Selecting the right search term in query-based systems for deduplication

Die Wahl des Suchbegriffs in anfragebasierten Systemen zur Erkennung bibliographischer Dubletten
Sélection de mot de recherche dans les systèmes basés sur des requêtes en vue de détecter les doublons bibliographiques

Harald Jele

Essentially three approaches could be identified when choosing a proper search term to detect bibliographic duplicates. Stop words are excluded in all of them, then (1) just the first term of an entry will be selected or (2) that term is selected, which produces the smallest number of hits or finally (3) that term will be used, which has a certain number of hits below a defined threshold.
These three procedures are compared with each other here. The results derive from series of measurements done with bibliographic data from the Austrian Central Catalog.

Bei der Wahl eines günstigen Suchbegriffs zur Erkennung bibliographischer Dubletten sind im Wesentlichen drei Ansätze erkennbar. Stoppwörter werden in allen dreien ausgeschlossen, anschließend wird (1) der erste Begriff eines Eintrags gewählt, der aufgefunden wird oder es wird (2) jener gewählt, der die kleinste Treffermenge hervorruft oder letztlich (3) findet der Begriff Verwendung, dessen Treffermenge unter einem definierten Schwellwert liegt.
Diese drei Vorgehensweisen werden hier miteinander verglichen. Die Ergebnisse stammen aus Messreihen, die mit Titeldaten aus dem österreichischen Bibliothekenverbund durchgeführt wurden.

On a pu identifier essentiellement 3 approches dans la procédure de selection d'un terme susceptible de détecter les doublons bibliographiques. Ces trois approches excluent les mots vides (stopwords), ensuite (1) on selectionne le premier terme d'une entrée ou bien (2) on choisit celui pour lequel on obtient le plus petit nombre de résultats ou bien finalement (3) on prend le terme pour lequel le nombre de résultats se situe sous un seuil défini. Ces trois procédures sont ici comparés les unes aux autres.
Les résultats sont issus de séries de mesures effectuées sur une base de données bibliographiques du catalogue de la Bibliothèque nationale autrichienne.

1 Preface

If one tries to classify this article based on current literature, he or she may quickly be tempted to dismiss the discussed methods as not contemporary any more. With this brief preface, which provides some references on relevant literature, it will be shown, that such a fast judgment is certainly not valid.

When detecting bibliographic duplicates it is intented in particular not to compare too many entries because the final calculation of large numbers of duplicates is rather time consuming. Much more effective, of course, is any approach that keeps the amount of titles to be tested very small. On the other hand it has to be concerned, however, that a smaller amount of selected titles increases the probability of not recognizing some duplicates. Thus, in any case of restricting the somehow limited "search space", within which the verification of duplicates take place, efficiency and reliability with their necessary requirements are diametrically opposed.

The approach to proceed on the basis of keywords is one that meets the reality of working with bibliographic databases. This reality is quite often characterized by the fact that bibliographic entries (from very different databases), which are locally not available, have to be integrated into a local system. Thereby, the remote databases are addressed via interfaces (like Z39.50) and APIs (= Application Programming Interfaces like REST (= Representational State Transfer) or SOAP (= Simple Object Access Protocol)).
Therefore, the selected approach must be one, that can be used in such environments in a functional and effective way (see Schneider 1999). Such is the method of automatically loading the titles to be checked by means of a proper selected keyword.

As an alternative to a keyword-based approach current literature presents mainly methods such as SNM (= Sorted Neighborhood Method) (Hernández & Stolfo 1995 and Yan u. a. 2007) or the "Blocking Method" (Draisbach & Naumann 2009). Thereby small strings are extracted from the data (e. g. substrings from the author- and title-information), joined together to a new string and sorted alphanumerically. The so formed and sorted keys ("Sorting keys") are used in a further way to search for similar entries in their neighborhood using so-called "search windows" with a certain size (number of entries to be considered). The algorithms applied for searching in a certain width are called "Windowing" (Hernández & Stolfo 1995 as well as Hernández & Stolfo 1998) and "Blocking" (Ananthakrishna et al. 2002, Baxter et al. 2003 and Bilenko et al. 2006).

It is only possible to access the sorting keys if they are available in the remote database, or if they are created locally from the total amount of bibliographic entries, which is rarely the case in practice. Furthermore, external production of sorting keys is impractical and inefficient because the requested databases usually do not contain static information but rather have a steadily increasing number of titles that must be considered. The method to create and keep the whole sorting keys for the duration of the processes in the main memory, which is proposed in Draisbach and Naumann (cf. 2009, p. 2), disregards the fact that even an average bibliographic database keeps an amount of title entries which is almost too big for such an approach. The keys must therefore be stored in the database files. These methods, although they are judged as state-of-the-art, can't be considered for a practical application nowadays, when data is accessed remotely. In case of a local merge process in batch mode, however, they might be quite relevant.

Other approaches which are frequently discussed in current literature are based on hashing methods. For text-based applications particularly implementations of indexing using "Locality-sensitive hashing" (LSH) seem to be promising, although there are no relevant results published for indexing bibliographic data. The basis of a LSH-implementation is typically formed by a combination (addition) of simple kept hashes. Each collision of two or more values is seen as a similarity between the affected records. Under the most favorable conditions hash values are formed in a way that there are no singular values, if it can be assumed that the total amount includes duplicates (Paulevéa et al. 2010 and Stein & Potthast 2006). The relevance of such an approach has to be seen under the same conditions as the already mentioned approach using sorting keys: In practical use, such an approach can always be successful, if it is implemented in a local (and not in a remote) system. Otherwise, in each of the remote bibliographic databases these hash values have to be formed and kept accessible.

2 Introduction

In the field of bibliographic title deduplication query-based systems are characterized by the specific circumstance that a second query is automatically produced due to the result list of a first one. The purpose which is intended in the second query is to discover all duplicate title information of a database to those titles that are included in the result list of the first query.
The actual (first) query, respectively, the resulting title set, could be done on the one hand by a particular user of an online system. On the other hand, these can be made when merging different databases by a sequential processing of bibliographic data in batch mode.
When the result set is achieved by a user, the goal of such an approach may be a duplicate-free title display as well as an enhanced display of additional holding information which is not included in the first result set, but derives from the second. This might be essential for further user actions in a local library system like ordering titles and for interlibrary loan.
Another application of such an approach can be the cataloging of works in a library system. In this case, possible duplicates will be reported to the cataloger while retrieving or saving a title record. We won't follow this use here because of it's specific implications. Schneider (1999) can be seen as an example of a practical application for deduplication in the process of cataloging. He developed the system "ZACK" which is still quite widely used in Germany.

In Figure 1, the further process, which will be discussed here, is illustrated schematically: A user produces due to its query *q1* a result list {`m1`} which should be displayed free of duplicates. The list of titles {`m1`} can therefore be either deduplicated by an adequate detection method or

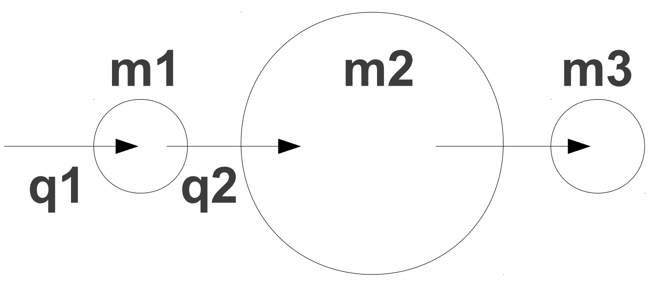

Figure 1: Scheme of the query system

can be furthermore expanded by those copies that are stored in the database with the amount of {m2} but were not recognized by query *q1*. Thereby, any record of the result list {m1} is checked by *q2* against {m2} without any user intervention and a set {m3} is formed, which is enriched with the copies (i. e. the items and holdings information) of the duplicate titles from {m2}. Such a scenario can be implemented – as already mentioned – in an online catalog which offers local ordering functions to registered users or includes non mediated (fully automated) interlibrary loan orders.

In most cases, such an approach for the automated comparison of several bibliographic databases, is implemented by a batch process. Doing so, the bibliographic information of the smaller data amount {m1} will be processed sequentially and compared with the titles from the larger amount {m2}. Every single title from {m1} must be analyzed and, subsequently, those terms, that can be used for creating appropriate queries *q2* against the whole title information {m2} must be extracted. The result list {m3} of query *q2* is used instead of the former {m2} for the successional check for duplicates.

In both approaches (in the online and in batch mode), the major challenge is the selection of an appropriate search term for the second query *q2*.

3 Usual procedures

To form the request, not all entries of a structured bibliographic data record are used. Usually, only the categories (tags) for the author information, the corporate names and the categories that contain the whole title information will be considered. A record without any information in at least one of these tags is ignored for the next steps. In case that content is available in one of these categories, they will be checked for their further practicability. In the first place, terms or entries, that are marked as stop words, are not usable for a query.

Three approaches can be found in the relevant literature as well as in the accessible routines for selecting the right query term:

1. the first term, which is not a stop word is used as the query term,
2. that term, which is not a stop word and has the smallest number of hits, is used as the query term,
3. the first term, which is not a stop word and has a certain number of hits below a defined threshold, is used as the query term.

These three approaches to determine a proper request term will be compared, and their advantages and disadvantages will be identified. The evaluation of the aspects, which has to be considered, mainly takes place by measuring the substantial values, that have been raised on concrete and real, existing bibliographic records, as they can be found in an average-sized Austrian university library. With this fact, namely, that no "artificial" or special selected data is used for measuring, the relevance of the results should be illustrated in a better way than more theoretical values could do.

All results refer to bibliographic data of monographs (books). Basically these are also applicable to bibliographic data of articles within journals or anthologies.

4 Testing method

The entire bibliographic data of the University Library of Klagenfurt, which is stored as digital records in `MAB 2` format and can be accessed online, are approximately `700.000` title records. They were completely exported from the catalog.
Subsequently, those titles that represent not monographic works (this includes those, that have been published within a series or several volumes) were merged together in the way, that the information on author, corporate names and title were taken from the higher-level records to the lower ones. This has been performed only in the case, that no such information were available in the concerned categories of the "child records" (from the lower level). The merging procedure should ensure, that a minimum of bibliographic records are concerned by the fact, that they must be ignored due to a lack of entries.
Afterwords, the entries of the parent records were removed from {m1} and {m2} in the case that the content of their categories could be inserted into the child record.
The number of categories, that were used for the further check on duplicates, are those mentioned in Jele (2009): to form the person's name, the contents from the `MAB 2` tags `100`, `100b`, `100c`, `100f` and `359` were taken; for the corporate names the tags `200`, `200b` and `200c`; for the edition tag `403`; for the place of publication `410` and `410a`; for the publisher the tags `412` and `412a`; for the year of publication `425a`, `425b` and `425c`; for the pagi-

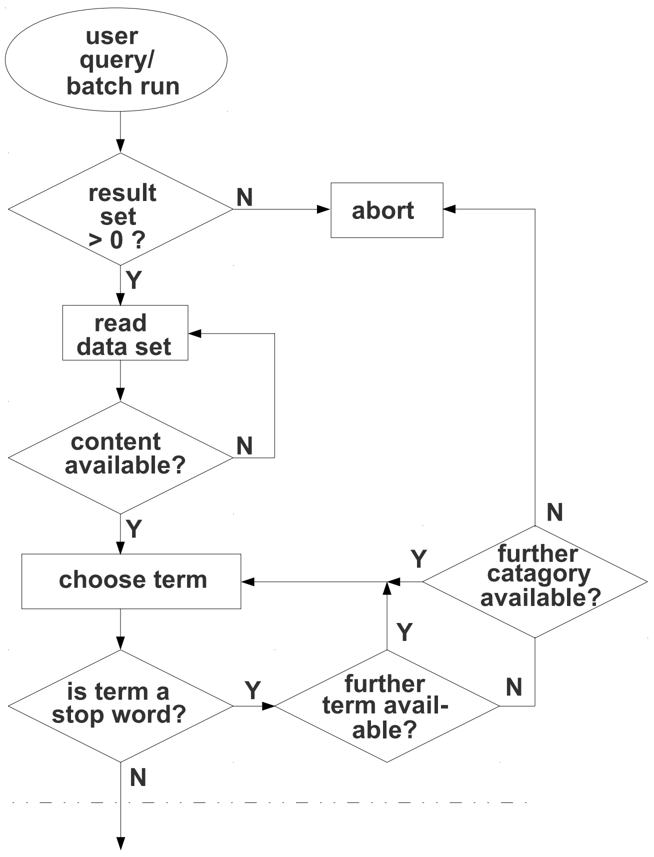

Figure 2: The logic of term discovery in a flowchart. The horizontal line indicates the transition to the following routines.

nation `433`, `433a` and `433b` and for the ISBN the tags `540a`, `540b` and `540`.

The result of the merged title records was loaded into a `MySQL` database on a simple standard desktop computer. `Ubuntu Linux` in version `11.04` was the used operating system. All database queries and deduplication methods have been implemented in the programming language `Perl`. The analysis of the results was performed using the free statistical software `R` and the added package `ggplot2` (see also Wickham (2009)). For the handling and visualization of large datasets, the recommendations in Unwin, Theus & Hofmann (2006) were considered.

To keep in mind is the fact, that all the figures given here should not be interpreted as absolute values, but only in their relation. The features and the capability of commercial desktop computers change substantially every few months. Therefore it can be assumed, that a short time after the publication of this paper, already significantly better results can be achieved with the current hardware (that means essentially a quite shorter computation time). Nevertheless, the relationship between the printed results should remain valid, even under changing circumstances.

The following measurements were taken with all three approaches:

- the duration time, required for the determination of the potential duplicates,
- the duration time, that elapses until the appropriate query term is found,
- the amount of result data records, caused by the needed requests,
- the number of categories (tags) with content, which were provided in each title record,
- and the number of categories, which had to be analyzed to identify the appropriate query term.

The generated amount of bibliographic records has been tested against itself. This means, corresponding to the above illustration 1, that {`m1`} is equal to {`m2`}.

Figure 2 shows the preliminary analysis of a bibliographic data record by a flow chart:
After having verified, that a query result was fetched by the current request (an amount {`m1`} is present), the first record from the result set will be read. In addition, this record will be analyzed according to its content in the appropriate categories, which are listed above. If any content is available, the first term of the first category, that has entries (i. e. personal or corporate names and title entries from the bibliographic record), will be taken. As long as the first term is not a stop word, it is used for further processing. It depends on the currently chosen approach, whether this term is used for the further query or not. In case, that the particular term is a stop word, successively the next term will be taken and checked in the same way.
Is there finally not a single term present, that was qualified upon this logic, the record will be ignored and the next record will be read and analyzed.

In all cases the fields where used in this order:
1. person's name,
2. corporate name and
3. title field.

4.1 The first approach

The decision to use the first term, which is not a stop word, from the considered categories as a query term for the second request provides the advantages, that this approach

- is easy and fast to implement, because no very complex or in particular difficult tests have to be done on the single records
- and it usually leads to a fast response of the system.

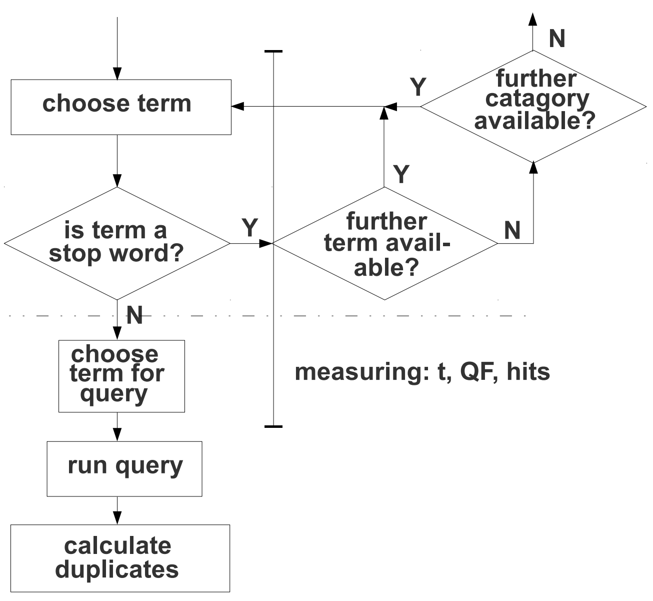

Figure 3: Flow chart from the first approach: The first term, which is not a stop word, is used as the query term.

For checking a small amount of data ($<$ 10.000 data records) as well as in environments, where no large quantities are to be deduplicated and no very large result sets are expected, this approach is certainly an appropriate one.
To note is the fact, that a database query is usually much more time consuming than the calculation of duplicates afterwards. Therefore it can be assumed, if little time is put into finding an appropriate request term, in the normal case, a very short response time can be observed in such a system.

Under the specific circumstances, which were already mentioned above, namely, if the bibliographic data base on the amount of an average Austrian university library, the major disadvantage in the implementation and launch of this approach appears quickly:

- the amount of resulting datasets can be occasionally very high, because each term can be used for the request, as long as the term is not a stop word, and it seems to the average, that many terms exist, which quickly lead to more than 1.000 hits.

For a system, that handles the display of results of online queries, this circumstance may be less relevant than for the deduplication of bibliographic records in a batch process: The expectable high amount of the resulting records must (afterwords) still be analyzed for the calculation of title duplicates. That is, that at the end of such an operation a particularly large amount of data is to be expected. Processing and analyzing the result as a following step can be extensive by itself.

Figure 3 shows the logic of this very simple kept approach in a flow chart: Once the first term, which is not

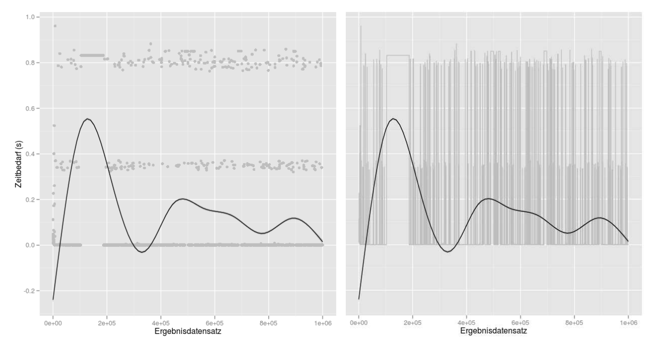

Figure 4: Approach 1: Measuring the time duration per record to identify the query term in a batch process.
The request is made by the first term, which is not a stop word. To the left, the representation of the values is printed in a scatter plot, right in a line chart.
Legend: x-axis = number of result sets, y-axis = time duration.

a stop word, is found, it will be used as a query term, and the request will be executed. In the figure also the two checkpoints within the logic, which were used for the measurement in detail (i. e.: the duration per record; the results are shown in Figure 4), are noted.

Subsequently, the above mentioned advantages and disadvantages were to review. For the measurement, a subset of 10.000 bibliographic data records were extracted and checked against the total amount.
The resulting total time was 6.245,762 s ($\approx$ 1 h 45 min). This time duration already includes the calculation of the duplicates. The number of result records was 1.049.057.

Figure 4 shows the process of the time required per output record. The scatter plot (on the left side in Figure 4) clearly shows that the measured values are not distributed continuously between the maximum and minimum characteristics. Rather, a significant accumulation of three values can be observed: 0 s, 0.35 s and 0.8 s. These values derive from the fact, that the query term is formed from the entries, that appear in one of three categories.
Can the term be formed out of the person's name (= the first category, which is used), it will be used for the request. In this case the measurement reaches a value very close to zero seconds. In addition, if the first term of the person's name, according to the shown logic in Figure 3, is used for a request (this is usually the surname), the resulting time duration is close to the measurement precision (this with "high resolution" to three decimal). Otherwise, the resulting value is typically located around 0.03 s.
Do the contents of the categories for the person's name not lead to any proper query term, the entries of the corporate's name are checked. This additional rerun does not causes, according to the appropriate measurement, a continuous increase in the time duration. Although the record has been fully stored in memory, and

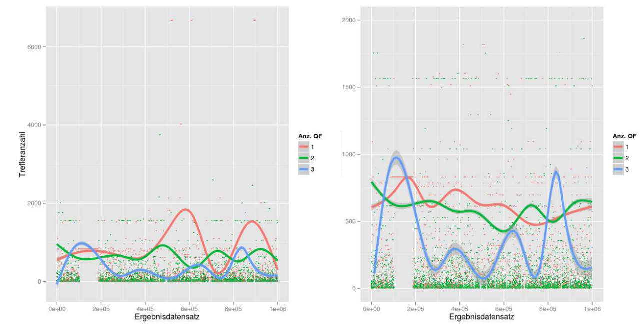

Figure 5: Approach 1: The number of result records generated per request.
{m1} = 10.000, {m2} = 400.000. QF = Number of categories (tags) with useful content, available to form the query term (max = 3, min = 1).
Legend: x-axis = number of result sets, y-axis = number of hits, Anz. QF = number of query fields.

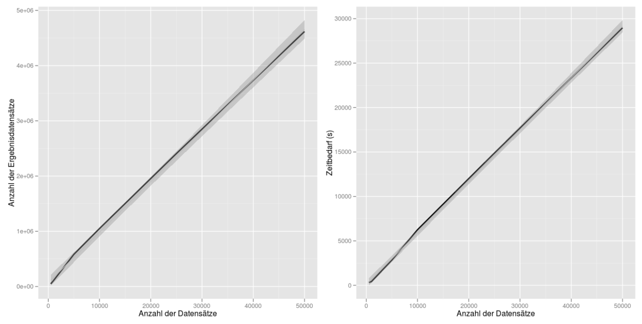

Figure 6: Approach 1: The number of resulting data sets, as well as the time needed for deduplication, depending on the number of records to be checked.
The values shown correspond to those from Table 1.
Legend: x-axis = number of records, y-axis (left) = number of resulting data sets, y-axis (right) = time needed for deduplication.

no new database query must be executed, the effort in time increases rather erratically. Rerunning the whole logical sequence to determine the query term, the certain values appear, which are shown in Figure 4 as an accumulation of the results by the value of 0.35 s.
The same effect, namely, the non continuous increase in the time required, is achieved, even if the procedure can't form any query term from the entries for corporate names. It is then to acquire the term from the contents of the corresponding title information. The effort, connected with this very last program step, finally causes a doubling of 0.35 s ($\approx$0.4 s) to 0.8 s.

In this context, the analysis of the time consumation per bibliographic record can be interesting, when the records are processed in a batch run: Requests that have be done once or more times by using the same query terms will be kept in the cache of the database. This fact causes that, later on, the same query can be dealt with significantly shorter response times. This circumstance is shown in Figure 4, but can only be identified by overlaying the individual values by an appropriate smoothing function.

This means for each series of measurements, which are compared to each other, that prior to the start of each iteration the database cache (in this case the query cache) has to be cleared. Otherwise, subsequent measurements would benefit from previous and lead to false results. The distortion of the results, based on the total flow, is hereby relatively low. In case that the same sequential batch run is repeated, the second run requires 6.149,775 s over 6.245,762 s from the first run. The difference is about 2 min at a cycle time of 1 h 45 min by processing 10.000 records. When processing the total amount of approximately 400.000 title records, the difference between the first and the second run is significantly lower, because the maximum of the available and finally deployable query cache can not be increased to a infinite value, belonging to the same hardware.

Besides measuring the time duration for each record, that passes the process to determine a proper query term, counting the number of result records is relevant. Already mentioned was the fact, that the implementation of the first approach, using the first term for the request which is not a stop word, occasionally generates a large amount of resulting data sets, which subsequently have to be checked on possible duplicates. The advantage, that this method very quickly leads to the appropriate query term, in such a case would turn into the disadvantage of a higher time consuming calculation, when detecting the bibliographic duplicates.
The already mentioned total number of result sets (1.049.057), which are generated by the sequential process, indicates this fact. Since this fact (of a possibly high number of result records), seen as an individual case (per single record and not in an overall process), did not become obvious in the practical application in an online system, it was investigated here in detail.

Figure 5 shows the individual values in a scatter plot on the left. Regarding to the number of categories, that are available to form the query term, the measuring points were shown in different colors: blue for the presence of three, green for two and red for just one category.
This figure clearly shows, that the presence of three categories is also advantageous when the query term can be formed out of the very first entry, which is not a stop word. However, are there only two or is there even only one category present, this circumstance, based on the single request, leads to a much larger number of hits (result sets). This can only be recognized by overlaying the individual values by a appropriate smoothing function.
Figure 5 shows on the right side only those values which lead to requests with a result < 2.000. This figure shows also the circumstance, that the vast majority of requests leads to results which are significantly smaller than 500. This may be considered as a first indication for a useful value (< 500) when introducing a threshold.

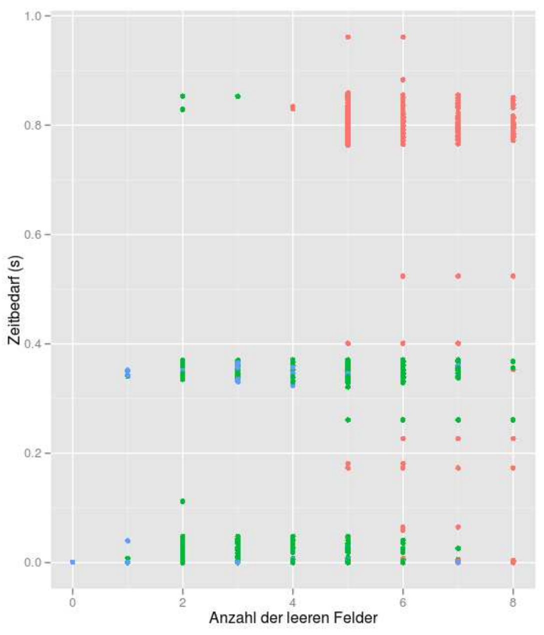

Figure 7: Approach 1: The time required for determining the query term per record, regarding the number of empty categories in the bibliographic record.
`{m1} = 10.000`, `{m2} = 400.000`. `QF` = Number of categories (tags) with useful content, available to form the query term (max = `3`, min = `1`).
Legend: x-axis = number of empty fields, y-axis = time required, Anz. QF = number of query fields.

It has already been shown in the analysis of Figure 4 that the number of those categories, which can be used to identify the query term, is a considerable degree for the time duration that is needed. The fact, that the time needed does not have a continuous but rather a discrete increase, shows also, that the number of terms, which are included in one category, has a far less effect on finding the request term, then the fact how many categories must be evaluated.

Figure 7 shows the relationship between the time required for producing the request term per record and the number of empty categories in the belonging title records. By analyzing the illustration, it can be seen, that from ten categories which are maximal available in the merged bibliographic data sets, up to eight entries can be without any content. For the evaluation the number of useful categories (shown in Figure 7 by the factor `QF` (Queryfields)) is actually of interest.
Nevertheless, here is also shown, that the effort to find a usable category for determining the query term is directly related to the number of categories which are empty. The analysis of the results, that were given in the previous figures do not suggest this fact yet. From the already shown figures, only the relationship between time consumation and the existing number of actually usable categories can be concluded. When processing the bibliographic data it has to be considered, that the fact of empty categories belong mainly to those, that actually would be needed for the next steps.

Bibl. Records	Result Sets	Time (s)
500	47881	335.450
1000	103754	418.308
5000	581753	2845.639
10000	1049057	6245.762
25000	2407018	14873.324
50000	4617445	28967.480

Table 1: Approach 1: The number of result data sets, as well as the time needed for deduplication in dependence on the number of records to be checked.

To estimate the number of resulting data sets as a function of the number of bibiliographic records a series of measurements has been formed on this approach which values are shown in Table 1. The regarding results show a linear relationship between the amount of bibliographic records to be checked in `{m1}` and the resulting amount in `{m3}`: Doubling the amount `{m1}` produces a doubling of the result sets of `{m3}`. The same situation applies to the resulting time duration, which is finally required for deduplication. Here, too, the double amount of `{m1}` leads to a doubling of the time spent. This relationship is shown in Figure 6.

In summary, the measurements to the first approach clearly show, that a higher number of available and usable categories leads to a lower value of needed time to determine the query term. This fact is quite expectable. Though, it was rather unexpected, that the effort per record takes a size, which can be determined in advance of deduplication, because it directly depends on the number of existing categories.
The highest density of results is achieved on the basis of query terms, that have a number of hits, which is smaller than `500`. By introducing thresholds (see approach 3), this value may already be seen as a useful reference. The disadvantage of the high amount of result records is clearly demonstrated; although, this fact can only be shown in the following comparison with the results from the second and third approach.

4.2 The second approach

In the case of processing a sequential batchjob to detect bibliographic duplicates – contrary to the deduplication of result records in online systems – large result sets are always to be expected, even when matching smaller quantities ($<$ `10.000` records). The optimal approach to minimize the result is to use that query term for the request, which is not a stop word, and which number of hits per single record is the smallest.
One of the major disadvantages of this approach is that the query term can only be determined by a much more elaborated check: All terms which are not stop words

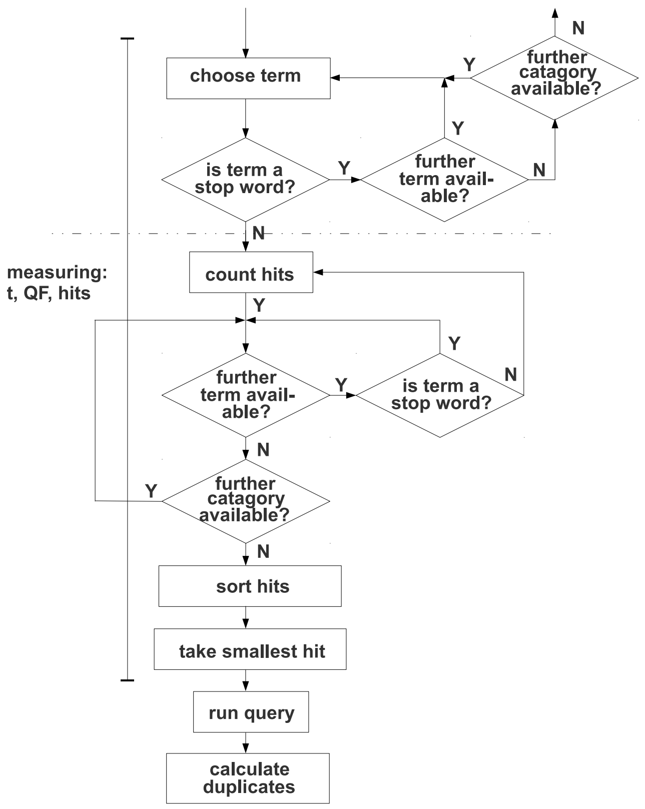

Figure 8: Flow chart from the second approach: That term is used as the query term, which is not a stop word and the number of hits is the smallest.

must be taken from every single record, out of `1 - 3` categories, and for every term the particular amount of hits must be counted. Afterwords, the term with the smallest number of hits must be taken and the request has to be done.

Under the already mentioned conditions, namely that a subset of `10.000` bibliographic records (as {`m1`}) was extracted of the total amount from `400.000` datasets and checked for duplicates within the whole amount {`m3`}, a total duration of `26.815,551 s` (≈`7 h 27 min`) could be measured. Thereby `210.661` resulting records were produced. This implies about a fourfold increase in the amount of time duration and simultaneously a fifth of the amount of result sets compared with the first approach. Therefore, the minimization of results in any case has to be discussed, when the whole process appears to slow down in such a significant matter. In all cases, in which large amounts of data need to be checked for bibliographic duplicates, this method can be seen as a reasonable one, because the following analysis of the results are easier to realize and less complex.

Figure 8 illustrates the logic to find the proper query term in a flow chart: All terms are successively extracted from the appropriate categories (tags) and will be further checked on stop words. Then the current number of hits is counted (in case they are used as a query term). Finally that term is chosen for the request, which is not

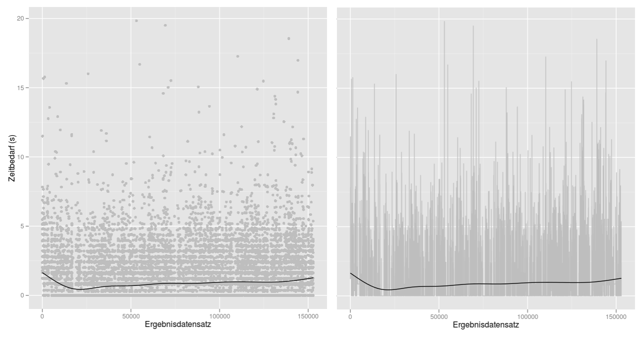

Figure 9: Approach 2: Measuring the time duration per record for determining the query term in a batch process.
The request is made by that term which is not a stop word and produces the smallest number of hits. On the left side the illustration of the values in a scatter plot, right as a line graph.
Legend: x-axis = resulting data set, y-axis = time duration.

a stop word, and which produces the smallest number of hits by the request. For each record this logic must be run, often for a hugh quantity of terms (namely all, extracted from the categories, which are used to form the query term). Furthermore, each run includes two database queries (checking for stop words and counting hits). Therefore it has to be expected that the appearance of large amounts of terms within a bibliographic record leads to a considerable need of time duration.

When looking at the individual results, which are shown in Figure 9, one can note, that the range from `0` to `5 s` is the most densely occupied and that "not only a few" individual results are located above this range. It can be seen as a conclusion of the batch run, that from analyzing the individual results the overall result can not be predicted (as it is in the first approach). The number of values, above shown in Figure 9 above `5 s` is quite remarkable. Finally, the approach to use the term with the smallest number of hits for a query leads to (only) a fourfold increase in the time required.

It has to be decided in each individual case, whether this method can be used for an appropriate integration into an online system. The specific fact should be considered in a particular way, that online systems, currently based on the technology of search engines, are optimized to react with the shortest possible response times. Under these circumstances it may therefore seem even strange to again extent the optimized response time with this approach, which optimizes the minimum of achievable result sets.

Contrary to the results of approach 1, the representation of hits per result record in approach 2 shows, that the number of hits can be significantly minimized. The highest density of hits in Figure 5 (corresponding to the approach 1) is located between the values `0` and `500`, but with approach 2 the highest density is settled in a much lower range, between `0` and `50`. Figure 10 shows, that

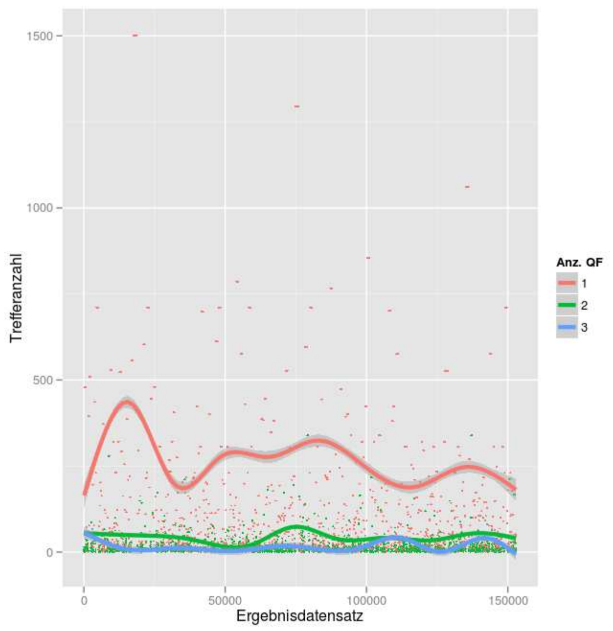

Figure 10: Approach 2: The number of result records generated per request.
`{m1} = 10.000`, `{m2} = 400.000`. `QF` = Number of categories (tags) with useful content, available to form the query term (max = `3`, min = `1`).
Legend: x-axis = number of result sets, y-axis = number of hits, Anz. QF = number of query fields.

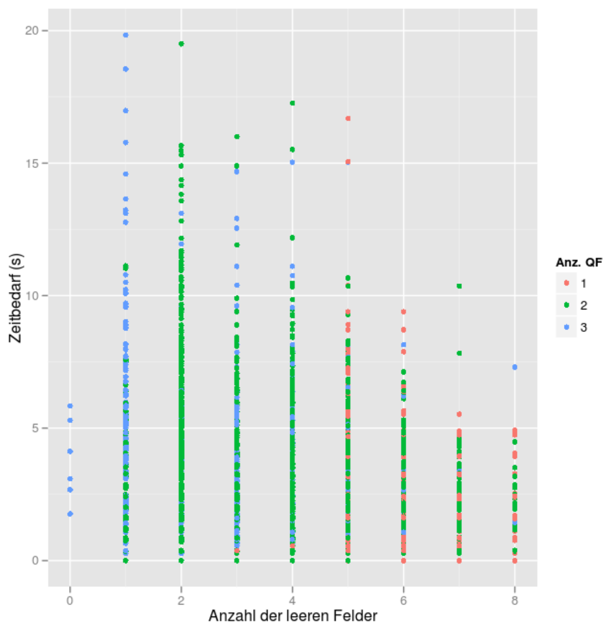

Figure 11: Approach 2: The time required for determining the query term per record, regarding the number of empty categories in the bibliographic record.
`{m1} = 10.000`, `{m2} = 400.000`. `QF` = Number of categories (tags) with useful content, available to form the query term (max = `3`, min = `1`).
Legend: x-axis = number of empty fields, y-axis = time required, Anz. QF = number of query fields.

for the case in which `2` or all `3` categories contain usable entries, the number of result sets per query is very low (sometimes < 10). The overall result clearly expresses this fact: The total number of result sets shrinks to one fifth compared with the first approach.

Figure 10 shows for the individual case the (expectable) fact, that the presence of fewer entries for determining the request term with asmaller amounts of hits also leads to the fact that fewer terms are present. Contrary to the results achieved with quantities of < 50 result records, in the case of only one usable catagory (for forming the request term), obviously values that are common in Figure 10 vary around the number of `250`. This can be discovered by overlaying the individual results in the scatter plot by an appropriate smoothing function.

The fact, that a small number of usable categories leads to higher numbers of hits per request, is also evident as the circumstance, that this further leads to an increased amount of time. Figure 11 shows the relationship between the number of empty data fields, present in the individual data records, and the time required to determine the query term as a function of the number of non usable categories. The analysis of the relevant information in the scatter plot is supported by a color coding: A higher number of empty fields also results a (expectable) lower number of usable categories and finally leads to a smaller amount of time to determine the query term. Described conversely, a larger amount of usable categories produces a higher amount of time needed, but it leads finally, as previously described, to a significantly reduced number of result data sets. This fact means for the individual case, that distinctably smaller result sets can only be achieved by much higher time consuming. When considering the whole process, this does not have such a big effect, because a much smaller number of hits causes also a remarkable reduction in the time required to calculate and evaluate the bibliographic duplicates.

In summary, the second approach illustrates, that the time duration for a single data record can be rather considerable, so this approach will probably apply in a batch run only if the number of result records must be very small for further reasons.
The decision, whether this approach can also be used in an online system, depends on the total, “felt” time, which is added to the normal behavior of the system, optimized for a very quick response. In individual cases, the approach 2 can be perceived as a far more disturbing. The overall behavior to a potential user can only be assessed through specific observation.

4.3 The third approach

The advantages and disadvantages, that arise from the two previously described approaches, can be beneficial supplemented by a quasi-compromise to a third ap-

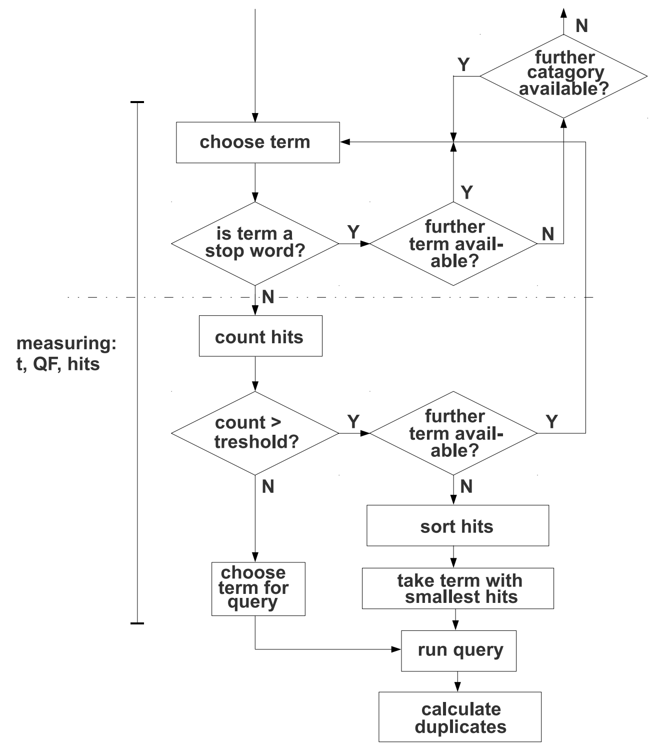

Figure 12: Flow chart from the third approach: That term is used as a query term which is not a stop word and its number of hits is below a defined threshold.

proach: The advantage to find an appropriate query term very quickly, without considering in any case all potential entries (= time optimization), can be combined with the advantage, that not every term comes into selection, which keeps the number of resulting data sets rather small (= optimization of the amount). By introducing a threshold for the maximum number of results per query term, the first term with a number of results below this threshold is used for the proper query term – without checking the other, or even all, terms. Such an approach provides a relatively fast response time (close to that from approach 1) and, at the same time, a relatively small total number of hits (in the best case very close to that of approach 2).
Therefore, in approach 3, the first term, which has a certain number of hits below the defined threshold (and which is not a stop word) is used as the query term for the deduplication afterwards.

The high density of result values in the range from `0` to `500`, which can be seen in Figure 5, had already guessed a potential threshold value, which may be somewhere below `500`. For example, `100` is stated as an effective threshold i. a. in Lohrum (1999, p. 3). The given reason is, that this amount of result sets can be checked quickly for bibliographic duplicates, and larger numbers of hits would be much more time consuming to process. Regarding to the recent results, it can be shown through the different measurements and the interpretation of their representation in Figure 4, that a possibly necessary, additional database query to determine the request term is much more time consuming than the deduplication of a larger result set.

Treshold	Time	Amount
2	26815.551	210661
25	9022.699	238164
50	7696.171	262276
100	6620.200	305345
200	6020.932	376623
300	5723.736	466029
400	5653.442	509084
500	5880.583	692117
600	5924.068	720809
700	5921.980	790049
800	5953.812	825637
900	6183.725	952801
1000	6217.125	958996

Table 2: Approach 3: Size of the threshold, measured time and total number of hits.

Dealing with the third approach it is to be tested, if a particularly effective threshold can be determined empirically and to show, what other implications can be observed when introducing this method.

Figure 12 shows the logic used, when the routines are seeking for the query term: From the first available category, the first term, that is not a stop word, is used, and the number of hits will be counted in a request. If the amount is less than the threshold, the term in use is the proper query term and the achieved result set contains the bibliographic data, which will be deduplicated by the following process. Is the number of hits above the threshold or equal to it, the programmed logic looks for a further term.
Finally, is there no appropriate query term with a certain number of hits below the threshold, the term with the smallest amount of result sets is used for the further request. The lower the threshold is defined, the more often this will take place. That is, with a minimization of the threshold value, the results of this procedure become more congruent to those of the second approach and increasing the value brings the results closer to those of the first approach.

To gain an overview of the behavior of such a process, thirteen different thresholds in the range of `2` to `1.000` were defined. Subsequently, the amount of `10.000` bibliographic records were checked against the total amount of `400.000` in a batch run for duplicates. In each run the entire duration was measured. The analysis for the measurement of the individual results (that means: the certain number of hits per single query term, the time required for the determination of the request term and the number of available, qualified categories) did not produce any new information additional to those of approach 1 and 2. Here, the achieved values are not documented here in detail and separately for each threshold.

Table 2 shows the measured values for the time required and the total numbers of hits, which were identified with the used threshold values in the range from 2 to 1.000.

Figure 13 illustrates this relationship by a graph:
Is the threshold value defined below the size of 200, the amount of needed time to find the request term primarily rises continuously (see Figure 13 left). When falling below the value of 100, time and effort on complexity rise very steeply (almost exponentially).
Will the threshold be increased gradually from the initial value (here: 200) in single steps of 100, at first a slight decrease in the amount of time needed appears, which rises slowly again from the threshold of 400.

This specific behavior is motivated by the fact, that minimizing the threshold means, that ultimately all of the terms from the qualified categories have to be checked to see, if they result in a request with a number of hits, which is below the defined threshold. The time required for this check appears to be significantly larger than those savings, which occurs when very small numbers of result sets must be deduplicated. The slightly and continuous increase in the time required above a threshold value of 400 is due to the fact that, starting from this value, the time required for determining the request term is stagnating, but at the same time the amount of result sets, which must be deduplicated, continuously increases (see Figure 13 right).
This behavior is congruent with the assumption in their effects, which was derived from the scatter plot of Figure 5: Even since the first term, which can be used for a request, leads in the vast majority of bibliographic records to a smaller number of hits than 500, hardly bibliographic records can be found, which produce a certain number of hits “above” the threshold of 400 with another term, which must be checked.

Figure 13 (right) shows the almost linear relationship between the size of the threshold and the amount of result records. In case that the amount of result data sets keeps in further consequence (if this amount will be optimized for example) a major role, a value between 100 and 200 may be well regarded as practical. This can be further relevant in case of deduplicating large data amounts in a batch run. Than the fact is to be considered, that only the increase of the threshold from 100 to 200 already leads to an enlargement in the total number of hits by 24 %, although the involved time duration remains approximately the same.
On the other hand, in case of an deduplication of query results in an online system, a faster response time can be reached with a threshold of 400, when it can be assumed, that deduplication by computing (calculating) power is much faster than any further, possibly necessary database queries for determining the better query term with less results.

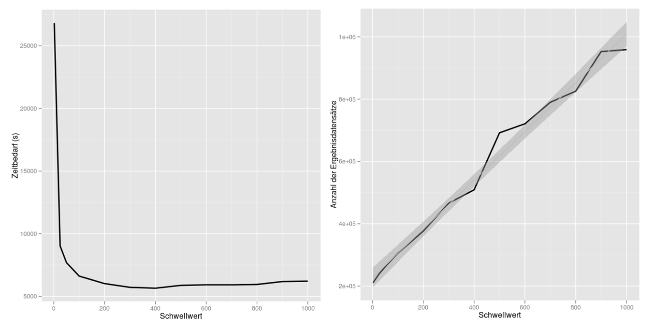

Figure 13: Approach 3: The total time required for identifying bibliographic duplicates depending on the threshold (= left illustration).
The number of produced result data sets, depending on the threshold for identifying bibliographic duplicates (= right illustration). {m1} = 10.000, {m2} = 400.000.
Legend: x-axis = threshold, y-axis (left) = time required, y-axis (right) = number of result records.

5 Summary

The pros and cons of each approach, cited in the text, could be essentially confirmed.

In the first approach the advantage appears obviously, that the choice to accept the first term, which is not a stop word, for the query term may represent a method, that is necessarily qualified for an integration into an online system. Particularly, this approach is appropriate, if the relevant online system has been optimized for short response times. On the other hand, for deduplication in a batch process, the disadvantage of this approach appears clearly: The large amount of result sets, which are caused by the query term, produces a significant effort in the further process to evaluate the duplicates.

For the batch process, approach 2 is always appropriate, when there is no big time pressure on the procedure to achieve the results, and if the amount of bibliographic records, which has to be deduplicated, is not very large. By identifying that request term, which achieves the least amount of hits, a much larger time duration occurs (about a fourfold increase compared to the first approach), but at the same time the number of hits is reduced to one-fifth. For an integration into an online system, the implementation of this approach may lead to an unwanted slowdown of the overall system.

The third approach, which can be seen as a compromise between the two previously mentioned, may be interesting as well for a batch process as for an integration into an online system. When choosing an adequate threshold, it has to be considered, that either short response times (by a larger threshold) or low numbers of resulting data sets (with a smaller threshold) can be achieved. In this case, the fact, which is shown in Figure 13 on the left, might be interesting: By increasing the threshold gradually, a slight decrease in the time required can be

observed between the values of 200 and 400 and then an equally light, but very steady rise appears.
This is due to the fact, that in typical environments of bibliographic data, such as those which is used here, the vast majority of terms, that can be used for a request, cause significantly less than 400 results (see Figure 5). From this treshold value, the effort, to identify the query term through multiple requests, can be omitted: The query term, discovered in the first or at least the second attempt, already fulfills the requirements and can be used.
For the use in the discussed approaches and measurements, 200 has shown as an appropriate value for both, the integration into an online system and for the batch process.
Decreasing the threshold to < 100, has shown, that the actual benefit, to achieve lower numbers of result sets, became unimportant and was rather detrimental, because the time required to detect a query term rapidly rised.
After all, there are some open questions. For example it is still not veryfied whether it is better to start with the person's name, the title field or the corporate name.

6 References

Ananthakrishna, Rohit; Chaudhuri, Surajit & Ganti, Venkatesh (2002): Eliminating fuzzy duplicates in data warehouses. In: Proceedings of the 28th international conference on Very Large Data Bases. VLDB '02, p. 586 - 597.

Baxter, Rohan; Christen, Peter & Churches, Tim (2003): A Comparison of Fast Blocking Methods for Record Linkage. In: SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, Washington DC.

Bilenko, Mikhail; Kamath, Beena & Mooney, Raymond (2006): Adaptive Blocking: Learning to Scale Up Record Linkage. In: Sixth International Conference on Data Mining. ICDM '06, p. 87 - 96.
Online:
`http://dx.doi.org/10.1109/ICDM.2006.13`

Draisbach, Uwe & Naumann, Felix (2009): A comparison and generalization of blocking and windowing algorithms for duplicate dedection. In: Proceedings of the International Workshop on Quality in Databases, p. 51 -56.
Online:
`http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_Naumann/publications/2009/QDB09_crc.pdf`

Draisbach, Uwe & Naumann, Felix (2010): DuDe: The Duplicate Detection Toolkit. In: 8th International Workshop on Quality in Databases. QDB'10. Singapore.
Online:
`http://www.vldb2010.org/proceedings/files/vldb_2010_workshop/QDB_2010/Paper5_Draisbach_Naumann.pdf`

Hernández, Mauricio & Stolfo, Salvatore (1995): The merge/purge problem for large databases. In: SIGMOD '95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data, p. 127- 138.
Online:
`http://dx.doi.org/10.1145/568271.223807`

Hernández, Mauricio & Stolfo, Salvatore (1998): Realworld Data is Dirty: Data Cleansing and The Merge/Purge Problem. In: Data Mining and Knowledge Discovery. Vol. 2 (Issue 1), p. 9 -37.
Online:
`http://dx.doi.org/10.1023/A:1009761603038`

Jele, Harald (2009): Erkennung bibliographischer Dubletten mittels Trigrammen: Messungen zur Performanz. In: BIT-Online (Issue 3, 2009), p. 265 - 272 (= Part 1) and BIT-Online (Issue 4, 2009), p. 385 - 390 (= Part 2).
Pre-Print online:
`http://wwwu.uni-klu.ac.at/hjele/publikationen/ngramme/2009_ngramme_main.pdf`

Lohrum, Stefan; Schneider, Wolfram & Willenborg, Josef (1999): De-duplication in KOBV. Konrad-Zuse-Zentrum für Informationstechnik Berlin. Preprint SC 99-05 (June 1999).
Online:
`http://opus4.kobv.de/opus4-zib/files/393/SC-99-05.pdf`

Paulevéa, Loïc; Jégoub, Hervé & Amsalegc, Laurent (2010): Locality sensitive hashing: A comparison of hash function types and querying mechanisms. In: Pattern Recognition Letters (Vol. 31, Iss. 11), p. 1348 - 1358.
Online:
`http://dx.doi.org/10.1016/j.patrec.2010.04.004`

Schneider, Wolfram (1999): Ein verteiltes Bibliotheks-Informationssystem auf Basis des Z39.50 Protokolls. Diploma Thesis, Technische Universität Berlin.
Online:
`http://wolfram.schneider.org/lv/diplom/diplom.pdf`

Stein, Benno & Potthast, Martin (2006): Hashingbasierte Indizierung: Anwendungsszenarien, Theorie und Methoden. In: Workshop Special Interest Group Information Retrieval (FGIR 06) (= Hildesheimer Informatikberichte), p. 159 - 166.
Online:
`http://nbn-resolving.de/urn:nbn:de:gbv:hil2-opus-672`

Unwin, Antony; Theus, Martin & Hofmann, Heike (2006): Graphics of Large Datasets. Visualizing a Million. Springer, New York

Wickham, Hadley (2009): ggplot2. Elegant Graphics for Data Analysis. Springer, Dordrecht.
Partly online:
`http://books.google.com/books?isbn=9780387981420`

Yan, Su; Lee, Dongwon; Kan Min-Yen & Giles, Lee (2007): Adaptive sorted neighborhood methods for efficient record linkage. In: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries. JCDL'07, p. 185 - 194.
Online:
`http://dx.doi.org/10.1145/1255175.1255213`

Dr. Harald Jele
works at University of Klagenfurt
Address:
Robert Musil-Institut
Bahnhofstraße 50
9020 Klagenfurt, Österreich
E-Mail:harald.jele@uni-klu.ac.at