Towards a flexible author name disambiguation framework

Łukasz Bolikowski[1] and Piotr Jan Dendek[1,2]

[1] Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw, ul. Pawińskiego 5A blok D, 02-106 Warsaw, Poland
[2] Faculty of Electronics and Information Technology, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract. In this paper we propose a flexible, modular framework for author name disambiguation. Our solution consists of the core which orchestrates the disambiguation process, and replaceable modules performing concrete tasks. The approach is suitable for distributed computing, in particular it maps well to the MapReduce framework. We describe each component in detail and discuss possible alternatives. Finally, we propose procedures for calibration and evaluation of the described system.

Keywords: name disambiguation, problem decomposition, scoring functions, single-linkage clustering, MapReduce framework, machine learning

1 Introduction

A person's name may be presented in several different forms, for example: "M. Brown", "Michael Brown", "M. A. Brown", "Michael A. Brown", or "Michael Arthur Brown". On the other hand, the same name form may, depending on the context, refer to several different people. Furthermore, when processed by a computer system, a name form may be distorted due to deficiencies of the software used, e.g. due to OCR errors, or incompatible handling of diacritics.

In a digital library storing scientific publications, such as the European Digital Mathematics Library (EuDML), it is of great interest to associate the names of a document's contributors with their identities. This piece of information is crucial for several purposes, including: presentation of all the works of a particular author in a concise form; analysis of researchers' co-operation network and identification of communities [12]; or assessing the impact of individual researchers.

The problem of associating names with identities is commonly referred to as name disambiguation. Torvik and Smalheiser [13] show that almost 2/3 of authors in MEDLINE have an ambiguous name (surname + first initial), thus providing a good motivation for further study.

Several approaches to the problem have been proposed in the literature. Han *et al.* [4] offer two solutions based on supervised learning: one using naive Bayes, the other using Support Vector Machines. Mann and Yarowsky [8], as well as Pedersen *et al.* [11] and Han *et al.* [5] propose various unsupervised clustering approaches. Kang *et al.* [7] stress the importance of co-authorship analysis in the name disambiguation process. Torvik and Smalheiser [13] propose a 5-step procedure to disambiguate names in an entire collection of documents.

In related research, Galvez and Moya-Anegón [3] employ finite-state graphs to standarize name variants, while Pavelec *et al.* [10] address the problem of finding the author of an anonymous document using stylometry.

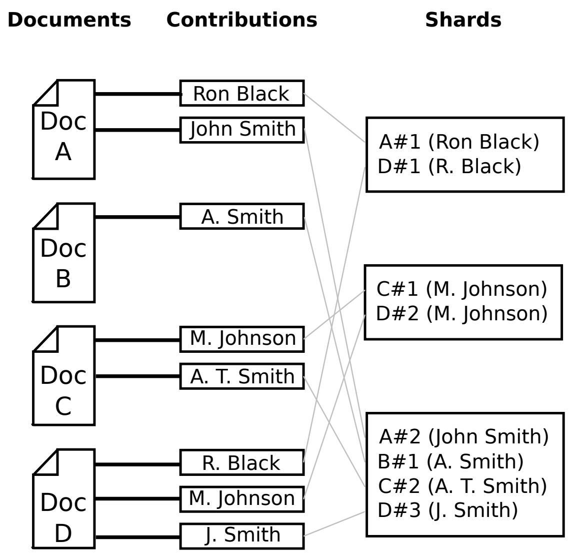


Fig. 1. Documents, contributions, and shards. Decomposition of contribution space into shards is performed according to some hash function, for example: surname of the contributor.

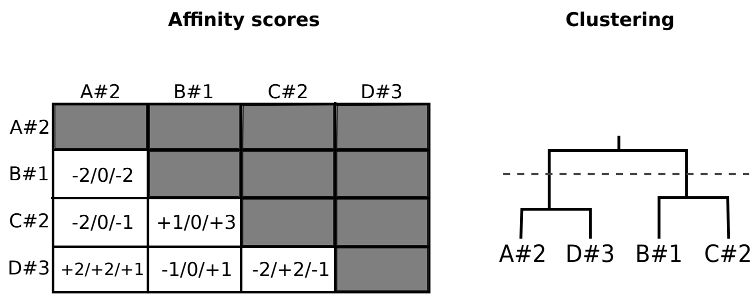
2 Author name disambiguation framework

2.1 Vocabulary and assumptions

Let us begin by establishing a vocabulary that will be used throughout the rest of this paper and stating certain assumptions about the objects discussed.

A **document** is an article or a book, referenced by an identifier that is unique within the collection. We assume to have an access to each document's metadata, which at the very least contain names of all the authors.

A **contribution** is an occurrence of an author's name in a document's metadata. Thus, a contribution refers to exactly one document and to exactly one person. For the sake of clarity, let us identify a contribution by concatenating the document's identifier with "#" and with (one-based) position of the name on the list of authors of the document. For example, if a document with identifier



	A#2	B#1	C#2	D#3
A#2				
B#1	-2/0/-2			
C#2	-2/0/-1	+1/0/+3		
D#3	+2/+2/+1	-1/0/+1	-2/+2/-1	

Fig. 2. For each pair of contributions within a shard, affinity is calculated based on several attributes. Next, a clustering algorithm is executed to extract author identities.

`124532` has three authors: J. Stone, M. Black and A. Smith, then the contribution of M. Black to the document is identified by `124532#2`.

Next, a **shard** is a group of contributions such that a **hash function** yields the same value for all the contributions in the shard. We are free to choose any hash function which satisfies the following condition: all the contributions of the same person should have the same hash. The converse does not have to be true, though: contributions of several different people may have the same hash. A case satisfying the above conditions is shown in Figure 3. There are contributions (visualised as “document” icons) gathered into shards (ellipses). Papers written by the same person (“human” icon) should be in the same shard. It may occur that papers of more then one person will appear in the same shard, which is valid and expected. The described situation is the result of a well-constructed hash function. In contrast, shards in Figure 4 are the result of a defective hash function, which splits contributions of same person into more then one shard.

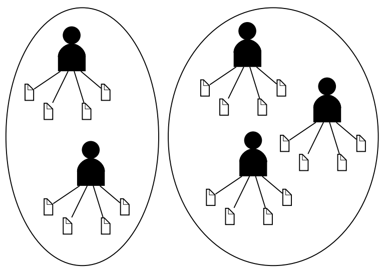


Fig. 3. Contributions (document icons) of the same person (human icons) should be present in the same shard (ellipses).

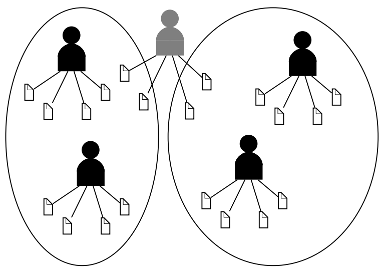


Fig. 4. The result of a defective hash function is a division of contributions (document icons) of the same person (human icons) into different shards (ellipses). The grey person’s papers are split in such a manner.

The purpose of a hash function is to decompose the problem space into manageable shards. A good example of a hash function is a function which returns the lower-cased surname of a name with all the diacritic marks removed (cf. Fig. 1).

An **attribute** (or a **feature**) is a function that extracts some feature from each contribution, such as: year of publication of the document, list of keywords in the document, or e-mail address of the person. Such an attribute may be either taken from the document's metadata, or inferred from the full-text. Whenever we write "an attribute of a contribution", we mean the value of the attribute for the contribution. When a given contribution lacks a given attribute, we shall denote the value by "$\perp$".

Affinity of a pair of contributions is a real number which is a measure of our confidence that the two contributions are made by the same person. It is a weighted sum of **atomic affinities** calculated for individual attributes. We assume that for each atomic affinity is a function that returns values from the range $[-1, 1]$. When returned value is equal 1 it means that according to that function two contributions share the same personality for sure. Zero value point out that function could not determine if contributions share the same person or not. Finally, -1 value indicate that two contributions are made by two different persons. The relative imporance of atomic affinities (and thus, indirectly, of attributes) is controlled by **weights**, which are non-negative real numbers.

Finally, a cluster of contributions believed to be made by the same person will sometimes be referred to as an **identity**.

2.2 Name disambiguation flow

Our name disambiguation procedure, presented in Figure 5, takes a collection of documents on input, and yields sets of contributions likely to be made by the same person on output. The procedure consists of three steps:

1. all the contributions in the input collection of documents are partitioned into shards (cf. Figure 1);
2. for each pair of contributions within each shard, its affinity is calculated (cf. Figure 2);
3. contributions within each shard are clustered (cf. Figure 2).

As a result of the last step, we have groups of contributions suspected to be made by the same person. The entire solution maps well to the Google's MapReduce framework [2].

Each part of the process is described in the following part of the article.

2.3 Partition of the input collection

The problem domain of name disambiguation is typically in the order of millions of contributions. Since a part of our solution employs pairwise comparison (quadratic computational complexity), it is crucial to decompose the domain beforehand, thus reducing the computational effort.

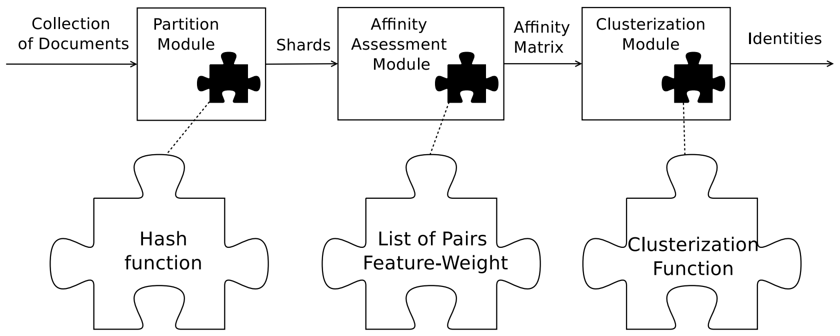


Fig. 5. Name disambiguation procedure stages. Puzzles represent replaceable parts of the solution.

When looking for same authorship of contributions, it is also apparent that some contributions do not need to be compared (e.g. they were written in time interval of two hundred years). Using this information, we can extract from the collection a working set of interest, in which documents are similar in some way, simultaneously filtering out other contributions, unlikely to be made by the same author.

For the sake of simplicity, one can restrict our attention to filters which partition the contributions into disjunctive working sets. Let us call such filters "hash functions", and the resulting working sets—"shards".

The framework is flexible and allows us to plug in any hash function satisfying the conditions stated in Subsection 2.1. We have chosen a basic hash function which takes the surname, lower-cases it, and removes all the diacritic marks.

Torvik and Smalheiser [13] found that misspellings, differences in authors' surnames notations or differences of complete surnames (e.g., "Olle Goig" vs. "Olle-Goig", "Le Roith" vs. "LeRoith", or "J. Benson" and "J. Flynn" both referring to Judy Benson Flynn) appear in approximately 1.8% of contributions. Moreover, the hash function described earlier is resilient to the majority of these discrepancies. Therefore, we have decided to ignore such mistakes.

However, if one chooses to account for misspellings, it can be achieved by using a more sophisticated hash function, for example one feeding the surname to the Soundex algorithm. (although differences in pronunciation across countries may spoil the result). It is not advised to use editorial distance, because in this case shards may overlap. Another way to extend the basic hash function is to include the initial of the first name in the hash and removing non-alphabetic characters (e.g.dashes) and diacritic marks from a surname. The biggest disadvantage of the described hash function is that it works well with surnames written with Latin alphabet. If one needs to proceed surnames written in different alphabets, they may write appropriated function to translate them into same alphabet representation. Whereas if one works on a set of surnames written in same alphabet, they do not need to employ any sophisticated surname translation function. On a side note, this illustrates the flexibility of the proposed framework.

The final product of the partition performed using the basic hash function is a large number of small shards and a small number of larger shards. In the collection of Polish History Museum (about 100 thousand documents, mostly arts and humanities), we have found that roughly 55% of all contributions are in shards of size 10 or less, while the largest shard was in the order of a thousand. In MEDLINE (over 15 million documents, mostly life science) when using surname + initial of the first name as the hash, Torvik and Smalheiser [13] found that the largest shard ("J. Lee") contains almost 16 thousand contributions. Given this order of magnitude of shard sizes, it is feasible to run an analysis of quadratic time complexity on each of the shards.

To sum up, the described decomposition of contributions dramatically reduces the computational complexity of the second step in the procedure. As an added benefit, the individual shards can be processed in parallel, further reducing the wall-clock time of the entire process.

2.4 Pairwise contribution comparison

Once the domain is decomposed into shards, we need to establish affinities of pairs of contributions in each shard. The total affinity is the sum of atomic affinities, assessed from an extensible set of features (a list of feature examples is presented in Table 1).

A feature is a method, which takes two contributions on input and returns a real number from the $[-1, 1]$ range. The resulting number is multiplied by the weight of the feature, giving an atomic affinity. Finally, atomic affinities are summed up to one number—total affinity.

For each shard, the output of the procedure is a matrix of contribution affinities. The matrix is passed to the next stage of the name disambiguation process.

Feature weight Some features, such as e-mail, can alone prove that two contributions share the same identity. Other features are only weak indicators, e.g. contributing to the same journal. The weight of a feature is introduced to reflect the feature's impact on the name disambiguation process.

Feature aspects Referring to the Table 1 we can point out a few aspects of a feature:

- Discretization level
  1. **Discrete.** Some features are highly discrete, e.g. e-mail feature.
  2. **Continuous.** Some features give a continuous result, e.g. continuous time distance feature.
- Polarisation level
  1. **Polarised.** A feature can be highly positive (negative) indicator and simultaneously weak negative (positive) indicator, e.g. e-mail or journal feature.

Name	**Description**
Time distance (continuous)	$= \begin{cases} 0 & \text{year}(c_1) = \perp \\ & \vee\ \text{year}(c_2) = \perp \\ -1 & \lvert\text{year}(c_1) - \text{year}(c_2)\rvert > 70 \\ \left(\frac{\text{year}(c_1)-\text{year}(c_2)}{70}\right)^2 & \text{otherwise} \end{cases}$
Time distance (discrete)	$= \begin{cases} 0 & \text{year}(c_1) = \perp \vee \text{year}(c_2) = \perp \\ -1 & \lvert\text{year}(c_1) - \text{year}(c_2)\rvert > 70 \\ 1 & \text{otherwise} \end{cases}$
Journal	$= \begin{cases} 0 & \text{journal}(c_1) = \perp \vee \text{journal}(c_2) = \perp \\ 1 & \text{journal}(c_1) = \text{journal}(c_2) \\ -0.1 & \text{otherwise} \end{cases}$
Email	$= \begin{cases} 0 & \text{email}(c_1) = \perp \vee \text{email}(c_2) = \perp \\ 1 & \text{email}(c_1) = \text{email}(c_2) \\ -0.1 & \text{otherwise} \end{cases}$
Language	$= \begin{cases} 0 & \text{language}(c_1) = \perp \vee \text{language}(c_2) = \perp \\ 0.05 & \text{language}(c_1) = \mathit{eng} \vee \text{language}(c_2) = \mathit{eng} \\ 0.1 & \text{language}(c_1) = \text{language}(c_2) \\ -1 & \text{otherwise} \end{cases}$
Keywords (discrete)	$= \begin{cases} 0 & \text{keyword}(c_1) = \emptyset \vee \text{keyword}(c_2) = \emptyset \\ -1 & \frac{\lvert\text{keyword}(c_1)\cap\text{keyword}(c_2)\rvert}{\lvert\text{keyword}(c_1)\cup\text{keyword}(c_2)\rvert} < 0.25 \\ 1 & \text{otherwise} \end{cases}$
Keywords (continuous)	$= \begin{cases} 0 & \text{keyword}(c_1) = \emptyset \\ & \vee\ \text{keyword}(c_2) = \emptyset \\ \frac{\lvert\text{keyword}(c_1)\cap\text{keyword}(c_2)\rvert}{\lvert\text{keyword}(c_1)\cup\text{keyword}(c_2)\rvert} * 2 - 1 & \text{otherwise} \end{cases}$
Self-citation	$= \begin{cases} 1 & name(c_1) = name(reference(c_1)) \\ 0 & \text{otherwise} \end{cases}$
Co-authorship	$= \begin{cases} 0.7 & \lvert\text{coauthors}(c_1) \cap \text{coauthors}(c_2)\rvert = 1 \\ 1 & \lvert\text{coauthors}(c_1) \cap \text{coauthors}(c_2)\rvert > 1 \\ 0 & \text{otherwise} \end{cases}$

Table 1. The list of the feature examples

   2. **Fair.** A feature can be equally important as positive and negative indicator, e.g. discrete time distance features.
- Structure
   1. **Flat structure.** A feature can focus on two documents connected to given contributor, e.g. year feature.
   2. **Graph structure.** A feature can check connections between larger quantity of documents, e.g. co-authorship, self-citation features.

One of advantages of the framework is a possibility of flexible feature addition and weight assignation, thanks to the usage of Spring Framework. The only restriction is that feature methods must be written in Java programming language.

2.5 Clustering process

Last part of the name disambiguation process is clusterization of contributors based on similarity matrix obtained in the previous step. As previously, one can use specially prepared clusterizer to reach desired result. In our case, we decide to use Single-linkage Hierarchical Agglomerative Clustering (described in [9]) with customization. This algorithm takes in each step two "active" contributors with top level score. If this score is below a given threshold (referred as T) the procedure is ended. Otherwise similarity level of contributions (σ) is recalculated in following manner:

$$\forall_{1<i<N}\forall_{i\neq a}\forall_{i\neq b}\sigma(c_a,c_i)=\sigma(c_b,c_i)=\begin{cases} -\infty & \sigma(c_a,c_i)<T \\ & \vee\,\sigma(c_b,c_i)<T \\ \sigma(c_a,c_i) & \sigma(c_a,c_i)>\sigma(c_b,c_i) \\ \sigma(c_b,c_i) & \sigma(c_a,c_i)\leqslant\sigma(c_b,c_i) \end{cases} \tag{1}$$

After recalculation, one of the contributors is deactivated. The process is repeated till either threshold is reached or there is only one active contributor left.

We choose simple-linkage clustering because it has desirable behaviour. If A is close to B, which is close to C, then merging A and B does not set the merged cluster further apart from C (which may be the case in Complete Link Clustering). Moreover, this type of clusterization provides $\mathcal{O}(N^2)$ time complexity, which is well acceptable. However, in other approaches other clusterizers can be applied. In Figure 6 one can see an example of how the customized Single-Linkage Clusterizer works.

2.6 Framework structure

Each part of the name disambiguation framework is written in Java programming language, with usage of Spring Framework and Sesame RDF Store. Thanks to usage of Spring Framework one can easily add their own elements of solution or replace them. Sesame RDF Query Language (SeRQL), in which database queries are written, is designed to support operations over graph structures.

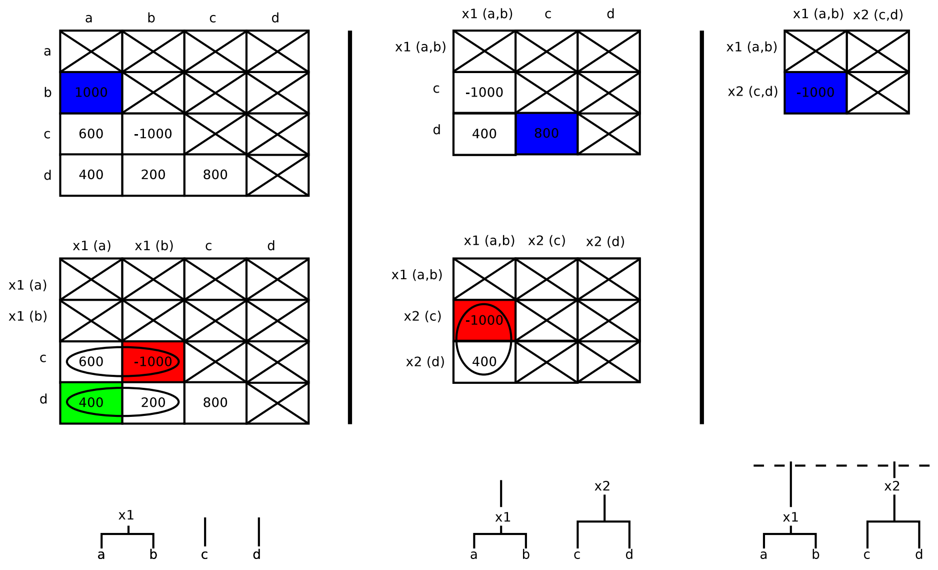


Fig. 6. Example of customized single-linkage clustering. First step is finding biggest affinity of two components in the affinity matrix. Then, if the value is negative clusterizing process is ended, whereas when the value is positive, contributions are merged into the same cluster. Affinities of the new cluster and other contributions or clusters are calculated based on recently taken components according to Equation 1.

2.7 Distributed computation

The presented framework is well-suited for distributed computation. Each shard is processed independently from all the others, and thus each can be processed on a different computing node.

The entire process can be implemented using Google's MapReduce [2]. In the "map" phase, for each document all its contributions are emitted, with the result of the hash function as the key. In the "reduce" phase, all the contributions with the same key, i.e. with the same hash function (a shard!) are processed together. Therefore, affinity assessment and clusterization are both performed during the "reduce" phase.

3 Future work

3.1 Training and evaluation

We have developed evaluation tools for measuring precision and accuracy of the framework output for a range of parameters. We are currently implementing a supervised learning algorithm (AdaBoost [6]) to automatically calibrate the weights associated with atomic affinities. However, we need an authority file to compare the results produced by our framework with the reality. Thanks to our co-operation with Zentralblatt MATH, we have recently obtained a high-quality authority file for the purpose of training and evaluating our framework.

3.2 Attributes from motifs

Another direction of framework extension is automatic feature generation, including weights. This approach is based on search for graph sequences with Apriori [1] algorithm. One of stages in Apriori algorithm is check of support and confidence coefficient, which can be treated as weight. A closer look at the support coefficient helps us determine how discriminative an indicator is. Does it generally give positive (negative) weights, or only to contributions of the same author?

4 Summary

We have presented a framework for author name disambiguation in a collection of documents. The framework has three “degrees of freedom” (cf. Figure 5): one may freely choose a hash function, feature functions together with their weights, and a clusterization function.

We were primarily interested in presenting the framework itself: establishing a vocabulary, defining components and their roles, defining workflows, proposing evaluation procedures. Presenting or evaluating a particular instance of the framework was not our goal. Nevertheless, we did hint at possible implementations of the individual components of the framework, and we outlined a plan of evaluation of an implementation that is currently under way at ICM.

The presented solution might be integrated into the European Digital Mathematics Library (EuDML) to handle contributions that are not present in the Zentralblatt MATH authority file.

5 Acknowledgements

EuDML project is partly financed by the European Union through its Competitiveness and Innovation Programme (Information and Communications Technologies Policy Support Programme, “Open access to scientific information”, Grant Agreement no. 250,503).

The authors would like to thank Zentralblatt MATH for providing their authority file for the purpose training and evaluation of our name disambiguation module, and the anonymous reviewers for their valuable comments.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB. vol. 1215, pp. 487–499. Citeseer (1994)

2. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM 51(1), 1–13 (2004)
3. Galvez, C., Moya-Anegón, F.: Approximate personal name-matching through finite-state graphs. Journal of the American Society for Information Science and Technology 58(13), 1960–1976 (Nov 2007)
4. Han, H., Giles, L., Zha, H., Li, C., Tsioutsiouliklis, K.: 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)
5. Han, H., Zha, H., Giles, C.L.: 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. pp. 334–343. ACM, New York, NY, USA (2005)
6. Hastie, T., Tibshirani, R., Friedman, J.: Elements of Statistical Learing. Springer (2009)
7. Kang, I., Na, S., Lee, S., Jung, H., Kim, P., Sung, W., Lee, J.: On co-authorship for author disambiguation. Information Processing & Management 45(1), 84–97 (Jan 2009)
8. Mann, G.S., Yarowsky, D.: Unsupervised personal name disambiguation. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -. pp. 33–40. Association for Computational Linguistics, Morristown, NJ, USA (2003)
9. Manning, C., D., Raghavan P., Schütze, H.: Introduction to Information Retrieval (2008)
10. Pavelec, D., Oliveira, L.S., Justino, E., Nobre Neto, F.D., Batista, L.V.: Compression and stylometry for author identification. 2009 International Joint Conference on Neural Networks pp. 2445–2450 (Jun 2009)
11. Pedersen, T., Kulkarni, A., Angheluta, R., Kozareva, Z., Solorio, T.: An unsupervised language independent method of name discrimination using second order co-occurrence features, pp. 208–222 (2006)
12. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: Extraction and mining of academic social networks. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 990–998. ACM (2008)
13. Torvik, V.I., Smalheiser, N.R.: Author name disambiguation in MEDLINE. ACM Transactions on Knowledge Discovery from Data 3(3), 1–29 (Jul 2009)