RefConcile – Automated Online Reconciliation of Bibliographic References

Guido Sautter[1], Klemens Böhm[1], David King[2]

[1] Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76128 Karlsruhe, Germany
[2] The Open University, Walton Hall, Milton Keynes MK7 6AA, UK
sautter@ipd.uka.de, boehm@ipd.uka.de, David.King@open.ac.uk

Abstract. Comprehensive bibliographies often rely on community contributions. In such settings, de-duplication is mandatory for the bibliography to be useful. Ideally, de-duplication works online, i.e., when adding new references, so the bibliography remains duplicate-free at all times. While de-duplication is well researched, generic approaches do not achieve the result quality required for automated reconciliation. To overcome this problem, we propose a new duplicate detection and reconciliation technique called RefConcile. Aiming specifically at bibliographic references, it uses dedicated blocking and matching techniques tailored to this type of data. Our evaluation based on a large real-world collection of bibliographic references shows that RefConcile scales well, and that it detects and reconciles duplicates highly accurately.

Keywords: Bibliographic References, Data Cleansing, Record Linkage

1 Introduction

Compiling bibliographies covering entire scientific domains is challenging. An additional challenge for bibliography platforms that rely on a user community for the addition of new data is a tradeoff between comprehensiveness and data quality. This is because, to maintain data quality, and to keep the rate of duplicates low in particular, many platforms include interactive duplicate detection procedures in the data upload process, i.e., no full automation. An example of such a platform is ZooBank [24]. Pursuing comprehensiveness, other platforms strip all data quality assurance measures from the upload process to simplify contributing, for instance the Catalogue of Life [4] and RefBank [27]. This incurs a large number of duplicate records; be it variant representations of the same record or records that differ merely by errors like misspellings.

A bibliography maintained by a community has to meet these conflicting requirements: (a) uploading references should incur little intellectual effort, (b) newly uploaded references should become available right away, and (c) there should be few erroneous or duplicate references. Thus, the bibliography platform has to check immediately if new references are duplicates of ones already present and reconcile them if necessary. Furthermore, de-duplication should be lightweight, i.e., consume as few resources as possible. This work proposes a de-duplication method tailored to bibliographic references to meet this combination of requirements.

Considerable research effort has gone into the clean-up of data sets: **Data de-duplication** [10], also known as **Entity Matching** [20], and **Record Linkage** [2] deal with the detection of duplicate records in large data sets, while **Data Cleansing** [13] aims at reconciling detected duplicates. Conceptually, entity matching compares all records in a data set pair-wise to find out which ones are duplicates of others. Its complexity thus is $O(n^2)$. To reduce computational cost, a blocking step groups the records into blocks prior to the matching. Blocks comprise records that might be duplicates of one another, and the matching step only compares records in the same block. There are three categories of blocking methods: disjoint, overlapping, and fuzzy. Disjoint methods assign each record to exactly one block, overlapping ones yield sorted blocks that overlap 'at the edges', and fuzzy methods assign records to several blocks. Draisbach [9] surveys different blocking strategies; their complexities vary from O(n) to O(n log n). Blocking is more effective the smaller the blocks are because this reduces the number of comparisons in the matching step. To meet above requirements, this paper devises an effective blocking method for bibliographic references; its complexity is O(n).

Entity matching in general has received considerable attention recently; cf. Köpcke [20]. The frameworks surveyed there also report results for bibliography data. However, the largest bibliographic data set evaluated consists of roughly 67,000 references to less than 6,000 works, with the best reported accuracy at only 89%.

Duplicate detection and reconciliation immediately after the arrival of new references further requires a blocking method that works incrementally. Re-computing the entire blocking every time would incur prohibitive effort. Reducing the workload by resorting to periodic cleanup leaves un-reconciled records in the data set most of the time and is not user-friendly.

To facilitate automated de-duplication in community contributed bibliographies, this paper proposes **RefConcile**, a technique tailored to both contemporary and legacy bibliographic references. The rationale is that a technique designed for this type of data can provide the accuracy and performance required, unlike generic approaches. To build RefConcile, we first thoroughly investigate bibliographic references and their peculiarities: For instance, identifying the last name of an author is far from trivial depending on the representation of author names; this may incur ambiguities. Based on our findings, we then propose a blocking method specifically tailored to bibliography data and the requirements outlined above. Regarding the six blocking methods surveyed in [9], our proposal is between key-based blocking and canopy clustering. This is because we use very fuzzy blocking keys for indexing to efficiently compute an ad hoc canopy around a newly added reference. Third, we propose a vote based reconciliation technique that works attribute by attribute.

Our evaluation on RefBank data demonstrates that RefConcile outperforms generic entity matching approaches with bibliographic data: The data set consists of about 150,000 bibliographic references, with about 20% duplicates. RefConcile detects such references with 95.6% f-measure at 99.7% precision. It takes less than one second per newly added reference.

2 Bibliographic References

This section describes bibliographic references and their individual attributes in detail, with a focus on how attribute values may vary in different representations of the same reference. Note that RefConcile works with atomized references. Reference parsing is a related, but distinct problem, and there are separate algorithms for solving it [5, 11, 26]. Thus, RefConcile handles different reference styles only with regard to their specific representation of individual attributes, especially author names (see below); ordering and intermediate punctuation of attributes is not an issue. Further, this section discusses the most commonly found errors in bibliographic references. Finally, this section introduces similarity measures commonly used in duplicate detection and related search tasks.

2.1 Attributes of References

To better understand the peculiarities of bibliographic references and their attributes, we have investigated a sample of several thousand references from the RefBank data set. This section describes our findings.

Author names are challenging for duplicate detection because the representation of a person name can vary considerably. Several problem patterns have been identified [26], namely double last names, leading and middle initials, noble titles, affixes indicating generation, and infixes like *van*. Reference style also plays an important role, with first and last name occurring in multiple orders and with various punctuation schemes, even within the same reference. Further, infixes and affixes may or may not be given, and if they are, their position and punctuation can vary considerably. The (fictional) author name *Alexis Ulysses van Thor*, for instance, can be given as anything from *van Thor*, *A.U. van* and *Thor AU* to *A. Ulysses van Thor*. Additional variation arises from names transliterated to Latin script, e.g., from Cyrillic. The last names *Iakowlew*, *Jakowlev*, and *Yakovlew*, for instance, may all refer to the same person. Further, sometimes only the first author of a referenced work is given, with *et al.* standing in for all others; we also found instances with the authors missing altogether.

The **title** of the referenced work does not vary as much. Safe for very few errors, the words are always in the same order, if with varying capitalization. However, long titles may sometimes be truncated, and individual words can be misspelled.

The **name of the journal** an article was published in is challenging for duplicate detection. Long journal names in particular tend to be reworked heavily: stop words may be omitted, and other words abbreviated to different degrees, down to their initial letter in extreme cases. The journal *Transactions of the Royal Entomological Society of London*, for instance, may be shortened to *T. Roy. Ent. Soc. Lond.* The order of the words remains the same, however. The use and degree of abbreviations is often motivated by spatial constraints in the bibliography of a publication. ISSN provides a list of suggested abbreviations [15]. However, it does not cover all abbreviations that occur in practice. The variation in abbreviated names has also spawned many web sites, such as those offered by CalTech [3] and the University of Leeds [29], which catalogue the variations to help with clarification. However, both web sites only list one suggested abbreviation per journal, with further ones hidden behind a login in [3].

The **publisher** of a referenced book or book chapter can also take a multitude of forms, especially when used in conjunction with the location(s) of the publisher, as the order of these two elements can vary. In addition, the representation of the name itself can vary considerably: The publisher name *Pensoft Publishers, Sofia, Moscow*, for instance, can be given as *Moscow: Pensoft Publishers*, or even simply as *Pensoft*.

Numeric attributes, like the **year of publication**, the **volume or issue number** in references to journal articles, and the **pagination**, exhibit little variation. However, the latter two may be lacking in some references. Thus, duplicate detection has to cope with their presence or absence.

2.2 Common Errors

Many duplicate references originate from slight variations in spelling. One reason for such differences is the varying transliteration schemes used for author names, as explained above. A more common reason for differences, however, are typographical errors. Davies [6] found that such errors are not equally frequent throughout a bibliographic reference: (1) misspellings are more likely to occur in the middle or at the end of a word rather than at the beginning; in particular, the first letter of a word is nearly always correct; (2) misspellings are most likely to occur in author names, somewhat less likely in titles, and least likely in journal and publisher names. We exploit these findings in the design of our blocking technique.

2.3 String Similarity Measures

This section describes measures for assessing the similarity (or difference) between two strings, with **similarity = 1 – distance**.

The **Levenshtein Distance** [22] (also referred to as the edit distance) between two strings is the number of single-character edits needed to change one word into the other using *insertion*, *deletion*, and *substitution*. If the two strings are equal, the Levenshtein distance is 0. The cost to compute it between two strings of lengths m and n is O(m × n). This makes it impractical to compute between full references, but it is well applicable to individual terms within a reference.

The **Jaro-Winkler Distance** [17, 30] measures similarity of strings: 0 indicates no similarity, and 1 is an exact match. The Jaro-Winkler distance is derived from earlier work [16] addressing the problem of person name variations, such as McDonald and MacDonald, in census taking. As such it is well suited to matching author names, as well as short strings in general.

Term Frequency - Inverse Document Frequency (**TF-IDF** for short) [18] indicates the significance of a word within an individual document that is part of a collection of documents. Its normal use is as a weighting factor in information retrieval and text mining, indicating how significant a given word is when used to select individual documents from the collection. Likewise, it is suited to weighting words in titles.

N-grams are the basis of several string similarity measures that are widely used in information retrieval, most commonly with n=3. Generating the n-gram representations of two strings is of linear effort, and comparison works in constant time (dependent on n^3). However, it is still relatively expensive in comparison to computing the Levenshtein distance when strings are short. In addition, n-gram based similarity measures abstract from the order of the n-grams, which can be detrimental to accuracy.

3 Related Work

Data de-duplication in general is closely related to the problems of Data Cleansing [13], Record Linkage [2], Entity Matching [20, 21], and clustering [7]. This section reviews respective techniques and assesses their applicability to the specific problem of keeping a community-contributed collection of bibliographic references clean.

The term **Data Cleansing** [13] refers to identification and cleanup of duplicate records in a data set. The closely related terms **Record Linkage** [2] and **Entity Matching** [20, 21] refer to the same activity when merging several data sets. Commonly, this works in two steps: blocking and matching. The latter often is computationally expensive especially if the former is not effective. Subsequent reconciliation may require input from human experts in the general case.

Köpcke et al. [20, 21] provide extensive surveys of entity matching techniques and respective frameworks. However, all approaches discussed there are generic, i.e., designed for entity matching in general. Exploiting domain-specific knowledge is orthogonal to the issues studied there. Generic approaches must learn all features in the text being analyzed. Yet phenomena like abbreviations and changes to the order and separating punctuation of first and last name are hard to infer. We speculate that this pursuit of generic entity matching is why the result quality of these frameworks on bibliographic data drops as data sets grow larger. According to [20], it decreases from 99% f-measure with 5,000 records to 89% with 67,000 records.

Draisbach [9] provides an overview of blocking techniques: **Sorted neighborhoods** do not appear to be well-suited to our setting: They are sensitive to changes to the order of the parts of author names and to abbreviations, let alone to omitted leading stop words. In addition, the size of the neighborhood would have to be very large for common author names like *Smith*. **Bigram indexing** is ineffective in our setting, as long strings like titles just yield too many combinations, and it is likely inefficient for the same reason. **Canopy clustering** in its generic form is computationally too expensive for our setting: If the fraction of duplicates in the dataset is low, its complexity approximates $O(n^2)$ with tight thresholds; if thresholds are relaxed, canopies grow impractically large. However, the incremental blocking method we propose can be seen as the efficient index-based computation of an ad hoc canopy around a newly added reference. **Blocking keys** are easy to use incrementally and, if constructed properly, are both efficient and effective. However, their construction is not trivial in our setting. This is because the construction has to cope with abbreviations and with different orderings of author name parts in situations where the last name may be ambiguous. On the other hand, the index entries we propose to use to efficiently compute canopies can be seen as fuzzy blocking keys.

Blocking is closely related to clustering, and incremental blocking is related to incremental clustering techniques. Much work has gone into clustering stream data [8, 12, 23, 31, 32]. However, all these approaches use rather rigorous pruning techniques to reduce memory consumption. This is not applicable in our setting, as blocking needs to retain all records, not only a few representatives ones.

4 Reference Reconciliation – the RefConcile Algorithm

This section describes the RefConcile technique in detail, in particular its custom tailored blocking mechanism, its matching technique, and how it reconciles references found to be duplicates.

4.1 Blocking

The purpose of blocking is to reduce the number of pair-wise reference comparisons in the matching step as far as possible, in our setting the number of comparisons to a newly added reference. Blocking in general is somewhat similar to hashing: it distributes records across multiple bins (called blocks), and subsequent matching works inside individual bins. With fuzzy blocking methods, records can also be assigned to multiple bins instead of one. To be effective, there should be many bins with few records in each. On the other hand, blocking should not assign any actual duplicates to different bins, as matching then cannot recognize them. Furthermore, blocking ideally has to be independent of the number of references already in the data set.

To meet these constraints, RefConcile uses fuzzy blocking keys [9] composed from all attributes of the bibliographic references. The keys are fuzzy in a sense that their individual parts consist of sets or ranges rather than single values. Two keys are considered equal if all the sets have a non-empty intersection and all the ranges overlap. If an attribute is not present in a reference, the corresponding part of the blocking key is a wildcard set or range that has a non-empty intersection with each non-wildcard set, or a wildcard range that overlaps with every non-wildcard range, respectively.

The rationale behind this approach can be demonstrated with an example: consider the following two representations of the fictional author name from Section 2.1: *Thor AU* and *Ulysses THOR*. However, it is possible that the first representation actually refers to a different author, someone with the first name *Thor* and the last name *AU*. The part of the blocking key that represents the author name has to reflect these possible alternatives. We use sets in alphabetical order as an easy form of normalization, thereby abstracting different name part orders. Hence, for *Thor AU*, the key would be be *{A, T}*, and for *Ulysses THOR*, *{T, U}*. RefConcile treats the individual attributes as follows:

- **Author names:** First, discard all letter blocks that do not contain any capital letters and then all fully capitalized ones that do not contain a vowel, as they are probably blocks of initials. The remaining letter blocks are probably the author's last name. Construct a set containing the first capital letter from each block, normalized to alphabetical order, conflating the letters I, J, and Y, as well as V and W, to allow for differences in transliteration. Because all but the first author can be substituted with "et al.", RefConcile uses only the first author. Note that, while the order of the individual parts of a name can vary, author names as a whole exhibit little variation, especially the first few authors [6].

- **Title:** Discard all stop words, as well as single letters. Construct a set containing the first letter of each remaining word. In addition, create a range for the number of remaining words, plus and minus one third.
- **Year of publication:** Create a range covering the given year plus and minus one.
- **Journal name:** Discard all stop words. Construct a set containing the first capital letter from each remaining word.
- **Publisher name / location:** Discard all stop words. Construct a set containing the first capital letter from each remaining word.
- **Volume / issue numbers:** Create a range covering the given volume or issue number plus and minus one.
- **Pagination:** Use the given page range.

In addition to the attributes, RefConcile includes the **type of work** a reference refers to in the blocking key, distinguishing the following reference types: *journal article*, *journal volume*, *book chapter*, *book*, *proceedings paper*, *proceedings volume*, and *online resource*. The type of work is determined based on which attributes are present.

To optimize runtime, we propose persisting the blocking keys in a database. Modern relational database engines keep the computational effort for retrieving a block of references largely independent of the number of references in the data set.

4.2 Matching

RefConcile's matching step uses a decision tree to classify reference pairs as duplicates or non-duplicates, plus several hand crafted kill rules. The latter are motivated by phenomena we observed during our investigation of the RefBank data set. For instance, they prevent matching references to individual volumes of multi-part works. Such references tend to differ only in the part number, which often is included in the title. Conceptually, the kill rules are nodes we manually added to the otherwise learned decision tree. To train the decision tree, we labeled a sampled set of training examples, computed all the similarity measures described in Section 2.1 for each pair of references, and then used Weka [14] for the learning. In live deployment, however, it would be inefficient to compute all the pair-wise similarity measures for all references in a block, or even only for each pair comprising the newly added reference that triggers the matching. We therefore hard-coded the learned decision tree and implemented it to compute the similarity measures on the fly, i.e., right before their first use. If a decision is reached far up the tree, this saves computation of similarity values that would have been used only further down the tree. Furthermore, to avoid overfitting to the training set, we slightly relaxed the decision thresholds.

4.3 Duplicate Reconciliation

Once a group of duplicate bibliographic references has been recognized, RefConcile links them together and selects a common representative for the group.

If there are three or more duplicates in the group, this works by selecting the attribute values that are most frequent across the duplicate group, individually for each attribute. This yields a reconciled value for each attribute. The rationale is that the common representative reference should have the most agreed-upon value for each attribute, and that typographical errors are sufficiently random not to become the majority value. If there is a reference in the group all of whose attributes have the reconciled values, it becomes the representative. If there is no such reference, RefConcile generates it by inserting the reconciled values into a template, and then adds it to the data set. While this adds a further duplicate record, it is the only way of providing a representative reference. Example 2 illustrates this.

Suppose RefConcile found the following threesome of duplicates (errors bold):
Thor, AU, Cond, SE (2012) Bibliographic**al** duplicates. Journal of TPDL **9**: 8-1**6**
Thor, AU, Co**ri**d, SE. Bibliographic duplicates. Journal of T**B**DL 8: 8-15, 201**3**
Tho**p**, AU, Cond, SE. Bibliographic duplicates. Journal of TPDL 8 (2012): **9**-15

The element wise majority vote yields the correct reference:
Thor, AU, Cond, SE. Bibliographic duplicates. Journal of TPDL 8 (2012): 8-15

Example 2. Reconciling three references that all contain minor errors

For groups of two duplicates, majority voting is not applicable. As a heuristic, RefConcile then selects the more recently added reference as the representative. The rationale is that the more recent duplicate might well be a correction of the other one.

5 Evaluation

This section reports on a thorough evaluation of the RefConcile algorithm based on the RefBank data set, a real-world community contributed collection of bibliographic references. We can only address the most interesting aspects here for lack of space.

The RefBank Data Set. RefBank is an open, multi-node platform collecting bibliographic references from various sources, including community contribution. Apart from storing the same reference string (in a character-by-character sense) only once, RefBank does not implement any duplicate detection or reconciliation measures. The data set grows constantly, as new references are added. At the time of our experiments, the data set comprised over 160,000 individual references. An atomized version was available for about 150,000 of them, and they are our test set.

Experimental Setup. To simulate a continuously growing reference data set, we start our experiments with an empty data set and then add the references one by one. Each addition of a reference prompts RefConcile to search for duplicates, and to reconcile any ones found. The experiments were conducted on a 4 x 2.0MHz 64-bit machine with 8GB of main memory running Ubuntu Linux 2.6.22, PostgreSQL 9.1, and a JVM 1.6 from Sun/Oracle.

Experimental Results. Table 1 displays our results. The high precision of 99.7% indicates that RefConcile rarely ever wrongfully labels a pair of references a duplicate. The recall of 91.9% means that RefConcile correctly finds 9 out of 10 duplicate relations. This corresponds to 95.6% in f-measure.

Even with around 150.000 references in the database, the fuzzy blocking returns only 9.14 candidate duplicates per reference on average. This means that the matching has fewer than 10 possible duplicates to deal with, underlining the scalability of

RefConcile. Implemented on top of a relational database, the incremental blocking takes only 5.9 microseconds for each pair wise reference comparison. Matching takes an average 6.4 milliseconds for each pair of references.

Table 1. Evaluation results

Precision / Recall / F-measure	99.7% / 91.9% / **95.6%**
Avg. Number of Candidates	9.14
Avg. Time for Candidate Retrieval	5.9 µs / reference, 47.8 ms / candidate found
Avg. Time for Candidate Assessment	6.4 ms / candidate, 216.8 ms / duplicate found

To assess the quality of automated references reconciliation, we manually inspected the generated cluster representatives, We found that only some 5-10% contained errors. This is in line with our expectations.

6 Conclusions

Relying on community contribution is popular to compile comprehensive bibliographies. Experience with ZooBank [24] shows that embedding duplicate detection procedures in the data input process that require intellectual input alienates users. To foster contribution, other platforms completely waive respective measures. This in turn affects data quality.

To ensure data quality automatically as contributors add new references, we have proposed a duplicate detection and reconciliation technique named RefConcile. Our evaluation on a real-world data set shows that it works well in practice: It finds and reconciles duplicate references highly accurately, and it scales very well.

Acknowledgements. This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under grant agreement no 261532 (ViBRANT – Virtual Biodiversity Research and Access Network for Taxonomy). The RefConcile algorithm is in productive use as part of RefBank[1], the platform of ViBRANT to build its Bibliography of Life.

7 References

1. Beall, J. Measuring duplicate metadata records in library databases. Library Hi Tech News 27: 10 – 12. 20120. doi: 10.1108/07419051011110595
2. Blakely, T., Salmond, C. Probabilistic record linkage and a method to calculate the positive predictive value. Int. J. Epidem. 31: 1246–1252. 2002. doi:10.1093/ije/31.6.1246
3. CalTech: http://library.3.edu/reference/abbreviations/
4. Catalogue of Life: http://www.catalogueoflife.org/
5. Cortez, E., da Silva, A. S., et al. Fluxcim: flexible unsupervised extraction of citation metadata. In Proceedings of JCDL 2007. Vancouver, BC, Canada, 2007.

[1] RefBank: http://plazi.cs.umb.edu/RefBank/

6. Davies, K. Reference accuracy in library and information science journals. Aslib Proceedings 64 (4): 373-387. 2012.
7. Defays, D. An efficient algorithm for a complete link method. The Computer Journal (British Computer Society) 20 (4): 364–366. 1977.
8. Domingos, P., Hulten, G. A general method for scaling up machine learning algorithms and its application to clustering. In: Proc. Intern. Workshop on Machine Learning 2001.
9. Draisbach, U., Naumann, F. A comparison and generalization of blocking and windowing algorithms for duplicate detection. Proc. QDB Workshop at VLDB, 2009.
10. Geer, D. Reducing the storage burden via data deduplication. Computer 41: 15-17. 2008.
11. Giles, C. L., Councill, I., Kan, M.-Y. ParsCit: an open-source CRF reference string parsing package. In Proceedings of LREC 2008, Marrakech, Morocco, 2008.
12. Guha, S., Mishra, N., Motwani, R., & O'Callaghan, L. (2000). Clustering data streams. In: Proc. Symp. on Foundations of Computer Science 2000: 359-366, 2000.
13. Han, J., Kamber, M. Data Mining: Concepts and Techniques, Morgan Kaufmann, 2001. ISBN 1-55860-489-8.
14. Holmes, G., Donkin, A., Witten, I.H. Weka: A machine learning workbench. In Proc. AUS NZ Conf. Intel. Inf. Syst., Brisbane, Australia, 1994.
15. ISSN: http://www.issn.org/2-22661-LTWA-online.php
16. Jaro, M. A. Advances in record linkage methodology as applied to the 1985 census of Tampa Florida. Journal of the American Statistical Society 84 (406): 414–20. 1989.
17. Jaro, M. A. Probabilistic linkage of large public health data file. Statistics in Medicine 14 (5-7): 491–498. 1995. doi:10.1002/sim.4780140510
18. Jones, K. S. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28 (1): 11–21. 1972. doi: 10.1108/eb026526
19. Köpcke, H., Rahm, E. Training selection for tuning entity matching. Proc. Workshop on Quality in Databases and Management of Uncertain Data: 3-12, 2008.
20. Köpcke, H., Rahm, E., Frameworks for entity matching: A comparison, Data & Knowledge Engineering, 69 (2): 197-210. 2010. doi: 10.1016/j.datak.2009.10.003.
21. Köpcke, H., Thor, A., Rahm, E. Evaluation of entity resolution approaches on real-world match problems. Proceedings of VLDB Endowment 3: 484-493. 2010.
22. Levenshtein, V.I. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Proceedings 10: 707–710. 1966.
23. O'Callaghan, L., et al. Streaming-data algorithms for high-quality clustering. In: Proc. Inntern. Conf. on Data Engineering, 2002.
24. Polaszek, A. A universal register for animal names. Nature 437 (477): 477. 2005. doi: 10.1038/437477a
25. PubMed Central: http://www.ncbi.nlm.nih.gov/pmc/
26. Sautter, G., Böhm, K., Improved Bibliographic Reference Parsing Based on Repeated Patterns. In Proc. TPDL 2012, Paphos, Cyprus, 2012.
27. Sautter, G., King, D., Morse, D. Towards a universal bibliography – the RefBank approach. In Proc. TDWG 2012, Beijing, China, 2012.
28. Thor, A., Rahm, E. MOMA – a mapping-based object matching system. Proc. 3rd Bien. Conf. Innov. Data Syst. Research (CIDR '07): 247-258, 2007.
29. University of Leeds: http://www.efm.leeds.ac.uk/~mark/ISIabbr/
30. Winkler, W. E. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. Proc. Sect. Surv. Res. Methods (ASA): 354–359. 1990.
31. Cao, F., Ester, M., Qian, W., & Zhou, A. Density-based clustering over an evolving data stream with noise. Proc. Intern. Conf. Data Mining: 328-339. 2006.
32. Wang, H., Yu, Y., Wang, Q., & Wan, Y. A density-based clustering structure mining algorithm for data streams. Proc. Intern. Workshop Big Data: 69-76. 2012.