other case where the edge results from a common paper of all three authors corresponds to the case of a direct coauthorship.

We argue that our approach is more general or just simpler to describe in the sense that it suffices to augment the degree of coauthorship to change the similarity measure and we involve only paths between authorships in the basic approach without environments. In the approach of connected components, one has to enlarge the environment so that the considered subgraphs decomposes into fewer connected components. However, which approach is chosen depends on the decision if vertices are split or merged during disambiguation.

Limitations of Coauthor Information

Availability. A general problem of involving coauthor information is that it provides no information for publications written by a single author. Obviously, this authorship has not any coauthorships. Therefore, it is impossible to make a statement concerning the belonging to a particular author using only coauthor information.

Usefulness. The second problem was already sketched when discussing connected components. In the real world, the coauthors of an author usually do not stick together in one connected component. Therefore, a disambiguation algorithm should involve other feature types.

Since our implemented algorithm has not any problems in terms of leaving authorhips of distinct authors mixed up as a single author, but inversely with the recognition of authorships that belong together. Environments constitute a remedy for problems of the first kind. They restrict the influence of paths and therefore make authorships more different. What we need is the opposite. some criteria that make authorships that are unsimilar according to coauthorships more similar. It seems to be most promising to increase the degree of coauthorship and, of course, involve feature types that are not based on coauthor information.

6.2.3 Involving Information Concerning the Publication Venue

We developed two possibilities to transform the vector based usage of publication venues to a graph based usage. The goal is to provide further semantical information concerning the relationship between publication venues.

Entry Points of DBLP

DBLP provides a coarse categorization of publication venues in form of 18 *entry points*[1], e.g. "Algorithms", "Information Retrieval", "Digital Libraries" etc. These entry points provide a first structure in form of a tree that is very shallow: there is a (virtual) root, 18 inner nodes representing the entry points on one level and leaves representing the venues.

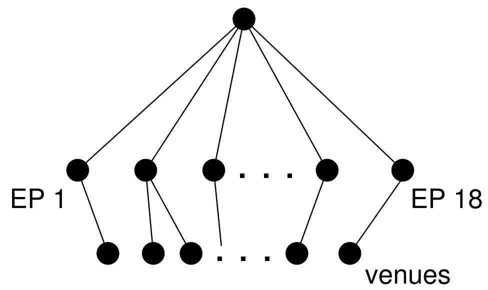


Figure 6.8: The tree structure of entry points

Then two authorships of publications at venues reached by the same entry point are considered to be more similar as two authorships of publications at venues reached by different entry points. A problem are authors that research at the borders of such entry points: it is probable that they publish at venues lying in some entry points next to the border. The existence of such authors in turn tighten a relationship between these entry points and particular venues in them. We call such relations "interconnections" expressing that there are relations between entry points. As a consequence, the tree structure of Figure 6.8 should be enriched with horizontal edges on the level of the leaves and if the number of such horizontal edges passes some threshold, also between the corresponding entry points on the level of inner nodes. Another improvement would be to use some more detailed ontology on publication venues, but as far as we know, there does not exist any. For example, the ACM portal provides such ontologies, but the leaves are missing or rather the edges from the leaves to their parents. It would be nice to have some algorithm that is able to extend the tree also into the depth. This task is equivalent to the one of automatic generation of ontologies.

Following up the idea to connect venues that have common authors leads to an undirected weighted graph which is illustrated in Figure 6.9.

Publication Venue Graph

The idea is to construct a graph where vertices represent publication venues and two vertices are connected by a weighted edge if there are authors that have published at both events. Based on the number of such authors a (normalized) weight for the edge is determined. Then the

[1]http://www.informatik.uni-trier.de/ ley/db/subjects.html

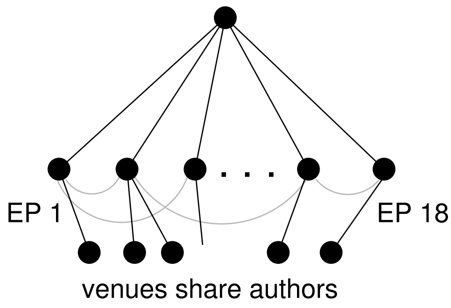


Figure 6.9: "Interconnected" entry points

similarity measure between two publication is defined as the sum of weights of the shortest path. A simplified version considers an unweighted graph and the shorter the shortest path between to vertices is, the more similar are these vertices.

Involving the information about publication venues might improve clustering. As already seen, features of this type approximate the research area of authors. From the graph point of view, information about publication venues may improve our clustering algorithm in the following way: as already illustrated in Section 6.2.2, our clustering algorithm has difficulties when the coauthors of a homonymous author can be partitioned into several groups between which there are not any edges in the coauthor graph. In such a situation, an author looks like several authors, but in reality it may be the case that there is only one person that collaborates or collaborated with different sub-communities, e.g. working groups. Publications with students, e.g. within a master thesis, are examples of publications that can not be assigned correctly. In such cases, further information as the one about publication venues may help.

The main problem of this idea lies in the computation of shortest paths which is expensive. Due to the lack of time, we did not implement such venue graphs or a tree-based measure that involves this kind of information. Therefore, no experimental results about usability and functionality (in the sense of improving accuracy) are given here. Further work can explore the possibilities and limitations of such measures.

6.2.4 Involving Publication Titles

Analogously to publication venues, keywords from publication titles, too, can be evidence for research topics. The integration of title keywords into a graph-based approach requires more effort. The idea is to build an ontology. Then, the similarity between two keywords can be measured similarly to publication venues based on the graph: the length of the shortest path expresses how near two keywords are; the shorter the path, the higher the similarity.

The ACM Computing Classification System

ACM[2] created a taxonomy of computer science which is available in different formats, e.g. as a hypertext or as an XML file. Currently, there are 11 top levels, e.g. "General Literature", "Hardware", "Mathematics of Computing" etc., each one has one or two sublevels. Publications at ACM have keywords that are explicitly given and integrated into the taxonomy. Therefore, this taxonomy can be used directly for ACM publications. Since we decided to focus on DBLP because of its high data quality, we did not further explore this taxonomy. Finding a way to make it "globally" usable is an interesting research topic. Once again, it is missing a generic algorithm that can automatically integrate arbitrary keywords into the taxonomy. With such an algorithm,

[2]http://www.acm.org

it would be possible to extract keywords from a publication title and use them to classify the publication with this title. Or, to integrate keywords into the graph-based approach, it would be possible to define a tree-based or graph-based similarity measure according to the taxonomy.

Generating an Ontology

Keywords coming from publication titles can also be put into a semantical relation and presented as a connected graph. Vertices represent keywords and edges can have different meanings. In the simplest approach, the graph is a tree and edges present a kind of "is a"-relation. The leaves of the tree represent the keywords. Upper levels depict a classification, e.g. the taxonomy of ACM. As already stated, it is necessary to integrate the keywords as leaves. For usability of the taxonomy, it is important that it is constructed *generically*. Then, it can be extended automatically. A manually built taxonomy would serve as well at the beginning, but is not of scientific interest and not the task of this work.

We explored other information sources such as e.g. dmoz[3] an open directory project trying to classify Web content. The idea of dmoz is analogously to Wikipedia[4] that anyone can become an editor and help in the enlargement and improvement of the project. Unfortunately, some searches for keywords coming from titles of publications showed that the granularity is not fine enough. This is a very interesting, important and challenging research topic that should be explored in the future.

6.2.5 Other Features Types

Some promising types of features have not been considered yet, neither in other approaches nor in our approach. We discuss here how they can be involved.

Affiliation. The term "affiliation" denotes the institute at which someone works. For researchers, this can be some university, research departments of enterprises, e.g. SAP Research[5], or other research institutions, e.g. there are many "Fraunhofer Institutes" in Germany which are organized by the "Fraunhofer-Gesellschaft"[6].

It seems not very probable that two homonymous researchers work have the same affiliation. Therefore, this is a promising feature candidate which helps to distinguish different persons and therefore increases precision on the one hand. On the other hand, recall might get worse. This can be seen if one considers the life of a researcher. At least in Germany, it is impossible to stay at one institution for the whole research life. A researcher has to leave his "home university" at least after finishing the Ph.D. studies because he cannot get any contract for an unlimited period. Therefore, "older" researchers will have published having several affiliations and two papers of one author that were published during different stations in the career, i.e. being employed at different institutions, will be falsely classified to belong to two persons. To avoid or at least reduce this effect, the publication year described below might help.

Email addresses. It is not possible that two persons share the same email-address (in the sense of having two separate post boxes). Therefore, an email address disambiguates a homonymous person using it. Analogously to affiliations, ambiguities can be introduced in the sense that a person may have "synonymous" email addresses and therefore might be considered to be several

[3]http://dmoz.org
[4]http://en.wikipedia.org
[5]http://www.sap.com
[6]http://www.fraunhofer.de

persons. For example, if a person changes the affiliation, his email address probably changes, too. Here, too, email addresses in combination with publication years might reduce the problem.

Personal homepages. The above statements hold true also for URIs of personal homepages. The URI of a personal homepage of an author is unique, but it may change with changing affiliation. The content of such homepages may produce further information that can be used. Then, additional techniques of processing natural text and of scoping with the inhomogeneity of the Web are needed. Since such techniques are not the focus of this project, we refer to related work and to future work.

Publication year. As sketched already several times, the publication year may explain differing affiliations, email addresses and homepages as well as the number of connected components in the coauthor graph and differing research interests. Notice that with the change of affiliation there is also a shift of research interests. Due to the lack of time given by the limitations of the frame of this work, we could not investigate how exactly the publication year can be involved. We propose to use the publication year in some feature weights.

FOAF data. FOAF[7] stands for "friend of a friend" and follows the idea to build up a social network involving technologies of the Semantic Web, namely RDF, In [AMNR+06] it is described that a large amount of FOAF data of authors occurring in DBLP can be found using the search engine SWOOGLE[8] which searches the semantic Web. In the FOAF network, every individual is identified uniquely, therefore name disambiguation can be solved if authorships can be assigned to FOAF identities. Searches for homonymous names using SWOOGLE showed that there is not enough FOAF information for ambiguous authors available. Therefore, we did not further investigate this approach but refer to future work.

6.2.6 Summary

Concluding this chapter, we give a summary of the presented approach. The main idea that is followed during the whole chapter is to process graphs.

We described three feature types in detail and proposed additionally other promising feature types. These feature types can be categorized in another way according to a particular property. We noticed that some of them differ in the fact if they depend of the actual database or depend of some "god given" knowledge that is independent of a database. For example, coauthor information totally depends on the actual state of the database. Relations between authors change with each insertion of new articles. In contrast, an ontology of computer science integrating title keywords into a tree is "god given", i.e. generated independently from the database. For example the fact that there are two entry points or top level categories "Software" and "Hardware" does not change with insertion of new keywords. Therefore, the distance of a keyword of one category to a keyword of another category does not change when inserting new keywords as long as the ontology is a tree and there are not any edges depending on the existence of leaves except the edges incident to leaves. Finally, there are hybrid feature types that show both properties, e.g. a graph-based involvement of publication venues uses a classification of venues enriched with information extracted from the venue graph or even the coauthor graph. Table 6.1 summarizes the feature types that were presented in this section.

[7] http://www.foaf-project.org/
[8] http://swoogle.umbc.edu/

Feature	**Availability**	**Usefulness**
Coauthor information		
coauthor graph	yes	most probably
Paper title keywords		
ontology on keywords	no	to examine
Venue title keywords		
Entry points	yes	to examine
Venue Graph	to be constructed	to examine
Other information		
publication year	yes	to examine
affiliation	no	to examine
email address	no	to examine
homepage	partly	to examine

Table 6.1: List of feature candidates

6.3 A Graph Based Clustering Algorithm

The algorithm that is developed in this section is based on hierarchical agglomerative clustering which is described in Section 4.1.2. This agglomerative approach fits well to our approach that considers each authorship to belong to a distinct author at the beginning and then merges authorships that belong in reality to the same author. Furthermore, HAC is a general algorithm by means that it requires only the existence of some similarity measure between objects and between objects and sets of objects (and between two sets of objects). Therefore, it provides a simple way to test different similarity measures. The algorithm is an unsupervised approach which does not need any training data. Furthermore, the clustering algorithm does not depend on any randomized initial partition. Therefore, it suffices to conduct experiments only once since HAC is more or less independent from the order of data objects. We say “more ore less” since this property depends also on the similarity measure and used feature types and if there is a propagation of values if some similarity value changes.

We adapted HAC to cluster graphs in the following way. The constructed coauthorship/coauthor graph is enriched with additional edges between coauthorships that are weighted with the appropriate similarities. Then, the algorithm works as follows:

1. Construct the similarity graph by computing all similarities.
2. Take the pair of authorships or of sets of authorships that is most similar to each other, i.e. take the two vertices of the similarity graph that are connected by the highest weighted edge.
3. Merge these authorships, i.e. contract the edge with the maximal similarity value. The resulting vertex contains the authorships of the two vertices that were incident to the contracted edge.
4. Update the coauthorships edges and similarity values.
5. The algorithm terminates if all similarity values are equal to 0 or ideally if there remain only k clusters, where k is the given number of clusters. Otherwise, continue with the next iteration starting at step 2.

Alternatively to the assumption that the correct number k of clusters is given, one could determine some threshold that a similarity measure must pass. If the maximal value does not pass this threshold, the algorithm terminates. We used the former version, i.e. knowing k in advance since it seems tedious work to determine an optimal value for the threshold that may depend on the similarity measure.

CHAPTER

SEVEN

A Graph Based Clustering Algorithm

This chapter describes the realization of some of the ideas of the previous chapter. The logical design is illustrated, details about the implementation are given and finally, some experiments are reported that help to evaluate the approach. Notice that the intention is not to provide some ready-to-use application or tool, but only to justify the theoretical idea of the previous chapter by giving a proof of concept.

7.1 Design of a Clustering Algorithm for Graphs

7.1.1 Requirements

The task is to design an algorithm that processes an undirected weighted graph. The graph has two different kinds of (weighted) edges. Edges of the first sort mean "has coauthored with" and are therefore called *coauthorship edges*, whereas the second type of edges, *similarity edges* connects all vertices representing authorships with the author name in question (i.e. with the ambiguous name). The vertices of the graph represent authors and authorships. Authors are considered to be sets of authorships. Coauthorship edges are provided by publications and publication venues, similarity edges are added by computations based on the coauthorship edges according to the chosen similarity measure. Figure 7.1 presents a graph with three authorships A, A', A''. The grey edges are similarity edges. In this work, such a graph is called an *extended coauthor graph*.

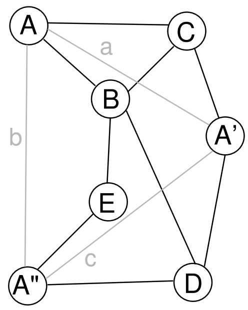


Figure 7.1: An extended coauthor graph: a, b, c denote the weights of the edges that correspond to the similarity values between the authorships

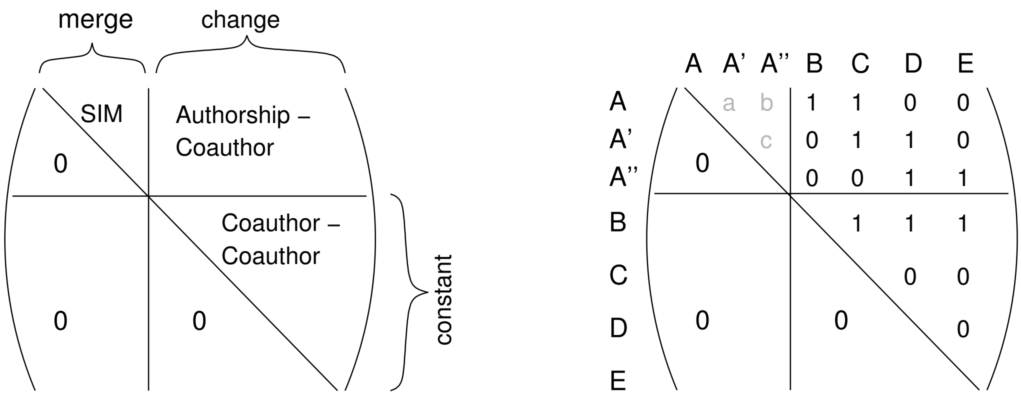
We decided to represent the graph by means of a modified adjacency matrix. The matrix representing the graph in Figure 7.1 is illustrated in Figure Remember that an element a_{ij} of an adjacency matrix $\mathcal{A}$ is set to 1 or to the weight of the corresponding edge if vertex i is connected with vertex j by an edge in the graph. Both types of edges are stored in a single matrix. Since edges representing coauthorships as well as edges expressing similarities are undirected, the adjacency matrix is symmetric. Therefore, we just use the upper triangular matrix.

7.1.2 Data Structures

Several representations of graphs are possible. Most naturally, but maybe expensive in memory consumption, would be an object oriented approach that directly models graphs as compositions of objects representing vertices and edges. The classical way is to use matrices, e.g. adjacency matrices. In this thesis graphs are represented by adjacency matrices since they are flexible and we are not bound to design decisions made by the authors of graph libraries.

Extended Adjacency Matrix

Since edges of the coauthor graph are undirected, the corresponding adjacency matrix ${}^{n}\mathcal{A}^{n}$ is symmetric. The dimension n is the sum of the number of authorships plus the total number of coauthors that are involved in an article of an authorship. The symmetry of the matrix allows to ignore the lower left part of the matrix. It suffices to maintain the upper triangular matrix. Notice that the trace of the matrix, too, is not needed. This fact is expressed in Figure 7.2 that illustrates the modified adjacency matrix by a general representation and by a concrete example. The matrix is then divided into three parts in which the values have different meanings. The parts are named after the types of operations applied to them, another possible naming is after the type of edges they contain. The matrix is called *extended* because there are two different kind of edges represented in the matrix.



(a) General illustration of the extended adjacency matrix

(b) The extended adjacency matrix representing the coauthor graph of Figure 7.1

Figure 7.2: The parts of an extended adjacency matrix

merging part / similarity part The upper left part contains only similarity adjacencies between authorships that are not fully merged to disambiguated authors. In this part of the matrix, merge operations will be applied.

changing part / mixed coauthorships part The upper right part contains coauthorship adjacencies between ambiguous authorships and disambiguated authorships (more exactly sets of authorships).

constant part / disambiguated coauthorships part The lower right part contains only authorship adjacencies between disambiguated authorships, i.e. coauthors. Therefore, no changes will be applied to this part. It does not come into play if only direct coauthorships are considered by the similarity measure, but if indirect coauthorships are considered.

Sets of Authorships

The collection of authorships to disambiguate is ordered, i.e. authorships have fixed positions in a list. Keeping in mind that edges between authorships are contracted during clustering, authorships are given as nested lists. The index of an authorship (list) in the list is the same as in the matrix.

Operations on the Matrix

The clustering algorithm is composed by the following operations that are applied to different parts of the matrix and to the set of authorhips:

merge This method merges two vertices representing different authorships. The tasks include *contracting* an edge connecting two authorships, *merging* the two authorships in the set of authorships and *updating* the matrix.

- **contraction** This method contracts an edge, i.e. eliminates one row and one column in the matrix and updates the other row and column.
- **mergeAuthorships** This method combines the two authorships in a single list and deletes the two old lists.
- **update** updates values in the similarity part of the matrix and in the mixed coauthorships part.

find_max This method returns the edge (i.e. pair of vertices or entry in the adjacency matrix) with maximum similarity value.

7.1.3 Similarity Measure

In this work, two similarity measures are used. The edges of the coauthor graph are considered to be unweighted for the benefits of simplicity.

"direct coauthorship" similarity measure The similarity between two authorships or sets of authorships is measured in terms of the number of common direct coauthors. If necessary this number can be normalized.

"indirect coauthorship" similarity measure The similarity between two authorships or sets of authorships is measured in terms of the number of common indirect coauthors. If necessary this number can be normalized.

7.2 Implementation

This section gives an overview of the implementation of our approach. This implementation should serve as *proof of concept*. Due to the limited amount of time that was available for the implementation, there is room for improvements and optimization. A re-implementation is delayed to future work and may yield further interesting results: various similarity measures can be compared with each other. Notice that algorithms that process matrices are indeed scalable as the implementation of Google's PageRank demonstrates to the whole world.

7.2.1 A huge XML File: dblp.xml

In the Appendix the DTD of the dblp.xml is given in Listing B.1 as well as a small sample extracted from the dblp.xml in Listing B.2. These examples illustrate the structure of DBLP. The actual version of dblp.xml contains about 800000 articles and is of a size of about 350 MB.

The goal is to extract all relevant data that is needed for the disambiguation of authors having one particular name, e.g. "John Smith". Such an extracted data set is also called *name dataset*. After extraction, the whole dataset is held in memory during the entire clustering process. Different ways of extraction were tried.

XSLT

It was not possible to apply XSLT stylesheets to dblp.xml due to the size of the file. That is why we did not try other XML query languages such as XQuery, XPath or Xcerpt. It is probable that other languages are similarly limited regarding the amount of data.

"Pure" Java

First approach. This approach extracts only the data that is needed for the direct and indirect coauthorship measure, i.e. all publications (<article> and <inproceedings> elements) and publication venues (<proceedings> elements) of all homonymous authors, all such elements of their coauthors and all such elements of the coauthors of the coauthors. Duplicates are avoided. The chosen XML parser sequentially reads XML element after XML element. It is not possible to jump back. Therefore, several passes are needed: the first one that retrieves all publications of homonymous authors, a second one for retrieving all publications of coauthors and so on. In each pass, lots of control methods are executed that avoid duplicates. These methods are based on searching through collections. We used the Java Vector data structure for the collections because they provide lots of useful methods. The resulting parser was too slow, even a single pass took too much time.

Second approach. This approach extracts all data and hold it in memory during clustering. The parser reads the data file in a single pass and does not need to perform controls that avoid duplicates. Java on the used computer cannot handle this: an exception java.lang.OutOfMemoryError: Java heap space is thrown even after setting the size of the heap space to a maximal value that is recommended for a computer equipped with 1 GB RAM. This setting is done by starting the virtual machine with the arguments –Xms512m –Xmx512m –XX:PermSize=128m –XX:MaxPermSize =128m.

Java in Combination with a Database

PostgreSQL database. The final solution is to migrate the data of dblp.xml to a database system. We have chosen PostgreSQL 8.0 because Postgres has several advantages. For example, it is an *open source* database and free available. The resulting database consists of several tables. Queries that extracted only relevant data took about 400 seconds. There is potential to optimize the structure of tables, views and queries. Relevant data is extracted once and then saved persistently as serialized objects containing all relevant information. Serialization is done by implementing the serialized interface of Java. That is why we did not spend more time on the optimization of the database and queries.

Structure of the database. During the migration process some auxiliary tables were created that are not needed for the extraction of relevant data. In the following list, these tables are omitted. Only relations that come into use are listed.

```
                         List of relations
 Schema |                 Name                  | Type  | Owner
--------+---------------------------------------+-------+-------

 public | article                               | table | klaas
 public | article_authorship                    | table | klaas
 public | article_authorships_nosingles         | view  | klaas
 public | article_nosingles                     | view  | klaas
 public | author_nbarticles                     | view  | klaas
 public | author_nbarticles_nosingle            | view  | klaas
 public | combined_authorship                   | view  | klaas
 public | combined_authorship_nosingles         | view  | klaas
 public | venue                                 | table | klaas
 public | venue_authorship                      | table | klaas
 public | venue_authorship_nosingles            | view  | klaas
 public | venue_nosingles                       | view  | klaas
```

Listing 7.1: Relevant tables and views

The most important relation is combined_authorship_nosingles which is a view resulting from the combination of article_authorships_nosingles and venue_authorship_nosingles. The appendix of names of relations _nosingles expresses that authorships concerning papers that are authored by a single person are omitted.

Queries For processing coauthor information, we extracted the required data from only one view, namely from combined_authorship_nosingles. There are two types of queries. The first one retrieves the data concerning all homonymous authors plus relevant data for a name, e.g. "V Kumar". Queries of the second type retrieve the data concerning a pair of homonymous authors plus relevant data for a name, e.g. "Vipin Kumar, Vijay Kumar". This second type of query is shown in Listing 7.2.

```sql
SELECT DISTINCT * FROM combined_authorship_nosingles
WHERE key IN
   (SELECT key FROM combined_authorship_nosingles
    WHERE author IN
      (SELECT author FROM combined_authorship_nosingles
       WHERE key IN (SELECT key FROM combined_authorship_nosingles
       WHERE author IN ('Vipin Kumar','Vijay Kumar'))))
ORDER BY key
```

Listing 7.2: SQL query: relevant records for a pair of names include indirect coauthors

7.2.2 XML Parser

The parser is based on the XMLPullParser[1] that implements the XMLPull API. The parser functions event-driven, i.e. the class DblpParser3 has to implement the actions for the possible events

[1]available from http://www.xmlpull.org

START_DOCUMENT, START_TAG, TEXT, END_TAG and END_DOCUMENT. It is the parser that asks for the next event. This can be seen as a special kind of iterator. The call of the method next() returns the next event. The parser then processes this event.

The output of the parser are text files. Each line represents a record that is inserted into the database in the next step, e.g. a data record for an authorship has the form name|articlekey, a sample record is Virginia Mary Lo|journals/ac/Lo94. The character | separates the fields of one record. The database is filled by using the COPY command of PostgreSQL that copies the content of such text files into tables.

The classes that are used by the parser were refactored and modified after the usage of the parser. Since the parser was not needed any more, we did not adjust the parser. The source code of the parser is given in the appendix in Listing A.1.

7.2.3 In-Memory Representation of the Data

Author **and** Authorship

The classes Author and Authorship represent authors and authorships in Java. They are used by the DblpParser3 in the first migration step that writes data from the dblp.xml into text files as well as in the structures involved in the clustering process. An UML class diagram of these two classes is given in Figure 7.3.

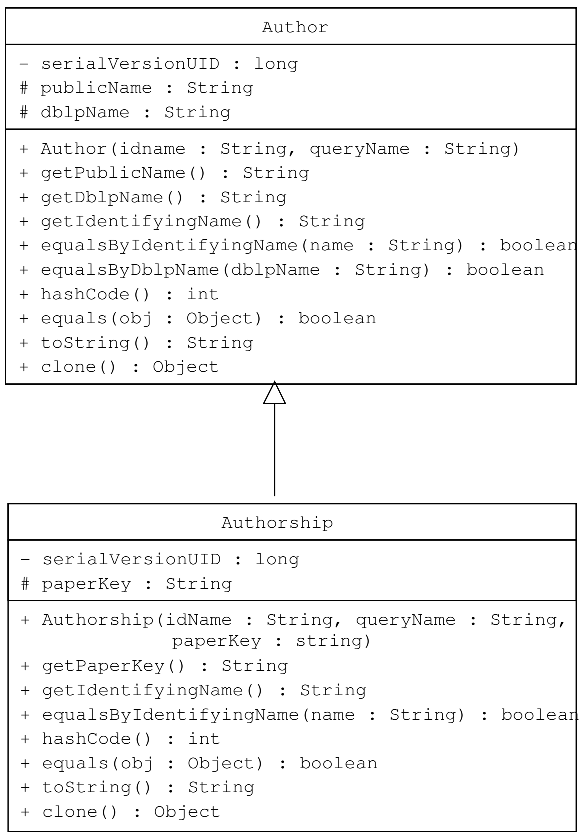


Figure 7.3: The classes Author and Authorship

Fields. The following fields characterize authors and authorships. The difference between them is expressed by the additional field paperKey belonging to the Authorship class.

dblpName The name that is used for an author in DBLP, e.g. "Vipin Kumar".

publicName The publicly visible name that is visible for the clustering algorithm, e.g. "V Kumar".

paperKey This field belongs only to the Authorship class and stores the key of the paper that belongs to the authorship.

Operations. Besides getters and setters, there are some interesting methods to compare authors with author names that are implemented differently in super- and subclass. A string is equivalent to an author if the dblpName is equal to the string. In contrast, a string is equivalent to an authorship if additionally the paperKey is contained in the string.

AuthorVector **and** ClusterVector

Structure of AuthorVector**.** The class AuthorVector is a structure that manages a list of authorships and authors. The structure is based on a vector of vectors of authors: Vector<Vector<Author>> . Notice that these vectors contain also Authorship objects. The idea is that during clustering such vectors of authorships are merged if these authorships belong to the same author. At the beginning of clustering, these vectors of authorships contain a single element. Not until the first contraction of a similarity edge comes the vector structure into play. The Matrix data structure that is described next has a field that points to an AuthorVector object to store the processed authors and the actual clustering structure. In Figure 7.4 an UML class diagram is given. Both the Matrix data structure and the AuthorVector constitute the core structures involved in the clustering process. However, the UML class diagram additionally shows the ClusterVector class because it is derived from the AuthorVector class. Notice that despite its structural similarity, objects of type ClusterVector are not involved in the process of clustering, but represent the result of a clustering process and support the evaluation process.

Operations of AuthorVector**.** Besides methods providing direct access to particular authors or vectors of authors, the most important methods are the following:

lastIndexofAuthorship returns the index of the highest index in the AuthorVector that contains a list of authorships.

contains returns **true** if a given Author object is contained.

addAuthor adds a given Author object (that may also be of type Authorship) preserving the order among authorships of the AuthorVector and avoiding duplicates.

mergeAuthorships merges two vectors of authors/authorships and removes the old ones.

Structure of ClusterVector**.** The subclass ClusterVector has additional fields to store the number of authorships, the correct number of clusters and the number of obtained clusters. Figure 7.4 also illustrates this. Notice that our clustering algorithm stops too early if the similarity measure drops to 0. As a consequence, too many clusters remain. Therefore, clusters are selected according to a heuristic that is described below. The constructor uses auxiliary methods to construct relevant objects that are stored in the following fields:

confusion points to an object of type ConfusionMatrix.

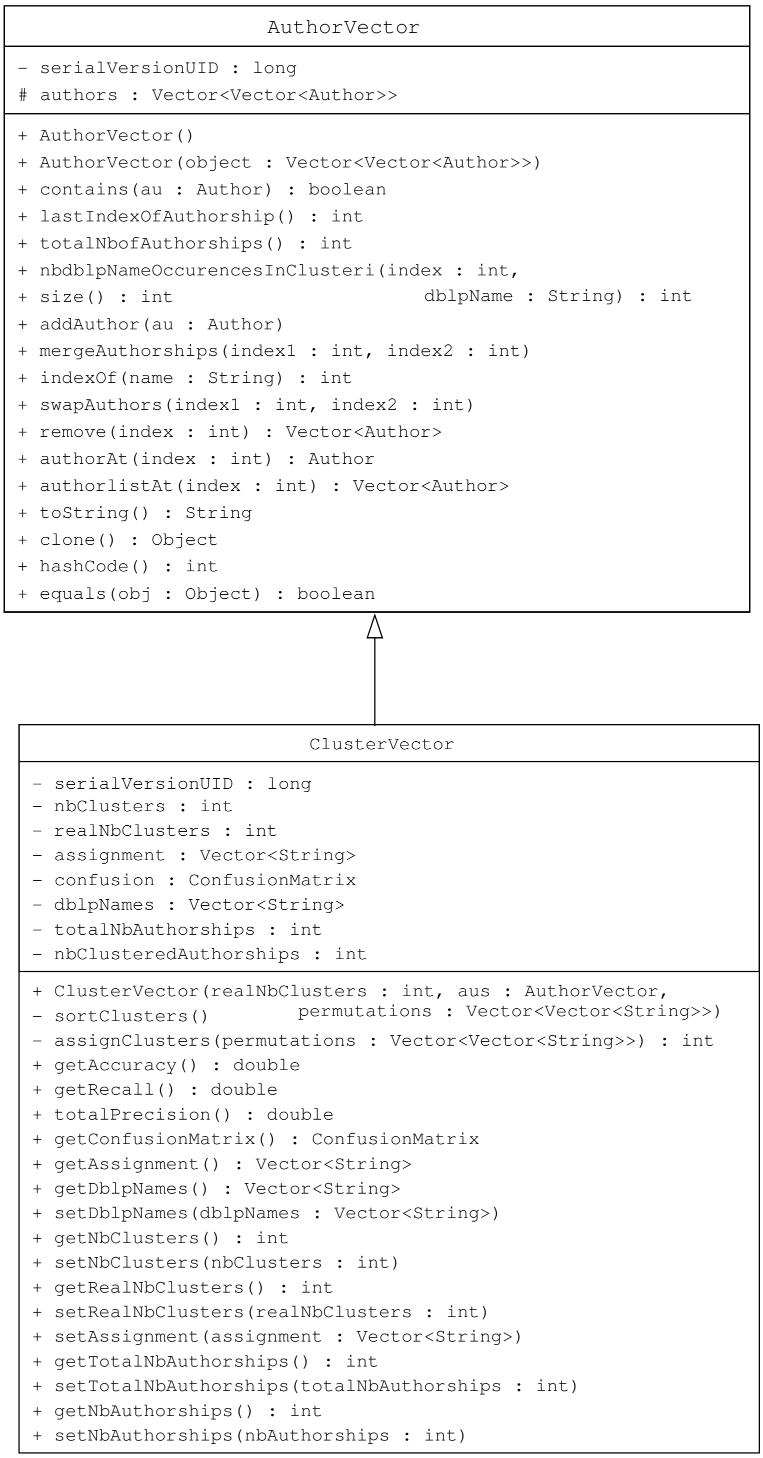


Figure 7.4: The classes AuthorVector and ClusterVector

assignment is of type Vector<String> and contains the (DBLP) names in the same order like the order of the clusters.

dblpNames of type Vector<String> contains the (unique) names as occurring in DBLP of the homonymous authors.

During construction, the assignment of non-ambiguous names to the obtained clusters is computed using a heuristic. Non-ambiguous names are taken directly from DBLP. The clusters are ordered by the number of name occurrences. First, the accurate cluster for the first dblpName is searched: the cluster in which the name occurs most often is taken. The clusters are ordered by their size in decreasing order. Then, the cluster for the second name is searched and so on. This heuristic is not optimal in all cases as Figure 7.5 shows. To assure an optimal assignment, the heuristic has to be applied for all permutations of dblpNames so that the best assignment can be chosen. If the correct number of clusters is obtained, one can compute all permutations of dblpNames and test which permutation is optimal regarding disambiguation accuracy.

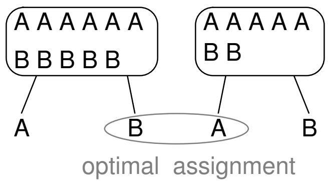


Figure 7.5: Sub-optimal assignment: A for the cluster on the left hand side, B for the cluster on the right hand side

If in the case of Figure 7.5 first A and then B is assigned, the assignment is not optimal: $6 + 2 = 8$ correctly clustered authorships, whereas an inverse assignment yields $5 + 5 = 10$ correctly clustered authorships.

Operations of ClusterVector. Besides getters, setters and auxiliary methods, the following methods are important:

totalPrecision returns the average precision of the considered clusters.

getRecall returns the proportion of authorships that belong to the selected clusters relative to the total number of clustered authorships.

getAccuracy returns the achieved accuracy.

The class Paper

The class Paper serves to store coauthor relationships and other features concerning authorships, e.g. publication title and venue. It is subject of future work to exploit such information in the implementation of other similarity measures.

The fields as shown in Figure 7.6 are self-explanatory, the class does not provide any special methods, but serves as an auxiliary class for parsing and for the construction of authors, authorships and of the matrix representing the coauthor graph.

The Class Matrix

Structure and construction. The matrix representing an extended coauthor graph is implemented by the class Matrix. The fields are the following:

```
Paper
----------------------------------------------------------
- title : String
- key : String
- authors : Vector<Author>
- year : short
- venue : String
- venueKey : String
----------------------------------------------------------
+ Paper(key : String)
+ addAuthor(name : String, fixAuthorname : String,
            isAuthorship : boolean) : Author
+ numberAuthors() : int
+ getAuthor(index : int) : Author
+ getTitle() : String
+ setTitle(title : String)
+ getVenue() : String
+ setVenue(venue : String)
+ getVenueKey() : String
+ setVenueKey(venueKey : String)
+ getYear() : short
+ setYear(year : short)
+ getKey() : String
+ hashCode() : int
+ equals(obj : Object) : boolean
+ toString() : String
+ venueString() : String
+ articleString() : String
```

Figure 7.6: Paper

matrix of type **double**[][] describes the actual matrix.

authors of type AuthorVector contains all authorships and coauthors. The i^{th} entry in the vector corresponds to the i^{th} column and row in the matrix.

sm of type SimilarityMeasure contains an instance of the used similarity measure.

dim of type **int** contains the actual dimension of the whole matrix.

dimSimPart of type **int** contains the dimension of the sub-matrix comprising the similarity part. This dimension is equal to the number of distinguished authors consisting of authorships. At the beginning, dimSimPart is equal to the number of authorships of homonymous persons. During clustering it is reduced, in the ideal case to the actual number of clusters, i.e. authors.

As shown by the class diagram in Figure 7.7, the constructor needs besides other arguments a numerical constant denoting the type of the similarity measure. The matrix is constructed using the similarity measure denoted by this constant.

In the evaluation framework a class MatrixGenerator exists that encapsulates the generation of matrices in a method with the signature Matrix newMatrix(**int** smid). The benefit is that only the MatrixGenerator needs to be serialized, i.e. saved persistently and (re)loaded for experiments. All data relevant for a name dataset is loaded at once into the framework. The similarity measure can be chosen dynamically. Thus, it is not necessary to save several matrices for each dataset, one per similarity measure, but just the MatrixGenerator. The construction of matrices is provided only by the constructor of the Matrix class and comprises the following tasks: the fields authors, dim and dimSimPart are set according to the actual parameters. A new two-dimensional array is created as well as the similarity measure corresponding to the actual parameter smid. Then, the matrix is filled with coauthor information and the similarity values are computed and set in the matrix.

Provided operations. The Matrix provides besides getters and setters further public methods that implement operations on the matrix that are needed during clustering.

```
                         Matrix
+ SIM_DIR_COAUTHORS : int
+ SIM_IND_COAUTHORS : int
- matrix : double[][]
- dimSimPart : int
- dim : int
- authors : AuthorVector
- sm : Matrix.SimilarityMeasure
---------------------------------------------------------
+ Matrix(dim : int, dimSimPart : int, papers : Vector<Paper>,
         authors : AuthorVector, smid : int)
- addPaper(p : Paper)
- conc(index1 : int, index2 : int) : double
- conc(d1 : double, d2 : double) : double
+ mergeAuthorships(index1 : int, index2 : int)
+ contraction(index1 : int, index2 : int)
+ update(index1 : int, index2 : int)
+ toString() : String
+ getIJ(indexI : int, indexJ : int) : double
+ setIJ(indexI : int, indexJ : int, value : double)
+ getIJ(nameI : String, nameJ : String) : double
+ setIJ(nameI : String, nameJ : String, value : double)
+ name2index(name : String) : int
+ fillSimilarities()
- genSim(id : int)
+ clone() : Matrix
+ sim(index1 : int, index2 : int) : double
+ hashCode() : int
+ equals(obj : Object) : boolean
```

Figure 7.7: The class Matrix (ordinary getters and setters omitted)

merge calls the mergeAuthorships method of the AuthorVector object.

update updates the coauthor information for merged authorships at index1 and index2 and recalculates the similarities for it; actually only the ones with index1 involved, but there may be similarity measures that require recalculation for all similarities.

contraction(index1, index2) contracts the edge (index1, index2) of the similarity graph: the similarity values in column index1 are updated; the matrix is copied into a new two-dimensional vector with one dimension less. Figure 7.8 illustrates this.

Similarity measures. Similarity measures are implemented as inner classes of the Matrix class that provide the method **double** sim(**int** i, **int** j) to compute the similarity between the authorship at index i and the authorship at index j. We implemented two similarity measures as shown in Figure 7.9. Listing A.6 contains the Java code of these two similarity measures.

"direct coauthorship" similarity measure To compute the similarity between two authorships at index i and j , it suffices to consider the *mixed authorship* part: count the number of columns that have values > 0 in both rows i and j. For example, in Figure 7.10(a) the authorships i and j share the coauthors C and F.

"indirect coauthorship" similarity measure To compute the similarity between two authorships at index i and j, both the upper and lower right part are considered. The relevant entries are located in the lower part: the black boxes in the example in Figure 7.10(b) mark these entries. The grey boxes mark the entries that are determined first, but since they are located in the lower triangular matrix that is not filled with values, the symmetric counterparts are taken. Values greater than 0 contribute to a similarity value ¿ 0 for indices i and j. Which entries are relevant is determined by the upper part: the coauthors of the authorships i and j are the ones that have at the indices i and j values greater than 0. The common coauthors

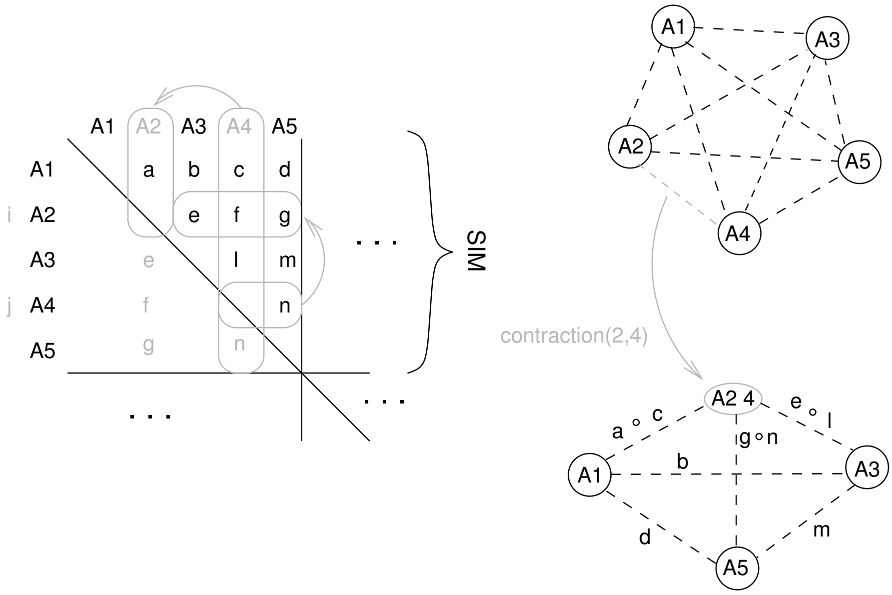


Figure 7.8: Contraction of an edge: the left hand side illustrates the manipulation of the matrix. $A4$ is concatenated with $A2$ by some operation $\circ$, the row and the column $A4$ is then deleted. The right hand side illustrates the effect on the graph. Coauthorship edges are omitted.

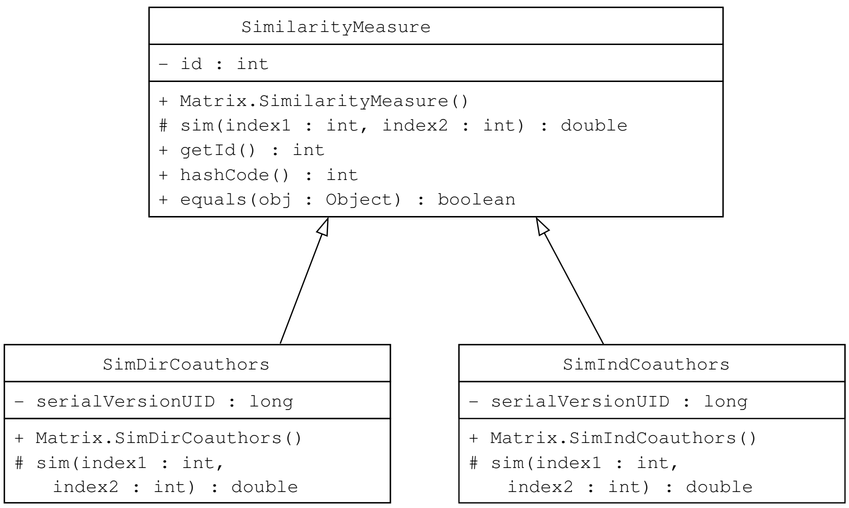
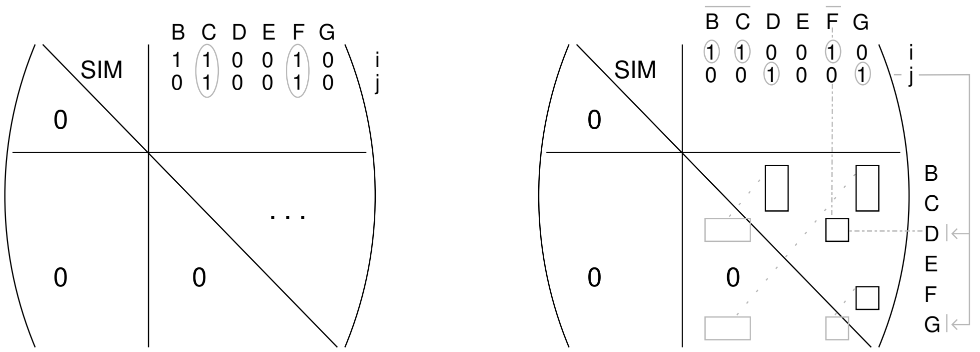


Figure 7.9: The classes for similarity measures: the sub-classes override the method sim according to the definition of the similarity measure

of these coauthors in turn are the common *indirect* coauthors of authorships i and j. For example, in Figure 7.10(b) author D is an indirect coauthor of authorship i if the values of the entry (F, D) is greater than 0.



(a) The direct coauthorship similarity measure

(b) The indirect coauthorship similarity measure

Figure 7.10: Direct and indirect coauthorship similarity measures

7.2.4 Clustering Algorithm

The clustering algorithm that is described in Section 6.3 is implemented in a straight-forward manner as the method cluster of the class Clustering which is shown in Figure 7.11. A Clustering object is constructed for a particular Matrix object. The cluster method clusters according to a given threshold or to a given number of clusters. The method stops before reaching the correct number of clusters if the similarity values drop to 0. The experiments are conducted using the variant that requires the correct number of clusters as input.

Clustering
- matrix : Matrix
+ Clustering(mat : Matrix) + cluster(treshold : double) : AuthorVector + cluster(nbClusters : int) : AuthorVector + prettyPrintCluster() : String - merge(index1 : int, index2 : int) - find_max(ij_max : int[]) : double + toString() : String

Figure 7.11: The class Clustering

7.2.5 Evaluation Framework

ConfusionMatrix

Structure. As shown by Figure 7.12 the core element of the class is a two-dimensional array of type **int** [][] . "Incomplete" clusterings where more clusters than the correct numbers of clusters remain lead to complications in the evaluation of clusterings. Fields that store the correct and obtained number of clusters are necessary as well as the total number of authorships and the sizes of all obtained clusters. The constructor requires a ClusterVector object as input and generates the two-dimensional array of the size of number of obtained clusters, fills in the adequate values and stores the mentioned values. The number of correct authorships in a cluster are obtained by comparing the dblpName that is assigned to the cluster with the contained authorships and counting the matches. This functionality is provided by the class AuthorVector in form of the method nbdblpNameOccurencesInClusteri(index, dblpName).

```
ConfusionMatrix
-------------------------------------------
- serialVersionUID : long
- confusion : int[][]
- clusterSizes : int[]
- totalNbAuthorships : int
- nbClusters : int
- realNbClusters : int
-------------------------------------------
+ ConfusionMatrix(aus : ClusterVector)
+ avgRecall() : double
+ ithRecall(i : int) : double
+ avgPrecision() : double
+ jthPrecision(j : int) : double
+ avgAccuracy() : double
+ absoluteValueAccuracy() : int
+ ithAccuracy(i : int) : double
```

Figure 7.12: ConfusionMatrix

Operations. The method used for evaluation in the case of our experiments is avgAccuracy. It returns the disambiguation accuracy and is computed as

$$\frac{\text{sum of trace elements}}{\text{total number of authorships}}$$

Notice that only a correct number of clusters is considered in the sum. In our experiments, only the authorships of two clusters contribute to the numerator, but all authorships contained in the clustered name dataset contribute to the denominator.

Evaluation

The experiments are conducted by a main method that organizes the clustering of sample data and outputs the results. The following steps are performed:

- Read the MatrixGenerator object from persistent memory and construct a matrix with a particular similarity measure.
- Perform clustering.
- Construct a confusion matrix and output the accuracy, percentage of clustered objects and total precision achieved in the clusters.

7.3 Experiments

7.3.1 Preparation of datasets

From <article> and inproceedings entries in DBLP, only the authors, the title, the key, the venue key and the year are extracted and put into the PostgreSQL database. From <proceedings> elements, the authors or editors, the title, the key and the year are extracted and put into the database. This data migration is done in two steps. In a first step, all XML elements mentioned above are read by a parser (DBLPParser3) and written into a couple of text files where one line represents one data record. Each file corresponds directly to a table of the database and allows a fast batch insertion of all data records of a file at once. This process was already described in detail in Section 7.2.1 and in Section 7.2.2.

We eliminated all papers that were authored by a single person and selected from the remaining set ten homonymous author names of authors with at least ten publications. We prepared two datasets for each chosen name. The first one contains the relevant data of all authors that pass our selection. For the second one, we selected the two authors that were the most active ones among the homonymous authors in the first set, i.e. the two authors that have published the largest number of papers.

Table 7.1 lists the pairs of homonymous names together with the sizes of the datasets and the time needed for generation of serializable data objects. To give an idea what execution times mean, the used computer is equipped with an "AMD Athlon(tm) XP 2000+" and 1 GB of RAM.

Name	nb. homonymous authors	nb. authorships	nb. coauthors	nb. papers	Query time (in sec.)	Constr. time (in sec.)
Key-Sun Choi, Kiyoung Choi	2	95	932	946	456.36	0.39
Anoop Gupta, Amarnath Gupta	2	160	3179	2828	452.6	2.87
Vipin Kumar, Vijay Kumar	2	283	4798	5462	462.24	7.52
James E. Smith, John R. Smith	2	158	3674	3998	451.59	4.53
Prithviraj Banerjee, Kaustav Banerjee	2	255	2883	3360	450.03	2.78
Friedrich L. Bauer, Michael A. Bauer	2	84	1198	1057	449.1	0.22
Philip A. Bernstein, Arthur J. Bernstein	2	143	5426	7507	464.94	11.57
Thomas Hofmann, Martin Hofmann	2	74	2480	2521	454.2	1.41
Michael C. Mulder, Henry Martyn Mulder	2	28	719	550	460.51	0.15
Luís Moniz Pereira, Fernando C. N. Pereira	2	189	2012	2647	468.31	1.51
Mean	**2**	**146.9**	**2730.1**	**3087.6**	**456.99**	**3.29**

Table 7.1: Size of name datasets containing only pairs of homonymous authors

The meaning of the other columns in Table 7.1 and also in Table 7.2 is explained as follows. The number of homonymous authors corresponds to the number of clusters that should be obtained after clustering. The number of authorships is the total number of papers produced by the homonymous authors. Notice that the list of authors of a paper does not contain more than one homonymous author. The reasons are the following. We do not consider multi name disambiguation and the case that a name occurs twice in a paper is not very probable. Furthermore, this name would already be recognized to belong to two persons. The number of coauthors denotes the total number of coauthors and indirect coauthors of the homonymous authors. The large values demonstrate that the number of coauthors increases exponentially with the degree of coauthorship. Analogously, the number of considered papers increases. Such numbers lead to quite long answer times of the SQL queries, but the construction time of MatrixGenerator objects is acceptable. Table 7.2 shows the sizes of the datasets containing all homonymous authors that were selected for a name. The implementation could not manage these amounts of data. We assume that a better implementation is able to cope with these datasets. Indeed there is room for improvements of the implementation. Instead of Java vectors, structures that use memory in a more economic way would help. Possibly, some objects are held in memory and not freed for garbage collection although they are not used anymore. An improved implementation pays more attention to garbage collection. Furthermore, redundancies among data objects should be eliminated, maybe the entire design can be simplified.

Name	nb. homonymous authors	nb. authorships	nb. coauthors	nb. papers	Query time (in sec.)	Constr. time (in sec.)
K Choi	4	130	3545	3473	888.41	6.89
A Gupta	15	483	12103	14090	529.61	124.52
V Kumar	5	327	6756	7432	519.75	30.91
J Smith	9	289	10880	10174	514.98	84.32
Banerjee	12	449	10605	11784	523.66	92.65
Bauer	11	247	8316	7697	518.87	44.74
Bernstein	6	214	9217	11470	521.07	61.97
Hofmann	4	117	6647	5785	526.27	23.51
Mulder	3	39	3028	2171	523.71	5.21
Pereira	7	290	8638	9154	524.9	43.4
Mean	**7.6**	**258.5**	**7973.5**	**8323**	**559.12**	**51.81**

Table 7.2: Size of name datasets containing all chosen homonymous authors

7.3.2 Conducted Experiments

Each of the prepared name datasets is clustered twice, once with the "direct coauthors" similarity measure and once using the "indirect coauthors" similarity measure. It was planned to repeat experiments for the datasets containing more than two homonymous persons. Due to the large amount of data to process, these experiments require an implementation that is optimized concerning the usage of memory. An optimized implementation would not have any problems with the larger datasets of the prepared dataset collection. However, we assume that further results would not yield in other conclusions.

7.3.3 Results – Coauthor Information Revisited

As already mentioned in Section 6.2.2, the structure of the coauthor graph poses insuperable barriers to algorithms that use only coauthor information. That is why we restricted the set of authorhips to publications that have at least 2 authors. But there is another circumstance that has bad consequences: in most cases, researchers look like several persons according to their collaboration habits.

The results of our experiments support this. Table 7.3 presents the achieved accuracies for the "direct coauthor" similarity measure, the number of obtained clusters and the achieved precision in the chosen clusters. The recall denotes the percentage of authorships that are contained in the produced clusters and depends on the size of the chosen clusters.

For three datasets no results were obtained due to the size of datasets. Tuning the virtual machine could not solve the problem of heap space overflows. Due to the small amount of time dedicated for the implementation, we could not improve efficiency. Table 7.4 presents the same kind of results achieved by the algorithm by using the "indirect coauthors" similarity measure.

Name Pairs	Obtained nb. of clusters	Accuracy	Precision	Recall	Clustering time (in sec.)
Key-Sun Choi, Kiyoung Choi	30	0.56	1	0.56	6.12
Anoop Gupta, Amarnath Gupta	17	0.69	1	0.69	232.29
Vipin Kumar, Vijay Kumar					
James E. Smith, John R. Smith	35	0.53	1	0.53	251.18
Prithviraj Banerjee, Kaustav Banerjee					
Friedrich L. Bauer, Michael A. Bauer	10	0.89	1	0.89	13.68
Philip A. Bernstein, Arthur J. Bernstein					
Thomas Hofmann, Martin Hofmann	25	0.38	1	0.38	24.24
Michael C. Mulder, Henry Martyn Mulder	10	0.64	1	0.64	0.56
Luís Moniz Pereira, Fernando C. N. Pereira	28	0.67	0.99	0.68	144.1
Mean	**22.14**	**0.62**	**1**	**0.62**	**96.02**

Table 7.3: Clustering results with “direct coauthors” similarity measure

Name Pairs	Obtained nb. of clusters	Accuracy	Precision	Recall	Clustering time (in sec.)
Key-Sun Choi, Kiyoung Choi	30	0.56	1	0.56	10.62
Anoop Gupta, Amarnath Gupta	17	0.69	1	0.69	223.49
Vipin Kumar, Vijay Kumar					
James E. Smith, John R. Smith	35	0.53	1	0.53	288.24
Prithviraj Banerjee, Kaustav Banerjee					
Friedrich L. Bauer, Michael A. Bauer	9	0.9	1	0.9	11.33
Philip A. Bernstein, Arthur J. Bernstein					
Thomas Hofmann, Martin Hofmann	24	0.38	1	0.38	25.85
Michael C. Mulder, Henry Martyn Mulder	10	0.64	1	0.64	0.91
Luís Moniz Pereira, Fernando C. N. Pereira	28	0.67	0.99	0.68	129.28
Mean	**21.86**	**0.62**	**1**	**0.63**	**98.53**

Table 7.4: Clustering results with “indirect coauthors” similarity measure

Notice that the correct number of clusters is 2. The large discrepancy is the explanation for low accuracies. We admit that the average accuracies of about 60% are not very much better than the ones an algorithm would achieve that randomly assigns authorships to clusters. Indeed such an algorithm would achieve an accuracy of about 50% since there are only two clusters for each name. Table 7.4 shows that in the selected cases, the "indirect coauthors" similarity measure does not improve disambiguation accuracy. The number of obtained clusters is reduced by 1 in three out of seven cases. We suppose that increasing the degree of coauthorship might further reduce the number of clusters and in consequence improve accuracies.

We assumed that the data we extracted from DBLP is free of errors and compared with DBLP to check the results of our algorithm. DBLP is indeed very reliable but not free of errors. It is possible that the "ground truth" we assumed is not correct in all cases. *Ground truth* denotes the knowledge of the proper clusters representing the authors in the real world. However, we believe that errors in DBLP are negligible. Due to the limited schedule for this project, we could not manually check the ground truth of the datasets.

The experiments support the assumption that coauthor information is a discriminative criterion for name disambiguation. It is even too discriminative. In DBLP, each author has his own page listing the publications he participated in as well as his coauthors. The presentation of the coauthors uses colors to partition the coauthors into groups. Two coauthors are in the same group if they are connected, if they have collaborated or if there is a chain of publications in the set of the author's publications that connect them. Such groups depict different facets of a person by means of different stages in the career and life of a researcher or by means of different research interests. It is believed[2] that well established researchers have only one or few colors in their list of coauthors. The community of researchers is a small world and it is more probable that the coauthors of a researcher also know each other and publish together. Missing connections between groups may result from the fact that our view on the coauthor graph is incomplete or just do not exist. We believe that both cases happen in reality.

Under these considerations our algorithm performs as well as he can within the "natural" limits of coauthor information. Improvement cannot be achieved by using only coauthor information. Other feature types must be involved and exploited. Future work should integrate publication title keywords and publication venues since these feature types promise to establish further connections between differently colored groups.

More knowledge about the distribution of colors in DBLP would allow to give an upper border for disambiguation accuracy using coauthor information alone.

7.3.4 Remedies

As described above, the problem is a basic one and cannot be solved by changing the clustering algorithm alone, but by involving other features. The similarity measure is the only point where even small changes may result in significant improvements. We propose to involve publication venues and an ontology of paper title keywords. An extensive analysis of the coauthor graph may reveal further knowledge about the structure of the social network. Such knowledge might influence a graph based similarity measure. Furthermore, a factor based on the publication year might weaken small similarity measures that are due to different stages in the career of a researcher. If abstracts or full texts of publications were *broadly* available, it would be possible to conduct text analysis using techniques from natural language processing. There are interesting works concerning automatic categorization of texts that yield good results.

[2]http://www.informatik.uni-trier.de/ ley/db/about/coauthorColors.html

7.3.5 Comparison to other Algorithms

It is difficult to compare our results to the ones presented in Chapter 5. The disambiguation accuracies cannot be compared directly because the experiments were conducted on datasets that vary in the difficulty for clustering. The assumption that the number of colors is similar for the authors that have not yet been clustered allows to transform the results to the cases including these authors. If they, too, do not have any "misleading" coauthor information, the disambiguation accuracy depends only on the proportion of authorships related to the largest connected components in the coauthor graph to all considered authorships. Misleading information would occur in the situation where two homonymous authors have common coauthors. It is believed that this case occurs seldomly. We expect that the results for full name datasets would be similar to the achieved ones. This can be seen as follows: our results show that the problem is that what is shown by the colors of coauthors on DBLP author pages. An author usually collaborates with distinct groups of coauthors that are not connected by the coauthorship relation. The distribution of such colors may vary, but if lots of examples are taken, it should be uniformly distributed. Notice that, as our results show, the disambiguation accuracy depends almost only on the proportion of the size of the citations belonging to the largest coauthor group to all considered citations. This proportion, too, seems to be uniformly distributed. Therefore, adding more homonymous persons to datasets should not have any impact on the average disambiguation accuracy. These considerations allow a tentative comparison. Our algorithm does not work as well as the more complex algorithms presented in Chapter 5, but in the case of using only coauthor information, other algorithms achieve similar results.

Part IV

Conclusion

CHAPTER

EIGHT

Conclusion

8.1 Summary

In this work, we gave a detailed overview of approaches to name disambiguation. We identified the *multi name disambiguation problem* as opposed to the *single name disambiguation problem*. Due to the variety of tasks and approaches, we restricted the overview to approaches that try to solve the problem of disambiguating homonymous authors in DBLP. Based on this overview we developed our own approach focusing on graph structures. We implemented some of our ideas to give a proof of concept. For evaluation purposes, we constructed name datasets and reported our experiences with the processing of that kind of data. The results of our experiments allow to draw conclusions about the nature of coauthor information.

8.2 Future Work and Research Directions

There are various ideas that have not been investigated in this thesis. Further research may reveal interesting findings. One can start at several points.

Multi Name Disambiguation Problem

In the introduction in Section 1.3 we identified the problem of several ambiguous names occurring simultaneously in the list of authors of a publication. The probability of such co-occurrences of ambiguous names is small, but nevertheless it would be interesting to investigate the problem. An approach respecting multi name disambiguation may improve in general the deployment of coauthor information. Furthermore the task is quite complicated and constitutes a challenging research topic.

Name Disambiguation on the Web

A challenging research topic is to extend name disambiguation in DBLP to name disambiguation on the Web, i.e. disambiguating personal names that occur on Web pages. Approaches for this task require the use of more techniques that can deal with natural text and the heterogeneity of the Web.

Social Network analysis

The community of researchers who are interested in computer science or related fields is a large social network with many facets. On the one hand, there are many individuals coming from different cultures and having different goals and motivations. On the other hand, these individuals have more in common than individuals in other networks. There are overlappings of research interests: researchers have a drive to collaborate together, but keep the own interests at the same time and a high fluctuation of individuals entering and leaving the community. Therefore, an extensive analysis of the social network looks promising. New conclusions may allow to exploit coauthor information in a more effective way. For example, *hubs* may be recognized. Hubs are individuals that have many connections in the network. They are useful for individuals to shorten the distance to many other individuals. According to the structure of the network, individuals could be classified. The information that an individual performs the role of a hub can be exploited as follows. A hub has direct connections to other hubs and is surrounded by "small" individuals that have only few connections. Usually, within a working group there is only one hub or maybe two. Surely, the number of hubs depends on the size and structure of the working group. Professors are probably hubs in most cases, but PhD. students probably not. Therefore, a connection from one hub to another is an indication that this edge connects two working groups. The knowledge of borders between working groups can be exploited for name disambiguation. It is not very probable that two persons having the same name are working in the same group.

Integration of other Social Networks

There are other sources of information in form of social networks. For example, FOAF data can be exploited. The information is given in a machine-processable form and therefore convenient for automatic information extraction and interpretation. Thanks to the structured format of FOAF data, computers "know" about the semantics of the data. To exploit FOAF data for name disambiguation, it is necessary to integrate FOAF data that is extracted from the Web and data extracted from DBLP. In [AMNR$^+$06] an example is given where this is achieved for a sub-community of researchers interested in the semantic Web. The authors of [AMNR$^+$06] adapt a reconciliation algorithm that is presented in [Don05]. An analysis of such an integrated network may reveal new insights that allow to improve the way coauthor information is deployed for name disambiguation.

Semantic Web Technologies

As proposed in Section 6.2.3 and in Section 6.2.4, ontologies may improve name disambiguation. Unfortunately, there are no satisfactory ontologies available at the moment or a generic insertion of keywords into the tree of the ontology is missing. Both the generation of ontologies and the insertion of keywords into ontologies are interesting research topics.

Natural Language Processing

Analyzing full texts of publications may reveal interesting facts about writing habits of authors. It would be interesting to investigate if such habits differ sufficiently to distinguish authors. An obstacle may be that usually papers are not written by a single person. In consequence, writing styles are mixed. Perhaps, such mixed styles can be matched with the list of authors, so that a false person can be recognized. For example, it has been shown that authors have a characteristic *zipRatio*, a value depending on the compression ratio using zip. Therefore, this value may also be of use.

Implementation of a Disambiguation Framework

For further investigations it would be useful to have some framework for evaluation purposes. This framework includes name datasets that provide all information that is needed by various approaches, e.g. an approach that exploits deeply the social network, the datasets should contain the information about an entire sub-community. If research concentrates on the involvement and evaluation of different feature types, this framework should consist of an algorithm that allows to integrate different similarity measures as components, e.g. our algorithm presented in Section 6.1.2 is able to deal with different similarity measures and it even does not know how these similarities are computed. An implementation should also encapsulate such similarity measures so that an evaluation framework can use them interchangeably. Of course, implementations of similarity measures depend on the representation of the data which should be provided in an efficient way. Therefore, the conception of data structures and of the framework should be as general as possible to preserve the possibility of extensions.

8.3 Concluding Remarks

This work showed that the task of name disambiguation comprises lots of different facets. Several research areas investigate the topic, but there are still many questions to be answered. Good solutions are of common interest.

Part V

Appendix

APPENDIX

A

Code Examples

A.1 Parsing dblp.xml: DblpParser3

Listing A.1 presents the structure of the parser. It is illustrated how papers are recognized and authors and paper titles are assigned. Analoguous parts, e.g. the extraction of venues and other properties like the publication year are omitted. Three dots " ... " mark places where code is omitted.

```
public class DblpParser3 {
    // ***** constants *****
    // possible paper entries without:|phdthesis|mastersthesis|www
    private final String pubTypes = "(|article|inproceedings|incollection|)";
    private final String venueTypes ="(|proceedings|book|)";
    // ***** final fields to set by constructor*****
    private  final String XML_FILE; // the dblp file which is used
    // for the XmlParser ...
    // for writing the files used for putting data into the database
    private PrintWriter article_out; ...
    // ***** fields used for computation *****
    private Paper paper, venue; // used to build currently read paper entry

    public DblpParser3 (String xml_file) throws XmlPullParserException,
        FileNotFoundException {
        // create new parser that reads the file xml_file
    }

    // process the document, i.e. the events fetched from XmlPullParser
    // events are START_DOCUMENT, END_DOCUMENT, START_TAG, END_TAG
    private void processDocument() throws XmlPullParserException, IOException{
        int eventType = xpp.getEventType();
        do{
            // only once: START_DOCUMENT, nothing to do
            if(eventType == XmlPullParser.START_DOCUMENT) {
                // measure starting time
            }
            // START_TAG: deligate to processStartTag
            else if(eventType == XmlPullParser.START_TAG) {
                processStartTag();
            }
            // END_TAG: deligate to processEndTag
            else if(eventType == XmlPullParser.END_TAG) {
                processEndTag();
            }
```

```
            eventType = xpp.next();
        } while ((eventType != XmlPullParser.END_DOCUMENT));
        // measure end time
    }

    // check if xml tagname as occuring in xml_file is publication type
    // returns true iff tagname is a publication tag and therefore is name of
    // a tag starting a publication element
    private boolean isPublicationXMLElement(String tagname){
        // "|" is needed because otherwise <ee> is recognized
        return pubTypes.contains("|"+tagname+"|");
    }

    // returns the first part of a key that is needed for a venueKey
    // e.g. "journals/ai/article79" or "conf/ai/1979-13"
    // "journals/ai/"
    public String extractFirstPartofKey(String s){ ... }

    // recognize type of xml element and and process it
    public void processStartTag() throws XmlPullParserException, IOException{
        String tagname = xpp.getName(); // name of opening tag
        // publication element
        if (isPublicationXMLElement(tagname)){
            String key = null;
            key = xpp.getAttributeValue(null, "key"); // key cannot be missing
            paper = new Paper(key); // create a new one
            paper.setVenueKey(this.extractFirstPartofKey(key));
        }
        // venue element ...
        // author element of a new paper, i.e. one which needs to be processed
        else if (tagname.equals("author")){
            if (xpp.next() == XmlPullParser.TEXT){
                String text = xpp.getText();
                if (paper != null){
                    paper.addAuthor(text, null,false);
                }...
            }
        }
        // title element
        else if (tagname.equals("title")){
            if (xpp.next() == XmlPullParser.TEXT){
                String title = xpp.getText().replace("|", "*");
                if (paper != null)
                    paper.setTitle(title);
                    ...
            }
        }
        // venue element in paper ...
        // year element in paper ...
    }

    // if ending tag closes author element: add paper (only if not already
        contained)
    public void processEndTag () throws XmlPullParserException, IOException {
        // name of closing tag
        String tagname = xpp.getName();
        // end publication element
        if(isPublicationXMLElement(tagname)) {
            article_out.println(paper.articleString());
            for (int i = 0; i < paper.numberAuthors(); i++){
                article_authorship_out.println(paper.getAuthor(i).getDblpName
                    ()+"|"+paper.getKey());
            }
            paper = null;
        }
```

```
        // end venue element ...
    }

    public void parse(String article_file, String venue_file, String
        article_authorship_file,
                    String venue_authorship_file) throws
                        XmlPullParserException, IOException {
        article_out = new PrintWriter(
            new BufferedWriter(new FileWriter(article_file)));
        ...
        processDocument();
        // flush buffers and close outputstreams
        article_out.flush();
        article_out.close();
        ...
    }
}
```

Listing A.1: DBLParser3

A.2 Clustering and Evaluation

The source code of the fundamental classes that are needed for clustering and evaluation is listed here. Once again, the listings are incomplete. Only interesting methods are given.

A.2.1 Author and Authorship

The classes Author and Authorship model non-ambiguous and ambiguous authors and authorships. Authors model the coauthors that are assumed to be already disambiguated. It is not necessary to memorize the papers they have published. Ambiguous authors are split into authorships according to the papers they have published. Therefore, authorships store the key of the publication of an authorship. These two classes provide mainly data. The provided functionality is limited to comparisons between authors, authorships, and strings that are differently implemented in the super- and the subclass. The class Paper is not presented here because it does not contain any interesting details, but consumes too much space in this work.

```
public class Author implements Serializable{
    protected final String publicName; //e.g. "Banerjee", "A Gupta"
    protected final String dblpName; //e.g. "Fernando Banerjee","Ann Gupta"

    public Author(final String idname, final String queryName) { ... }

    // returns the name that identifies nonambiguously the entity
    // for an author, it is the dblpName
    public String getIdentifyingName(){ return this.dblpName; }

    // checks if a name (of the form "dblpName" or "dblpName_paperKey")
    // identifies this author object: true iff dblpName is the same as name
    public boolean equalsByIdentifyingName(String name){
        return dblpName.equals(name);
    }

    // return true if dblpName of author equal to given name
    public boolean equalsByDblpName(String dblpName){
        return this.dblpName.equals(dblpName);
    }

    public boolean equals(Object obj) { /* equal if dblpNames are equal */ }

    public String toString(){ return dblpName; }
```

```
}
```

Listing A.2: Author

The class Authorship overrides the comparison methods equalsByIdentifyingName and equals as well as the method toString.

```
public class Authorship extends Author implements Serializable{
    protected final String paperKey; // "_" not in venue or paper key

    public Authorship (final String idName, final String queryName,
                       final String paperKey)
    {      super(idName, queryName); this.paperKey = paperKey; }

    // an authorship is identified by a combination of dblpName and paperKey
    public String getIdentifyingName()
    { return super.dblpName.concat("_"+paperKey); }

    // in the case of an authorship, true iff dblpName and paperKey are equal
    public boolean equalsByIdentifyingName(String name){
        StringTokenizer st = new StringTokenizer(name,"_");
        if (st.countTokens() != 2){ return false; }
        else{ return (super.equalsByIdentifyingName(st.nextToken())) &&
                  (paperKey.equals(st.nextToken()));
        }
    }

    public boolean equals(Object obj) { /* involves paperKey */ }

    public String toString(){return super.getDblpName().concat("-"+paperKey);}
}
```

Listing A.3: Authorship

A.2.2 AuthorVector and ClusterVector

These two classes serve to collect authors and authorships. The class AuthorVector is used during parsing and clustering. The class ClusterVector is used for evaluation. For this purpose, a ConfusionMatrix object is contained and additional methods are provided to allow the evaluation of clusterings in terms of precision and recall.

```
public class AuthorVector implements Serializable{
    protected Vector<Vector<Author>> authors;
    // generates an AuthorVector with a new instance of authors
    public AuthorVector(Vector<Vector<Author>> object) {
        this.authors = (Vector<Vector<Author>>)object.clone();
    }

    // tests if already an equal author object (i.e. with equal dblpname) or
    // authorship object (i.e. with equal dblpName and paperKey)
    // is contained in the VectorVector
    public boolean contains(Author au){
        for (int i = 0; i < authors.size(); i++){
            if (authors.elementAt(i).contains(au))
                return true;
        }
        return false;
    }

    // an AuthorVector is constructed with all authorships at the beginning
    // and authors in the second part of the vector
    // returns the highest index of an authorship: number of authorships -1
    public int lastIndexOfAuthorship(){
        for (int i = 0; i < authors.size(); i++){
```

```
            if(!(authors.elementAt(i).elementAt(0) instanceof Authorship))
                return i-1;
        }
        return authors.size()-1;
    }

    // returns the total number of Authorships
    public int totalNbofAuthorships(){
        int res = 0;
        for (int i = 0; i < lastIndexOfAuthorship(); i++){
            res += authors.get(i).size();
        }
        return res;
    }

    // returns number of distinguished authors including coauthors
    public int size(){ return authors.size(); }

    // adds an author only if not already an equal author object is contained;
    // the object is added at the position accordingly to the object type:
    // if the object carries an authorship, it is placed after the last
        authorship,
    // otherwise at the end
    public void addAuthor(Author au){
        if (!this.contains(au)){// equal object already element of vector?
            int index;
            // index for a new authorship
            if (au instanceof Authorship){ index = this.lastIndexOfAuthorship
                ()+1; }
            // index for a new (co)author
            else{ index = authors.size(); }
            // each author(ship) element has the form of a vector of authors (
                containing one element)
            Vector<Author> v = new Vector<Author>();
            v.add(au);
            authors.add(index, v);
        }
    }

    // merge the two vectors of authors/authorships;
    public void mergeAuthorships(int index1, int index2){
        // append the elements of vector at index2 to vector at index1
        authors.elementAt(index1).addAll(authors.elementAt(index2));
        authors.remove(index2); // remove the vector at index2
    }

    //finds the index of an author-element by name (public or dblp)
    public int indexOf(String name) {
        int res = -1;
        for (int i = 0; i < authors.size(); i++){
            // each element needs to be searched by name
            Vector<Author> aus = authors.elementAt(i);
            for (int j = 0; j < aus.size(); j++){
                Author au = aus.elementAt(j);
                if (au.equalsByIdentifyingName(name))
                    res = i; break;
            }
        }
        return res;
    }

    public void swapAuthors(int index1, int index2){
        Vector<Author> tmp = authors.get(index1);
        authors.set(index1, authors.elementAt(index2));
        authors.set(index2, tmp);
    }
```

```
    public Vector<Author> remove(int index){ return authors.remove(index); }

    public Author authorAt(int index) {
        return authors.elementAt(index).firstElement();
    }

    public Vector<Author> authorlistAt(int index){
        return authors.elementAt(index);
    }
}
```

Listing A.4: AuthorVector

Since in our experiments only the case occurs that too many clusters remain, clusters and names are brought together only by using a heuristic as explained in Section 7.2.3. Therefore, in the following extract the parts computing an optimal assignment are omitted. The assignment is implemented in the sortClusters method, the permutations are not needed in this version.

```
public class ClusterVector extends AuthorVector implements Serializable{
    private int nbClusters; // number of Clusters obtained by clustering
    private int realNbClusters; // the correct number of Clusters
    private Vector<String> assignment; // permutation with optimal accuracy
    private ConfusionMatrix confusion; // conf matrix for all clusters
    private Vector<String> dblpNames; // a list of all dblpnames
    private int totalNbAuthorships; // total number of authorships
    private int nbClusteredAuthorships;// nb authorships in considered
        clusters

    public ClusterVector(int realNbClusters, AuthorVector aus,
                         Vector<Vector<String>> permutations){
        super();
        nbClusters = aus.lastIndexOfAuthorship()+1;
        this.realNbClusters = realNbClusters;
        dblpNames = (Vector<String>) permutations.firstElement().clone();
        totalNbAuthorships = 0;
        // only the real clusters are hold in authors, i.e. no coauthors, only
            authorships
        for (int i = 0; i < nbClusters; i++){
            authors.add((Vector<Author>) aus.authorlistAt(i));
            totalNbAuthorships += aus.authorlistAt(i).size();
        }
        // heuristic for assignment, if there are too many clusters
        if (nbClusters > realNbClusters)
            sortClusters();

        nbClusteredAuthorships = 0;
        for (int i = 0; i < realNbClusters; i++){
            nbClusteredAuthorships += this.authorlistAt(i).size();
        } // in case of too many clusters, assignment by sortClusters()
        assignment = permutations.elementAt(0);
        confusion = new ConfusionMatrix(this);
    }

    // counts the number of occurrences of a name in a particular cluster
    public int nbdblpNameOccurencesInClusteri(int index, String dblpName){
        int res = 0;
        for (int i = 0; i < this.authorlistAt(index).size(); i++){
            if (this.authorlistAt(index).elementAt(i).equalsByDblpName(
                dblpName))
                res++;
        }
        return res;
    }

    // sort the clusters according to their number of occurences of names;
```

```
    // search for the first dblpname the cluster with the most occurences of
        this name
    // then do the same for the next name, but consider only the remaining
        clusters;
    // and so on; is not optimal - optimal solution would be "6 aus 49"
    private void sortClusters() {
        // find the best cluster for each name
        for(int i = 0; i < realNbClusters; i++){
            // find the cluster of ith name
            int maxindex = i; // the index of the whished cluster
            for (int j = i+1; j < nbClusters; j++){
                if(nbdblpNameOccurencesInClusteri(j, dblpNames.elementAt(i)) >
                   nbdblpNameOccurencesInClusteri(maxindex, dblpNames.
                       elementAt(i)))
                    swapAuthors(maxindex, j);
            }
        }
    }

    // accuracy relative to total number of authorships
    public double getAccuracy(){ return confusion.avgAccuracy(); }
    // proportion of clustered authorships to all authorships
    public double getRecall(){
        return (double)nbClusteredAuthorships / (double)totalNbAuthorships;
    }
    // average precision in the considered clusters
    public double totalPrecision(){
        return (double)confusion.absoluteValueAccuracy()/(double)
            nbClusteredAuthorships;
    }
}
```

Listing A.5: ClusterVector

A.2.3 Clustering

The extended coauthor graph is modeled by the class Matrix that is described in Section 7.2.3. Besides the representation of the coauthor graph, the class provides other functionality. Comprised are the manipulation of the graph that is needed during clustering and the computation of similarity measures that are encapsulated as inner classes.

```
public class Matrix implements Serializable{
    public static final int SIM_DIR_COAUTHORS = 0;
    public static final int SIM_IND_COAUTHORS = 1;
    private double[][] matrix;
    private int dimSimPart, dim;
    private AuthorVector authors;
    private SimilarityMeasure sm;

    public Matrix(int dim, int dimSimPart, final Vector<Paper> papers,
                  AuthorVector authors, int smid){
        this.authors = (AuthorVector) authors.clone(); //authorships first
        this.dim = dim; // dimension of whole matrix
        this.dimSimPart = dimSimPart; // dimension of merging part
        this.matrix = new double[dim][dim]; // extended adjacency matrix

        for (int i = 0; i < papers.size(); i++){
            Paper p = papers.elementAt(i);
            this.addPaper(p); // fill values of coauthorships
        }
        genSim(smid); // create similarity measure
        fillSimilarities(); // fill merging part with values
    }
```

```java
// set coauthorship relations that are expressed by paper p
private void addPaper(Paper p){
    // for each author of paper p, update matrix by coauthor relations
    for (int i = 0; i < p.numberAuthors(); i++){
        String nameI, nameJ;
        // flags checking that not both of the authors are ambiguous
        boolean iIsAuthorship, jIsAuthorship;
        //for the test below
        nameI = p.getAuthor(i).getIdentifyingName();
        iIsAuthorship = p.getAuthor(i) instanceof Authorship;

        for (int j = i+1; j < p.numberAuthors(); j++){
            nameJ = p.getAuthor(j).getIdentifyingName();
            jIsAuthorship = p.getAuthor(j) instanceof Authorship;
            // count the coauthorships between nameI and nameJ
            int newValue = (int)this.getIJ(nameI, nameJ);
            newValue++;
            // small prob. of collab. of 2 considered ambiguous authors
            // do not count such coauthorships: no place in matrix for it
            if (!(iIsAuthorship && jIsAuthorship))
                this.setIJ(nameI, nameJ, (double)newValue);
        }
    }
}

// combines two similarity values: recomputes the similarity
private double conc(int index1, int index2){ return sim(index1, index2); }

// merges two Authorships in the authors-vector
public void mergeAuthorships(int index1, int index2){
    authors.mergeAuthorships(index1, index2);
}

// contracts the edge (index1, index2) of the similarity graph:
// similarity values in column index1 are updated;
// matrix is copied into a new (smaller) two-dimensional vector
public void contraction(int index1, int index2){
    if (index1 > index2) // message because something went wrong
        System.out.println("Error: indices in wrong order");

    // set values of index1 column in old matrix before copying
    for(int i = 0; i < getDimSimPart(); i++){
        setIJ(i, index1, conc(i,index2));
    }
    // copy matrix in new (dim-1)x(dim-1)-matrix in 3 steps
    // get new matrix
    double[][] m = new double[getDim()-1][getDim()-1];

    // copy upper left part from (0,0) to (index2 excl, index2 excl)
    for (int i = 0; i < index2; i++){
        for (int j = i+1; j < index2; j++)
            m[i][j] = getIJ(i, j); // identical to old matrix
    }
    // copy upper right part from (0, index2+1) to (index2 excl, dim)
    // BUT into the part from (0,index2) to (index2 excl,dim)
    for (int i = 0; i < index2; i++){
        for (int j = index2; j < getDim()-1; j++)
            m[i][j] = getIJ(i, j+1);
    }
    // copy lower right part from (index2, index2) to the end
    for (int i= index2; i < getDim()-1;i++){
        for (int j = i+1; j < getDim()-1;j++)
            m[i][j] = getIJ(i+1, j+1);
    }
    dimSimPart--; dim--; // reduce dimensions of matrix
```

```
        setMatrix(m); // new matrix without row and column index2
    }

    // updates the coauthorinformation for merged authorships and recalculates
    // similarities for it; actually only the ones with index1 involved,
    // other similarity measures may require recalculation of all similarities
    public void update(int index1, int index2){
        // update second part of matrix
        for (int j = getDimSimPart(); j < getDim(); j++)
            setIJ(index1, j, getIJ(index2, j)+getIJ(index1, j));
        // update similarites: recalculate values in the row of index1
        for (int j = index1+1; j < getDimSimPart(); j++)
            setIJ(index1, j, sim(index1, j));

        fillSimilarities(); // no problem if conc = recalculate similarity
    }

    // fills the similarity part, called by constructor
    public void fillSimilarities(){
        for (int i = 0; i<dimSimPart; i++)
            for (int j = i+1; j<dimSimPart; j++)
                matrix[i][j] = sm.sim(i,j);
    }

    private void genSim(int id){
        switch(id){
            case Matrix.SIM_DIR_COAUTHORS : sm = new SimDirCoauthors(); break;
            case Matrix.SIM_IND_COAUTHORS : sm = new SimIndCoauthors(); break;
            default: sm = null;
        }
    }

    public abstract class SimilarityMeasure{
        protected abstract double sim(int index1, int index2);
        private int id;

        public int getId() {
            return id;
        }
    }

    public class SimDirCoauthors extends SimilarityMeasure implements
        Serializable{
        protected double sim(int index1, int index2) {
            int res = 0;
            if (index1 == index2)
                return (double)res;

            for (int j = dimSimPart; j < dim; j++)
                if((matrix[index1][j] > 0) && (matrix[index2][j]>0))
                    res++;

            // maybe it is better to return normalized values
            if (res == 0)
                return (double)res;
            else
                return (double)res;//return (double)res/(dim-dimSimPart);
        }
    }

    public class SimIndCoauthors extends SimilarityMeasure implements
        Serializable{
        protected double sim(int index1, int index2) {
            int res = 0;
            if (index1 == index2)
                return (double)res;
```

```
            for (int j1 = dimSimPart; j1 < dim; j1++){
                if((getIJ(index1,j1) > 0)&& (getIJ(index2,j1))==0){
                    for (int j2 = j1+1; j2 < dim; j2++)
                        if ((getIJ(index2,j2) > 0) && (getIJ(index1,j2)==0)){
                            res += getIJ(j1,j2);// no condition needed
                        } //since if matrix[j1][j2] == 0, nothing happens
                } else{
                    if((getIJ(index2,j1) > 0)&& (getIJ(index1,j1))==0){
                        for (int j2 = j1+1; j2 < dim; j2++)
                            if ((getIJ(index1,j2) > 0)&&(getIJ(index2,j2)==0))
                                res += getIJ(j1,j2);// no condition needed
                    }
                }
            }
            return (double) res;
        }
    }

    // computes the similarity between two given authorships
    public double sim(int index1, int index2){ return sm.sim(index1, index2);
        }
}
```

Listing A.6: Matrix

The clustering process is encapsulated in the class Clustering that manipulates an object of type Matrix. The core method is cluster that is provided in two versions. The one that requires the correct number of clusters that should remain after clustering is included in the extract in Listing A.7, the other one in which the termination criterion is expressed by a threshold is omitted since it is not used in the conducted experiments.

```
public class Clustering {
    private Matrix matrix;

    public Clustering(Matrix mat){ matrix = mat; }
    // HAC-like clustering, stops if only nbClusters cluster remain or
    // max similarty == 0
    public AuthorVector cluster(int nbClusters){
        int[] ij_max = new int[2];//contains indexI and indexJ of max. element
        double nextMerge = find_max(ij_max);
        while ((matrix.getAuthors().lastIndexOfAuthorship()+1 >
                nbClusters)&&(nextMerge>0.0)){
            merge(ij_max[0], ij_max[1]);
            nextMerge = find_max(ij_max);
        }
        return matrix.getAuthors();
    }

    // merge the authorships at index1 and index2 in matrix and authors
    private void merge(int index1, int index2){
        // index1 is the smaller one
        int tmp1 = index1; int tmp2 = index2;
        index1 = Math.min(tmp1, tmp2); index2 = Math.max(tmp1, tmp2);
        // update coauthor information and similarity values
        matrix.update(index1, index2);
        // before contracting the edge (index1,index2) because
        // in contraction, row and column index2 are already deleted
        matrix.contraction(index1, index2);
        // merge these authorships in the AuthorVector object
        matrix.mergeAuthorships(index1, index2);
    }

    // find the two authorships (sets) which are next to be merged
    private double find_max(int[] ij_max){
        double max = 0;
```

```
        for (int i = 0; i < matrix.getDimSimPart(); i++)
            for (int j = i+1; j < matrix.getDimSimPart(); j++)
                if (matrix.getIJ(i, j) > max){
                    max = matrix.getIJ(i, j);
                    ij_max[0] = i; ij_max[1] = j;
                }
        return max;
    }
}
```

Listing A.7: Clustering

A.2.4 Evaluation

The class ConfusionMatrix implements the idea of a confusion matrix that records for each name the number of falsely and correctly clustered authorships. The code fragment that is presented in Listing A.8 contains only one method that returns a quality measure. Due to the fact that too many clusters remain in our experiments, the method absoluteValueAccuracy is the only one that is used to evaluate the clustering algorithm. The class ClusterVector implements methods that determine precision and recall.

```
public class ConfusionMatrix implements Serializable{
    private int[][] confusion;
    private int[] clusterSizes;
    private int totalNbAuthorships, nbClusters, realNbClusters;

    public ConfusionMatrix(ClusterVector aus){
        nbClusters = aus.getNbClusters();
        this.realNbClusters = aus.getRealNbClusters();
        confusion = new int[nbClusters][nbClusters];
        clusterSizes = new int[nbClusters];
        totalNbAuthorships = aus.getTotalNbAuthorships();;

        for (int i = 0; i < realNbClusters; i++){
            clusterSizes[i] = aus.authorlistAt(i).size();
            for (int j = 0; j < realNbClusters; j++){
                String ithname = aus.getAssignment().elementAt(j);
                confusion[i][j] = aus.nbdblpNameOccurencesInClusteri(i,
                    ithname);
            }
        }
    }

    public int absoluteValueAccuracy(){
        int res = 0;
        for (int i = 0; i < realNbClusters; i++)
            res += confusion[i][i];
        return res;
    }
    ...
}
```

Listing A.8: ConfusionMatrix

APPENDIX

B

DBLP Data

In the following, the actual DTD for XML data of DBLP is given together with a small sample of of the XML file.

B.1 DTD

This DTD has been downloaded from the DBLP Web site[1].

```
<!--
    DBLP XML Records are available from

        http://dblp.uni-trier.de/xml/

    Copyright 2001, 2003 by Michael Ley (ley@uni-trier.de)

        Copying of the "DBLP" bibliography collection is permitted provided
            that
        the copies are not made or distributed for direct commercial advantage
             and
        credit for the source is given. To copy or republish otherwise
            requires
        specific permission. ACM, IEEE Computer Society, and The VLDB
            Endowment
        have the permission to publish DBLP on CDROM/DVD and on their Web
            servers.
-->

<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
               phdthesis|mastersthesis|www)*>
<!ENTITY % field "author|editor|title|booktitle|pages|year|address|journal|
    volume|number|month|url|ee|cdrom|cite|publisher|note|crossref|isbn|series|
    school|chapter">

<!ELEMENT article        (%field;)*>
<!ATTLIST article        key CDATA #REQUIRED
                         reviewid CDATA #IMPLIED
                         rating CDATA #IMPLIED
                         mdate CDATA #IMPLIED>

<!ELEMENT inproceedings  (%field;)*>
<!ATTLIST inproceedings  key CDATA #REQUIRED
```

[1] http://www.informatik.uni-trier.de/~ley/db/about/dblp.dtd

```
                              mdate CDATA #IMPLIED>

<!ELEMENT proceedings    (%field;)*>
<!ATTLIST proceedings    key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT book           (%field;)*>
<!ATTLIST book           key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT incollection   (%field;)*>
<!ATTLIST incollection   key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT phdthesis      (%field;)*>
<!ATTLIST phdthesis      key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT mastersthesis  (%field;)*>
<!ATTLIST mastersthesis  key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT www            (%field;)*>
<!ATTLIST www            key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>

<!ELEMENT author     (#PCDATA)>
<!ELEMENT editor     (#PCDATA)>
<!ELEMENT address    (#PCDATA)>

<!ENTITY % titlecontents  "#PCDATA|sub|sup|i|tt|ref">
<!ELEMENT title      (%titlecontents;)*>
<!ELEMENT booktitle  (#PCDATA)>
<!ELEMENT pages      (#PCDATA)>
<!ELEMENT year       (#PCDATA)>
<!ELEMENT journal    (#PCDATA)>
<!ELEMENT volume     (#PCDATA)>
<!ELEMENT number     (#PCDATA)>
<!ELEMENT month      (#PCDATA)>
<!ELEMENT url        (#PCDATA)>
<!ELEMENT ee         (#PCDATA)>
<!ELEMENT cdrom      (#PCDATA)>
<!ELEMENT cite       (#PCDATA)>
<!ELEMENT school     (#PCDATA)>
<!ELEMENT publisher  (#PCDATA)>
<!ATTLIST publisher  href CDATA #IMPLIED
>
<!ELEMENT note       (#PCDATA)>
<!ATTLIST cite       label CDATA #IMPLIED>
<!ELEMENT crossref   (#PCDATA)>
<!ELEMENT isbn       (#PCDATA)>
<!ELEMENT chapter    (#PCDATA)>
<!ELEMENT series     (#PCDATA)>
<!ATTLIST series     href CDATA #IMPLIED
>
<!ELEMENT layout     ANY>
<!ATTLIST layout     logo CDATA #IMPLIED
>

<!ELEMENT ref  (#PCDATA)>
<!ATTLIST ref  href CDATA #REQUIRED>
<!ELEMENT sup  (%titlecontents;)*>
<!ELEMENT sub  (%titlecontents;)*>
<!ELEMENT i    (%titlecontents;)*>
<!ELEMENT tt   (%titlecontents;)*>

<!ENTITY reg   "&#174;">
```

```
<!ENTITY micro "&#181;">
<!ENTITY times "&#215;">

<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % HTMLlat1 PUBLIC
       "ISO 8879-1986//ENTITIES Added Latin 1//EN//XML">
-->
<!-- This version of the entity set can be used with any SGML document
     which uses ISO 8859-1 or ISO 10646 as its document character
     set. This includes XML documents and ISO HTML documents.
-->

    <!ENTITY Agrave  "&#192;" ><!-- capital A, grave accent -->
    <!ENTITY Aacute  "&#193;" ><!-- capital A, acute accent -->
    <!ENTITY Acirc   "&#194;" ><!-- capital A, circumflex accent -->

    ⋮
```

Listing B.1: A dtd for DBLP

B.2 XML Data

The following extract illustrates the XML data that is read by the parser and considered in this work. The presented data records are randomly chosen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>

<article mdate="2002-01-03" key="journals/ac/Dasgupta89">
<author>Subrata Dasgupta</author>
<title>The Structure of Design Processes.</title>
<pages>1-67</pages>
<year>1989</year>
<volume>28</volume>
<journal>Advances in Computers</journal>
<url>db/journals/ac/ac28.html#Dasgupta89</url>
</article>

<inproceedings mdate="2006-04-04" key="journals/tochi/GuimbretiereMW05">
<author>Fran&ccedil;ois Guimbreti&egrave;re</author>
<author>Andrew Martin</author>
<author>Terry Winograd</author>
<title>Benefits of merging command selection and direct manipulation.</title>
<pages>460-476</pages>
<year>2005</year>
<booktitle>ACM Trans. Comput.-Hum. Interact.</booktitle>
<ee>http://doi.acm.org/10.1145/1096737.1096742</ee>
<url>db/journals/tochi/tochi12.html#GuimbretiereMW05</url>
</inproceedings>

<incollection mdate="2003-01-21" key="journals/lncs/Atzeni93">
<author>Paolo Atzeni</author>
<title>LOGIDATA+: Overview.</title>
<pages>1-6</pages>
<year>1993</year>
<crossref>books/sp/Atzeni93</crossref>
<booktitle>LOGIDATA+: Deductive Databases with Complex Objects</booktitle>
```

```
<url>db/journals/lncs/lncs701.html#Atzeni93</url>
</incollection>

<proceedings mdate="2005-04-26" key="journals/tcsb/2005-1">
<editor>Corrado Priami</editor>
<title>Transactions on Computational Systems Biology I</title>
<volume>3380</volume>
<year>2005</year>
<isbn>3-540-25422-6</isbn>
<booktitle>T. Comp. Sys. Biology</booktitle>
<series href="db/journals/lncs.html">Lecture Notes in Computer Science
</series>
<publisher>Springer</publisher>
<url>db/journals/tcsb/tcsb1.html</url>
</proceedings>

<book mdate="2002-01-03" key="books/infix/Zachmann97">
<author>Andreas Zachmann</author>
<title>Typsichere Objektbankmigration.</title>
<publisher>Infix Verlag, St. Augustin, Germany</publisher>
<year>1997</year>
<series href="db/series/disdbis/index.html">DISDBIS</series>
<volume>26</volume>
<isbn>3-89601-426-9</isbn>
</book>

</dblp>
```


Listing B.2: A small part of XML data from DBLP

INDEX

BIBLIOGRAPHY

[ABR64] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 – 837, 1964.

[ALLM05] Ali Al-Lawati, Dongwon Lee, and Patrick McDaniel. Blocking-aware private record linkage. In Laure Berti-Equille, Carlo Batini, and Divesh Srivastava, editors, *IQIS 2005, International Workshop on Information Quality in Information Systems, 17 June 2005, Baltimore, Maryland, USA (SIGMOD 2005 Workshop)*, pages 59–68. ACM, 2005.

[AMNR+06] Boanerges Aleman-Meza, Meenakshi Nagarajan, Cartic Ramakrishnan, Li Ding, Pranam Kolari, Amit P. Sheth, I. Budak Arpinar, Anupam Joshi, and Tim Finin. Semantic analytics on social networks: experiences in addressing the problem of conflict of interest detection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 407–416, New York, NY, USA, 2006. ACM Press.

[Bay63] Thomas Bayes. An essay towards solving a problem in the doctrine of chances. In *Philosophical Transactions of the Royal Society*, number 53, pages 370–418, 1763.

[BB98] Amit Bagga and Breck Baldwin. Entity-Based Cross-Document Coreferencing Using the Vector Space Model. In Christian Boitet and Pete Whitelock, editors, *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 79–85, San Francisco, California, 1998. Morgan Kaufmann Publishers.

[BB00] Amit Bagga and Alan W. Biermann. A methodology for cross-document coreference. In *Proceedings of the Fifth Joint Conference on Information Sciences*, pages 207–210, 2000.

[BD83] Dina Bitton and David J. DeWitt. Duplicate record elimination in large data files. *ACM Trans. Database Syst.*, 8(2):255–265, 1983.

[BGV92] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

[BL06] Kurt Bryan and Tanya Leise. The$ 25,000,000,000 Eigenvector: The Linear Algebra behind Google. *SIAM Review*, 48(3):569–581, 2006.

[BM05] Ron Bekkerman and Andrew McCallum. Disambiguating Web appearances of people in a social network. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 463–470. ACM, 2005.

[BMC+03] Mikhail Bilenko, Raymond J. Mooney, William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. Adaptive Name Matching in Information Integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[BS92] Christine L. Borgman and Susan L. Siegfried. Getty's Synoname and Its Cousins: A Survey of Applications of Personal Name-Matching Algorithms. *JASIS*, 43(7):459–476, 1992.

[BYRN99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CK99] Erica Chisholm and T. G. Kolga. New Term Weighting Formulas for the Vector Space Method in Information Retrieval. Technical report, Computer Science and Mathematics Division, Oak Ridge National Laboratory, 1999.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In Subbarao Kambhampati and Craig A. Knoblock, editors, *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico*, pages 73–78, 2003.

[Cun05] Hamish Cunningham. Information Extraction, Automatic. *Encyclopedia of Language and Linguistics, 2nd Edition*, 2005.

[Die05] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, electronic edition of 3. edition edition, 2005.

[Don05] Dong, X. and Halevy, A. and Madhavan, J. Reference reconciliation in complex information spaces. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96, 2005.

[DP96] Pedro Domingos and Michael J. Pazzani. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In *International Conference on Machine Learning*, pages 105–112, 1996.

[EL05] Ergin Elmacioglu and Dongwon Lee. On six degrees of separation in DBLP-DB and more. *j-SIGMOD*, 34(2):33–40, June 2005.

[Fei04] Dror G. Feitelson. On identifying name equivalences in digital libraries. *Inf. Res*, 9(4), 2004.

[Fis00] Gerd Fischer. *Lineare Algebra*. Vieweg, 12. edition edition, 2000.

[Fri97] Jerome H. Friedman. On Bias, Variance, 0/1–Loss, and the Curse-of-Dimensionality. *Data Min. Knowl. Discov.*, 1(1):55–77, 1997.

[FS69] I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[GG04] R. Guha and A. Garg. Disambiguating People in Search. In *Proceedings of the 13th international conference on World Wide Web, New York, USA*, 2004.

[GIKS03] Luis Gravano, Panagiotis G. Ipeirotis, Nick Koudas, and Divesh Srivastava. Text joins in an RDBMS for web data integration. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 90–101, New York, NY, USA, 2003. ACM Press.

[HGM$^+$03] Hui Han, C. Lee Giles, Eren Manavoglu, Hongyuan Zha, Zhenyue Zhang, and Edward A. Fox. Automatic document metadata extraction using support vector machines. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, pages 37–48, Washington, DC, USA, 2003. IEEE Computer Society.

[HGZ$^+$04] Hui Han, C. Lee Giles, Hongyuan Zha, Cheng Li, and Kostas Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. In Hsinchun Chen, Howard D. Wactlar, Ching chih Chen, Ee-Peng Lim, and Michael G. Christel, editors, *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2004, Tuscon, AZ, USA, June 7-11, 2004, Proceedings*, pages 296–305. ACM, 2004.

[HH04] A. Hassan and R. Holt. The Small World of Software Reverse Engineering. In *Proceedings of WCRE 2004: Working Conference on Reverse Engineering*, 2004.

[HOL04] Yoojin Hong, Byung-Won On, and Dongwon Lee. System Support for Name Authority Control Problem in Digital Libraries: OpenDBLP Approach. In Rachel Heery and Liz Lyon, editors, *Research and Advanced Technology for Digital Libraries, 8th European Conference, ECDL 2004, Bath, UK, September 12-17, 2004, Proceedings*, volume 3232 of *Lecture Notes in Computer Science*, pages 134–144. Springer, 2004.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 127–138. ACM Press, 1995.

[HW79] J. A. Hartigan and M. A. Wong. A K-Means Clustering Algorithm. *Applied Statistics*, 28:100–108, 1979.

[HXZG05] Hui Han, Wei Xu, Hongyuan Zha, and C. Lee Giles. A hierarchical naive Bayes mixture model for name disambiguation in author citations. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005*, pages 1065–1069. ACM, 2005.

[HySK04] Masanori Harada, Shin ya Sato, and Kazuhiro Kazama. Finding authoritative people from the web. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 306–313, New York, NY, USA, 2004. ACM Press.

[HZG03] Hui Han, Hongyuan Zha, and C. Lee Giles. A Model-based K-means Algorithm for Name Disambiguation. In *Proceedings of the Workshop at ISWC-03*, October 07 2003.

[HZG05] Hui Han, Hongyuan Zha, and C. Lee Giles. Name disambiguation in author citations using a K-way spectral clustering method. In *JCDL'05: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries*, Tools & techniques: identifying names of people and places, pages 334–343, 2005.

[Joh67] S. C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32(3):241–254, 1967.

[KLR$^+$04a] Stefan Klink, Michael Ley, Emma Rabbidge, Patrick Reuther, Bernd Walter, and Alexander Weber. Browsing and Visualizing Digital Bibliographic Data. In *VisSym*, pages 237–242, 2004.

[KLR$^+$04b] Stefan Klink, Michael Ley, Emma Rabbidge, Patrick Reuther, Bernd Walter, and Alexander Weber. Visualising and Mining Digital Bibliographic Data. In Peter Dadam and Manfred Reichert, editors, *INFORMATIK 2004 - Informatik verbindet, Band 2, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Ulm, 20.-24. September 2004*, volume 51 of *LNI*, pages 193–197. GI, 2004.

[Kro93] Robert Krovetz. Viewing morphology as an inference process. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–202, New York, NY, USA, 1993. ACM Press.

[LBG05] Levon Lloyd, Varun Bhagwan, and Daniel Gruhl. Disambiguation of References to Individuals. Technical report, 2005.

[LHK04] Mong-Li Lee, Wynne Hsu, and Vijay Kothari. Cleaning the Spurious Links in Data. *IEEE Intelligent Systems*, 19(2):28–33, 2004.

[LLL00] Mong-Li Lee, Tok Wang Ling, and Wai Lup Low. IntelliClean: a knowledge-based intelligent data cleaner. In *Knowledge Discovery and Data Mining*, pages 290–294, 2000.

[LOKP05] Dongwon Lee, Byung-Won On, Jaewoo Kang, and Sanghyun Park. Effective and scalable solutions for mixed and split citation problems in digital libraries. In *IQIS '05: Proceedings of the 2nd international workshop on Information quality in information systems*, pages 69–76, New York, NY, USA, 2005. ACM Press.

[LR06] Michael Ley and Patrick Reuther. Maintaining an Online Bibliographical Database: The Problem of Data Quality. In Gilbert Ritschard and Chabane Djeraba, editors, *Extraction et gestion des connaissances (EGC'2006), Actes des sixièmes journées Extraction et Gestion des Connaissances, Lille, France, 17-20 janvier 2006, 2 Volumes*, volume RNTI-E-6 of *Revue des Nouvelles Technologies de l'Information*, pages 5–10. Cépaduès-Éditions, 2006.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, Califonia, 1967. University of California Press.

[ME97] A.E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. *Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, 1997.

[Mil67] S. Milgram. The small world problem. *Psychology Today*, 2(1):60–67, 1967.

[MY03] G. Mann and D. Yarowsky. Unsupervised personal name disambiguation. *In CoNLL, Edmonton, Alberta.*, 2003.

[NM01] Tadashi Nomoto and Yuji Matsumoto. An Experimental Comparison of Supervised and Unsupervised Approaches to Text Summarization. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 630–632, Washington, DC, USA, 2001. IEEE Computer Society.

[OMR06] Simon Overell, João Magalhães, and Stefan Rüger. Place disambiguation with co-occurrence models. In Alessandro Nardi, Carol Peters, and Jose Luis Vicedo, editors, *CLEF 2006 Workshop, Working notes*, September 2006.

[PTH+03] Yves Petinot, Pradeep B. Teregowda, Hui Han, C. Lee Giles, Steve Lawrence, Arvind Rangaswamy, and Nirmal Pal. eBizSearch: an OAI-compliant digital library for eBusiness. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, pages 199–209, Washington, DC, USA, 2003. IEEE Computer Society.

[Reu06] Patrick Reuther. Personal Name Matching: New Test Collections and a Social Network based Approach. Technical report, 2006.

[RK] Yael Ravin and Zunaid Kazi. Is Hillary Rodham Clinton the President? Disambiguating Names across Documents.

[Rob04] S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.

[RT05] E. Rahm and A. Thor. Citation analysis of database publications. *ACM SIGMOD Record*, 34(4):48–53, 2005.

[RW06] Patrick Reuther and Bernd Walter. Survey on test collections and techniques for personal name matching. *IJMSO*, 1(2):89–99, 2006.

[Sal89] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[SM00] J. Schneider and A. Moore. *A locally weighted learning tutorial using vizier*. Carnegie Mellon University, the Robotics Institute, 2000.

[SvR00] M. M. M. Snyman and M. Jansen van Rensburg. Revolutionizing name authority control. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pages 185–194, New York, NY, USA, 2000. ACM Press.

[TKL06] Yee Fan Tan, Min-Yen Kan, and Dongwon Lee. Search Engine Driven Author Disambiguation. *JCDL*, 2006.

[VL63] V. N. Vapnik and A. Lerner. Pattern recognition using generalised portrait method. *Automation and Remote Control*, 24, 1963.

[vL06] U. von Luxburg. A tutorial on spectral clustering. Technical Report 149, Max Planck Institute for Biological Cybernetics, 08 2006.

[VPKV04] A. Valarakos, G. Paliouras, V. Karkaletsis, and G. Vouros. A Name-Matching Algorithm for Supporting Ontology Enrichment. *Proceedings of SETN*, 4, 2004.

[vRRP80] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. New models in probabilistic information retrieval. Technical report, 1980.

[Wat03] Duncan J. Watts. *Six Degrees: The Science of a Connected Age.* W. W. Norton, New York, 2003.

[WRC97] Nina Wacholder, Yael Ravin, and Misook Choi. Disambiguation of Proper Names in Text. In *ANLP*, pages 202–208, 1997.

[Y05] Soe-Tsyr Yuan and Jerry Sun 0002. Ontology-based structured cosine similarity in document summarization: with applications to mobile audio-based knowledge management. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(5):1028–1040, 2005.

[YCLK02] Shiren Ye, Tat-Seng Chua, Jimin Liu, and Jeremy R. Kei. Grouping Web Pages about Persons and Organizations for Information Extraction. In *ICADL '02: Proceedings of the 5th International Conference on Asian Digital Libraries*, pages 241–251, London, UK, 2002. Springer-Verlag.

[ZWS+05] Wenbo Zong, Dan Wu, Aixin Sun, Ee-Peng Lim, and Dion Hoe-Lian Goh. On assigning place names to geography related web pages. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 354–362, New York, NY, USA, 2005. ACM Press.