Author disambiguation in the YADDA2 software platform

Piotr Jan Dendek, Mariusz Wojewódzki, and Łukasz Bolikowski

Interdisciplinary Centre for Mathematical and Computational Modelling,
University of Warsaw
{p.dendek, m.wojewodzki, l.bolikowski}@icm.edu.pl

Abstract. SYNAT platform powered by the YADDA2 architecture has been extended with the Author Disambiguation Framework and the Query Framework. The former framework clusters occurrences of contributor names into identities of authors, the latter answers queries about authors and documents written by them. This paper presents an outline of the disambiguation algorithms, implementation of the query framework, integration into the platform and performance evaluation of the solution.

1 Introduction

1.1 Record deduplication

A common challenge among databases is a record deduplication, which is the term describing the situation when a real-world object is described by many separate records. "Record deduplication" itself is known in different communities as "record linkage" [1], "data cleaning" [2], "data scrubbing" [3], "mirror detection"[4], "instance matching" [5], etc. This issue may occur as a result of multiple formats used for representing an attribute, for example in the case of an address or a person name. The problem is particularly acute when information is gathered over a long period of time. Even when there is only one standard of representing a record, some misspelling or diacritics handling issues may occur. Another concern is merging data from multiple heterogeneous sources, when normalization levels and record definitions may not match. Last but not least, there are cases of automatic data acquisition (e.g. from the Internet), which is an instance of combining information from many origins. Those concerns are especially problematic when coping with big datasets.

The challenge of the accurate record linkage had been addressed many times over the last five decades. Elmagarmid *et al.* [6] et al. coherently enumerate all mainstream approaches to the record linkage. Authors focus on specific aspects of the problem, which are:

1. Data preparation, covering a parsing and a standardization procedures. [7–11]
2. Attribute matching techniques, as approximate string matching, token based and phonetic based. [12–15]

3. Duplicate detection, covering supervised and unsupervised machine learning techniques as well as hand-crafted ones. [16–19]
4. Problem decomposition approaches as blocking, k-nearest neighbour, clustering. [20–22]

1.2 Author disambiguation

Author disambiguation is an instance of the record linkage problem, where instances to match are authors, typically represented by first names, a surname, an affiliation and metadata of co-authored articles. It is clear that none of the mentioned attributes can single-handedly solve the entire problem. There are attributes that determine identity with a high degree of certainty, but they are frequently not present, e.g. an e-mail address appears only in 10% of articles [23].

In author disambiguation all typical object deduplication obstacles arise, beginning from many standards for writing a name ("J.Smith", "John Smith", "J.Smith Jr.", "Smith, J."), misspellings ("J.Smiht", "J.Smth"), an attribute value change over time ("Eleonore Smith", "Eleonore Smith-Black"), diacritic handling ("José Gonçalves", "Jose Goncalves","Jos? Gon?alves"), transliterations (e.g. translating "Angela Johnson" to Japanese equivalent " アンジェラ·ジョンソン " and back to English result is "anjira jyonson") [24] and extraction artifacts ("Smith Machine"). As Torvik and Smalheiser [25] investigated, about 1,3% authors whose e-mail addresses match have different surnames, most likely due to inconsistencies enumerated here.

The other attributes also need further consideration, e.g. some errors may occur in an e-mail address, like "jsmith@@institution.org" or "jsmith"institution.org", hence requiring a rectification step.

Many researchers explored specifically the subject of the author disambiguation. Han *et al.*[26] compared Naive Bayes and Support Vector Machines (SVM) classifiers for this task, whereas in [27], they examined efficiency of k-Way Spectral Clustering. Concurrently, Dai and Storkey [28] applied hierarchical Dirichlet process and nonparametric latent Dirichlet allocation models, whereas Levin and Heuser [29] included in their solution enhancements derived from the genetic programming.

Typically, authors tried to conduct pairwise comparisons on a set of records with the same value of a major feature (e.g. surname) to determine whether two candidate author items are the same. In contrast to performing analysis in respect of all given features, Qian *et al.* [23] proposed to perform initial clustering with a limited number of features to obtain High Precision Clusters in the first step and then merge clusters into High Recall Clusters in the second step. They also proposed to introduce a human judgement clustering in the final step. When utilizing user feedback, it is crucial to distinguish between experts and regular person, especially preventing acts of vandalism, considered as sending false information.

Culotta *et al.* [30] proposed to pre-assess each contributor block to determine the likelihood of duplicates. For example, if all authors are affiliated to a few

institutions or e-mail addresses then the cluster of candidate items is more likely to have duplicates than the one that contains contributions associated with a high number of e-mails and institutions. Authors claim that the usage of so called first-order features over sets of records may eventually reduce error rate that outperforms a regular binary classification by up to 60%. Tan *et al.* [31] decided to extend an available set of information by employing Internet search engines and adopting as a feature home pages containing given article.

The rest of this paper is organized as follows. Section 2 describes both the SYNAT project and the YADDA2 architecture. Section 3 presents the Author Disambiguation Framework (ADF) developed for purposes of YADDA2 [32] and the results of further examination [33]. Section 4 shows adaptation of the ADF to SYNAT platform with an emphasis on its presentation layer – the Query Framework (QF). Section 5 contains evaluation of the ADF and the QF. Finally, Section 6 concludes the paper and proposes further improvements.

2 SYNAT and YADDA2

SYNAT project aims to build an "Interdisciplinary System for Interactive Scientific and Scientific Technical Information." It is a strategic project commissioned by the Polish National Centre for Research and Development. The system is based on the YADDA2 architecture [34], presented in Figure 1, developed at ICM UW.

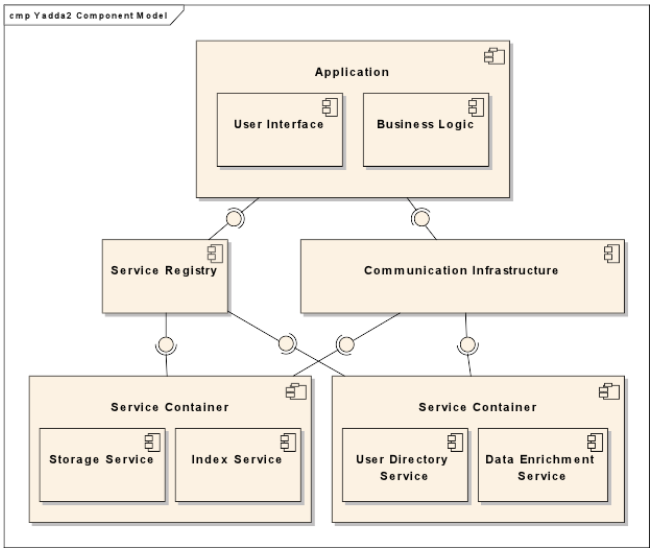

Fig. 1: Component model of the YADDA2 framework.

YADDA2 is an open, loosely-coupled, service-oriented and modular framework that facilitates development of digital repository applications. The framework contains a number of reusable modules that provide, among others: storage, relational and full-text indexing, process management, authorization/authentication and asynchronous communication. The above are so-called base services, providing general functionalities which are independent of the type of content being processed. On top of them, there is a number of more specialized compotents that implement a business logic layer.

The Author Disambiguation Framework, as well as the Query Framework described later in this paper, are good examples of such specialized components.

3 YADDA2 author disambiguation framework

3.1 Vocabulary

In our previous papers [32, 33] we have established the vocabulary for ADF description, which, after further adjustments, can be described as follows. A **contributor** entity reflects the fact that a **person** was a co-author of a **document**. When data about a contributor is extracted from document medatada, it may be treated as a one-to-one binding between a person and a document. For each document, the number of corresponding contributor instances is equal to the number of co-authors. Occasionally, we may also reuse the concept of a document to represent a different tangible outcome that can be attributed to a person or persons, for example, a log of user actions in an Internet service. Having a set of contributors, the goal is to cluster them into groups containing publications written by one person.

To do so in the efficient way, we perform a coarse-grained grouping into **blocks** of contributors according to a **hash function**. Typically, a hash function yields a result which is a function of surname, like a diacritics removal and lowercasing. Depending on authors' surnames, a more sophisticated hash function, e.g. Soundex or Double Metaphone phonetic transformations, may be chosen. This division step corresponds to the "map" phase of the MapReduce paradigm.

Consequently, the "reduce" stage is performed, in which **crude features** are calculated. A crude feature is an integer representing a number of common values, e.g. identical key words. Afterwards, we obtain a **feature** by scaling a crude feature into the [0,1] range, which is multiplied by a feature weight yielding an **atomic affinity**. A sum of atomic affinities is called a **total affinity** and constitutes the input data for a clustering algorithm.

As mentioned above, this approach is customizable with respect of a hash function, a set of features with associated weights and a clustering function. Finally, this solution is fully applicable in a MapReduce workflow.

3.2 Author Disambiguation Framework Flow

In [32] we presented the Author Disambiguation Framework flow, which is briefly summarised in this section. The ADF flow consists of five steps:

1. Data import. Data is transferred to an ADF database.
2. Blocking. All contributors are split into relatively small, computationally less expensive subsets.
3. Affinity calculation. Pairwise comparison against a set of crude features accompanied with their weights is performed in each block, eventually generating a total affinity.
4. Clustering. Contributors which are recognized as similar are inserted into the same group.
5. Result persistence. A connection between a contributor and a cluster is stored either in a database or in text files.

The framework may be initialized by passing two kinds of input data as presented in Figure 2:

1. a document collection from which contributors are extracted,
2. a collection that contains information about contributors with information associated to them and optionally contributors' documents.

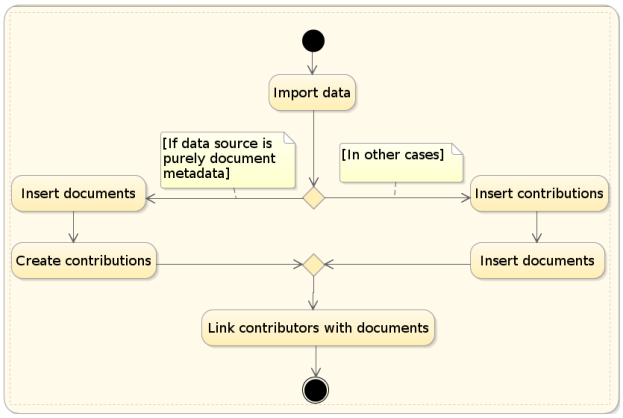

Fig. 2: The process of data import to the Author Disambiguation Framework. It may either extract data about contributors or utilize pre-generated ones.

Calculations on blocks may be processed in parallel taking advantage of a multicore computer architecture.

4 Implementation

4.1 Author Disambiguation Framework Implementation

The ADF has been implemented purely in Java using two databases: bigdata® and Sesame. In both of them the basic entity is a ordered triple, containing information about subject, predicate and object, $t_i = \langle s, p, o \rangle$, where the first two

are Uniform Resource Identifier (URI) objects and the last one can either be an URI or a literal (non-URI) object. ADF methods are implemented using Resource Description Framework Storage And Inference Layer (RDF SAIL) which is a standard set of interfaces defining an API for RDF repositories. As the result, it can connect to standard triple stores such as the ones mentioned above. bigdata® is capable of fitting about 12.7 billion triples in its hard drive journal file, whereas Sesame may accumulate 70 millions of triples in a memory[1]. Due to the size of a database and its localization (a hard drive file vs. memory) the second mentioned triple store outperforms the other one in terms of a communication time by about tree orders of magnitude.

Taking into account these facts, the ADF uses bigdata® to collect all imported data, whereas for each block, the ADF creates a cache in Sesame memory store where all data needed for calculating affinity are transferred. This particular approach proved to be the most fruitful, synergistic strategy in terms of the performance. Eventually, resulting person objects may be written back to a bigdata® or to CSV files.

4.2 Query Framework Implementation

Data structures Query Framework (QF), similarly to ADF, is written entirely in Java, but it takes advantage of the Neo4j database[2]. Neo4j, as the example of a NoSQL database has been the subject of detailed comparison [35] with the traditional, relational approach.

The reason for choosing Neo4j was its flexibility in a model construction as well as its high efficiency. A few specialized data structures have been applied to obtain a better performance. For instance, to increase a search performance, identifiers and attributes of stored objects are indexed with full-text indexes embedded into Neo4j.

The structure of data is built as follows. The top element is a **root**, which due to Neo4j restrictions always exists in a database, even if not inserted explicitly. A root is bound by a **root relation** with a **person**, which, as a reflection of a real-world author, should point by an **identity relation** to all **contributor** instances corresponding to that person. A **contributor** stores information about publications or activities associated with a **person** object. In case when a data source is solely documents metadata, contributor-document is a one-to-one relation. On the contrary, when a data source is derived from any other origin, contributor-document is a one-to-many relation (one-to-zero also applies).

The database structure is presented in Figure 3.

During import, the QF takes metadata (containing information about contributors and optionally documents) and information about persons from the ADF. Finally, the QF constructs its internal structures described in the following section.

[1] See: `http://www.w3.org/wiki/LargeTripleStores`
[2] See: `http://neo4j.org/`

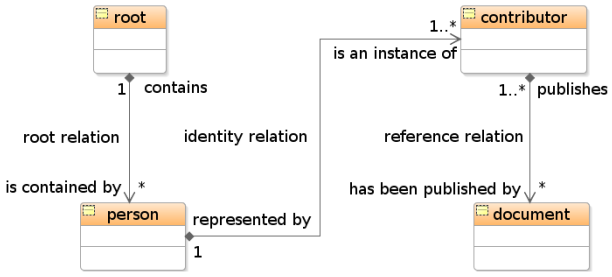

Fig. 3: Classes appearing in the Query Framework's Neo4j database. The top instance is singular root object, which always exists in the Neo4j store. Person objects are created as a reflection of the Author Disambiguation Framework results. Contributors are instances of persons. Documents are publications written by given contributor and transitively by a person.

Model creation Currently, data are imported from files generated in the process of disambiguation. In order to add to the service the information about a person:

1. The system checks if the person already exists to omit duplicates, which may occur whenever a next import extends only a few blocks of previously digested data.
2. Person is created and linked to the root.
3. Contributors and documents are created and connected in the top-down manner.

A contributor can exist without any document, as a result of employing another data source which does not provide details about publications and activities.

The above description is reflected in the the activity diagram in Figure 4.

Model usage Queries directed to the service may be routed through a website, or can be called directly by the appropriate method of the QF API. After loading the QF database, a user can construct queries as follows:

- "Find one person object" – used to construct queries that are aimed at finding one person with all contributors that are in identity relation with it. Input parameter is a person identity.
- "Find one person object and related publications" – used to construct queries that are designed to find one person and associated contributors together with dependent documents. Input parameters are: person identity, order and attribute criteria.
- "Find many person objects" – used to construct queries that are aimed at finding persons in a certain order with previously specified constraints. Input parameters are: order and attribute criteria.

Activities mentioned above are also reflected in Figure 5.

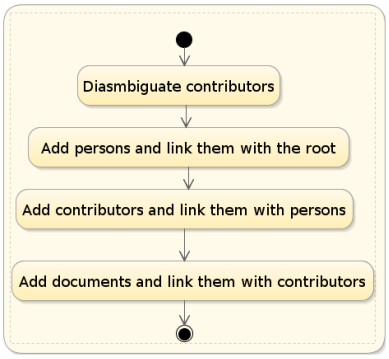

Fig. 4: The Query Framework preparation. The first step is the disambiguation procedure conducted by the Author Disambiguation Framework. Next, results of the ADF are inserted to the QF and connected with the Neo4j root. Then, the regular metadata are imported.

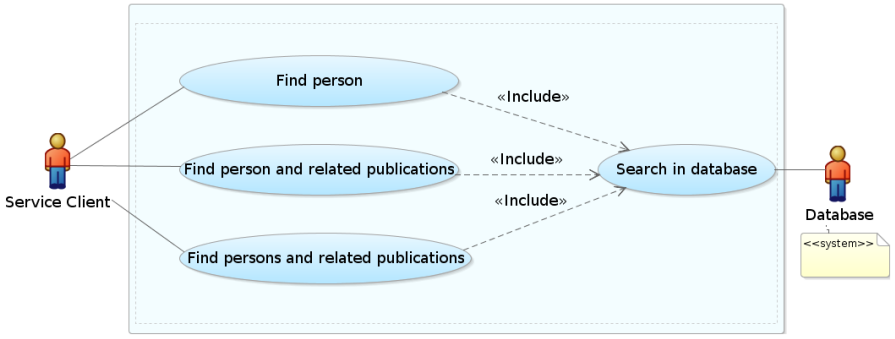

Fig. 5: Actions available in the Query Framework.

5 Evaluation

We conducted performance tests on a 4 cores machine (Intel®Core™i5 CPU U520@1.07GHz) with 4GB RAM under control of Ubuntu Linux OS, kernel version 2.6.32-41-server. We focused on the BazTech bibliographic database[3] containing document metadata with 257784 contributors.

5.1 Author Disambiguation Framework Performance

Let $BlockSize$ denote a set of contributors sizes and the function $BlockOcc(x)$ returns a number of occurences of a block of size x The disambiguation time for

[3] See: `http://baztech.icm.edu.pl/`

a block of size x can be closely approximated by the formula:

$$T_{disambiguation}(x) = 0,0608099 \cdot x + 0,0005811 \cdot x^2 \quad (1)$$

where the part $0,0005811 \cdot x^2$ is mainly the effect of performing pairwise comparison, but also employing a single-link hierarchical clustering, which has $\mathcal{O}(N^2)$ computation complexity. It has to be mentioned that a clustering procedure is at least one order of magnitude faster [36] then similarity computation, so choosing another clustering method would not have a crucial impact on an overall disambiguation time, which is given by the following equation:

$$T_{overall}(BlockSize) = \sum_{x \in BlockSize} BlockOcc(x) \cdot T_{disambiguation}(x) \quad (2)$$

5.2 Query Framework Performance

To evaluate the QF we carried out two types of tests. The first one was a data import of the BazTech metadata to the Neo4j database, which took about 2 hours. The second one covered three queries described in the Section 4.2. The "Find one person object" query repeated 10000 times for a different person finished in the total time of 3 seconds, whereas the query "Find one person object and related publications" executed 10000 times for a different person took about 6 seconds. The time increased as a natural effect of retrieving more information, located further from the root. The last query we examined, "Find 10000 person objects", was executed in 4 seconds. Described results are presented in the Table 1.

Table 1: Results of the Query Framework performance tests against provided query types.

Query goal	Number of repetitions	Overall time
Find one person object	10000	3s
Find one person object and related publications	10000	6s
Find 10000 person objects	1	4s

6 Conclusions

In this article we have presented the Author Disambiguation Framework developed in the YADDA2 architecture and adopted for the SYNAT platform by means of the Query Framework. We have described the details of disambiguation as well as the method of adaptation and a presentation layer. We proposed the time complexity of the disambiguation process accompanied with the Query Framework performance tests.

Future work will include applying more sophisticated disambiguation techniques, followed by the use of more efficient computing architectures. Ideally, each framework should rely on the same database and data structures. Furthermore, we plan to extend the disambiguation procedure to cover activities other then publishing. Another promising direction is the enhancement of disambiguation results with a user feedback to fully utilize SYNAT platform capabilities.

Acknowledgment

This work is supported by the National Centre for Research and Development (NCBiR) under Grant No. SP/I/1/77065/10 by the Strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".

References

1. I. P. Fellegi and A. B. Sunter, "A Theory for Record Linkage," Journal of the American Statistical Association, vol. 64, pp. 1183–1210, 1969.
2. K. Park, E. Becker, J. K. Vinjumur, Z. Le, and F. Makedon, "Human behavioral detection and data cleaning in assisted living environment using wireless sensor networks," in Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments - PETRA '09, (New York, New York, USA), pp. 1–8, ACM Press, 2009.
3. J. J. Berman, "Concept-Match Medical Data Scrubbing," Archives of Pathology & Laboratory Medicine, vol. 127, no. 6, pp. 680–686, 2003.
4. J. Widom, "Research problems in data warehousing," in Proceedings of the fourth international conference on Information and knowledge management - CIKM '95, (New York, New York, USA), pp. 25–30, ACM Press, 1995.
5. M. Aono and M. H. Seddiqui, "Scalability in ontology instance matching of large semantic knowledge base," in AIKED'10 Proceedings of the 9th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases, pp. 378–383, 2010.
6. A. Elmagarmid, P. Ipeirotis, and V. Verykios, "Duplicate Record Detection: A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 19, pp. 1–16, Jan. 2007.
7. V. Raman, "Potter's wheel: An interactive data cleaning system," International conference on Very Large Data, 2001.
8. C. Sutton, K. Rohanimanesh, and A. McCallum, "Dynamic conditional random fields," in Twenty-first international conference on Machine learning - ICML '04, (New York, New York, USA), p. 99, ACM Press, 2004.
9. E. Agichtein and V. Ganti, "Mining reference tables for automatic text segmentation," in Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04, (New York, New York, USA), p. 20, ACM Press, 2004.
10. V. Borkar, K. Deshmukh, and S. Sarawagi, "Automatic segmentation of text into structured records," in Proceedings of the 2001 ACM SIGMOD international conference on Management of data - SIGMOD '01, (New York, New York, USA), pp. 175–186, ACM Press, 2001.

11. A. McCallum and D. Freitag, "Maximum entropy Markov models for information extraction and segmentation," Proceedings of the Seventeenth International Conference on Machine Learning, 2000.
12. L. Philips, "The double metaphone search algorithm," C/C++ Users Journal, vol. 18, no. 6, pp. 38–43, 2000.
13. K. Kukich, "Technique for automatically correcting words in text," ACM Computing Surveys, vol. 24, pp. 377–439, Dec. 1992.
14. E. Ristad and P. Yianilos, "Learning string-edit distance," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, pp. 522–532, May 1998.
15. G. Navarro, "A guided tour to approximate string matching," ACM Computing Surveys, vol. 33, pp. 31–88, Mar. 2001.
16. S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," Information Systems, vol. 26, pp. 607–633, Dec. 2001.
17. W. W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02, (New York, New York, USA), p. 475, ACM Press, 2002.
18. V. S. Verykios and G. V. Moustakides, "A generalized cost optimal decision model for record matching," in Proceedings of the 2004 international workshop on Information quality in informational systems - IQIS '04, (New York, New York, USA), p. 20, ACM Press, 2004.
19. V. Verykios, G. Moustakides, and M. Elfeky, "A Bayesian decision model for cost optimal record matching," The VLDB Journal The International Journal on Very Large Data Bases, vol. 12, pp. 28–40, May 2003.
20. A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, (New York, New York, USA), pp. 169–178, ACM Press, 2000.
21. A. Monge and C. Elkan, "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records," in Proc. Second ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery, pp. 23–29, 1997.
22. M. A. Hernández and S. J. Stolfo, "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem," Data Mining and Knowledge Discovery, vol. 2, no. 1, pp. 9–37, 1998.
23. Y. Qian, Y. Hu, J. Cui, Q. Zheng, and Z. Nie, "Combining Machine Learning and Human Judgment in Author Disambiguation Framework," in Proceedings of the 20th ACM International Conference on Information and Knowledge Management, pp. 1241–1246, ACM Press, 2011.
24. K. Knight and J. Graehl, "Machine Transliteration," Computational Linguistics, vol. 24, no. 4, pp. 599–612, 1998.
25. V. I. Torvik and N. R. Smalheiser, "Author name disambiguation in MEDLINE," ACM Transactions on Knowledge Discovery from Data, vol. 3, pp. 1–29, July 2009.
26. H. Han, L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis, "Two supervised learning approaches for name disambiguation in author citations," Proceedings of the 2004 joint ACM/IEEE conference on Digital libraries - JCDL '04, p. 296, 2004.
27. H. Han, H. Zha, and C. L. 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, (New York, NY, USA), pp. 334–343, ACM, 2005.

28. A. M. Dai and A. J. Storkey, "Author Disambiguation: A Nonparametric Topic and Co-authorship Model," in NIPS Workshop on Applications for Topic Models Text and Beyond, pp. 1–4, 2009.
29. F. H. Levin and C. A. Heuser, "Using Genetic Programming to Evaluate the Impact of Social Network Analysis in Author Name Disambiguation," in Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management Buenos Aires Argentina May 1720 2010 (A. H. F. Laender and L. V. S. Lakshmanan, eds.), vol. 619 of CEUR Workshop Proceedings, Citeseer, 2010.
30. A. Culotta, P. Kanani, R. Hall, M. Wick, and A. McCallum, "Author Disambiguation using Error-driven Machine Learning with a Ranking Loss Function," in Sixth International Workshop on Information Integration on the Web, 2007.
31. Y. F. Tan, M. Y. Kan, and D. Lee, "Search engine driven author disambiguation," in Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries - JCDL '06, (New York, New York, USA), p. 314, ACM Press, 2006.
32. L. Bolikowski and P. J. Dendek, "Towards a Flexible Author Name Disambiguation Framework," in Towards a Digital Mathematics Library (P. Sojka and T. Bouche, eds.), pp. 27–37, Masaryk University Press, 2011.
33. P. J. Dendek, L. Bolikowski, M. Lukasik, L. Bolikowski, and M. Lukasik, "Evaluation of Features for Author Name Disambiguation Using Linear Support Vector Machines," in Proceedings of the 10th IAPR International Workshop on Document Analysis Systems, pp. 440–444, 2012.
34. W. Sylwestrzak, T. Rosiek, and L. Bolikowski, "YADDA2 – Assemble Your Own Digital Library Application from Lego Bricks," in Proceedings of the 2012 ACM/IEEE on Joint Conference on Digital Libraries, 2012.
35. C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database," in Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10, (New York, New York, USA), p. 1, ACM Press, 2010.
36. H. Manning, C., D., Raghavan P., Schütze, "Introduction to Information Retrieval," 2008.