Université de Genève HEG – Département Information Documentaire

CESID
-
Diplôme universitaire de formation continue en information documentaire

Travail de recherche

Development of a flexible tool for the automatic comparison of bibliographic records. Application to sample collections

Développement d'un logiciel flexible pour la comparaison de notices bibliographiques et application à différentes collections

Alain BOREL
Jan KRAUSE

Supervision: Prof. Jaques SAVOY

2009

Table of Contents

I Résumé

La multiplication des catalogues bibliographiques électroniques (archives institutionnelles, catalogues de bibliothèque ou de fournisseurs, etc.) met à la disposition des spécialistes de l'information documentaire de très grandes quantités de métadonnées qu'il s'agit de traiter en masse en vue d'objectifs divers. Une problématique importante à ce niveau est la détermination de la similarité entre notices, que ce soit dans une optique bibliographique (décrivent-elles le même document ? Utilité pour la détection de doublons, l'étude du recouvrement entre collections...) ou plutôt thématique (suggestion de documents à l'usager, mais aussi gestion des contenus dans le cadre d'une politique documentaire locale ou en réseau, classification automatique de documents...).

Afin de répondre à ces besoins variés, nous proposons de créer un logiciel libre, multi-plateforme et flexible permettant l'implémentation de nombreuses stratégies pour la comparaison des notices. Dans une seconde phase, nous étudierons la pertinence et la performance de différents algorithmes face à une sélection de collections (taille, origine, type de documents décrits...).

II Abstract

Due to the multiplication of digital bibliographic catalogues (open repositories, library and bookseller catalogues), information specialists are facing the challenge of mass-processing huge amounts of metadata for various purposes. Among the many possible applications, determining the similarity between records is an important issue. Such a similarity can be interesting from a bibliographic point of view (i.e., do the records describe the same document, the answer to which can be useful for deduplication or for collection overlap studies) as well as from a thematic point of view (suggestion of documents to the user, as well as content management within the framework of a library policy, automatic classification of documents, and so on).

In order to fulfil such various needs, we propose a flexible, open-source, multiplatform software tool supporting the implementation of multiple strategies for record comparisons. In a second step, we study the relevance and performance of several algorithms applied to a selection of collections (size, origin, document types...).

III Introduction

In standard Library Information Systems (LIS), documents are represented by bibliographic records stored in standardized formats (Rivier 2007, pp.60-65). Since the analogic documents themselves cannot be directly connected to the catalogue, they provide a compact description of the collection (containing information such as titles, author names, subject keywords, abstracts, physical description of material objects...), with which a user can interact more easily. With the advent of digital libraries, direct access to documents has become possible. Nevertheless, bibliographic records have not become useless: they have been retained and generalized as metadata, that is structured digital data describing another digital object.

In the numerical era, it is trivial to observe that the duplication and communication of documents have become extremely simple, a quantum leap comparable with the one caused by the introduction of the printing press by Gutenberg in Europe. Of course, metadata have undergone the same mutation – a mutation made even easier by their structure and relatively small size, suitable for database applications. In the late 60s, the MARC (*MAchine Readable Catalogue*) standard for the representation of library records was established at the U.S. Library of Congress (Avram 1968), and soon it spread to many libraries worldwide. As integrated library systems (ILS) and library networks evolved during the late 20th century, these records have become more and more openly available, first to other libraries and then to third parties and the general public – originally through more or less user-friendly online catalogue interfaces, and soon through more direct protocols allowing the retrieval of complete records for custom processing, such as Z39.50 and OAI-PMH2 (Rivier 2007, pp.109-117). For librarians, it allows a rationalization of the cataloguing process: it is no longer necessary for each library to enter records manually for documents that have been catalogued somewhere else. For developers, it allows the creation of innovative applications based on a wealth of good-quality data.

The first application of record similarity determination is an almost immediate consequence of the above situation. Despite the advantage of harvesting records from others, there is still no central cataloguing authority. This means that the same document will frequently be catalogued by several independent sources (which could be libraries, institutional repositories, book or journal publishers, free or commercial databases...), yielding similar but not necessarily identical records. If we consider the ease of exchanging records, we see that the probability for one user of encountering two (or more) different records that actually describe the same document becomes significant. As long as we only talk about a few records at a time, it isn't a big deal and the human brain will notice the duplicates easily enough. When we consider thousands or millions of records, we have to recognize that despite its power, our distinguished organ is no longer up to the task. We need an automatic way to assess the similarity of two bibliographic records hidden among a large number of other bibliographic records, which can be useful for quality control or for comparative collection analyses. This problem is not new and various examples can be found in the literature, such as the Digital Index for Works in Computer Science (DIFWICS) (Hylton 1996), the QUALCAT project at the University of Bradford (Ridley 1992) and the creation of the COPAC union catalogue in Britain (Cousins 1998). It can be seen as a special case of the more general question of finding duplicate records in a database, which remains an active research topic today (Elmagarmid et al. 2007)

Another application of bibliographic similarity is information discovery. We would like to identify records that do not describe the same document, but rather documents similar to each other. One example of this application is relevance feedback for information retrieval: if one particular document has been selected as relevant by the user, the system will be able to suggest other similar documents with a reasonably high relevance probability. Another related example is information monitoring: using a collection of records representing one's needs, it is possible to point out automatically and periodically the most interesting documents amidst the torrent of new records offered by one or more information sources.

When it comes to similarity for information discovery, librarians can become users, too. In 1998, the International Federation of Library Associations and Institutions (IFLA) has proposed a new model of bibliographic description (IFLA 1998) called Functional Requirements for Bibliographic Records (FRBR). Broadly speaking, FRBR introduces a relationship between different editions (called *manifestations*) of the same book (*work* or *expression*). The conversion to the new model of a large library catalogue, or a union catalogue for a library network, by human workers is not a project that library managers can hope to « sell » to the funding authorities – and probably not to librarians, either. The case for an automatic procedure is compelling (Freire et al. 2007).

If we broaden the scope, we can consider the collection not only as an ensemble of documents (or document representations) but as the ensemble of the subjects that they discuss. The analysis of the text elements of bibliographic records and their similarity from one record to the next provides a way of grouping documents according to their subject, in a pseudo-semantic way, and reveal the structure of the collection - which can be as large as an entire scientific field if a good specialized database or subject repository exists - with respect to its information content. There are many possible goals to this grouping: automatic classification (an interesting application even for non-digital libraries), clustering of displayed records as an alternative to the standard (and often unattractive) interfaces of library catalogues and bibliographic databases), etc.

Turning back to a more literal similarity but looking beyond the bibliographic records to examine the digital documents as well, we find another important application: the detection of “plagiarism and covert multiple publication of the same data [that] are considered unacceptable because they undermine the public confidence in the scientific integrity” (Errami et al. 2009). Errami et al. wrote a text similarity based tool able to flag this kind of practices in the biomedical domain using the MEDLINE database (Medline 2007). An other plagiarism detection study using text similarity was conducted on the arXiv.org collections. It is based on the sentence composition of fulltext documents (Sorokina et al. 2006). In the Swiss academic community, this topic is also a concern. For example, the universities of Geneva (Bargadaà 2008) and Lausanne, as well as EPFL (Bogadi 2007) and the University of Zurich (ATS 2008) have publicly announced an anti-plagiarism policy. Various plagiarism detection tools exist. All of them require a lot of computing power and a vast knowledge base. So much so that institutions often choose commercially hosted solutions (like Turntin[1], Compilatio[2], or Ephorus[3]). There are of course open source academic plagiarism detection alternatives like Copy Tracker[4] and free of charges services like Plagium[5], SeeSources[6] or eTBLAST[7]. These tools need to access a huge quantity of data because they face plagiarism in all academic fields. To do so they seem to rely commonly on the powerful infrastructures of web search engines, and sometimes well structured metadata. For example, Plagium uses Yahoo's API, and eTBLAST is taking advantage of MEDLINE's metadata.

In this project, we have used the Python programming language to develop a flexible, open-source, multiplatform software tool supporting the implementation of multiple strategies for automatic MARCXML record comparisons. We extracted sample data sets from several real-world collections, namely the CERN Document Server (CERN 2009), the RERODOC digital library (RERO 2009) and the ETH e-collection (ETHZ 2009). We demonstrate the application of our new tool for the detection of duplicate records: several detection strategies were used to identify artificial near-duplicates added to our test collections. Other applications like information monitoring, plagiarism detection, and visualization of collections in the form of graphs have also been briefly tested.

---

[1] http://turnitin.com/
[2] http://www.compilatio.net/
[3] http://www.ephorus.com
[4] http://sourceforge.net/projects/antiplag/ , http://copytracker.ec-lille.fr
[5] http://www.plagium.com
[6] http://seesources.com/
[7] http://invention.swmed.edu/etblast/

IV Technical section

IV.1 MARCXML

MARCXML is an XML (*eXtensible Markup Language*) representation of the classic MARC format for library records (Avram 1968). The general structure of the data is shown in Figure 1. It is thoroughly described in an XML Schema hosted by the U.S. Library of Congress (LoC 2009). The top-level node is called *collection*, with any number of *record* nodes as children.

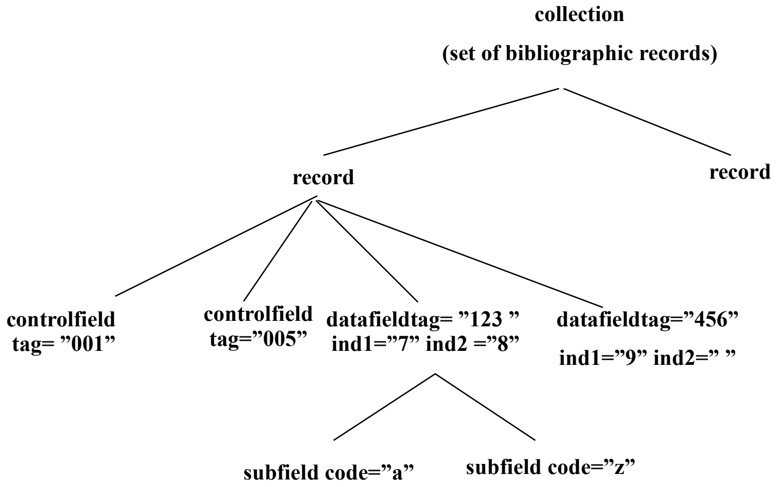

collection
(set of bibliographic records)
record
record
controlfield tag= ”001”
controlfield tag=”005”
datafieldtag= ”123 ” ind1=”7” ind2 =”8”
datafieldtag=”456” ind1=”9” ind2=” ”
subfield code=”a”
subfield code=”z”

Figure 1: Tree structure of MARCXML collections.

Each record contains various *controlfield* and *datafield* nodes. Both node types have a mandatory *tag* attribute with a 3-digit code. Furthermore, data fields carry two mandatory 1-character attributes *ind1* and *ind2* (the attribute character can also be a space) and can contain zero or more *subfield* nodes, with a 1-character *code* attribute. Only control fields and subfields contain actual character data, the difference being that control fields, limited to a few applications, contain only one single data element or a series of fixed-length elements (LoC 2008), whereas subfield content is essentially free-format text (barring cataloguing rule constraints, of course). The cataloguing conventions assign a *tag/ind1/ind2/code* to a particular usage, such as 245/ / /a (spoken as 245$a by librarians) for the main title of a document.

For the sake of convenience, in the following text, we will call a field any MARCXML node able to contain actual data (controlfield or subfield).

IV.2 Text similarity and information retrieval models

Although MARC records contain highly structured data, whose natural habitat is the library catalogue database, a database-like approach is not always the best way to process them. For example, as mentioned previously, MARC subfields contain free-format text, only limited by the cataloguing rules (with many variants and highly variable respect from the users) or the controls enforced by the cataloguing tool interface (quite variable as well). Thus one cannot always count on any field to contain anything more specific than text (such as numbers or dates), regardless of what the cataloguing rules say. Consequently, it can be useful to consider all kinds of text-comparison algorithms for the task of matching two records. A number of such methods have been developed along the years, starting with the simplest character-to-character comparison, followed by wildcards, fuzzy matching (Navarro 2001), regular expressions (Friedl 2006), and so on. These examples are of course just a short selection among many others, and an exhaustive list is certainly not our goal. In this work, we want to evaluate *information retrieval* methods. Broadly speaking, information retrieval can be defined as "[...] finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" (Manning et al. 2008, p.1) Such methods are normally intended to find the documents of a collection that provide the best possible match to a user-supplied query, based on statistical properties of the document content. However, the query itself can also be seen as a document in its own right. This is where information retrieval meets our purpose: we will consider MARC records and their subfields as queries performed over a set of records from one or several collections.

IV.2.1 The Boolean model

In this information retrieval model, queries are expressed by a series of terms, which can be either words for which the system will search, connected by logical operators (the AND operator being an implicit default in general). The system will analyse this query and retrieve the documents that contain the specified words, very much like a relational database. Depending on the operators used in the query, the system might have to retrieve intermediate result sets and calculate combinations: unions, intersections, differences. If the system is designed to support structured data, the user can specify in which data field one particular term should be found (a typical example would be `author:Einstein AND year:1905`). More elaborate systems will also understand an extended operator vocabulary, with keywords such as `NEAR` indicating the proximity of the connected terms in the requested documents.

This model gives the user a great deal of control over his query. However, the average user is rarely able to take advantage of this control, due to the non-intuitive character of Boolean logic. Furthermore, it is extremely sensitive to the presence or absence of terms in the query and documents. Documents that match the query only partially will be completely neglected by the system, and spelling errors will have a serious impact on the results. Furthermore, the standard Boolean system doesn't provide any way to sort the resulting documents with respect to their expected usefulness to the user (relevance ranking). In the hands of an inexperienced user, Boolean information retrieval systems will frequently return a huge number of hits with no clear usefulness (excessive recall in the case of too general a query) or no hits at all (if the query is too specific).

Although these inconveniences can be solved in part by assigning weights to the terms (Bookstein 1980), in the so-called *hybrid Boolean model* (Savoy 1997), we see that the Boolean approach is not a good choice for our project. The terms of the queries will be largely out of our control since the field contents are given by the collection, and a rigid character string comparison are not suitable for some applications.

Wildcards and fuzzy searches

As mentioned in the above section, the variable form of words in natural text can be a problem for information retrieval systems. Such variations can be due to spelling errors, spelling variants (UK versus US English, spelling reforms in various countries) or even grammatical inflections (singular versus plural forms, case declensions in some languages, etc.). In order to retrieve the relevant documents regardless of such variations, the system needs to account for them in some way. Since this is likely to happen in our collections and play a role in our applications, we will briefly discuss a few examples and their possible use in our project.

In text documents, some writing systems will induce a variation of the script symbols, which may be reflected in the computer character encoding or not. An obvious example is the upper-case versus lower case form of the Roman (as well as Cyrillic and Greek) letters. In the current project, we will mostly deal with Western European languages so changing all text to lower case will be sufficient. However, should the program be used for languages using non-Roman alphabets, some parts of the code will have to be modified to deal with their specificities.

We have to mention wildcards due to their frequent availability in database systems. These special characters are inserted in the query to represent either one or several unspecified characters. Thus they can be used to represent words in a generalized form, accounting for spelling or grammatical differences – as long as these differences are not too large: the liberal use of wildcards in a query can easily lead to a serious loss of precision (*c*t* will match *cat*, *cut*, *court*, *covenant*...). As in the case of Boolean and proximity operators, this feature will not be used in our project.

A quantitative evaluation of the difference between two character strings is provided by the *edit* *distance*, that is the minimal number of characters that need to be modified (added/subtracted or substituted) to the first in order to obtain the second (Levenshtein 1966). If the distance between two strings of similar lengths is much smaller than the length of the shorter string, one might suppose that they are equivalent (thus a few spelling errors in a text will be overlooked). This method is currently used in various library applications, such as the open-source Greenstone digital library software (Greenstone 2009) - thanks to the underlying Lucene search engine (Apache 2009).

The spell-checking capability of the Levenshtein distance makes it an attractive tool for our similarity engine, as we expect to observe relatively small distances when looking for duplicate records. We note that more tolerant techniques exist if one expects a high level of errors, such as the Soundex method (Knuth 1998, p.394) (useful for people names typed from “noisy” sources: handwritten text, phone calls...).

Grammatical variations can be taken into account though word *stemming* or *lemmatization* (Manning et al. 2008, pp.30-33). These techniques will reduce inflected words to a simple invariant form (*stem*) before they are used for comparison or matching, which makes the query less sensitive to the syntax of the documents and more to their semantic content. Obviously, these techniques are language-dependent. For English, the best-known example is Porter's stemmer (Porter 1980), which is based on the empirical removal of suffixes. At this point, we have not included any such features in the program but adding it should be straightforward since a Python implementation is available from the reference web site (Gupta 2008).

IV.2.2 The vector model

Introducing the vector model

In the Boolean model, a document may either be match or not match at all by a query. There is no finer distinction than that. This is not appropriate in many situations (Manning et al. 2008, p.100). For example, a query like “`pollution AND city AND environment AND co2 AND nox AND bicycle AND bus AND car AND train`” will not match a document, even if just one of those syntagms is absent. It is obvious that the document could still be relevant to the query. This flaw is crippling in our context because instead of trying to match relatively small queries to documents fields often of similar sizes are compared (like abstracts, titles). A dreadfully low recall would follow. Fortunately, the vector model solves this weakness by taking into account the similarity of the weights of each terms respectively in the query and in the documents.

Weighting

The weighting is based on the terms frequencies (*tf*), which are generally improved in two ways. The first improvement of the weight computation consists in getting rid of the effect produced by the document size on the absolute term frequency. The weight becomes the normalized term frequency (*ntf*). There are plenty of variations to calculate this, like for example, simply dividing each term frequency by the total number of terms in a document. However, we retained a more sensitive approach:

$$ntf_{d,t} = \frac{tf_{d,t}}{max_d(tf_{d,t})} \tag{1}$$

Where *ntf* is the normalized term frequency, *tf* is the frequency in a given document, the indices *d* and *t* refer respectively to document and term. This formula is a simple case of the well studied technique of maximum *tf* normalization (Manning et al. 2008, p.117)

The second way to improve the weight computation consists in taking into account the fact that some terms are less important than others. For example, in a collection about the Python programming language, even if the term 'Python' will probably be well represented in documents, it will have no discriminating power at all. In addition, the most frequent words (first ranks in the Zipf distribution) of natural language documents are generally function words that have little or ambiguous meaning (Rivier 2007). In other words, the normalized term frequency may be improved by balancing it with the normalized inverse document frequency (*nidf*). Again, there are several ways to calculate this quantity. We retained a common and efficient one that is using a logarithm to sharpen the desired effect on the weight.

$$nidf_t = \ln\left(\frac{N}{df_t}\right) \tag{2}$$

Where: *nidf* is the normalized inverse document frequency, *N* is the number of documents in the collection, df_t refers to the frequency of documents containing the term *t* (Manning et al. 2008, p.108)Finally, these two improvements on weight computations are combined in the *tf-idf* , respectively *ntf-nidf* approach. The *ntf-nidf* weight is simply the multiplication of the normalized term frequency (*ntf*) by the normalized inverse document frequency (*nidf*).

$$w_{dt} = ntf_{d,t} * nidf_t \tag{3}$$

Vector space scoring and similarity

This operation consists in using each terms weight in a document and a query to produce a score, representative of the similarity between document and query. This score is preferably normalized in the [0..1] interval for further use. The query terms weights and the document terms weights can be assimilated to vectors whose dimension equals the number of terms. After that, various methods for comparing or matching these vectors exist, and studies shown that "no algorithms or approach distinguished itself as being greatly superior to the others" (Noreault et al. 1981, pp.60-61). Generally, the assumption that the terms are independent, or orthogonal in linear algebra terminology, is made. It is the case of the three vectorial similarity measurements used in this project. This was however criticised (S. K. Wong et al. 1987).

In the following formulas: w symbolises weights, d a document, q a query, t a term and T the number of terms in the query.

Salton's cosine

Because we are dealing with vectors it seems almost a natural solution: as the angle between the vectors tends to 0, the cosine similarity will approach 1. On the other hand if the vectors are orthogonal the similarity will be 0. Algebraically it is expressed by the following formula (Lewis et al. 2006):

$$cosinus = \frac{\sum_{t=1}^{T} w_{qt} w_{dt}}{\sqrt{\sum_{t=1}^{T} {w_{qt}}^2 \sum_{t=1}^{T} {w_{dt}}^2}} \tag{4}$$

Dice

Following a set-like approach, the Dice similarity measure (Lewis et al. 2006) represents the ratio of the intersection of two sets over their union. As the elements belonging to the intersection of the sets are taken twice into account in the denominator, the numerator is multiplied by two in order to compensate.

$$Dice = \frac{2\sum_{t=1}^{T} w_{qt} w_{dt}}{\sum_{t=1}^{T} {w_{qt}}^2 + \sum_{t=1}^{T} {w_{dt}}^2} \tag{5}$$

Jaccard

The Jaccard measure is a bit similar to Dice's (Lewis et al. 2006). In opposition to the Dice equation, where the intersection was taken twice into account in the denominator, the Jaccard model subtracts this intersection, so that it is not necessary to multiply the numerator by two.

$$Jaccard = \frac{\sum_{t=1}^{T} w_{qt} w_{dt}}{\sum_{t=1}^{T} w_{qt}^{2} + \sum_{t=1}^{T} w_{dt}^{2} - \sum_{t=1}^{T} w_{qt} w_{dt}} \qquad (6)$$

IV.2.3 The probabilistic models

The goal of probabilistic information retrieval models is to rank documents according to their relevance to a user's needs (Crestani et al. 1998). In order to achieve this goal, they require sound foundations based on formal probability and statistical theories and efficient ways to evaluate said relevance probabilities.

The basic justification of probabilistic models for information retrieval is the Probability Ranking Principle (Robertson 1977), according to which an information retrieval system will perform optimally when documents are sorted in order of decreasing probability of relevance to a user's needs, based on the available information. This is of course trivially true when perfect information is available, that is when yes/no relevance judgements are known for all documents in the collection: if the full collection is shown in the output, retrieving first the relevant documents and then the irrelevant ones is obviously the best possible way to present the results. In practice, the available information about the documents and the user's needs will always be incomplete. Expressing the user's need by a specific query and matching this query not with the documents but with metadata representations is just one cause of this incompleteness (Fuhr 1992), but certainly not the only one. Nevertheless, one can demonstrate without perfect knowledge the Probability Ranking Principle still holds for arbitrary documents. For example, let us assign a cost C_r to the retrieval of a relevant document and a higher cost C_i to the retrieval of an irrelevant document (implying some kind of penalty when making the wrong decision). If we write the probability of document d with respect to query q as $P(R/q,d)$, we can express the rule for deciding to retrieve one specific document d_k at a given point in the enumeration of the collection as:

$$C_r P(R/q,d_k) + C_i(1-P(R/q,d_k)) \leq C_r P(R/q,d_j) + C_i(1-P(R/q,d_j)) \qquad (7)$$

for any document d_j that has not yet been retrieved, so as to minimize the expected cost. This is equivalent to:

$$(C_r - C_i) P(R/q,d_k) \leq (C_r - C_i) P(R/q,d_j) \qquad (8)$$

And since $C_r < C_i$ the rule becomes:

$$P(R/q,d_k) \geq P(R/q,d_j) \qquad (9)$$

The challenge in probabilistic information retrieval models is to estimate these probabilities with sufficient accuracy, and several increasingly sophisticated approaches have been proposed in the past thirty years. Reviewing them all has been the subject of various articles (two of which have been cited earlier), and is beyond the scope of this report. We will only briefly present what is perhaps the most simple probabilistic model, and show how some formulas used by other information retrieval models can arise naturally from the probabilistic foundations (Manning et al. 2008, pp.204-209).

With no loss of generality, we can replace the probability of relevance by the odds of relevance and rearrange probabilities using Bayes' theorem $P(a/b) = P(b/a)\,P(a)/P(b)$:

$$O(R/q,d)=\frac{P(R/q,d)}{P(\bar{R}/q,d)}=\frac{P(R/q)}{P(\bar{R}/q)}\frac{P(d/R,q)}{P(d/\bar{R},q)} \tag{10}$$

The first fraction is the odds of relevance for a given query and does not depend on the document. For pure ranking purposes, we can then neglect it. We now represent the document by a vector of binary numbers $d_t = \{0;\ 1\}$ indicating the presence or the absence of term t in document d. The Binary Independence Model rests on the assumption that the terms contribute independently to the probability of relevance:

$$\frac{P(d/R,q)}{P(d/\bar{R},q)}=\prod\frac{P(d_t/R,q)}{P(d_t/\bar{R},q)}=\prod\frac{P(d_t=1/R,q)}{P(d_t=1/\bar{R},q)}\prod\frac{P(d_t=0/R,q)}{P(d_t=0/\bar{R},q)} \tag{11}$$

Let us define $p_t = P(d_t=1/R,q)$ and $u_t = P(d_t=1/\bar{R},q)$, the probabilities of finding t in a relevant and in and irrelevant document, respectively. At this point, any value of t is allowed, so the expression remains somewhat inconvenient. We can simplify it greatly if we assume that terms not found in the query are as likely in relevant as in irrelevant documents. With this assumption, the product we need to evaluate becomes:

$$\prod_{t\in q\cap d}\frac{p_t}{u_t}\prod_{t\in q-d}\frac{1-p_t}{1-u_t} \tag{12}$$

where the first product runs over terms found in both the query and the document and the second one over terms found in the query but not in the document. We can make the second part document-independent, and thus constant for a given query and negligible for ranking purposes, by simultaneously multiplying and dividing by $(1 - p_t)/(1 - u_t)$ for all terms found in both the query and the document, so that for all of these we need to evaluate the following:

$$\frac{p_t}{1-p_t}\frac{1-u_t}{u_t} \tag{13}$$

By taking the logarithm, we turn our product into a sum over all query terms and the ranking function is finally

$$f=\sum \log\left(\frac{p_t}{1-p_t}\frac{1-u_t}{u_t}\right)=\sum \log\left(\frac{p_t}{1-p_t}\right)+\log\left(\frac{1-u_t}{u_t}\right) \tag{14}$$

Let us suppose that the number of relevant documents will be small with respect to the size of the collection. The irrelevant documents will then have essentially the same statistical properties as the whole collection and u_t will be approximately equal to the document frequency of term t, divided by the total number of documents, N. Hence, if the term is not too frequent in the collection, we find:

$$\log\left(\frac{1-u_t}{u_t}\right)\approx\log\left(\frac{N-df_t}{df_t}\right)\approx\log\left(\frac{N}{df_t}\right) \tag{15}$$

which justifies the use of the normalized inverse document frequency that we have already seen. We will not go further into the details.

In this project, we have implemented one probabilistic ranking function, based on the Okapi BM25 weighting scheme (Sparck Jones et al. 2000a) (Sparck Jones et al. 2000b). It uses a parametric formula taking into account the frequency of the query terms in both the query (tf_q) and the document (tf_d), as well as the document length L_d: (Manning et al. 2008, p.214)

$$BM25=\sum_t \log\left(\frac{N}{df_t}\right)\frac{(k_1+1)\,tf_d}{k_1\left((1-b)+b\frac{L_d}{L_{av}}\right)+tf_d}\,\frac{(k_3+1)\,tf_q}{k_3+tf_q} \tag{16}$$

L_{av} is the average document length in the collection and b a scaling parameter. The k_1 and k_3 parameter can be adjusted to optimize the performance of the system for a given collection, by tuning the importance of the query and document term frequencies in the evaluation.

IV.2.4 Global similarity computation

The methods presented above describe ways to obtain similarities by comparing fields. In this project we want to compare records. There are of course an infinity of possibilities to combine the similarities resulting from individual fields comparisons into a global value. We implemented several methods in our program, which we will present later on.

IV.3 Analysis of similarity output

IV.3.1 Sorting

A simple way to analyse similarity output is to sort the result in decreasing similarity order. In a deduplication context, the most interesting candidates will then be found at the top of the output.

IV.3.2 Clustering

In the case of important sets of results with high and significant similarities the mere sorting may not be sufficient. For example, a high number of similar records might be better presented as groups rather than as a long list of uncorrelated pairs. This is where clustering becomes useful. This method also allows to represent graphically browsable results. Clustering consists in the fragmentation of a set of documents into subsets of similar documents called clusters. There are two types of clustering methods: a hierarchical approach and a heuristic approach, the later being faster but often of lesser quality than the former (Manning et al. 2008, pp.321-368). This project is based on the Python programming language and both kinds of clustering tools are already available in that context.

IV.4 The Python programming language

Python is an universal and efficient high level programming language. Universal, because it runs on most operating systems and offers a broad range of modules covering extensively all fields of computer sciences. Efficient, as well in its unique syntax as in its execution speed.

Python has already found important applications in information retrieval. For example, Google uses it extensively. In fact Guido van Rossum, the creator of Python is currently employed by that Mountain View company[8].

Due to its specificities, it is ideal in the context of this project. The following language capabilities retained our attention in the perspective of realising this project. Many of the following informations were found in (Lutz 2008, pp.3-20) and in (GNU Linux Magazine 2009) :

Portability: Python runs on and is generally included by default in major operating systems (Linux, SOLARIS / OpenSOLARIS, FreeBSD, OpenBSD, Darwin / MacOSX, Silicon Graphics IRIX, IBM AIX) and it is easy to install on Microsoft Windows. It is also available for many other systems like QNX, OS/2, BeOS, AROS, Windows CE, Symbian, Playstation, etc.[9]

Syntax: This work involved a lot of prototyping and Python excels at that because its syntax is very expressive and synthetic, almost to the level of pseudo code. Furthermore, since indentation is used to define logical blocks, a very clean style of programming is imposed, which is ideal for the developers collaboration. Among Python's useful syntactic features, we can mention *dictionaries* (sometimes known as *associative arrays* in other programming languages). Python dictionaries

---

[8] See: http://en.wikipedia.org/wiki/Guido_van_Rossum

[9] http://www.python.org/download/other/

make it very easy to define database-like structures where an object can be accessed by means of a non-numeric key. An obvious application for our project is the retrieval of the frequency and inverse document frequency associated with a given term. Another important feature is that except for reserved keywords, all program elements are computable objects whose members and methods are available to the programmer. This includes functions, which is instrumental in giving our program its flexibility.

Modularity: As mentioned previously, the nature of this project requires a great deal of modularity. Python is highly modular and makes it positively easy to create, import and use modules and packages. Many high-level functionality are included in the standard modules library[10] which ships with the interpretor, and are sufficient for the core of this similarity framework. The possibility to rely only on the standard library for this framework's core functionalities simplifies its deployment and distribution. In addition, Python's impressive external modules collections (see the Python Package Index[11]), is also an asset in the perspective of developing advanced applications on top of the framework. For example, the ability to pilot Open Office[12] is useful to extract text from most office files in order to work on full-text similarity. An other example is NeworkX, a powerful « package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks »[13]. This is of course appropriate for analysing the similarity output files. To further this analysis, plenty of scientific packages are available, for instance matplotlib[14] (a Matlab like tool) focusses on matrix manipulation. For purely statistical purposes, the rpy[15] package allows to use GNU R-project resources from Python.

Speed: The number of records comparisons required to analyse a collection increases dramatically with the records cardinality, and bibliographic collections tends to be big and become huger. Clearly, speed is a central concern in our context. Python may seem to be quite slow compared with a compiled language, even if it is a fast interpreted language. However, it allows many speed optimisations. Of course, the clear programming style helps the programmer to make high level code adjustments to yield speed optimisations. Further more, a formal optimisation can be conducted on the basis of Python profiling tools[16]. In addition, Psyco[17], an automatic speed enhancer (comparable with Just-in-Time compilers used for other interpreted languages) is available. Critical

---

[10] http://docs.python.org/3.1/modindex.html
[11] http://pypi.python.org/
[12] http://wiki.services.openoffice.org/wiki/Python
[13] http://networkx.lanl.gov/
[14] http://matplotlib.sourceforge.net/
[15] http://rpy.sourceforge.net/
[16] http://docs.python.org/library/profile.html
[17] http://psyco.sourceforge.net/]

code may also be written in C: the principal Python interpreter is written in C as well as many modules (standard library and external). Fortran might also be used thanks to pyfortran[18].

Existing bibliographic applications: Python is used in a number of bibliographic open source applications: CDS Invenio[19] (an Integrated Library System developed at CERN), PyBibtex[20] (an improvement over BibTeX), Bibus[21] (references mangagment software), PyBliographer[22] (bibliographic data manaement), Comix[23] and Calibre[24] (ebooks reading and management). The use of Python for the constitution of this framework might create synergies with some of theses projects in the future.

Multi-paradigms: object-oriented and functional programming are both fully supported. They can even be mixed within one script (it is not recommended). This project is almost entirely based on a functional programming approach (except for the XHTML and XML generation library used by several applications). Anyway, object oriented methods might become useful in the future, in particular to abstract interactions with SQL databases using an object relational mapper (ORM) like SQLAlchemy[25] (several ORMs are available in Python).

Web oriented : further developments of this project may lead to or become linked with existing web-based applications. For starters, web based GUIs would be useful because they are straightforward to implement, cross-platform and easy to access remotely. Secondly, libraries generally tend to strengthen their presence on the web. In that regard, Python is well equiped for web development. It is much faster than PHP (GNU Linux Magazine 2009), and offers modern and efficient web frameworks based on the view-model-controler (VMC) principles and using object relational mapping (ORM),. Two popular examples are Django[26] and Zope[27]. By the way, Python comes with built-in HTTP service functionalities, useful to build cross-platform Graphical User Interfaces (GUIs). Finally, Python works well with the popular Apache2 web server via mod_python[28] or mod_wsgi[29].

---

[18] http://sourceforge.net/projects/pyfortran/
[19] http://cdsware.cern.ch/invenio/
[20] http://pybtex.sourceforge.net/
[21] http://bibus-biblio.sourceforge.net/
[22] http://pybliographer.org/
[23] http://comix.sourceforge.net/
[24] http://calibre.kovidgoyal.net/
[25] http://www.sqlalchemy.org/
[26] http://www.djangoproject.com/
[27] http://www.zope.org/
[28] http://www.modpython.org/
[29] http://code.google.com/p/modwsgi/

V The MarcXimiL similarity framework

The name MarcXimiL was chosen to underline this tool's function (similarity analysis) and its native input format (MARCXML).

V.1 Framework licencing and availability

The MarcXimiL framework is placed under the free and open source GNU General Public Licence 3.0 (GPL 3.0)[30]. The framework's website is hosted by SourceForge, and its source code may be downloaded from there: http://marcximil.sourceforge.net

V.2 Framework portability

Currently, the following versions of Python are supported: 2.4.x, 2.5.x, 2.6.x, 3.0.x, 3.1.x. The framework's core has been tested on several operating systems with success: Linux (Ubuntu 9.04 , SuSE 10.0), Solais (OpenSolaris 2009.06 [OS 5.10]), MacOSX (10.5.7), Windows (XP-SP2, Seven). On all these systems MarcXimiL works out of the box and with these OS's default configuration, except on Windows in witch it is necessary to install Python.

This broad portability was possible because no other requirement than a basic Python installation is mandatory for the framework's core. MarcXimiL should also run on many other platforms, whenever a supported version of Python has been ported on them. MarcXimiL was mostly developed on Ubuntu 9.04 using both Python 2.6.2 and 3.0.1.

V.3 Framework installation

V.3.1 Basic installation

Unpack the archive......done. All the core functionalities will work. No further setup is required.

NB: Windows users should download and install Python first: http://www.python.org/download/

V.3.2 Additional tools

The demonstration tools build on top of the framework are not as portable as the core for two reasons:

1. These are all in early stage of development, often unstable, and only provided as an illustration of the possible uses of MarcXimiL. Generally, their portability could be improved a lot.

---

[30] http://www.gnu.org/licenses/gpl.html

2. Some of these tools use modules that do not exist for all Python versions and/or platforms.
3. Python 3.x compatibility was privileged when possible in perspective of the future.
4. POSIX compliant systems were privileged over Windows.

The following table sums up the compatibility of the different tools coming with this framework:

Table 1: MarcXimiL compatibility

Component	Description	Supported OS	Suported Python
Core – similarity.py	Similarity analysis	All:Linux, Mac, UNIX, Windows	All: >= 2.4.x and 3.x
Core – batch.py	Perform batch similarity analyses	Same as above	Same as above
Core – sort.py	Sort and truncates the outputs	Same as above	Same as above
Core – colldescr.py	Performs a statistical analysis of a collection	Same as above	Same as above
Core – oai.py	OAI-PMH2 metadata harvesting	Same as above	Same as above
Core – text2xmlmarc.py	Converts VTLS Virtua text MARC to MARCXML.	Same as above	Same as above
enrich.py (prototype)	« More litke this » calalog enhancement	Same as above	Same as above
monitor.py (prototype)	Infomation monitoring - using Invenio. (downloading, analysing, presenting)	Linux	3.x
plagiarism.py (prototype)	Managing knowledge base and detection.	Linux	>= 2.4.x (not 3.x) Requiered: UNO, OpenOffice, xpdf
visualize.py (prototype)	2D Graph generaion (output analysis)	Linux	>= 2.4.x (not 3.x), NetworkX
semantic.py (prototype)	MARCXML semantic relations editor to create graphs with visualize.py	Linux	3.x

All these tools will work on Ubuntu 9.04. To install the required dependencies in one go just type in a shell (the right version of Python is selected automatically at execution):

```
sudo apt-get install xpdf python-uno python-networkx python-matplotlib python3
```

On Windows, it is also possible to install several version of Python simultaneously and select the suitable version for each tool. This may be done by editing the .bat files and indicating within them the path of the desired interpreter (these files may be found in ./bin/windows.zip).

V.4 Framework structure

V.4.1 An introduction to this application's core structure

The programs structure reflects the determination to achieve three main goals:

1. portability made easy (automatic adaptation to the OS and Python version)

2. simplicity of installation (installation is limited to unpacking the archive)
3. modularity (flexibility at all levels, whenever it is reasonably possible)

Portability was achieved by coding in a way that is supported as well in Python 2 and 3. Because Python 3 differs a bit from Python 2, a few version specific pieces of code had to be produced, and these are selected automatically as the Python version is detected at runtime. Older versions than 2.4 are not supported, because of the lack of built-in set functionalities that are vital for this application.

The broad OS support is quite straight-forward in Python: complying with POSIX norms does the trick for most systems. It was however necessary to write specific equivalents for Windows that are automatically triggered if this OS is detected.

The simplicity of installation was obtained by using relative paths. Top level scripts detect the current location and then modules are found and loaded on that basis.

The modularity requirement was reached through the fragmentation of the framework in Python modules. Measures were taken in order to end up with thematic modules, that is to say, each level of flexibility is represented by a particular module containing a range of functions that perform similar tasks and that the end user may select according to their needs. All these options can be set up in the main configuration file.

V.4.2 Whithin the file system

This application is organized in a standard UNIX-like structure in the file system:

• **./bin/**	top level scripts that the user may call directly
• **./doc/**	documentation
• **./etc/**	configuration files
• **./lib/**	libraries, consisting of simple Python modules
• **./tmp/**	temporary files, like the ones produced by OAI-PMH2 harvesting
• **./var/**	data
• **./var/log/**	log files and results output
• **./var/rec_cache/**	fragments of loaded collection(s) dumped on disk to limit RAM usage

The most important files are:

• ./bin/**similariy.py**	the main toplevel program
• ./etc/**similarity_config.py**	the main configuration file

Most of the code may be found in the modules stored in the **./lib/** directory. The flexibility of this application is reflected by a number of thematic modules: as mentioned before, most modules contain related functions among which users may choose the most adapted to their needs.

• **load_records.py**	functions used to load MARCXML records (called by the top level similarity.py script)
• **parse_records.py**	functions importing data from MARCXML records (called by load_records functions)
• **compare_records.py**	functions determining the order of records comparison within collection(s) (called by load_records functions)
• **global_similarity.py**	functions to integrate the similarity of all fields in one value (called by compare_records, through records_structure, defined in the main configuration file)
• **compare_fields.py**	functions used to compare fields (called by globlal_similartiy functions)
• **compare_fields_helpers.py**	algorithms needed by various functions in compare_fields (called by compare_fields functions)
• **output.py**	functions writing results and logs (called by compare_records functions)
• **globalvars.py**	global variables definitions, used by many modules

Other modules, that are not listed just above, are stored in that directory. These are not used by the framework's core but by applications built on top of it.

V.5 Framework execution phases

V.5.1 Loading records

The first important phase in a similarity analysis process will load one or more collections. To do this, the function records_load in the load_records module is called on each collection by the top level similarity.py script (N.B.: all modules are stored in ./lib). The XML collection is then loaded within a global variable. This make it accessible to the various modules that will work with it without having to duplicate it in memory, because it might be quite large.

V.5.2 Parsing fields

An efficient parsing on big collections excluded the use of a DOM parser. That is why the records_load function will call the micro_dom function that was written to speed up the parsing. Micro_dom parses the MARCXML and returns sequentially all records in the form of XML strings. These much smaller elements are then analysed by the Python minidom parser: fields selected in the configuration file will be looked up. Since there are three types of MARC fields, three different functions will be applied: parse_controlfield, parse_nonrep and parse_multi. These functions are located in the parse_records module. A fourth function enables to concatenate fields. This permits to analyse related fields together such as a the title-subtitle pair.

V.5.3 Pre-processing of fields

Each extracted field is then submitted by the records_load function to the represent_field function that has been selected for that field in the main configuration file. Depending on what function was chosen, the field will either be returned untouched (if a RAW similarity function family was associated to the field, this will be explained more fully below), or trigger a pre-processing (if a two-pass similarity function is involved, like the ones in the WC, SHINGLES, or INITIALS families). This stage was introduced there for speed optimization reasons: by running the pre-processing within the loading phase, one additional pass through the records is avoided later. The pre-processing will generally involve a text normalisation (diacritics, punctuation, and so on are normalized by the functions of the compare_field_helpers.py module). On top of that, specific procedures related to the comparison function associated with a field will be executed.

V.5.4 The caching subsystem

The loaded, parsed and sometimes pre-processed records are progressively stored into the caching subsystem. The goal of this device is to limit memory usage in the case of very large collections. The general principle is that a fixed number (defined in the main configuration file) of records are stored in RAM for each loaded collection. The rest is dumped to the hard drive in the ./var/rec_cache directory. To do so, each collection is divided into segments of the same size that are serialized on the hard drive using a powerful built-in Python objects representation format called Pickle. This is managed directly by the records_load function. Individual hard disk caches are created for each loaded collection.

At comparison time, when a record is needed, it may be called up by the get_cached_record function, located in the compare_records module. This function will directly return a record if it is stored in the RAM cache (that takes the form of a global dictionary named globalvars.reccache). Otherwise, the segment containing the record is automatically loaded into the RAM cache and then the record is returned. If the RAM cache is already full, the oldest records it contains are eliminated prior to storing the new ones. In other words, the RAM cache follows simply the first in first out

(FIFO) queue principle. The order or arrivals of record in the RAM cache is stored in globalvars.rec_cache_queue.

V.5.5 Managing comparisons

After loading the records, the records_load function comes to an end, and the program flows back to the similarity.py top level script. The function records_comp is then called up. This function is selected in the main configuration file. But these functions are actually defined in the compare_records module. They define the order and the way in which records are compared with each other. The three most important ones are:

1. **records_comp_single**: This function may be used if only one collection is loaded. It will execute a comparison of all records within that collection with each other. That is to say each pair of records is only compared once: for example if the record A has been compared with the record B, B will not be compared to A afterwards. In other words, the comparisons will follow a triangular matrix pattern. This is useful because the similarity computation functions built in this framework are generally symmetrical in the sense that the (A, B) and the (B, A) comparisons will produce the same result. For this reason this comparison pattern will save almost half the process duration. This records_comp function is the best option for de-duplication purposes applied to one collection.
2. **records_comp_2collections_caching** : This function may only be used if two collections are loaded. All the records of the first collection are compared to all the records of the second. It is useful for information monitoring. That is to say to use a collection of known interesting records to find similar ones in an unknown flow of records. This function may be applied to plagiarism detection as well: a collection representing parts (sentences or paragraphs) of the document to check for plagiarism can be compared with a collection of the parts of a set of documents that are plagiarized candidates.
3. **records_comp_multiple_caching**: This function can be used with any number of loaded collections. All possible comparisons will be made. The function will compare all records of each collections with the records of the other collections as well as each collection's records amongst themselves. For now, it is not put in practical use, but it is described here because of its general aspect.

In all cases the record_comp functions will call up the record_rules defined in the main configuration file on each selected pairs of records. The record_rules, points to a function of the global_similarity module and is used to control the execution of field comparisons within a record and compute global similarities between records pairs.

All records_comp functions share a common programming interface and they are defined as follows (simplified example) :

Code sample 1: Example records_comp function

```
def records_comp_2collections( record_type, apply_rules, output_function ):
    from similarity_config import INPUT_FILES
    # writing header to output
    dummy_result=record_type.copy()
    dummy_result['similarity']=0.0
    output_function(dummy_result,0.0,'head')
    # executing comparisons
    collection_i = 0
    collection_j = 1
    for ri in range(globalvars.rec_cache_n[INPUT_FILES[collection_i]]):
        for rj in range(globalvars.rec_cache_n[INPUT_FILES[collection_j]]):
            output_function( apply_rules( record_type, \
                    get_cached_record(ri, INPUT_FILES[collection_i]), \
                    get_cached_record(rj, INPUT_FILES[collection_j]) ), \
                    globalvars.output_threshold )
    # writing footer to output
    output_function(dummy_result,0.0,'tail')
```

This type of function must be aware of 5 variables:

1. apply_rules: a pointer to a global similarity computation functions. These are described in the next sub-section. This pointer is set up in the main configuration file.
2. record_type: this information must be passed down to the 'apply_rules' function
3. output_function: a pointer to the selected output function. This pointer is set up in the main configuration file. The output is written within the execution of the record_comp function.
4. INPUT_FILES: so that the program knows which collections were loaded.
5. globalvars.reccache: necessary to be able to access the cached records through the get_cached_record function.

V.5.6 Computing the global similarity

The global_similarity module functions are designed to compute a global similarity for each records pair on the basis of the similarities of the fields the records pairs are constituted of. These functions will also control the execution of field comparison within a record. This permits speed optimisations, for example by means of functions that will stop field comparisons as fast as possible if certain conditions are met. Strategies have been developed that take advantage of this capability.

All global similarity functions share a common programming interface and they are defined as follows:

```
def global_similarity_function(record_structure, rec1, rec2):
```

The record structure parameter contains the definition of each field. It comes from the main configuration file. The rec1 and rec2 represent the records to be compared.

The fields of a record may be accessed using a `for` loop on the record_structure variable. A field comparison can be executed as follows (example for the 'doi' field defined in the main configuration file), assigning the result to an element of an `output` dictionary:

```
output['doi'] = execute_comp('doi', record_structure, rec1, rec2)
```

If one or both of the compared field are missing `None` (an empty Python variable) is returned.

The global similarity is simply called 'similarity', for example:

```
output['similarity'] = max( [ output['doi'], output['title'] ] )
```

A description of a selection of global similarity functions follows.

Maxsim

The similarity is computed for all selected fields, and the global similarity is simply the maximum similarity between the fields. When that function is used, the 'recids' field must be defined in the main configuration file and should represent some kind of digital identifiers. That field will only be used for output purposes.

As this function is the shortest of the global similarity function its full code is given here:

Code sample 2: Maxsim global similarity function

```
def maxsim(record_structure, rec1, rec2):
    output = {}
    output['recids'] = execute_comp('recids', record_structure, rec1, rec2)
    output['similarity'] = 0.0
    for field in record_structure:
       output_field = execute_comp(field, record_structure, rec1, rec2)
```

```
                          if    (output_field    is    not    None)    and    (not
type(output_field)==type('abc'))):
           if output_field > output['similarity']:
               output['similarity'] = output_field
    return output
```

The test `(not type(output_field)==type('abc')))` is necessary because some fields, like the record identifiers are simply concatenated with a separator so that a trace of what was compared is transmitted to the output. This concatenation is performed by the fields_concat__raw located in the compare_field module. With this test, we make sure that only numerical or `None` values are actually used for the global similarity value.

Means

In this range of global similarity functions, similarity is systematically computed for all selected fields and the global similarity returned is based on the mean similarity of all fields. Our program proposes three types of weighted means: the geometric mean (function: geometric_mean), the harmonic mean (function: harmonic_mean) and the arithmetic mean (function: arithmetic_mean). For each field, the weights must be set up in the main configuration file (this will be explained afterwards).

We note that although the records identifiers (recids) are not strictly mandatory for this strategy, they are practically required in order to make the program output understandable.

Breakout means

These three functions are almost the same as the above means, with field comparisons run in alphabetical order. However, if any field comparison yields a similarity under 0.8, the function will skip all subsequent field comparisons for the current record and replace them with the dummy value 1e-42. The three corresponding function names are harmonic_mean_breakout, arithmetic_mean_breakout and geometric_mean_breakout.

Boundaries

This strategy computes the global similarity using the weighted arithmetic average. There are however two variations with the weighted arithmetic mean function:

1. for each field, a 'threshold' parameter enables avoiding to take a field into account if it is not reached (this threshold is user defined in the similarity main configuration file)

2. the 'global-threshold' parameter, user defined in the same way, enables to return global similarity = 1e-10 if it is not reached

The 'recids' field is mandatory in the main configuration file and must be set up to accommodate record identifiers. That field will only be used for output purposes.

Boundaries_max

In this strategy, the global similarity is based on a maximum value, a bit like the maxsim function presented previously. There are however two differences:

1. for each field, a 'threshold' parameter enables avoiding to take a field into account if it is not reached (this threshold is user defined in the similarity main configuration file)
2. the 'global-threshold' parameter, user defined in the same way, enables to return global similarity = 1e-10 if it is not reached

The 'recids' field is mandatory in the main configuration file and must be setup to contain record identifiers. That field will only be used for output purposes.

Ubiquist

This function uses the following fields, that must be defined with these exact identifiers in the similarity configuration file:

- recids : record identifiers (only for output purposes)
- title_dice : title similarity
- authors : authors similarity
- year : years of publication similarity
- doi : digital object identifiers
- abstract : the abstract

The comparison of these fields are all executed (it is not well optimized for speed). Then a sequence of operations is started:

1. Base global similarity is set to a low value of 1e-10
2. If two identical digital object identifiers are found, the global similarity will be set to 1.0 and the treatment stops.
3. Otherwise, if the authors similarity is >= 0.85 and the year similarity is >=0.9 the global

similarity is set temporarily to 0.85 * the authors similarity. The idea here is that if a similar group of authors publishes in defined period of time (two years), the records are likely to be at least related. Or they might be near duplicates. The output is 0.85 * the authors similarity, in order not to give too much importance to the authors similarity. For example, if only one author is involved and that this person has a common surname too much false positives could be generated. The value of 0.85 was empirically because it seems to be a critical similarity value returned by other algorithms (Dice...), in the sense that pairs of fields that show an higher similarity than this are often real duplicates.

4. Then the abstract an the title similarity is computed. And the maximum is taken of the temporary global similarity, the title similarity and the abstract simlarity.

Abstract_fallback

The returned global similarity is simply the abstract similarity which is generally a very informative field. If at least one abstract is missing in a compared pair of record, this function will attempt to compute and return the title similarity as the global similarity.

This function uses the following fields, that must be defined with these exact identifiers in the similarity configuration file:

- recids: record identifiers (only for output purposes)
- title: title similarity
- abstract: abstract similarity

V.5.7 Comparison functions

All comparison functions share a common programming interface:

```
def functionname__ENDING(fieldtype, field1, field2, options = None)
```

First we note that the comparison functions are divided into several families depending on the required preprocessing. The function family is indicated by the ending of the function name, with two underline characters acting as a separator. The `fieldtype` argument is an identifier for the field or fields to compare, `field1` and `field2` are the field contents for both records in the current pair. The `options` argument, with an empty default value, is reserved for future use.

RAW family

In this function family, the field contents are directly compared pairwise, without any influence of the rest of the collection.

`years_comp__raw()`

comparison of publication years, the result value is set to 1 minus 0.1 times the difference between the two record dates, in years, provided that the first four characters of the compared fields can be interpreted as an integer number. The minimal returned value is 0.

`identifiers_comp__raw()`

standard string comparison, after removal of possible line breaks, dashes ('-'), and tabulations. This function is case-insensitive, and it will ignore the following prefixes: 'doi:', 'pmid:', 'issn:', 'isbn:', '-', 'oai:'.

`freq_comp__raw()`

comparison of the terms and term frequencies, taking only one pair of records into account. For each term found in both fields A and B, the score is raised by 1.0 minus the term frequency difference in A and B divided by the sum of the term frequencies. The term frequency for each term found only in A or B is then subtracted from the score.

`freq_comp_norm__raw()`

same as `freq_comp__raw()` except for an ad hoc normalization:

$$\mathit{freq\ comp\ normraw}(A,B)=\frac{1}{2}\left\{\frac{2\,\mathit{freq\ comp\ raw}(A,B)}{\mathit{freq\ comp\ raw}(A,A)+\mathit{freq\ comp\ raw}(B,B)}+1\right\}$$

`authors_comp__raw()`

set-theoretical comparison of two author lists (number of elements in the intersection of the sets, divided by the number of elements in the union of both => 1.0 if the sets are identical). Authors names are normalized prior to comparison: only the last name and the first initial of the first name is considered. If a comma is present, the words before it as considered to be last names, otherwise the last word of the field is taken as the last name.

`items_comp__raw()`

set-theoretical similarity calculation; this function returns the ratio of the number of elements (typically: words) common to fields A and B divided by the the number of distinct elements in A and B.

`levenshtein_comp__raw()`

fuzzy comparison based on the Levenshtein edit distance between two strings (Levenshtein 1966), normalized as a decaying exponential exp($-L/L_{par}$), where L_{par} is a field-specific configurable parameter related to the tolerance one assigns to the function. The Levenshtein distance calculation itself is performed using the Python implementation of Hetland (Hetland n.d.)

`fields_concat__raw()`

this function is not really a comparison, as it only produces the concatenation of the fields with a separator. However, it is useful to keep it in the RAW family of comparison functions as it makes it easy to display the field contents in the output while retaining the normal syntax of the matching patterns.

WC family

The functions of this family use a global, collection-wide dictionary of terms and term frequencies for the relevant fields, which has to be built before the record similarity calculations can start.

`ntfnidf_vectorcosine__wc()`

calculation of the vector-model ranking function (Salton et al. 1975), using the cosine distance definition (Salton 1991), normalized term frequencies (*ntf*) and normalized inverse document frequencies (*nidf*). Unlike in the normal use of the vector model, the product of the two vectors is performed over all terms of both fields. For our similarity calculations, there is no distinction between the “query” and the “document” so we treat them as equals.

`ntfnidf_vectorjaccard__wc()`

same as above, but using the Jaccard distance definition.

`ntfnidf_vectordice__wc()`

same as above, but using the Dice distance definition.

`okapibm25__wc()`

calculation of the Okapi BM25 probabilistic ranking function (Manning et al. 2008, p.214), modified to yield a result in the [0.0; 1.0]. For an ad hoc normalisation, we divide the standard function by $(k_1+1)*(k_3+1)*T_q$, where k_1 and k_3 are the adjustable BM25 parameters and T_q is the number of terms in the field content considered as the query. One has to note that the maximal value is in general not obtained by comparing one document with itself. Furthermore, the function is not invariant with respect to the permutation of its arguments: BM25(d1,d2) != BM25(d2,d1).

INITIALS family

In this family, the initial character of each term is extracted instead of the full field content.

`items_comp__initals()`

the INITIALS equivalent to `items_comp__raw.`

SHINGLES family

This family is very similar to the WC family. But instead of counting words it counts so-called *word bags*, that is to say all possible groups of *n* adjacent words. In this case *n* is set by default to 4, as suggested in the literature. This practice is usually applied in information retrieval to detect near duplicates. Its interest by contrast with the previously presented strategies is that it takes the order of words into account (Manning et al. 2008, p.403).

V.5.8 Writing output

As mentioned earlier, the result of the comparison of record pairs is finally transmitted by the records_comp function to the output function selected in similarity configuration file, and the records_comp function moves to the next pair of records, and so on. When the last pair of records has been compared, the program comes to an end. In addition, a threshold on the global similarity may be setup to reduce the output size: if the score is less than the threshold, the result is not written to the output.

Several output functions can be selected:

`report_quickndirty()`

This function is available mostly for testing purposes. It simply converts the pair similarity computation result (a Python dictionary) to a string. This string is written to the `results.log` file, followed by a new line character.

`report_tab()`

Each pair of compared records is stored on one line. The fields of the similarity computation results (including the global similarity, the individual scores for each field and record identifiers) are written in tabulation-separated columns, in the alphabetical order of the field names. The first line describes the columns content. In this form, the results are easy to read in a spreadsheet application, as in statistical environments like GNU R-Project, or manipulated with *NIX utilities like `sort`, `awk`, `split`, `grep`, `head`, `tail`, `wc`, etc.

`report_xml()`

The report_xml function saves the results in an ad-hoc xml format, for future machine to machine data exchanges: this XML can easily be converted into other XML formats, for example by means of an XSLT processor for example. The XML tree structure is simple, with a `<similarity_output>` root node containing a number of `<result>` nodes corresponding to all record-pair comparisons. Each `<result>` in turns contains `<field>` subnodes corresponding to each field of the record structure, plus the global similarity as above. The field name and similarity values are in turn contained in the `<name>` and `<data>` subnodes of a `<field>`.

This is how it should look in the configuration file:

```
    report = report_tab
    globalvars.output_threshold = 0.3 # use -1 to output everthing
```

By default, the output file is stored in ./var/log.

All output functions share the same interface, with 3 input arguments and no output:

`comp_result`: the Python dictionary result produced by the record comparison function.

`threshold`: the minimal global value for which the result will be written to the output file, as specified by the `globalvars.output_threshold` variable.

`param`: a character string with the default value 'body'. When the magical values 'head' or 'tail' are given (typically at the beginning and the end of the record comparison loop), the output function will produce the relevant header and ending for a particular file format (examples: the XML heading and root node opening for `report_XML` and the column titles for `report_tab`).

V.6 Program setup

All user definable options may be set up in the main configuration file ./etc/**similarity_config.py**. They deal essentially with the selection of collections to analyse, the structure of the records of interest, and the presentation of results.

V.6.1 Selection of collections

One or more MARCXML collection can be loaded at a time. The collection file must be located in the ./var/ directory. Note that Unicode UTF-8 character encoding is mandatory. On *NIX systems one can easily convert encodings using the `iconv` utility[31].

---

[31] http://www.gnu.org/software/libiconv/documentation/libiconv/iconv.1.html

A configuration file snippet follows. It will load a collection named « marcxml-collection.xml ». In addition, the order of records comparison within collection(s) is set up. In this case, all possible record pairs will be compared only once. This set up is reasonable for deduplication.

```
INPUT_FILES = ['marcxml-collection.xml']
records_comp = records_comp_single
```

Another configuration allows to load two collections, and then compare the records of the former collection to the records in the latter. This set up is useful for collection overlap studies, as well as for information monitoring, that is to say to use a collection of known records to find similar ones in an unknown flow of records. Another application is plagiarism detection: comparing a set of potentially plagiarized data with a set of original documents.

```
INPUT_FILES = ['collection1.xml', 'collection2.xml']
records_comp = records_comp_2collections_caching
```

A more general function enables to load as many collection collections as wanted and then compare each loaded record with all the others (within an between collections).

```
INPUT_FILES = ['c1.xml', 'c2.xml', 'c3.xml']
records_comp = records_comp_multiple_caching
```

V.6.2 Structure of records & similarity

A part of the configuration file is used to:

- select the fields to be analysed within each record,
- select their name in the output,
- select the parsing function used to extract each field,
- select the similarity function to be apply to each field (with optional parameters)
- select the function that will compute each record pair global similarity value

This is done like this:

Code sample 3: Example record structure definition

```
record_structure = { \
        '01recid'   :{'marc'      : '001',
                      'weight'    : 0,
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
        '02year'    :{'marc'      : '260  c',
                      'weight'    : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
        '03authors' :{'marc'      : ['100  a', '700 a'],
                      'weight'    : 3,
                      'parse-func': parse_multi,
                      'comp-func' : authors_comp__raw },
        '04title'   :{'marc'      : '245  a',
                      'weight'    : 3,
                      'parse-func': parse_nonrep,
                      'comp-func' : levenshtein_raw },
        '05title'   :{'marc'      : '245  a',
                      'weight'    : 3,
                      'parse-func': parse_nonrep,
                      'comp-func' :
ntfnidf_vectordice_comp__wc },
        '06abstract':{'marc'      : '520  a',
                      'weight'    : 3,
                      'parse-func': parse_nonrep,
                      'comp-func' :
ntfnidf_vectordice_comp__wc }}

                      record_rules = geometric_mean
```

The keys on the left, like '01recid', are used to label the results in the output.

The `marc` keys are used to select the fields to analyse. There are 3 types of MARC fields:

1. Controlfields, composed of only 3 digits ('001' for example), and they are parsed by the parse_controlfield function which is selected with the 'parse-func' key
2. Non-repeatable fields (i.e. fields that may appear only once according to the cataloguing convention), identified by 3 digits, two indicators and a subfield code, for example: '245 a' (in this case the two indicators are not used and represented by a space). This type of field is parsed using parse_nonrep as parsing function ('parse-func').
3. Repeatable fields, like authors ('100 a' and '700 a') are parsed by the parse_rep function. This function is capable of extracting several repeatable fields at once. Here, it will take the primary author (in '100 a') as well as the other authors (in '700 a').

The `comp-func` keys are used to select the similarity comparison functions.

Finally, record_rules enables to choose the global similarity function, the weighted geometric mean is used in this example. That is why each field as been given a 'weight' parameter. All similarity functions return values between 0.0 and 1.0, except if at least one of the two compared fields is non existent in a record. In that case, `None` is returned and the geometric mean function will ignore that field.

Note that the same field may be selected more than once. In the above example, different similarity algorithms are applied to the '245 a' field. This is useful, because when the global similarity is computed, some strategies might for example take the maximal value several occurrences. It makes sense for example if one similarity function is good at detecting misspellings (like the Levenshtein function) while the other is better at detecting permutations, additions and suppression of words (like the Dice function).

V.6.3 Caching options

In order to limit the memory usage while processing very large collections, one may configure the caching subsystem, which is activated by default. The caching subsystem fragments the loaded collection(s) and writes the fragments on the hard drive. It is possible to set up the number of records stored in RAM and the number of records composing the fragments. Twenty thousands records will typically take something between 500Mo and 1Go in RAM. But this can vary depending on the record structure and the average length of fields. For good performance, one should use the largest possible RAM cache..

In the configuration files, the parameters may by adjusted like so:

Code sample 4: Example record cache parameters

```
# activating cache
globalvars.caching = True
# segment length (in records) for hard disk storage
globalvars.rec_cache_segment_length = 300
# cache length (in records) stored in RAM
              globalvars.rec_cache_length = 30000
```

V.6.4 Other options

Similarity configuration is flexible, and quite a lot of functions are available. For a more accurate view of the possibilities take a look at the demo similarity configuration files, stored in ./etc/configuration-examples .

V.7 Additionnal tools

Some additional utilities were written to help working with this framework. They are all located in the ./bin/ folder.

V.7.1 oai.py

This tool will perform a basic OAI-PMH2 harvesting in order to download MARCXML collections from data providers. It can be set up via the ./etc/oai_config.py file, which offer the basic OAI-PMH2 harvesting options, except that it is for now limited to the MARCXML format (Dublin Core is not supported). The harvesting consists in downloading OAI-MARCXML fragments that are subsequently converted to simple MARCXML and reassembled into one single collection. More informations on the protocol may be found on the OAI-PMH2 official specification web page[32].

This is an example of oai.py configuration options in ./etc/oai_config.py:

Code sample 5: Example oai_config.py file for OAI-PMH2 record harvesting

```
# harvesting timing
oai_politeness_delay = 60 # delay between queries in seconds
oai_error_delay = 600 # if an error occurs, wait this amount
oai_error_retries = 5 # tolerated errors before aborting
# harvesing setup
oai_baseurl = 'http://cdsweb.cern.ch/oai2d/'
oai_metadataPrefix = 'marcxml' # do not change that
oai_from = '2008-10-01' # Optional: may be set to None
oai_until = '2008-10-08' # Optional: may be set to None
oai_set = None # Optional: may be set to None
```

At the end of the run, the harvested collection is transferred in the ./var directory, where all collections are stored. It is renamed according to the the date and time when the harvesting was started using the following pattern: YYYYMMDD-hhmmss.xml .

V.7.2 batch.py

This utility runs the MarcXimiL program in batch mode with a series of similarity configuration files.

SYNTAX: `./batch.py [<config-files-directory>]`

It will look for files named after the flowing pattern: similarity_config__* (note the double

---

[32] http://www.openarchives.org/OAI/openarchivesprotocol.html

underscore before the *) . Then, a directory in ./var/log/ named after the current date and time will be created to store the outputs., e.g.: ./var/log/20100324-122536

For each configuration file found, an output will be created and named after the configuration file in question using the * part of the pattern and adding .dat at the end of it. Furthermore, the duration of each process if measured and written in a file named after the same pattern *.dat.duration.

By default, this script will look for configuration files in ./etc/ . But if it is given an argument, it will consider it as a directory and look for the configuration files in ./etc/that_directory

V.7.3 sort.py

This script will sort a tabulated output file according to the global similarity, and then truncate it to 1000 lines of output plus the first line containing headers.

SYNTAX: `./sort.py [<directory-to-process>]`

The *.dat files in ./var/log will be used by default, in particular ./var/log/output.dat, which is the program default output file. Alternatively it is possible to specify a sub-directory located in ./var/log, for example 20101128-190150

WARNING: This script is not optimized and may take a lot of time and resources while processing long logs. To avoid this, use a statistical analysis tool like GNU R-Project or set up a reasonably high output threshold in the similarity configuration file.

V.7.4 colldescr.py

This program will perform a statistical analysis of a MARCXML collection. The displayed information may help understand the composition of the collection, and therefore to set up the MarcXimiL framework more efficiently.

The following informations will be returned to the standard output:

- the number of records in the collection
- the average number of fields by record
- for each subfield: the percentage of its presence throughout the collection, and its average length

SYNTAX: `colldescr.py <collection_name.xml>`

The collection will be looked-up in the ./var directory, where all collections are stored.

V.7.5 check.py

This script gives informations about the software packages installed on your system that may be used by MarcXimiL. If these packages are absent, the script will tell you in what way they could be useful and how to get them.

V.7.6 text2xmlmarc.py

This utility converts VTLS Virtua text MARC to MARCXML.

V.7.7 Additional libraries

Two additional libraries have been developed and included in MarcXimiL. They are not used by the framework's core applications, but they contain useful functions for building tools on top of it:

1. **toox.py** is a pure Python object oriented XHTML and XML generation library. It is used to generate web interfaces and could also be applied to generate XML outputs.
2. **marcxmlload.py** is a pure Python library enabling to read and load MARCXML collections in Python data structures.

V.7.8 Other tools

Other tools may be found in the ./bin folder. These are application prototypes build on top of this framework to illustrate its possibilities. They are in early stages of development and are quite unstable. A full description of these is found the “Future developments” section.

V.8 Distribution of the development tasks

As this project is subject to academic evaluation, the developers equitably shared the work load as described in Table 2.

Table 2: Work distribution

What	**AB**	**JK**	**Comment**
Requirements and general schema	X	X	Working together.
Initial prototype		X	First basic but functional structure, including loading and parsing functions, text normalisation, word and frequencies computations, identifiers concatenation, author and date comparison functions.
Two-pass processing schema and tests	X	X	Working together.
Two-pass processing implementaiton	X		WC and initials families comparison algorithms, including Jacquard, Dice, Salton, OKAPI.
Caching subsystem		X	RAM usage limitation
Levenshtein comparison	X		Experimental similarity computation

What	AB	JK	Comment
Global similarity : classic means		X	Weighted arithmetic, geometric and harmonic means.
Global similarity : breakout means	X		Based on means but intended to be faster
Global similarity : custom strategies		X	Ubiquist, abstract_fallack, boundaries*, etc.
Test strategies and collections	X	X	Double blind test
Results evaluation	X	X	Doubble blind test, and then working together
Results discussion and conclusions	X	X	Working together
Global similarity integrated strategies	X	X	Working together
Report redaction	X	X	Attibution of parts following original workload.
Related tools		X	Visualize.py (2D net), batch.py, sort.py, check.py, etc.
Generalisation of OS and Python versions (windows, python 3.x)	X	X	Both developpers did their best to make it happen, and JKs actions to support Windows, Python 3.x , Python 2.4.x and 2.5.x as well were quite light.

V.9 Tests

V.9.1 Methodology

The tests were performed using a double-blind methodology. Both developers independently extracted several sets of 1990 MARC records from selected sources, and duplicated 10 of them with various alterations for a total of 2000 records. In the same manner, both developers used the MarcXimiL framework to create strategies and record structures optimized for criteria such as speed, precision, etc. The test collections were communicated to the other author with minimal information (dataset origin, specific changes in the MARC field definitions). Each developer applied his own strategies to all available collections, and sent his results for up to 1000 top similarities back to the collection creator for precision evaluation. The processing time was measured by our own batch.py utility while running the program under minimal system load (only the default desktop environment and a terminal window).

The raw precision at fixed recall values was evaluated by counting the occurrences of the created near-duplicates among the top N similarity scores (N = 10, 20, 50). The remaining pairs were identified as raw noise. Corrected values were obtained by examining the raw noise and removing the observed duplicates that were already part of the collection.

V.9.2 Test collections

CERNa dataset

The datasets of CERN 'a' through 'd' collections are based on the same 1990 records. These were simply the last 1990 records added to the articles collection the 4th of August 2009 stored within the

CERN Document Server.[33]

In the CERNa dataset, the first 10 records of the 1990-record collection were duplicated and 1 field was modified in each duplicate records. The affected fields were 100__$a (first author), 700__$a (general author), 245__$a (main title), 260__$c (year) and 520__$a (abstract). The changes were as follows:

1187507: truncated title(1 word out of 6).

1187474: truncated title (3 words out of 12).

1187449: truncated title (3 words out of 7).

1187448: author first names were removed.

1187436: several authors were removed (6 out of 14).

1187432: the only author name was misspelled (one error in the last name, one in the first name).

1187427: two sentences in the abstract were moved inside the field.

1187408: two sentences in the abstract were removed.

1187403: the date was changed by one year.

1187394: the date was changed by two years.

CERNb dataset

The CERNb dataset was built with the same method as the CERNa dataset, except that two fields were modified in the duplicate records.

1187507: truncated title (1 word out of 6), several authors removed (2 out of 4).

1187474: truncated title (3 words out of 12), the abstract was removed.

1187449: truncated title (3 words out of 7), the date was changed by two years.

1187448: the authors' first names were removed and the date changed by three years.

1187436: several authors were removed (6 out of 14) and 3 sentences were moved inside the abstract.

1187432: the only author's name was misspelled, as well as the title (5 misspelled word out of 11). There is no date.

1187427: two sentences were moved inside the abstract field. The title was misspelled (4 misspelled

---

[33] http://cdsweb.cern.ch

words out of 12).

1187408: two sentences were removed from abstract and the date was changed by three years.

1187403: the date was changed by one year and the title misspelled (5 misspelled words out of 21).

1187394: the date was changed by two years and one authors was removed (1 out of 2).

CERNc dataset

The CERNc dataset was built with the same method as the CERNa dataset, except that three fields were modified in all duplicate records..

1187507: truncated title (1 word out of 6). Some authors were deleted (2 out of 4). One sentence was removed from the abstract.

1187474: truncated title (3 words out of 12). The abstract and 1 author (out of 2) were removed.

1187449: truncated title (3 words out of 7). The date was changed by two years. The abstract was removed.

1187448: the authors' first names were removed. The date was changed by three years and the title totally modified.

1187436: several authors were removed (6 out of 14). 3 sentences were moved inside the abstract. The title was misspelled (2 misspelled words out of 12).

1187432: the only author was misspelled, as well as the title misspelled (5 misspelled words out of 11). The date was changed by one year.

1187427: two sentences were moved inside the abstract. The title was misspelled (4 misspelled words out of 12) and the date changed by six years.

1187408: two sentences were deleted from the abstract. The date was changed by 2 years

1187403: The date was changed by one year. The title was misspelled (5 misspelled words out of 21) and one authors removed (1 out of 4).

1187394: The date was changed by two years. One author was removed (1 out of 2) and the title was truncated (2 words out of 13).

CERNd dataset

The CERNd dataset was built with the same method as the CERNa dataset, but with more serious alterations: All abstracts were removed, and half the dates were changed by 1 year. Fields 773 were removed (all subfields: document source, such as journal title for an article, etc.). In addition, titles as well as digital identifiers (035__$a) variations were made:

1187507: 1 word was misspelled in the title.

1187474: 2 words were misspelled in the title.

1187449: 2 words were misspelled in the title.

1187448: 3 words were misspelled in the title.

1187436: 4 words were misspelled in the title.

1187432: 1 word was misspelled in the title, the digital identifier was removed.

1187427: 2 words were misspelled in the title, the digital identifier was removed.

1187408: 2 words were misspelled in the title, the digital identifier was removed.

1187403: 3 words were misspelled in the title, the digital identifier was removed.

1187394: 4 words were misspelled in the title, the digital identifier was removed.

ETHZ1 dataset

The ETHZ1 was built by harvesting the ETH e-collection using the OAI-PMH protocol. Three records were extracted at fixed positions from the first 663 batches of 30 records exported by the ETHZ server, and one more was added from the 715th batch. Ten records were extracted at fixed positions (10 20 30 50 80 130 210 340 550 890) in the initial collection of 1990 records and modified with the aim of reproducing real-world errors or variant interpretations of the cataloguing rules. The altered fields were 245__$a (main title), 245__$b (subtitle), 260__$a (year), 700__$a (author), 900__$a (PhD advisor)

10.3929/ethz-a-000080552: the PhD advisors were added to the authors.

10.3929/ethz-a-000085328: one compound word was modified in the title (oxydo-réduction -> oxydoréduction)

10.3929/ethz-a-000085643: the German umlauts in the title were replaced by the vowel + e spelling.

10.3929/ethz-a-000086597: the authors' first names were replaced by initials.

10.3929/ethz-a-000087872: the German umlauts were replaced by the vowel + e spelling in the title, and the year was deleted.

10.3929/ethz-a-000089154: no change.

10.3929/ethz-a-000090875: an artificial umlaut was added to the author's name (Brunner -> Brünner)

10.3929/ethz-a-000096651: the subtitle was removed.

10.3929/ethz-a-000152441: a variant title 245__$i was changed into a subtitle 245__$b.

10.3929/ethz-a-000578009: the subtitle was merged into the main title.

ETHZ2 dataset

The ETHZ2 dataset was built from the same original data as for ETHZ1, but with different positions in the first 663 record batches of the OAI-PMH export.

10.3929/ethz-a-000081089: one Greek letter in the title was changed to its full name in Roman characters (α-Melanotropin -> alpha-Melanotropin).

10.3929/ethz-a-000085338: the author was deleted and replaced by one of the PhD advisors.

10.3929/ethz-a-000085653: square brackets were added at the beginning and end of the title, as used by cataloguers to indicate an artificial title attributed to the document.

10.3929/ethz-a-000087354: one German umlaut in the title was replaced by the base vowel.

10.3929/ethz-a-000087882: the author's middle name was removed.

10.3929/ethz-a-000089164: the first few words of the title "Contribution à l'" were removed

10.3929/ethz-a-000090885: the author's first name was replaced by an initial and the year was deleted.

10.3929/ethz-a-000096662: the author's name was changed from the standatd last *Name*, *First Name* order to *First Name Last Name* (without comma).

10.3929/ethz-a-000153723: two characters were exchanged in the title (Polyedern -> Polyedren).

10.3929/ethz-a-000578047: a TeX-notation subscript was changed in the title (N_2 → N-2) and an artificial umlaut was added to the author's name.

RERO1 dataset

The initial data for the RERO1 and RERO2 datasets were harvested from the RERO DOC repository using the OAI-PMH protocol. About 3300 records were harvested in batches of 500. The 1990 reference records in the RERO1 dataset were extracted from batches 0 to 2 (500 records each) and the first 490 records of batch 3. The duplicated records were selected at the same positions as for the ETHZ datasets.

7391: the journal title was changed to an abbreviated form. The *Van* particle of an author name was moved to imitate a frequent confusion with such name (*Gogh, Vicent van* versus *van Gogh, Vincent*).

7808: parentheses were removed in the title. For one author, the French compound first name was replaced by two isolated initials.

7830: accented letters in the names of 2 authors were replaced by the base vowel. One character was added at the end of the last word of the title.

7853: a comma was replaced by a colon in the title. The journal title was replaced by an abbreviated form.

7890: a punctuation error in the first author's name was corrected. The HTML tags included in the title (<i>, </i>*)* were deleted.

8017: an umlaut in the title was replaced by the base vowel. The HTML tags included in the title (<i>, </i>*)* were deleted.

8164: the authors' full names were replaced by a Pubmed-like form (*Last Name, Initial without punctuation*). A likely variant journal title was used, and one word was abbreviated in the main title.

8344: one author and the year were deleted.

8708: two author names were modified (one two-syllable Chinese first name given as one word was divided into two; *Mac Raighne* was replaced by *McRaighne*). The dashes in the title were deleted.

9280: an *e* was added to one title word. The date given in the YYYY-MM-DD format was replaced by the year YYYY.

RERO2 dataset

The reference records of the RERO2 dataset were obtained from the RERO DOC OAI-PMH export by combining batches 2 to 4 and the first 490 records of batch 5. Since the total size of the RERO DOC collection, some overlap had to be accepted in the reference records. However, the fixed positions of the duplicated records within the reference set ensure their uniqueness.

4121: *ñ* was changed to *n* in one author's name. Part of the main title was turned into a subtitle.

4140: <i> HTML tags were inserted into the title. The order of the authors was modified, and an umlaut was removed from one of them.

4151: in the main title, *pseudospectral* was changed to *pseudo spectral.*

4205: a patronymic name was added to an author with a Russian name. An article was added to the main title and the journal title was changed to an abbreviated form.

4326: the complete title was replaced by a space.

4817: both one-syllable first names of a Chinese author were merged into a single word. A typo was inserted in one title word.

5020: two author names were modified by removing diacritics (1 umlaut and 1 cedilla). Two modifications on highly specialized terms were applied to the title (*pre-Variscan* -> *prevariscan*; *north-Gondwanan* → *north Gondwanan*).

5382: for all authors with an abbreviated compound first name, the initials were replaced by two artificial full names (*J.-C.* → *Jean Claude*). One compound word in the title was split (*groundwater* → *ground water*).

5800: full first names were replaced by initials. TeX-notation indices in the title were replaced by regular numbers.

6388: the punctuation of the title was modified and a dash was added. The year was deleted. A typo was inserted in the 2nd author's name.

Test collections statistical properties

Statistical properties of the test collections were gathered automatically by the colldescr.py script (Table 3).

Table 3: Field statistics for the test collections. Percentage of presence of the subfield in the collection (P), average length of the field in characters (L).

		001	**0247 a**	**035 a**	**100 a**	**245 a**	**260 c**	**269 c**	**520 a**	**700 a**	**773 c**	**773 g**	**773 t**	**9001 a**
CERNa	**P**	100	-	-	98	100	99	35	35	416	-	-	-	-
	L	7	-	-	12	64	4	10	728	10	-	-	-	-
CERNb	**P**	100	-	-	98	100	99	35	35	415	-	-	-	-
	L	7	-	-	12	64	4	10	728	10	-	-	-	-
CERNc	**P**	100	-	-	98	100	99	35	34	415	-	-	-	-
	L	7	-	-	12	64	4	10	728	10	-	-	-	-
CERNd	**P**	100	-	88	98	100	99	35	-	415	-	-	-	-
	L	7	-	13	12	64	4	10	-	10	-	-	-	-
ETHZ1	**P**	1	100	0	1	100	99	-	0	109	-	-	-	168
	L	9	24	15	16	74	4	-	17	16	-	-	-	16
ETHZ2	**P**	1	100	0	0	100	99	-	0	111	-	-	-	168
	L	9	24	14	17	74	4	-	17	16	-	-	-	16
RERO1	**P**	100	-	0	99	100	-	99	92	317	8	99	100	-
	L	4	-	10	16	93	-	6	1097	15	47	16	33	-
RERO2	**P**	100	-	0	99	100	-	97	78	298	13	100	100	-
	L	4	-	9	16	92	-	5	1059	14	41	17	35	-

V.9.3 Test strategies

In the double blind test, each author developed several deduplication strategies. Concretely, a strategy takes the form of a MarcXimiL configuration file (with small variations for each dataset since homologous fields are sometimes tagged with different MARC fields by institutions). In this section, these strategies have been put together and are described in details.

2geom, 2geombreak and okapigeom strategies

The 2 geom and 2geombreak strategies use the geometric and geometric_break methods with the same record structure, with the aim of examining the effect of the breakout option on speed and precision. The field labels are therefore given names whose alphabetical order reflect their supposed importance for duplicate detection. The record identifiers are examined first and directly passed to the output (no comparison is performed), followed by the year, document title (including a possible secondary title), authors (including thesis supervisors) and source information. Considering the significant differences in cataloguing practices between the RERO and ETHZ collections, several fields (identifier, year) are included in two different instances so that they are properly taken into account. In general, all fields have been given the same weight in the geometric mean, but the title was assumed to be more important and its weight was doubled.

The title similarity is computed the vector model and Dice's similarity measure. The author list is analysed in two steps, first by considering only the initials and then the full last names, the hypothesis being that the first comparison might be faster but less precise. Thus, in the breakout strategy, it would make sense to perform the quick comparison first so that the second one can be avoided most of the time. Finally, the source information (MARC 773), crucially important for articles, is analysed first by comparing the initials of the source title (which may be found in full form or abbreviated form depending on the cataloguing rules and their more or less serious enforcement), then by computing the Levenshtein distance between the fields that contain the page information. We note that both these comparisons are unreliable when dealing with poorly documented collections. Cataloguing standards tend to vary quite a lot for this MARC field.

Code sample 6: Record structure for the 2geom and 2geombreak strategies

```
record_structure = { \
    '01recid'        : {'marc'      : '001',
                      'weight'    : 0,
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
    '01recid2'       : {'marc'      : ['0247 a'],
                      'weight'    : 0,
                      'parse-func': parse_concat,
                      'comp-func' : fields_concat__raw },
    '02year'         : {'marc'      : '260  c',
                      'weight'    : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
    '03year2'         : {'marc'      : '269  c',
                      'weight'    : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
    '04title'        : {'marc'      : ['245  a', '245  b'],
                        'weight'    : 2,
                        'parse-func': parse_concat,
                        'comp-func' : ntfnidf_vectordice_comp__wc },
    '05authors1'      : {'marc'      : ['100  a', '700  a', '9001 a'],
                         'weight'    : 1,
                         'parse-func': parse_concat,
                         'comp-func' : items_comp__initials },
    '05authors2'      : {'marc'      : ['100  a', '700  a', '9001 a'],
                         'weight'    : 1,
                         'parse-func': parse_multi,
                         'comp-func' : authors_comp__raw },
    '06source1'      : {'marc'      : ['773  t', '7112 a'],
                       'weight'    : 1,
                       'parse-func': parse_concat,
                       'comp-func' : items_comp__initials },
    '07sourceinfo'   : {'marc'      : ['773  g', '773  c'],
                       'weight'    : 1,
                       'parse-func': parse_concat,
                       'comp-func' : levenshtein_comp__raw } }
```

The okapigeom strategy substitutes the probabilistic Okapi BM25 formula to the vector-model Dice similarity measure for the computation of the title similarity.

Initials and initialsbreak strategies

The goal of the initials strategy is to perform a very fast comparison while still considering all important fields for the duplicate detection. For this, the record structure used with the 2geom strategy was modified at the title and author levels: for both fields, a simple set-theoretical comparison of initials is performed.

Code sample 7: Record structure for the initials and initialsbreak strategies

```
record_structure = { \
    '01recid'        : {'marc'      : '001',
                      'weight'    : 0,
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
    '01recid2'       : {'marc'      : ['0247 a'],
                      'weight'    : 0,
                      'parse-func': parse_concat,
                      'comp-func' : fields_concat__raw },
    '02year'         : {'marc'      : '260  c',
                      'weight'    : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
    '03year2'         : {'marc'      : '269  c',
                      'weight'    : 1,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
    '04title'        : {'marc'      : ['245  a', '245  b'],
                        'weight'    : 2,
                        'parse-func': parse_concat,
                        'comp-func' : items_comp__initials },
    '05authors1'      : {'marc'      : ['100  a', '700  a', '9001 a'],
                         'weight'    : 1,
                         'parse-func': parse_concat,
                         'comp-func' : items_comp__initials },
    '06source1'      : {'marc'      : ['773  t', '7112 a'],
                       'weight'    : 1,
                       'parse-func': parse_concat,
                       'comp-func' : items_comp__initials },
    '07sourceinfo'   : {'marc'      : ['773  g', '773  c'],
                       'weight'    : 1,
                       'parse-func': parse_concat,
                      'comp-func' : levenshtein_comp__raw } }
```

Again, a breakout variant was introduced, yielding the initials_breakout strategy.

Abstract Fallback strategy

This strategy is based on the principle that the abstract field (520$a) is generally longer and therefore contains more information (see Table 3). Because this field is long, the Dice-based vector similarity algorithm was used. For instance, the Levenshtein similarity algorithm is much to slow for that purpose. The Dice algorithm was privileged over Salton's cosine and Jaccard functions because preliminary tests showed that it was a touch more efficient in the similarity value obtained.

Unfortunately, the abstract field is not always present, even in article collections. In this case, the strategy falls back on the titles when it happens, because this field is almost always present.

To implement this strategy, the abstract_fallback global similarity function that we described previously was written.

In the configuration file, the key points of this strategy are defined as follows:

Code sample 8: Program configuration for the abstract_fallback strategy

```
    INPUT_FILES = ['collection.xml']
    records_comp = records_comp_single
    report = report_tab
    globalvars.output_threshold = 0.3
    record_structure = { \
        'recids'   : {'marc'      : '001',
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
        'title'    : {'marc'      : '245  a',
                      'parse-func': parse_nonrep,
                      'comp-func' : ntfnidf_vectordice_comp__wc},
        'abstract' : {'marc'      : '520  a',
                      'parse-func': parse_nonrep,
                      'comp-func' : ntfnidf_vectordice_comp__wc}}
    record_rules = abstract_fallback
```

Boundaries_max strategy

This strategy is based on the principle that, it often is interesting to take fields into account only if they meet certain conditions. Therefore, it allows the end user to specify two optional parameters for each field:

1. 'threshold' : if this threshold is not attained for that fields similarity, that value will not be used in the global similarity computation
2. 'global-threshold' : if this threshold is not reached, a minimal global similarity for the record pair will be returned. The treatment stops at once.

To implement this strategy, the boundaries_max global similarity function that we described earlier was implemented.

A generic strategy was devised on that basis:

- The 'year' field is used to impose a low global record similarity between two record if they were published more than five years apart. This is done to avoid useless computations. In deed, publications that are more than 5 years apart are unlikely to be near duplicates, even a preprint and a corresponding article that would be published a long time afterwards (NB: This is not valid in some academic domains, like economy). Practically, this is done by setting a 'global-threshold' parameter of 0.5, on a year_comp__raw similarity output. NB: this last function calculates the difference in years between the fields and returns 1 – 0.1*(difference in years) as field similarity.

- The 'title' and 'abstract' fields are then analysed with a Dice based vectorial similarity field function. If either of these analyses returns a value lower than 0.5, the corresponding field is ignored. This way, because the global output threshold is set to 0.4, the similarity output file will generally be of acceptable length.
- Then the maximal field similarity is returns as global similarity, or a minimal value of 1e-10 is return if not applicable.

Code sample 9: Program configuration for the boundaries_max strategy

```
  INPUT_FILES = ['collection.xml']
  report = report_tab
  globalvars.output_threshold = 0.4
  record_structure = { \
        'recids'  : { 'marc'      : '001',
                      'parse-func': parse_controlfield,
                      'comp-func' : fields_concat__raw },
        'year'   :   {'marc'      : '260  c',
                      'global-threshold' : 0.5,
                      'parse-func': parse_nonrep,
                      'comp-func' : years_comp__raw },
        'title'  :   {'marc'      : '245  a',
                      'threshold' : 0.5,
                      'parse-func': parse_nonrep,
                      'comp-func' :
ntfnidf_vectordice_comp__wc },
        'abstract':  {'marc'      : '520  a',
                      'threshold' : 0.5,
                      'parse-func': parse_nonrep,
                      'comp-func' : ntfnidf_vectordice_comp__wc}}
    record_rules = boundaries_max
```

NB: An other strategy called Boundaries is available. It is based on the same principles than Boundaries Max, however the global similarity returned is a weighted average of the fields similarities that meet their thresholds. This second Boundaries strategy was not studied in this report, because preliminary tests showed that it is generally less effective than Boundaries Max.

Geometric_jk strategy

This strategy is based on a classic weighted average. The geometric mean was used since preliminary tests showed that it yielded a better recall in the first results than the arithmetic and harmonic versions.

A relatively small selection of fields was selected for the mean: year, authors, title, abstract. This fields were chosen, since they are the most important bibliographic fields that are present in almost all types of records (articles, preprints, books, periodicals). The abstract field is more frequent in preprints and articles than in other types of records, but it was included nonetheless because it was considered as is informative input when it is there.

In preliminary tests on a limited amount of records, several weighting strategies were tested. In most cases the weighting does not seem to do much difference. However, it seemed that attributing less importance to the year was slightly more effective.

Concretely, this strategy is defined as follows:

Code sample 10: Program configuration for the geometric_jk strategy

```
    INPUT_FILES = ['collection.xml']
    records_comp = records_comp_single
    report = report_tab
    globalvars.output_threshold = 0.3
    record_structure = { \
        '01recid'        : {'marc'      : '001',
                            'weight'    : 0,
                            'parse-func': parse_controlfield,
                            'comp-func' : fields_concat__raw },
        '02year'         : {'marc'      : '260  c',
                            'weight'    : 1,
                            'parse-func': parse_nonrep,
                            'comp-func' : years_comp__raw },
        '03authors'      : {'marc'      : ['100  a', '700 a'],
                            'weight'    : 3,
                            'parse-func': parse_multi,
                            'comp-func' : authors_comp__raw },
        '05title'        : {'marc'      : '245  a',
                            'weight'    : 3,
                            'parse-func': parse_nonrep,
                            'comp-func' : \
                                  ntfnidf_vectordice_comp__wc },
        '05abstract'     : {'marc'      : '520  a',
                            'weight'    : 3,
                            'parse-func': parse_nonrep,
                            'comp-func' : \
                                 ntfnidf_vectordice_comp__wc } }
    record_rules = geometric_mean
```

Maxsim strategy

This strategy is a simplified boundaries_max strategy: it will return the maximum field similarity as global similarity for a record (no thresholds can be defined). But it was set up in a way that was not exploited in boundaries_max.

The title similarity is computed twice using two different algorithms. The first is the Dice vector similarity function, and the second is based on the Levenshtein edit distance. While the former is good at detecting words permutations, deletions and additions, the latter is good at detecting misspellings. Because the maximal field similarity is then computed, the best characteristics of each algorithm are reflected in the global similarity depending on the compared record pair.

This strategy is defined as follows:

Code sample 11: Program configuration for the maxsim strategy

```
  INPUT_FILES = ['collection.xml']
  records_comp = records_comp_single
  report = report_tab
  globalvars.output_threshold = 0.4
  record_structure = { \
    'recids'            : {'marc'      : '001',
                            'parse-func': parse_controlfield,
                            'comp-func' : fields_concat__raw },
    'title_levenshtein' : {'marc' : '245  a',
                            'parse-func': parse_nonrep,
                            'comp-func' : levenshtein_comp__raw },
    'title_dice'        : {'marc'      : '245  a',
                            'parse-func': parse_nonrep,
                            'comp-func' : \
                                  ntfnidf_vectordice_comp__wc },
    'abstract'          : {'marc'      : '520  a',
                            'parse-func': parse_nonrep,
                            'comp-func' : \
                                  ntfnidf_vectordice_comp__wc}}
     record_rules = maxsim
```

Ubiquist strategy

This strategy uses the bibliometric properties of most collections to estimate similarities between records. To do so, a succession of tests is applied in order to obtain the global record similarity, as already previously described in the 'Computing the global similarity' section dedicated to the

Ubiquist function.

This strategy's formal definition follows:

Code sample 12: Program configuration for the ubiquist strategy

```
    INPUT_FILES = ['collection.xml']
    records_comp = records_comp_single
    globalvars.output_threshold = 0.65
    record_structure = { \
        'recids'        : {'marc'      : '001',
                           'parse-func': parse_controlfield,
                           'comp-func' : fields_concat__raw },
        'doi':             {'marc' : '035  a',
                           'parse-func': parse_nonrep,
                           'comp-func' : identifiers_comp__raw},
        'authors'       : {'marc'      : ['100  a', '700  a'],
                           'parse-func': parse_multi,
                           'comp-func' : authors_comp__raw },
        'year'          : {'marc' : '260  c',
                           'parse-func': parse_nonrep,
                           'comp-func' : years_comp__raw },
        'title_dice'      : {'marc'      : '245  a',
                             'parse-func': parse_nonrep,
                             'comp-func' : \
                                  ntfnidf_vectordice_comp__wc },
        'abstract'        : {'marc'      : '520  a',
                             'parse-func': parse_nonrep,
                             'comp-func' : \
                                  ntfnidf_vectordice_comp__wc } }
        record_rules = ubiquist
```

V.9.4 Quantitative results

Recall

Results were obtained separately in the double blind test, and then put together in the following table. The processing time was measured by running all tests on Ubuntu 9.04 with Psyco on a Dell Optiplex 760, equiped with Intel Duo CPUs (E7400 @ 2.80Ghz) and 3 Gb of RAM.

Table 4 reports the number of retrieved duplicate pairs among the first 10, 20 and 50 top similarity scores for all combinations of strategies and collections, as well the processing time for each test.

Table 4: Results - recall on test collections for all strategies

Collection	Strategy	Recall 10	Recall 20	Recall 50	Time [minutes]
CERNa	2geom	5	8	8	7.40
CERNa	2geombreak	6	8	9	5.93

Collection	Strategy	Recall 10	Recall 20	Recall 50	Time [minutes]
CERNa	abstract_fallback	9	9	10	3.85
CERNa	boundaries_max	10	10	10	4.28
CERNa	geometric_jk	6	8	8	4.92
CERNa	initials	6	8	9	4.18
CERNa	initialsbreak	8	8	9	3.90
CERNa	okapigeom	5	8	8	6.38
CERNa	maxsim	10	10	10	50.42
CERNa	ubiquist	10	10	10	6.43
CERNb	2geom	3	4	8	7.35
CERNb	2geombreak	3	3	4	5.93
CERNb	abstract_fallback	8	9	10	3.92
CERNb	boundaries_max	9	9	10	4.23
CERNb	geometric_jk	4	7	8	4.93
CERNb	initials	5	6	8	4.2
CERNb	initialsbreak	5	5	7	3.98
CERNb	okapigeom	3	5	8	6.35
CERNb	maxsim	9	9	9	51.6
CERNb	ubiquist	9	9	10	6.53
CERNc	2geom	1	2	7	7.37
CERNc	2geombreak	3	3	5	5.98
CERNc	abstract_fallback	6	7	8	3.88
CERNc	boundaries_max	7	7	8	4.23
CERNc	geometric_jk	3	7	8	4.93
CERNc	initials	4	5	7	4.33
CERNc	initialsbreak	5	5	7	3.88
CERNc	okapigeom	3	3	7	6.37
CERNc	maxsim	7	8	8	50.17
CERNc	ubiquist	7	8	8	6.45
CERNd	2geom	4	4	6	7.45
CERNd	2geombreak	3	3	5	6.02
CERNd	abstract_fallback	2	3	5	2.88
CERNd	boundaries_max	2	3	5	3.17
CERNd	geometric_jk	3	5	8	3.77
CERNd	initials	7	7	7	4.17
CERNd	initialsbreak	7	7	10	3.92
CERNd	okapigeom	3	5	6	6.43
CERNd	maxsim	4	4	8	49.12
CERNd	ubiquist	7	9	10	5.58
ETHZ1	2geom	9	9	9	6.60

Collection	**Strategy**	**Recall 10**	**Recall 20**	**Recall 50**	**Time [minutes]**
ETHZ1	2geombreak	6	10	10	5.17
ETHZ1	abstract_fallback	3	7	10	2.82
ETHZ1	boundaries_max	3	7	10	3.03
ETHZ1	geometric_jk	6	10	10	3.15
ETHZ1	initials	9	10	10	4.08
ETHZ1	initialsbreak	10	10	10	3.78
ETHZ1	okapigeom	8	9	10	5.53
ETHZ1	ubiquist	5	6	7	4.33
ETHZ2	2geom	10	10	10	6.68
ETHZ1	maxsim	3	7	10	65.02
ETHZ2	2geombreak	10	10	10	5.17
ETHZ2	abstract_fallback	7	6	10	2.82
ETHZ2	boundaries_max	5	9	10	3.07
ETHZ2	geometric_jk	7	9	10	3.17
ETHZ2	initials	9	9	9	4.10
ETHZ2	initialsbreak	8	8	9	3.88
ETHZ2	okapigeom	9	10	10	5.55
ETHZ2	ubiquist	5	6	7	4.32
ETHZ2	maxsim	5	10	10	63.15
RERO1	2geom	8	10	10	12.65
RERO1	2geombreak	6	8	9	9.05
RERO1	abstract_fallback	4	7	10	12.35
RERO1	boundaries_max	0	2	10	13.58
RERO1	geometric_jk	2	8	9	14.73
RERO1	initials	10	10	10	9.53
RERO1	initialsbreak	9	9	10	8.93
RERO1	okapigeom	6	9	10	11.4
RERO1	ubiquist	7	10	10	16.67
RERO1	maxsim	0	3	10	16.65
RERO2	2geom	5	7	10	10.46
RERO2	2geombreak	3	6	8	13.80
RERO2	abstract_fallback	4	7	10	9.46
RERO2	boundaries_max	3	4	10	10.50
RERO2	geometric_jk	2	4	8	11.19
RERO2	initials	9	10	10	10.53
RERO2	initialsbreak	6	6	9	10.05
RERO2	okapigeom	10	10	10	12.61
RERO2	ubiquist	8	10	10	12.91
RERO2	maxsim	3	4	10	102.34

In order to show the retrieval performance of each strategy in a more visual way and identify their possible weak points, we analyse the position of each duplicate pair in the output file in Table 5. For each duplicated record, we report whether the pair was found:

- +++ in the first 10 results ,
- ++ in the first 20 results ,
- + in the first 50 results ,
- the cell is left empty if the record is detected after the position 50

Table 5: Individual records recall for all strategies.

+++ : the record was retrieved in the 10 first results ; ++ in the 20 first results ; + in the 50 first results.

	2geom break	**2geom**	**initials break**	**initials**	**Okapi geom**	**boundaries _max**	**abstract**	**ubiquist**	**geometric**	**maxsim**
CERNa|1187394	+++	+++	+++	+++	++	+++	+++	+++	+++	+++
CERNa|1187403	+++	+++	+++	+++	+++	+++	+++	+	+++	+++
CERNa|1187408	+++	+++	+++	+++	+++	+++	+	+++	+++	+++
CERNa|1187427	+++	+++	+++	+++	+++	+++	+++	+++	+++	+++
CERNa|1187432	++		+++	+++		+++	+++	+++		+++
CERNa|1187436	+++	++	+++	+++	++	+++	+++	+++	+++	+++
CERNa|1187448	++					+++	+++	+++		+++
CERNa|1187449		++	++	++	+++	+++	+++	+++	++	+++
CERNa|1187474	+	++		+	++	+++	+++	+++	++	+++
CERNa|1187507	+++	+++	++	++	+++	+++	+++	+++	+++	+++
CERNb|1187394	+++	+	+	++	+	+++	+++	+++	+++	+++
CERNb|1187403		+	+++	+++	+	+++	+++	+	++	+++
CERNb|1187408		+++		+++	+++	+++	++	+++	+++	+++
CERNb|1187427		+	+++	+++	+	+++	+++	+++	+	+++
CERNb|1187432			+++	+++		+++	+++	+++		+++

	2geom break	**2geom**	**initials break**	**initials**	**Okapi geom**	**boundaries_max**	**abstract**	**ubiquist**	**geometric**	**maxsim**
CERNb| 1187436	+++	+++	+++	+++	++	+++	+++	+++	+++	+++
CERNb| 1187448						+++	+++	+++		+++
CERNb| 1187449		++	+	+	+++	+++	+++	+++	++	+++
CERNb| 1187474	+	+++		+	++	+	+		++	
CERNb| 1187507	+++	+			+++	+++	+++	+++	+++	+++
CERNc| 1187394	+++	+	+	++	+	+++	+++	+++	+++	+++
CERNc| 1187403		+	+++	+++	+	+++	+++	+	++	+++
CERNc| 1187408	+++	+++	+++	+++	+++	+++	++	+++	+++	+++
CERNc| 1187427							+++	+++	+	+++
CERNc| 1187432			+++	+++		+++	+++	+++		+++
CERNc| 1187436	+	+	+++	+++	+	+++	+++	+++	+++	+++
CERNc| 1187448						+++	+++	+++		+++
CERNc| 1187449		++	+	+	+++				++	
CERNc| 1187474	+	+		+	+	+	+		++	
CERNc| 1187507	+++	+			+++	+++		++	++	++
CERNd| 1187394	+++	+++	+++	+++	+++	+++	+++	+++	+++	+++
CERNd| 1187403	+++	+++	+++	+++	+++	+++	+++	+	+++	+++
CERNd| 1187408		+	+++	+++	++			+	+	
CERNd| 1187427	+	+++	+++	+++	++	+	+	++	++	+
CERNd| 1187432	+++	+++	+++	+++	+++	++	++	++	+++	+++
CERNd| 1187436								+++	+	
CERNd| 1187448								+++		+
CERNd| 1187449			+++	+++	+			+++		+

	2geom break	**2geom**	**initials break**	**initials**	**Okapi geom**	**boundaries _max**	**abstract**	**ubiquist**	**geometric**	**maxsim**
CERNd|1187474		+	+++	+++				+++	+	+
CERNd|1187507	+					+	+	+++	++	+++
ETHZ1|10.3929/ethz-a-000080552	+++	+++	+++	++	++	+	+	+++	++	++
ETHZ1|10.3929/ethz-a-000085328	+++	+++	+++	+++	+++	++	++		++	++
ETHZ1|10.3929/ethz-a-000085643	+++	++	+++	+++	+++	+	+		++	+
ETHZ1|10.3929/ethz-a-000086597	+++	+++	+++	+++	+++	++	++	+++	+++	++
ETHZ1|10.3929/ethz-a-000087872			+++	+++	+			+++		++
ETHZ1|10.3929/ethz-a-000089154	+++	+++	+++	+++	+++	++	++	+++	+++	++
ETHZ1|10.3929/ethz-a-000090875	+++	+++	+++	+++	+++	+++	+++	++	+++	+++
ETHZ1|10.3929/ethz-a-000096651	+++	++	+++	+++	+++	+++	+++	+++	+++	+++
ETHZ1|10.3929/ethz-a-000152441	+++	++	+++	+++	+++	+++	+++	+++	+++	+++
ETHZ1|10.3929/ethz-a-	+++	+++	+++	+++	+++	+	+	+	++	+

	2geom break	2geom	initials break	initials	Okapi geom	boundaries _max	abstract	ubiquist	geometric	maxsim
000578009										
ETHZ2|10.3929/ethz-a-000081089	+++	+++	+++	+++	++	++	++	+	++	++
ETHZ2|10.3929/ethz-a-000085338	++	++	+		++	+++	++	++	+++	+++
ETHZ2|10.3929/ethz-a-000085653	++	++		++	+++	+	+	+	+	++
ETHZ2|10.3929/ethz-a-000087354	+++	+++	+++	+++	+++	+++	++	+++	+++	+++
ETHZ2|10.3929/ethz-a-000087882	+++	+++	+++	+++	+++	+++	+	+++	+++	+++
ETHZ2|10.3929/ethz-a-000089164	+++	+++	+++	+++	+++	++	++	++	++	++
ETHZ2|10.3929/ethz-a-000090885	+++	+++	+++	+++	+++	+++	+++	+++	++	+++
ETHZ2|10.3929/ethz-a-000096662	+++	+++	+++	+++	+++	+++	+++	+++	+++	+++
ETHZ2|10.3929/ethz-a-000153723	+++	+++	+++	+++	+++	++	++	++	++	++
ETHZ2|10.3929/ethz-a-000578047	+++	+++	+++	+++	++	++	++	++	++	++

	2geom break	**2geom**	**initials break**	**initials**	**Okapi geom**	**boundaries_max**	**abstract**	**ubiquist**	**geometric**	**maxsim**
RERO1|4121	+++	+++	+++	+++	++	+	+	++	++	+
RERO1|4140	+++	+++	+++	+++	++	+	+	++	++	+
RERO1|4151	+++	+++	+++	+++	+++	+	+	+++	++	+
RERO1|4205	+++	+++	+++	+++	+++	+	+	+++	++	+
RERO1|4326	+++	+++	+++	+++	+++	+	+	+++	++	+
RERO1|4817	+	+++	+++	+++	+++	+	+	+++	++	+
RERO1|5020		++	+++	+++	++	+	+	+++	+	+
RERO1|5382	++	+++	+	+++	+++	+	++	+++	++	++
RERO1|5800	+++	+++	+++	+++	+++	++	++	+++	+++	++
RERO1|6388	++	++	+++	+++	+	++	++	++	++	++
RERO2|7391	++	++	+	+++	+++	+	+	++	++	+
RERO2|7808	++	+++	+	+++	+++	+	+	+++	++	+
RERO2|7830	++	++	+++	+++	++	+	+	++	+	+
RERO2|7853	+++	+++	+++	+++	+++	+	+	+++	++	+
RERO2|7890		+	+++	+++	++	+	++	+++		+
RERO2|8017		+		++	++	+	++	+++		+
RERO2|8164	+	+++	+++	+++	++	++	+++	+++	+	++
RERO2|8344	+++	+++	+++	+++	+++	+++	+++	+++	+++	+++
RERO2|8708	+	+	+	+++	+	+++	+++	++	+	+++
RERO2|9280	+++	+++	+++	+++	+++	+++	+++	+++	+	+++

Speed estimations

In order to estimate performances, timing measurements were performed. These were systematically performed as no other application was running (except sometimes top). Most measurements were made in Ubuntu, and the X11 server and Gnome were systematically running.

In order to estimate the accuracy of these measurements, the same similarity calculation was executed 20 times. The mean duration was of 12.4 minutes. The standard deviation was 0.08 minutes, and the difference between the longest and the shortest duration was 0.33 minutes, which represents approximatively 3% of the average duration. This shows that theses measurements are accurate enough to compare the relative speed between strategies.

Speed and operating system

The duration in minutes of a batch similarity computation based on three different collection of 2000 records, using a geometric mean on 5 fields (Ids, title:Dice, date, authors, abstract:Dice) has been measured on a Dell Latitude D620 (Intel Core Duo T2400 at 1.83GHz, 1Go RAM) . This machine was set up with a dual boot Windows XP/SP2 and Ubuntu 9.04.

Table 6: Timing [minutes] according to the operating system and Python version

Collection	Ubuntu Py 2.6.2	Ubuntu Py 2.6.2 Psyco	Ubuntu Py 3.0.1	XP Py 2.4.3	XP Py 2.5.4	XP Py 2.5.4 Psyco	XP Py 3.1.0
CERNa	11.8	8.9	10.9	10.4	11.5	8.6	8.8
ETHZ1	7.3	5.9	7.2	6.6	7.1	5.8	6.0
RERO1	34.4	26.3	26.6	30.4	34.2	25.6	20.4
Total	53.6	41.0	44.7	47.7	58.8	40.0	35.5

Speed and number of records

To estimate this, a CERN articles collection was downloaded the same way as the CERNa through CERNd collections except that 16'000 records were stored. The collections listed below correspond respectively to the 500, 1'000, 2'000, 4'000, 8'000, 16'000 first records of the downloaded data set. These estimations were performed on 3 strategies:

- Abstract_fallback, that systematically computes abstract similarity if applicable.
- Initials, a geometrical mean on a fast function applied to many fields.
- Initialsbreak, the same as just above but using a « breakout mean » strategy.

Table 7: Timing [minutes] dependance on the collection length.

Number of Records	Abstract_fallback	Initials	InitialsBreak
500	0.3	0.2	0.2
1000	1.0	0.6	0.6
2000	4.2	3.0	3.1
4000	23.6	16.9	15.1
8000	166.7	103.1	96.1
16000	1073.2	762.2	726.0

V.9.5 Discussion

Efficiency of strategies

Let us examine the average recall of each strategy on all collections considering the first 10, 20 and 50 results on Table 4. The values are reported in Table 8.

Table 8: Average recall values for the test strategies

Strategy	Recall 10	Recall 20	Recall 50	Average	Timing [min]
2geom	5.6	6.8	8.5	7.0	8.3
2geombreak	5.0	6.4	7.5	6.3	7.1
abstract_fallback	5.4	6.9	9.1	7.1	5.3
boundaries_max	4.9	6.4	9.1	6.8	5.8
geometric_jk	4.1	7.3	8.6	6.7	6.4
initials	7.4	8.1	8.8	8.1	5.6
initialsbreak	7.3	7.3	8.9	7.8	5.3
maxsim	5.1	6.9	9.4	7.1	56.1
okapigeom	5.9	7.4	8.6	7.3	7.6
ubiquist	7.2	8.5	9.0	8.2	7.9

At most levels, the ubiquist method is the winner. The exceptions to this are the recall at 10 records, where the initials and initialsbreak obtain a slightly better grade, and the recall at 50 records, for which the abstract_fallback, boundaries_max and maxsim methods perform marginally better.

However, the three best **unrelated** strategies that give the highest recall in the 10 first results are ubiquist, initials and okapigeom. Indeed, initialsbreak and abstract_fallback must be left aside if unrelated strategies are considered: initialsbreak is similar to initials (a bit faster, but less efficient at R20) and abstract_fallback is similar to ubiquist (though much simpler and less efficient).

Even the best strategy is not capable of retrieving all 10 test records in the top 10 positions for all collections. Initials has got an average of 7.4.

The ubiquist method probably owes part of its success to the carefully chosen threshold parameters used for the various fields. Thanks to these numbers, the impact of the individual field similarity functions on the overall score can be fine tuned. The loss in generality is clearly compensated by the precision gain. Taking the abstract into account also appears to be a winning strategy. From the point of view of information retrieval, the abstract considerably enriches the number of terms in the "query", which automatically makes a vector-space or probabilistic approach more precise. At the very least, we can expect that a high abstract similarity combined with other very similar fields to indicate a close relationship between two records, worthy of closer inspection.

If we compare the breakout strategies with their full-calculation equivalents, we observe a decrease in precision. This is not surprising since this strategy essentially involves overlooking various field comparisons for the sake of speed. Nevertheless, the decrease in precision remains small enough on average, so that the breakout strategies remain competitive compared with the other strategies. This shows that breaking out of a record similarity calculation when some ad hoc condition is encountered is a valid option for speed optimizations – provided that the breakout test is adequate for the considered collection.

The okapigeom method is conceptually a modification of the 2geom strategy where the title/subtitle comparison uses the probabilistic Okapi BM25 formula rather than the Dice distance from the vector-space model. On average, it performs slightly better than the 2geom equivalent, and with shorter computation times, so one might be tempted to use it systematically as a replacement for field comparisons based on the vector-space model. Furthermore, we used the default values for its various numerical parameters, so one can imagine that the precision would be improved by optimizing these parameters for a given collection. However, the BM25 function has at least one drawback: the resulting value is in not normalized in the form we found in the literature. In this project, we modified the function to ensure that the result would remain within the [0.0;1.0] domain, but this patch is far from optimal. Even if the results are properly distributed according to the similarity of the record pairs, we cannot predict the absolute values for arbitrary collections. It will thus be difficult to apply any method that would be based on a threshold value, whether for the global comparison strategy (such as our breakout methods) or for output or post-processing. Moreover, it makes the human control of the detected duplicates more tedious, with no clear way to decide when this control should stop. As a consequence, we feel that this method is not suitable for the specific application presented here.

Both the initials and the initialsbreak strategy performed surprisingly well in our tests. Despite the rather bold assumption of representing whole MARC fields contents by the initials of their terms without even considering their order in the field, these methods are only surpassed by the ubiquist strategy as far as the mean average precision is concerned. They are also among the fastest in our tests. Indeed, they can be seen as a loose variant of the well-established technique of generating keys or hash functions based on the abbreviated forms of selected field contents, and performing the duplicate detection analysis over these codes (Goyal 1987). However, it is not certain that this result could be extrapolated to arbitrary collections. As more and more records are processed in a large collection, the probability of collisions between codes generated from unrelated records increases. Furthermore, if the content of the fields becomes richer (a likely example would be the author list of many CERN publications, where hundreds of people are acknowledged), the probability of using up most of the alphabet becomes fairly high.

The maxsim, boundaries_max and abstract_fallback share a common behaviour as far as precision is concerned. On one hand, their average precision at 10 records is not better than that of the geometric mean methods. On the other hand, we observe that our duplicate pairs almost always appear within the top 50 similarity scores. The maxsim and boundaries_max methods clearly favour recall over precision: the maximal similarity score is returned, overlooking the less similar fields. As for the abstract_fallback method, it is meant more as a demonstration of the discriminating power of information retrieval methods than as a real duplicate detection strategy. The fact that it still manages to obtain a decent grade proves the usefulness of the approach.

Complementarity of strategies

Some of our engineered duplicates are harder to find than others. In order to measure this numerically, grades were assigned based on the data in Table 5 as follows (blank cell → 0 ; + → 1 ; ++ → 2 ; +++ → 3).

For each test record, the maximal, the minimal and the average values were computed over all strategies. The results are displayed in Figure 2. The 'max' data set shows that amongst all tested strategies, at least one was able to retrieve each engineered duplicate within the 10 first results (there is only one exception on the 80 near-duplicates). This is quite an achievement because, there are 10 near-duplicates in each collection that have a valid reason to be in the 10 first results.

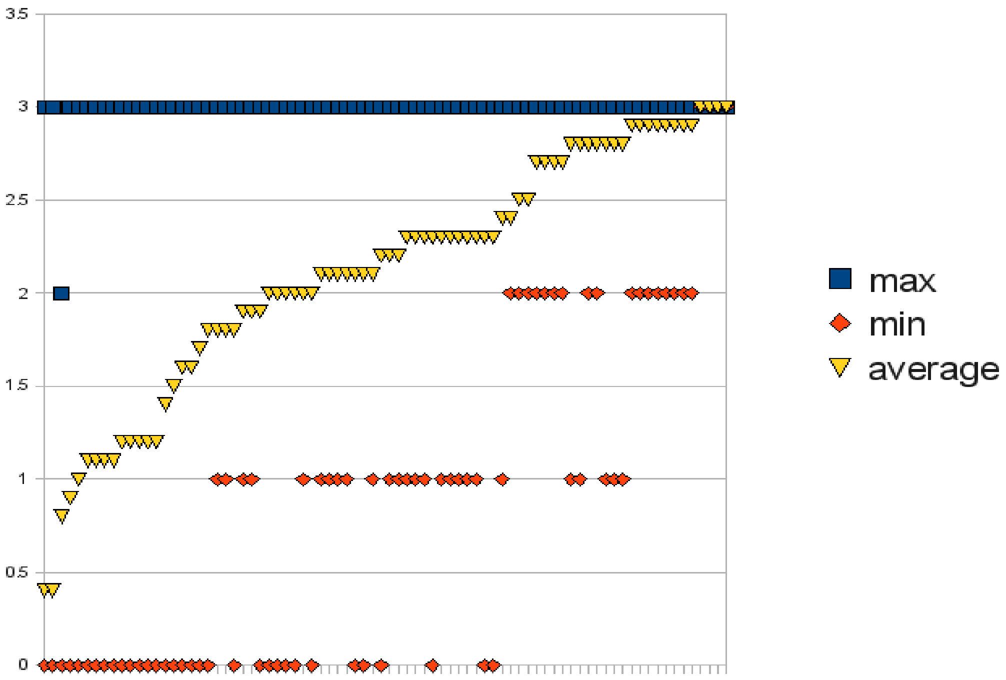


Figure 2: Qualitative recall according to Table 5 for all test records (sorted by average score)

In other words, even if no single strategy is capable of retrieving all 10 test records in all collections, at least one of our strategies almost always succeeded (with one exception): our strategies are complementary.

Fortunately, the integration of several outputs is straightforward and can be done automatically: all *n* first duplicate candidates pairs of each strategy can be gathered together, then the identical pairs regrouped, their similarity adjusted, and the results sorted again.

The most difficult part is to identify the most efficient and complementary strategies.

Using Table 5 helps analysing our best strategies under a different angle: how often did each of these strategy yield the best performance in the top 10 results (compared to the other strategies, even if it is a tie)? The results are summed up in Table 9.

Table 9: Percentage of best recall at 10 records for the initials, ubiquist and okapigeom strategies

Strategy	**Percentage of the best performance in the top 10**
Initials	73%
Ubiquist	66%
Okapigeom	50%

From this point of view, initials is the most efficient strategy because it was the best in the retrieval in 73% of the cases. But, what is the complementarity of these three strategies? This can be read on the Table 5, too.

- In 93% of the cases, initials OR ubiquist performed the best job compared to the other strategies. But in 46% of the cases initials AND ubiquist achieved the best marks together. So, 47% of the duplicates that they detect are not common to them.
- In 96% of the cases, initials OR ubiquist OR okapigeom performed the best job compared to the other strategies. But in 29% of the cases initials AND ubiquist AND okapigeom achieved the best marks together. So, 67% of the duplicates they detect are not common to the three.

In summary, these results show that the combined use of these 2 or 3 strategies is sufficient to detect the vast majority of the test duplicates because they are quite complementary.

Recall analysis

In a few cases, these strategies combined were not able to detect all test duplicates within the 10 first results. In absolute numbers only 6 of our 80 test duplicates are problematic. They may be subdivided as follows:

3 records were not found at all (even with considering the 3 best strategies combined) (Table 10). 3 records were found only in combining the 3 strategies (2 strategies were not enough) (Table 11).

Table 10: Recall failures at 10 for the initials, ubiquist and okapigeom strategies

Identifier	Comments
ETHZ2|10.3929/ethz-a-000085338	The initials strategy failed totally (not in the 50 first) in this case, but ubiquist and okapigeom found it in the first 20 results, so it is not catastrophic. It must be noted that exceptionally boundaries_max got it in the top 10. The modification applied was simple and quite likely: the main author was replaced by a PhD advisor.
CERNb|1187474	The title was truncated (3 words out of 12) and the abstract removed. But okapigeom was still able to get it in the 20 first results. The best score was this time exceptionally performed by the 2geom strategy.
CERNc|1187474	This is a very tough case. Only one strategy managed to get it in the first 20 records: the geometric strategy. In addition to the CERNb|1187474 modifications described above 1 author out of 3 was removed.

Table 11: Near-duplicates identified by the combination of all 3 best strategies

Identifier	Comments
ETHZ2|10.3929/ethz-a-000085653	Square brackets were added around the title (this is a catalogue rule indicating that an artificial title was attributed to the document). Okapigeom made the difference. However, ubiquist and initials managed to place it in the first 20 results.
CERNc|1187449	The title was truncated (3 word out of 7). The date was changed by 2 years. The abstract was removed... and the okapigeom strategy was able to put it in the first 10 results!
CERNc|1187507	Half of the authors were deleted (2 out of 4), The title was truncated (1 word out of 6), and 2 sentences were cut out of the abstract. Ubiquist was able to place this duplicate in the first 20 results, but initials failed totally (it was not even in the first 50 results). However, okapigeom succeed in getting it in the top ten.

The average curve on Figure 2 is related to the difficulty to find each duplicate. Let us consider the records that were in average the hardest to return in the top ten:

- In the 10 most difficult duplicates to find (lowest average), 9 belong to the CERNc and CERNd collections. This is not surprising because these two collections contain the test records that were most heavily modified: At least three fields of significance were altered in each record.
- The 10th trickiest duplicate is the RERO2|8017, in which HTML tags were added in the title and a diacritic suppressed. It can reasonably be assumed that the diacritic removal had no effect, because these are eliminated in the text preprocessing. The HTML tags adjacent to one particular word are not frequent and must have produced an important impact on the title similarities because of their *nidf*.
- The 11th trickiest duplicate is the ETHZ1|10.3929/ethz-a-000087872. The title contains a misspelling and the date field was removed. Misspellings may have quite a lot of impact: they can cause the removal of one indexation term as well as the introduction of a rare term that will have a strong impact because of its *nidf.* Unsurprisingly, the initials strategy performed well: is not affected by misspellings unless they touch the first character. The maxsim function, that is good at detecting misspellings because it uses the Levenshtein algorithm did also well.
- The 12th hardest test record to find was CERNb|1187448. All authors first names were removed. Effectively, none of them was recognized because of the way the author comparison function works. And the date was changed by 3 years, which is more than the breakout means based strategies as well as other strategies will tolerate (2 years). But the boundaries max, abstract fallback and ubiquist strategies that use the abstract were able to surmount this difficulty.

The 4 easiest duplicates to find have all an average of 3.0, which means that they were systematically retrieved in the 10 first results by all strategies:

1. CERNa|1187427 : the only modification was to move 2 sentences within the abstract. This was easy to spot since all algorithms that were applied to abstracts are based on therm frequencies.
2. RERO2|8344 : one author was removed (many strategies do not use the author field, and the ones that do take other parameters into account), and the year deleted.

3. ETHZ2|10.3929/ethz-a-000096662: The author name format was change from one supported format to another supported format (lastname, firstname → firstname lastname)
4. CERNd|1187394 : Only the title field was touched.

The 8 next easiest duplicates to find have all an average of 2.9 (they were retrieved in the 10 first results by many strategies):

- CERNc|1187408 : two sentences were cut out of a long abstract, the year changed by 2 years (this is tolerated in all strategies that use the date field).
- CERNb|1187436 : authors were removed (6 out of 14, and the author similarity stays over 50%), the abstract remixed (this does not affect term frequencies based algorithms applied to abstracts).
- CERNa|1187394 : date changed by two years : this is tolerated in all strategies (if they use the date field at all).
- ETHZ1|10.3929/ethz-a-000152441: the subtitle was changed by a variant. Many strategies do not use the subtitle field. The others tolerate variations.
- ETHZ2|10.3929/ethz-a-000090885: the author first name was replaced by an initial (in the author normalisation, this is done anyway and has therfore no effect), and the date removed
- ETHZ2|10.3929/ethz-a-000087354 : a diacritic was removed from a title... this has no effect because all diacritics are removed in the text normalisation process.
- ETHZ1|10.3929/ethz-a-000090875 : a diacritic was added to an author name... this has no effect because all diacritics are removed in the text normalisation process.
- ETHZ1|10.3929/ethz-a-000096651: the subtitle was removed. Many strategies do not use the subtitle field. The others tolerate variations.

Together, the analysis of the 12 respectively easiest and hardest to find test duplicates shows that:

- In summary, the difficulty to find a duplicate seems particularly linked to the number of fields that were altered, and then of course to the severity of the alterations. The failures to place a duplicate in the 10 (or 20) first results happen often when 3 fields or more have been altered.
- If one field has been reasonably altered in a record, it will be returned in the top 10 results by most strategies, whichever field the alteration concerns. With a proper strategy, the following alterations are well detected:

    - removing up to one third of authors
    - removing up to one third of a title
    - removing a few sentences in an abstract
    - changing the publication date by up to two years
    - words permutations have in general no effect (except in Levenshtein based algorithms... and shingling algorithms, that were not tested in this part of this report)
    - punctuation and diacritics are normalised, their alteration has generally no effect
    - author format variations don't have much influence (order of first and last names, abbreviated first names...)
- misspelled words in a title produce more effect than their deletion
- many strategies will still work if a few fields are deleted

Noise analysis

Now, let us examine the false positives within the 10 highest similarities, i.e. the noise, for the methods that obtained a reasonably good grade on a given collection (more than 5 duplicates found in the first 10 results) as they can provide useful inspiration for strategy refinements. False positives are generally due to similar records that may in some cases be considered as duplicates or just similar records that are both justified. Two documents published by the same authors, and whose titles differ only by one letter (sometimes even articles in the same issue of a journal) are not unheard of. (Lavanchy 1977) and (Tecon 1979) are examples of such a pathological case. Thus, if highly similar but distinct records are known to exist within a collection, it would be useful to mark them as such (by a specific field or by adding them to a stop list) so that the program can automatically skip them in the analysis.

a) CERN collections

As expected, the abstract_fallback, boundaries_max, ubiquist and maxsim strategies yielded the best results on the CERNa collection. These methods were originally developed to process this very collection (CERN records with not too many field alterations) so this is essentially a confirmation.

We got more or less equivalent results with the 2geom (Table 12), 2geombreak, geometric_jk (Table 13), and okapigeom (Table 14) methods. Except for the last one, the noise in the 2geombreak results was identical to the one with the 2geom method.

Table 12: Noise pairs reported by 2geom for the CERNa collection

Identifiers	Comments
1181589 | 1181575	The same author presented his work under the same title at 2 different conferences, with different abstracts
1185026 | 1185023	The author published two articles with the same titles, published in the same year in 2 different journals
1186848 | 1186846	Parts 1 and 2 of a continuing work, with titles different by 1 word, written over 2 years and distributed as preprints
1182408 | 1182407	Identical records except for 1 author and the page information
1182864 | 1182862	The same authors wrote 2 reports with very similar (but clearly different) titles

Table 13:Noise pairs reported by geometric_jk for the CERNa collection

Identifiers	Comments
1178196 | 1178188	2 norms with 2 different subtitles (no author)
1182949 | 1182948	The titles differ by just one word,
1182408 | 1182407	See 2geom results
1185026 | 1185023	See 2geom results

Table 14: Noise pairs reported by okapigeom for the CERNa collection

Identifiers	Comments
1185026 | 1185023	See 2geom results
1181589 | 1181575	See 2geom results
1182408 | 1182407	See 2geom results
1181140 | 1181137	2 preprints written in the same year by largely identical authors. The title of one document is a part of the title of the other
1182817 | 1182633	The same authors (in a different order) presented 2 contributions to the same conference, with 3 words out of 4 composing the shorter title also found in the longer one.

Table 15: Noise pairs reported by initials for the CERNa collection

Identifiers	Comments
1179074 | 1179072	Some shared words in the title, a majority of shared authors
1181925 | 1178561	Apparently the conference proceedings and article versions of the same work (one title contains an annotation, otherwise identical)
1182567 | 1182501	The conference name is the only identical feature of the records
1182987 | 1182271	Many shared words in the title, as well as shared authors, presented to the same conference.

There is also a significant overlap between the noise records generated by all 4 methods. Since they share a common fundamental approach to the problem (evaluation of the global similarity as the geometric mean of the individual field values), this is probably not surprising. As noted in the general discussion of the ubiquist strategy, adding the abstract to the comparison fields might improve the performance of these vector-space-model methods with this particular collection. The CERN Document Server generally contains rich metadata with the abstract available in most cases.

Table 16: Noise pairs reported by initialsbreak for the CERNa collection

Identifiers	Comments
1179074 | 1179072	See initials results
1181925 | 117856	See initials results
1182987 | 1182271	See initials results
1185052 | 1185041	Two articles by the same authors, with similar titles, published after a 2-year interval

The initials and initialsbreak methods obtained quite good results as well, but the noise records already show some cases of the hash collisions mentioned earlier (Tables 15 and 16). This suggests that this method for the field content representation should not be used for larger collections than ours. As mentioned earlier, enriching our comparison data with abstract might have a negative effect in this case. Obviously an abstract will contain a broader vocabulary than a title, which means that more varied initials will be found and that the intersect of two sets extracted from arbitrary records should increase on average.

The CERNb, CERNc and CERNd results show an increasing error rate, which is natural considering their construction. In particular, the methods based on a geometric mean suffer from a marked performance loss going from the CERNa to the CERNb collection. Only the initials, initialsbreak and ubiquist retain a reasonable detection efficiency with the CERNd collection. Except for one pseudo-hash collision for the initials strategy, the origin of the noise pairs for theses methods (Tables 17 and 18) seem to be curable with minor changes in the field and record similarity computations.

Table 17: Noise pairs reported by initials for the CERNd collection

Identifiers	Comments
1179074 | 1179072	2 papers with many identical title words (but in a different order) by essentially the same authors
1181925 | 1178561	Preprint and final published version of the same work
1182567 | 1182501	Titles and authors are totally unrelated, but the papers were presented at the same conference

Table 18: Noise pairs reported by ubiquist for the CERNd collection

Identifiers	Comments
1187324 | 1187320	2 contributions with the same main title at the same conference, with different secondary titles, by two different authors
1178196 | 1178188	2 norms with different subtitles (no author)
1182949 | 1182948	2 contributions at the same conference, one shared author and titles differ by just one word

b) ETHZ1

The performance of the methods developed with this collection in mind (2geom, 2geombreak, initials, initialsbreak, okapigeom) was good. In general, 90% of the duplicates were discovered in the top 20 similarity values if not the top 10 values. This holds for the geometric_jk method as well, which demonstrates again its close relationship with the other geometric mean methods. The abstract_fallback,boundaries_max, ubiquist and maxsim had more problems but managed to find all duplicates within the top 50 similarity scores. It is interesting to note that for this collection with few abstracts, the ubiquist method did not shine as brightly as with the CERN collections.

Table 19: Noise pairs reported by 2geombreak for the ETHZ1 collection

Identifiers	Comments
10.3929/ethz-a-005552594 | 10.3929/ethz-a-005589693	2 different authors produced dissertations with the same title on the same year.
10.3929/ethz-a-005431188 | 10.3929/ethz-a-005431667	Same problem as above with another title
10.3929/ethz-a-005394500 | 10.3929/ethz-a-005431667	Same problem as #2, except for a minor change in the title.
10.3929/ethz-a-005394500 | 10.3929/ethz-a-005431188	Same problem as #2 and #3, forming a triplet of near duplicates

Table 20: Noise pairs reported by geometric_jk for the ETHZ1 collection

Identifiers	Comments
10.3929/ethz-a-005552594 | 10.3929/ethz-a-005589693	See 2geombreak results
10.3929/ethz-a-005431188 | 10.3929/ethz-a-005431667	See 2geombreak results
10.3929/ethz-a-000362933 | 10.3929/ethz-a-000494141	Same problem as above with another title
10.3929/ethz-a-005394500 | 10.3929/ethz-a-005431667	See 2geombreak results

Once again, the abstract could be a useful way to distinguish between the documents produced by several students based on the same assignment by a teacher. However, the precision improvement will not be as important for records from the ETH e-Collection as for those from the CERN Document Server. Indeed, ETHZ1 and ETHZ2 are usually derivatives from the NEBIS library network union catalogue, which tends to emphasize formal cataloguing over record enrichment.

The initials method shows another example of hash collision (Table 21).

Table 21: Noise pairs reported by initials for the ETHZ1 collection

Identifiers	Comments
10.3929/ethz-a-004288363 | 10.3929/ethz-a-004756575	Several words are shared in the titles, 2 authors share the same initials

c) ETHZ2

The precision results with the ETHZ2 are similar to those with the ETHZ1 collection, except for the maxsim method which shows an improvement (all duplicate pairs were found within the top 20 similarity values).

The geometric_jk was not as good as the 2geom, 2geombreak and okapigeom method in this case.

Table 22: Noise pairs reported by geometric_jk for the ETHZ2 collection

Identifiers	Comments
10.3929/ethz-a-005395825 | 10.3929/ethz-a-005396163	2 different authors produced dissertations with the same title on the same year.
10.3929/ethz-a-005395824 | 10.3929/ethz-a-005395825	Same problem as above with another title
10.3929/ethz-a-005395824 | 10.3929/ethz-a-005396163	Same problem as #2

The okapigeom produced one noise record pair with many differences between the two records (Table 23). The lack of an absolute scale for the Okapi BM25 might be a cause.

Table 23: Noise pairs reported by okapigeom for the ETHZ2 collection

Identifiers	Comments
10.3929/ethz-a-004292962 | 10.3929/ethz-a-004292980	Two volumes of a work, with different subtitles and some variant authors.

Table 24: Noise pairs reported by initials for the ETHZ2 collection

Identifiers	Comments
10.3929/ethz-a-004325635 | 10.3929/ethz-a-004325666	A majority of shared words in the title and one shared author

d) RERO1

The initials strategy yielded very good results for this collection, with the others generally able to report most duplicate pairs within their top 20 similarity scores. The exceptions to this are the methods that tend to favour recall over precision: abstract_fallback, maxsim and boundaries_max. The geometric_jk strategy performed more poorly than its 2geom and okapigeom (developed with a better knowledge of the collection's field structure).

Table 25: Noise pairs reported by 2geom for the RERO1 collection

Identifiers	Comments
8410 | 8414	English records for two versions of the same article, published in English and Japanese respectively (as noted in the title and the journal name – the pagination is almost identical).
8104 | 8526	Only one word is different in the titles, which appears to be a mistake after fulltext examination. This might be counted as an actual duplicate, or at least as a case requiring human intervention

Table 26: Noise pairs reported by okapigeom for the RERO1 collection

Identifiers	Comments
4186 | 4187	Parts I and II (in Roman numbers) published over 2 years
8410 | 8414	See 2geom results
6387 | 6388	The same authors wrote two articles with very similar titles, published in the same journal at a few pages from each other.
8064 | 8065	Two parts of an ongoing work (with different subtitles) were published in two consecutive issues of the same journal.

Table 27: Noise pairs reported by initialsbreak for the RERO1 collection

Identifiers	Comments
4186 | 4187	See okapigeom results

Table 28: Noise pairs reported by ubiquist for the RERO1 collection

Identifiers	Comments
8410 | 8414	See 2geom results
5269 | 5270	The authors wrote 2 articles with rather different titles, but the abstracts are identical. Fulltext examination shows that the abstracts should be different, although similar. This is not actually a duplicate, but certainly something that requires cleaning up.
6672 | 6674	Same problem as #2, but the titles are closer and the real abstracts qualitatively farther from each other. The publication years are different as well.

e) RERO2

With the exceptions of the initials, okapigeom (with a perfect score) and ubiquist methods, most results were mediocre to poor.

Table 29: Noise pairs reported by initials for the RERO2 collection

Identifiers	Comments
11499 | 11504	The author wrote two articles with the same title, published in the same journal on two consecutive years. This particular case was often found in the top 10 or 20 similarity values with the other methods

Table 30: Noise pairs reported by ubiquist for the RERO2 collection

Identifiers	Comments
8410 | 8414	English records for two versions of the same article, published in English and Japanese respectively (as noted in the title and the journal name – the pagination is almost identical)
11176 | 11261	The same work was presented once to a conference and once as a book chapter

Improvement suggestions for the presented methods

We have discussed the limitations of the initials strategy, which temperate its generally good results. Nevertheless the fact that hash functions and complex keys are frequently used in the literature suggests that good precision could be achieved with somewhat larger collections already with minor improvements to the method. Instead of initials, digrams or trigrams could be used, as in the construction of the DIFWICS database (Hylton 1996), or even methods such as Soundex.

The methods based on a geometric mean will by significantly improved if they can take advantage of an abstract. However, its availability is far from constant from one collection to the other. Fortunately, the general trend in digital as well as classic libraries is to systematically enrich the records with information that the users might find useful. The abstract is a good example of such value-adding data, so we can hope that it will be included more and more commonly in the future.

The ubiquist stategy did a very good job on average. Nevertheless, its efficiency appears to be adversely affected by the absence of abstracts in a collection, especially when the variations between near duplicates happen in the title. In such cases, even alterations by a few characters can be tricky (see for example the ETHZ 10.3929/ethz-a-000085328, 10.3929/ethz-a-000085643, 10.3929/ethz-a-000578009, 10.3929/ethz-a-000081089 and 10.3929/ethz-a-000085653 records). To guard against this problem, a better treatment of other fields would be helpful. Inclusion of the subtitle field 245__$b in the comparison is one easy step into that direction, which will be beneficial for records derived from classic library records, with sophisticated field use but sparse text content. Source information, such as journal titles, volumes and pages is clearly useful for articles, book chapters, etc., but in order to take advantage of it a good documentation of the cataloguing rules for the examined collection is required as many different conventions exist.

Finally, as discussed earlier, the complementarity of some strategies suggests that combining their outputs, for example by adding the global similarities, will result in a very precise detection. Of course, the computational cost will be higher.

Speed and operating system

As shown in Table 7, the duration of the same similarity calculations varies between interpreter versions, due to the presence of Psyco, and between operating systems... but not very much so.

Windows XP was approximatively 10% faster than Ubuntu, while both using Python 2.6.2, the latest stable version of the Python 2 branch (which is for now the most important branch since many important python modules have not been ported to the 3 branches).

However, no Pysco binaries are available for Python 2.6.2 on Windows. And using Psyco on Ubuntu reverses the situation, and the run on Ubuntu becomes almost 15% faster than on XP.

Because Python 3.1.x has been improved, its use on XP is ~25% faster than python 2.6.2 again on

XP and ~15% faster than Python 2.6.2 with Psyco on Ubuntu. Python 3.1 will be integrated in Ubuntu in the October 2009 release (9.10).

Speed and number of records

Analyses of collections of more than 10'000 records will generally take more than one our on an average desktop computer. This is not fast enough, but it can luckily be improved in several ways discussed in the next section.

VI Perspectives and next developments

VI.1 Developments of the similarity framework itself

The framework is stable... but it can be improved!

VI.1.1 Speed optimisations

This is probably the very next aspect that will be tackled, because there is a lot of useful possible improvements:

Using multiprocessing

When running alone one multiple core machines, MarcXimiL will generally be assigned about 100% of one CPU. The other core(s) will stay quite idle. Introduced in Python 2.6, a multiprocessing module can help take advantage of all CPUs. Due to the flexibility of the framework it could easily be introduced at the record comparison functions level (functions of the compare_records module). These functions determine the pairs of records that are compared as well as the order in which it is done. The use of the multiprocessing module could simply replace the loop that iterates over record pairs comparisons and execute them in parallel.

High level code optimisations

Use python profiling tools to detect the functions that take the most resources and optimize them.

Compile the most critical functions

Compiled code is generally faster than Python. It could be done in C, the compilation managed by Python seamlessly at execution time if a GCC compiler is installed on the system.

Optimise global similarity functions

The functions that compute the global similarity also pilot the execution of each field comparison. If any field comparison is not required it should not be executed. For example, let's consider the case of a pair of records flaged with two different digital object identifier (DOI) of the same family (for example two PubMed Identifiers or PMIDs). A deduplication strategy could return at once 0.0 as the global similarity and stop immediately the treatment for this pair of records. This solution becomes more compelling as the use of DOI becomes more frequent and well organized.

_Intelligent record_comp functions_

These functions determine the pairs of records that will be compared as well as the order in which it will be done. Presently, they are quite basic. It is for example possible to compare all records of all loaded collections with each other. An existing speed improvement consist in comparing each pair of records only once (if A is compared to B, B will not be compared to A later). This is useful because most of the similarity functions built in this framework are symmetrical in that regard.

It would be even better to use this level to avoid doing potentially useless calculation. A lead could be to sort the records using an intelligent criteria (this is much faster than a similarity analysis) and then perform a full similarity only on adjacent records (in the broad sense of the term). Pre-clustering can also be used to reduce the number of full comparisons by several orders of magnitude, as shown by various authors (De Melo & Lopes 2005)

VI.1.2 MarcXimiL as a Python package

It would be useful to make a few adjustments to permit the importation of this framework as a package in external Python programs, like CDS Invenio. A Python package is a coherent set of modules that may be registered in the Python Package Index[34] and that is often takes the form of an 'egg', the Python package distribution format. This would also be helpful for the development of a graphical user interface, which is the subject of the next paragraph.

VI.1.3 Graphical user interface (GUI)

Such a tool would probably help new users to discover the capabilities of this framework. Even if the similarity configuration file is quite straight forward to understand and customize, a GUI might make the difference for users who are not accustomed to edit files (and this application also runs on Windows).

A nice GUI could be written using asynchronous JavaScript and XML (AJAX) in conjunction with the Python built-in web service capabilities and the MarcXimiL toox.py library for XHTML and XML generation. Thus no other software requirement other than a browser would be necessary, and remote access would be immediately added to the program capabilities.

On a related topic, a few modifications would permit to show the progression of the similarity computation process. This would be welcome, since it is often quite long. Another nice feature related to the interface would be a function capable of estimating the duration of a run on the basis the progression.

---

[34] http://pypi.python.org/pypi

VI.1.4 XML validation

A fast MARCXML validation tool in pure Python could be an asset: The similarity framework requires strictly and valid MARCXML. However, no validation test is performed prior to a run. Presently, the only way to make sure a collection is valid is to use an external tool like `xmllint`[35] (UNIX, Linux, MacOSX) in conjunction with the MARCXML Document Type Definition (DTD)[36].

VI.1.5 Limiting memory usage while loading collections

As the records are loaded, a lot of RAM is used for large collections. It is quite easy to remedy to that. However, it is not urgent since a critical RAM usage (like 4Gb) only concerns big collections of at least 50'000 records (it depends of course on the records structure and content). Before speed optimisations, analysing such a collection would take a quite long time to process anyway, and therefore the case should therefore not present itself at the moment. Furthermore, large collections are frequently delivered in chunks of manageable size, which our program could easily load in sequence. In any case, the if a critical usage of RAM is reached, the operating systems swap can kick in and help in that regard. Yet another way to manage big collections could be to use CDS Invenio.

VI.1.6 Using CDS Invenio data directly

As Invenio is written in Python, it is easy to import its search engine within MarcXimiL. Essentially, only three short modifications are required:

1. Instead of importing collections using file names in the configuration file, Invenio requests must be used.
2. The loading of records must be skipped, and the globalvars.reccache filled with the recids corresponding to the collection.
3. The get_cached_record function must be adapted too lookup records in Invenio.

An Invenio search can be performed in just two lines of code from within MarcXimiL, as documented in the Invenio search engine API documentation[37]:

```
>>> # import the function:
>>> from invenio.search_engine import perform_request_search
>>> # get all hits in a collection:
>>> perform_request_search(cc="ARTICLES")
```

---

[35] http://www.xmlsoft.org/xmllint.html

[36] http://www.loc.gov/marc/marcxml.html

[37] http://invenio-demo.cern.ch/help/hacking/search-engine-api

VI.2 Developments on top of the framework

Because of the flexibility of MarcXimiL, many interesting developments are possible on top of it. For demonstration purposes, applications prototypes are already bundled with the framework. These tools are potentially useful and should be developed further.

VI.2.1 Monitor.py

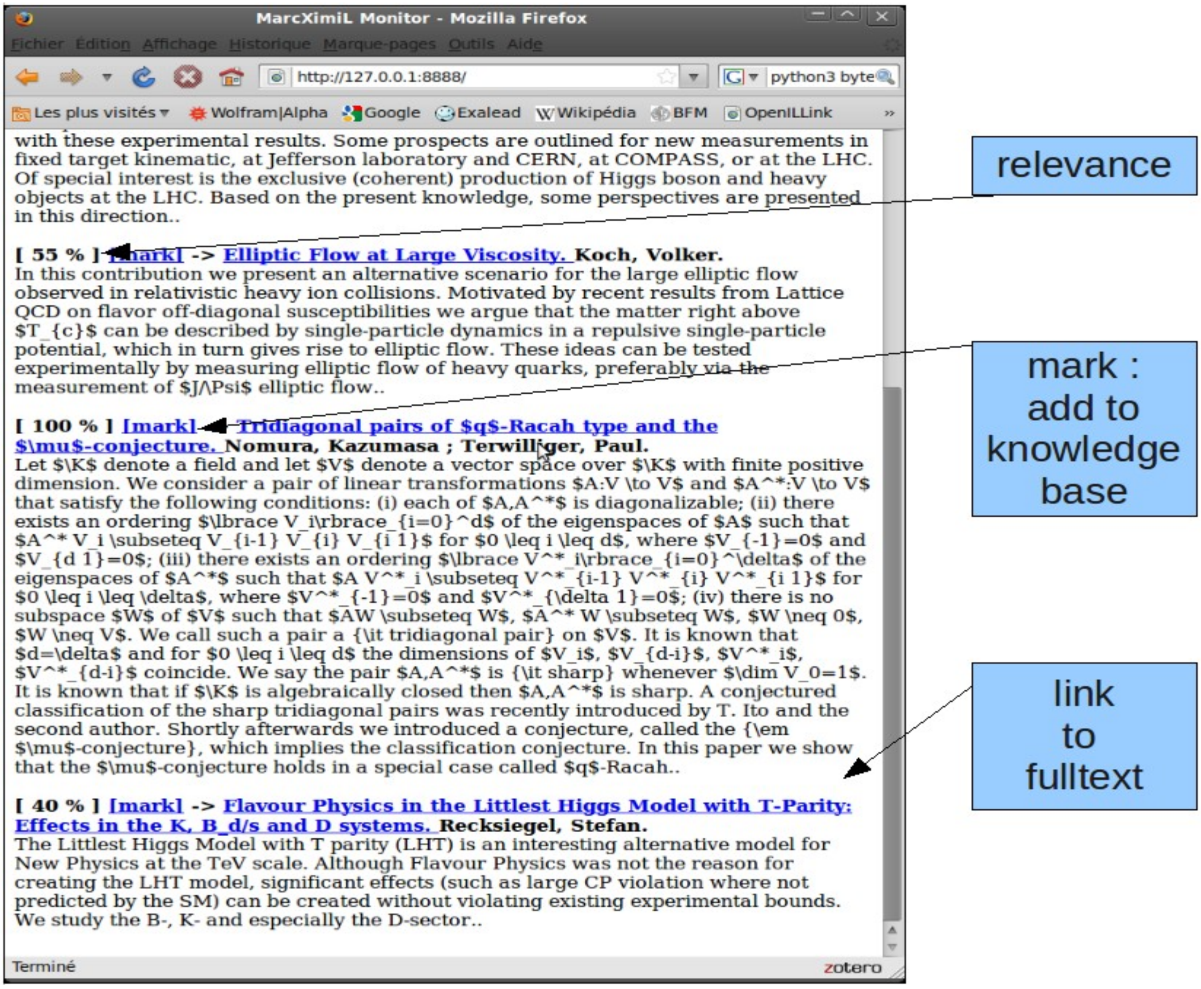
This application prototype enables MarcXimiL is to perform information monitoring on servers running CDS Invenio. Presently monitor.py is in early stage of development and is only provided as an example of this framework's capabilities. It is located in the ./bin folder .

The tool uses a knowledge base of articles deemed interesting by the user and filters on that basis potentially interesting the latest additions made on the server that are automatically downloaded. The similarity filtering is done on the basis of titles, abstracts and authors.

Monitor.py is already capable of learning from the users, who can mark retrieved records that they find of interest. This system could easily be improved by learning users needs through their behaviour, for instance by detecting the retrieved documents that they open.

The tricky part will be to render this tool attractive for the end user. This involves a nice and reactive interface. For reactivity purposes, it would probably be best if the download of new records and similarity analysis are executed in the background on a regular basis. The current web based interface could be retained and improved using the AJAX technology and an elegant design. An applet running directly on the user's desktop would certainly be a real asset. This way, the user could be kept transparently informed of incoming references and, if interested, visualize them directly by the simple expedient of one click in the CDS Invenio server. For now, the interface is provided by an ad-hoc lightweight HTTP server and browsable at http://127.0.0.1:8888.

An interesting aspect of this application for librarians might be to take care of the initial set up for their users: They could create the initial knowledge base collection (the reference used to find similar articles in data flows on the web), by filling it with the metadata of their user's publications and related work. Librarians may also improve the performances by setting up the incoming data flow using well chosen queries as well as appropriated data sources.



with these experimental results. Some prospects are outlined for new measurements in fixed target kinematic, at Jefferson laboratory and CERN, at COMPASS, or at the LHC. Of special interest is the exclusive (coherent) production of Higgs boson and heavy objects at the LHC. Based on the present knowledge, some perspectives are presented in this direction..

[55 %] [mark] -> Elliptic Flow at Large Viscosity. **Koch, Volker.**
In this contribution we present an alternative scenario for the large elliptic flow observed in relativistic heavy ion collisions. Motivated by recent results from Lattice QCD on flavor off-diagonal susceptibilities we argue that the matter right above T_{c} can be described by single-particle dynamics in a repulsive single-particle potential, which in turn gives rise to elliptic flow. These ideas can be tested experimentally by measuring elliptic flow of heavy quarks, preferably via the measurement of J/Ψ elliptic flow..

[100 %] [mark] -> Tridiagonal pairs of q-Racah type and the μ-conjecture. **Nomura, Kazumasa ; Terwilliger, Paul.**
Let $\K$ denote a field and let V denote a vector space over $\K$ with finite positive dimension. We consider a pair of linear transformations $A:V \to V$ and $A^*:V \to V$ that satisfy the following conditions: (i) each of A,A^* is diagonalizable; (ii) there exists an ordering $\lbrace V_i\rbrace_{i=0}^d$ of the eigenspaces of A such that $A^* V_i \subseteq V_{i-1} V_{i} V_{i 1}$ for $0 \leq i \leq d$, where $V_{-1}=0$ and $V_{d 1}=0$; (iii) there exists an ordering $\lbrace V^*_i\rbrace_{i=0}^\delta$ of the eigenspaces of A^* such that $A V^*_i \subseteq V^*_{i-1} V^*_{i} V^*_{i 1}$ for $0 \leq i \leq \delta$, where $V^*_{-1}=0$ and $V^*_{\delta 1}=0$; (iv) there is no subspace W of V such that $AW \subseteq W$, $A^* W \subseteq W$, $W \neq 0$, $W \neq V$. We call such a pair a {\it tridiagonal pair} on V. It is known that $d=\delta$ and for $0 \leq i \leq d$ the dimensions of V_i, V_{d-i}, V^*_i, V^*_{d-i} coincide. We say the pair A,A^* is {\it sharp} whenever $\dim V_0=1$. It is known that if $\K$ is algebraically closed then A,A^* is sharp. A conjectured classification of the sharp tridiagonal pairs was recently introduced by T. Ito and the second author. Shortly afterwards we introduced a conjecture, called the {\em μ-conjecture}, which implies the classification conjecture. In this paper we show that the μ-conjecture holds in a special case called q-Racah..

[40 %] [mark] -> Flavour Physics in the Littlest Higgs Model with T-Parity: Effects in the K, B_d/s and D systems. **Recksiegel, Stefan.**
The Littlest Higgs Model with T parity (LHT) is an interesting alternative model for New Physics at the TeV scale. Although Flavour Physics was not the reason for creating the LHT model, significant effects (such as large CP violation where not predicted by the SM) can be created without violating existing experimental bounds. We study the B-, K- and especially the D-sector..

Figure 3: The monitor.py interface

Another improvement of this tool would be to extend its compatibility with other types of data sources like PubMed for example. This particular example would be straightforward because their XML format can easily be converted to the MARCXML format and a well documented public API called EUtils[38] is available.

VI.2.2 Visualize.py

This application prototype enables to visualize similarity results in the form of a graph. In short, this is a way to assess quickly the relations between records, even in a complex and extended set of documents. Many applications may be derived from this tool. But presently visualize.py is in early stage of development and is provided as an example of this framework's capabilities. It is located in the ./bin folder. This tool may be use it on any similarity output file, typically ./var/log/output.dat .

---

[38] http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html

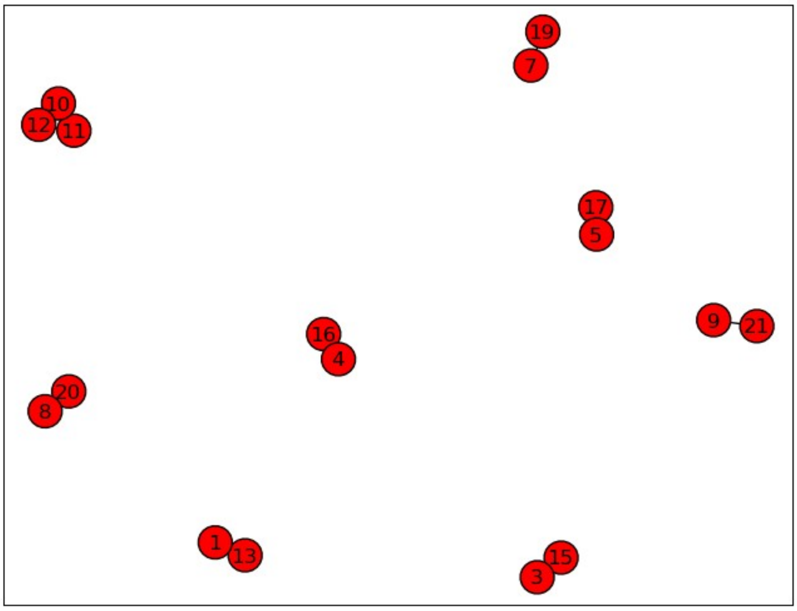


Figure 4: A visual representation of duplicates.

The simplest but not very useful way to use visualize.py is to show graphically potential duplicates, triplicates and so on. A more interesting development would be to add the ability to assign colours to subsets of records based on an analysis of fields related to record contents (like subject headings, decimal classification and/or abstracts), and finally represent everything on a graph. The result would provide would be a quick overview of the domains covered by the collection.

Another use of visualize.py is to assess automatically the links between the services and resources of a library and derive a tool to guide its uses to the resources they need. This has a lot of potential, since many expensive databases are under-used because users simply do not know about them. A quick test of this was conducted on a simplified structure of a medical library. The structures in Figures 5 and 6 was defined semantically using concepts listed in a MARC field, on which a similarity analysis was performed.

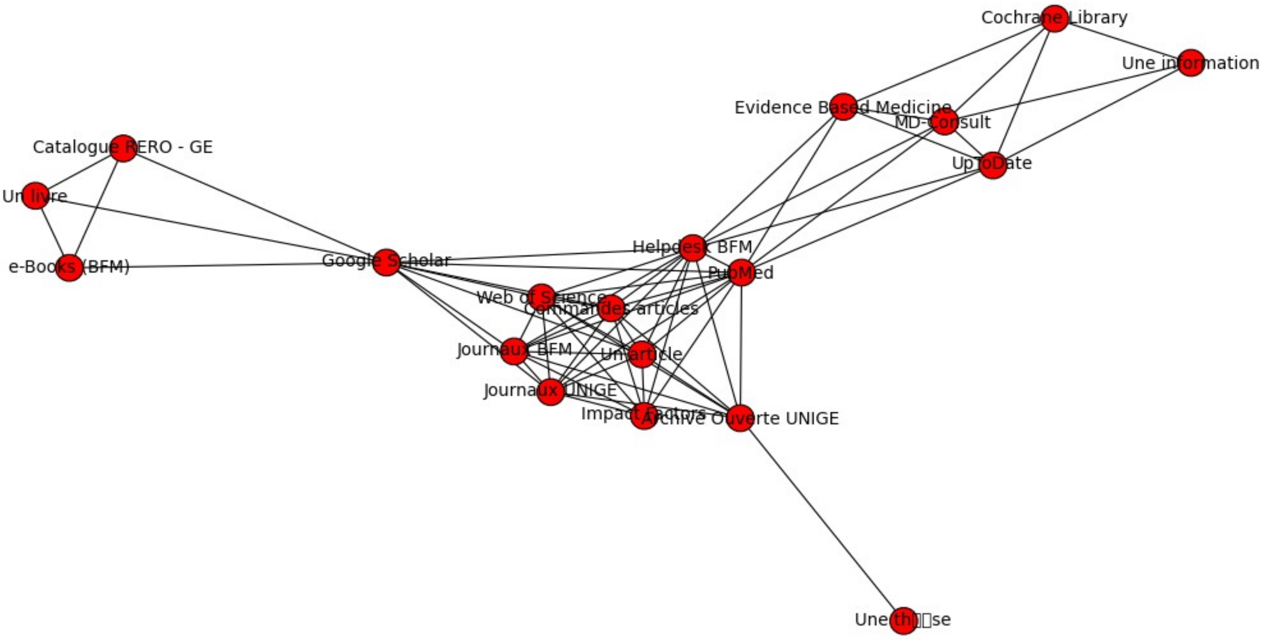


Figure 5: A visual representation of ressources and services of a library.

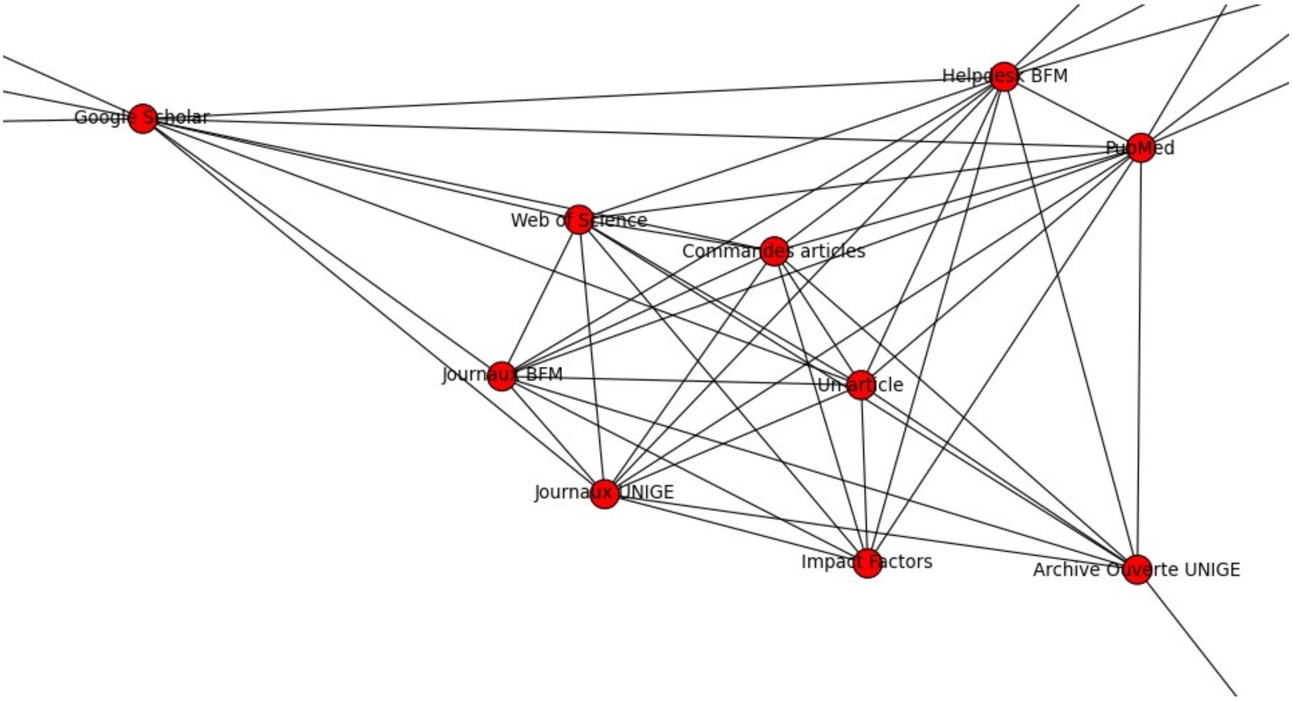


Figure 6: A visual representation of ressources and services of a library: zoom.

Even though this graph is derived of a hastily generated data set, and might at first seem messy, the results are very interesting. In this case, playing with this tool permitted to identify five entry points from which it is possible to access all studied resources of the library in only one click, namely: 'Un article', 'Un livre', 'Une thèse', 'Evidence Based Medicine', and 'Une information'.

A web application could then be devised on that basis and display these entry points. If a user is looking for an article he would move his mouse over 'Un article', and the following options would be suggested to him (these options are represented by the graph's edges):

1. Visit the 'PubMed' bibliographic database
2. Use the Web of Science database
3. Lookup impact factors
4. Order an article on line 'Commandes d'articles'
5. Browse the medical journals collection 'Journaux BFM'
6. Browse the institutional journal collection 'Journaux UNIGE'
7. Open the institutional repository 'Archive ouverte UNIGE'

In this example, only a few services and resources are listed, and already quite a complex network binds them together. In reality, most libraries provide many more services and resources than that. Because resources and services are thematically linked with others, it becomes a real challenge to present them in a comprehensive way on websites. And even if a good solution is found, every time a resource or service is removed or added a lot of updating is required. This tool has the potential automatise these tasks. Besides, MarcXimiL comes with a prototype called semantic.py, that will assist in managing the collections used to generate the graphs (see below).

SYNTAX: `./visualize.py similarity_file_name.dat label_column similarity_column`

- WARNING: the path of similarity_file_name.dat is relative to ./var/log/
- the label column is the one in witch the identifiers of the compared records are stored. The column numeration starts at 0
- the similarity_column is the column this script will use to generate the graph.

Example: `./visualize.py output.dat 0 1`

The NetworkX package is required. On Ubuntu install it using « sudo apt-get install python-networkx ». Binaries are available for Windows on the NetworkX website.

VI.2.3 Semantic.py

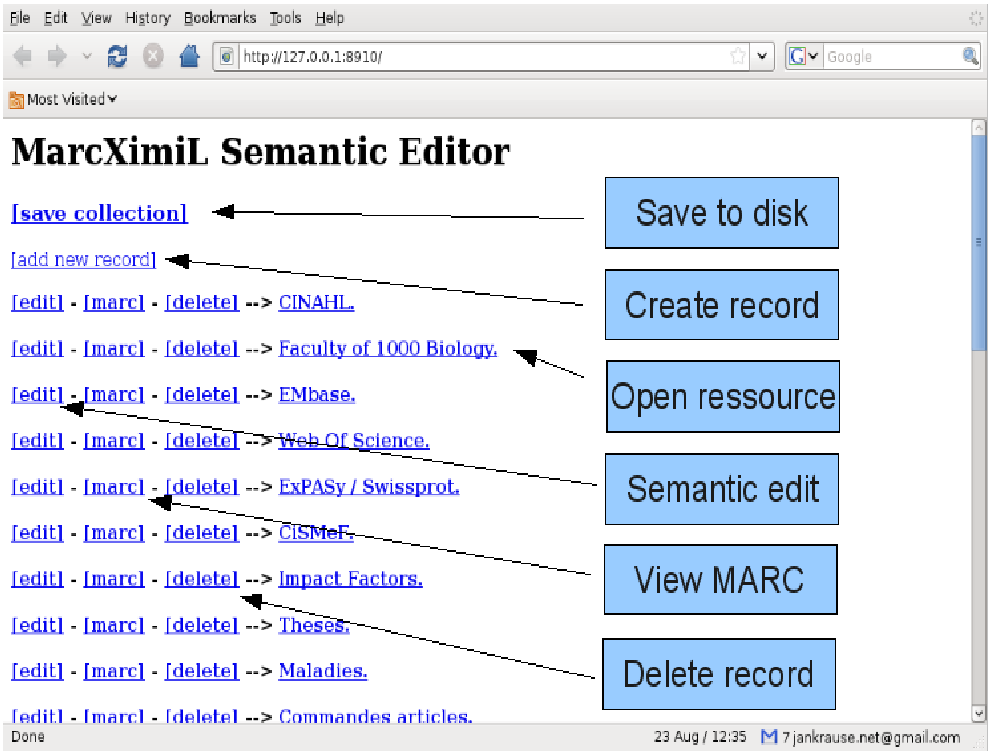
This tool will assist managing MARCXML collections describing semantically linked concepts. The obtained collections may be used to generate graphs using the previously presented `visualize.py` utility.

This is a graphical tool. The interface is served on a local port by an ad-hoc light-weight Python web server. It enables to:

- navigate within a collection
- create records
- delete records
- visualize and edit records in a simple way (no knowledge of MARC is required)
- visualize the records in MARC

Screenshots:

Browsing a collection:



MarcXimiL Semantic Editor

[save collection]

[add new record]

[edit] - [marc] - [delete] --> CINAHL.

[edit] - [marc] - [delete] --> Faculty of 1000 Biology.

[edit] - [marc] - [delete] --> EMbase.

[edit] - [marc] - [delete] --> Web Of Science.

[edit] - [marc] - [delete] --> ExPASy / Swissprot.

[edit] - [marc] - [delete] --> CISMeF.

[edit] - [marc] - [delete] --> Impact Factors.

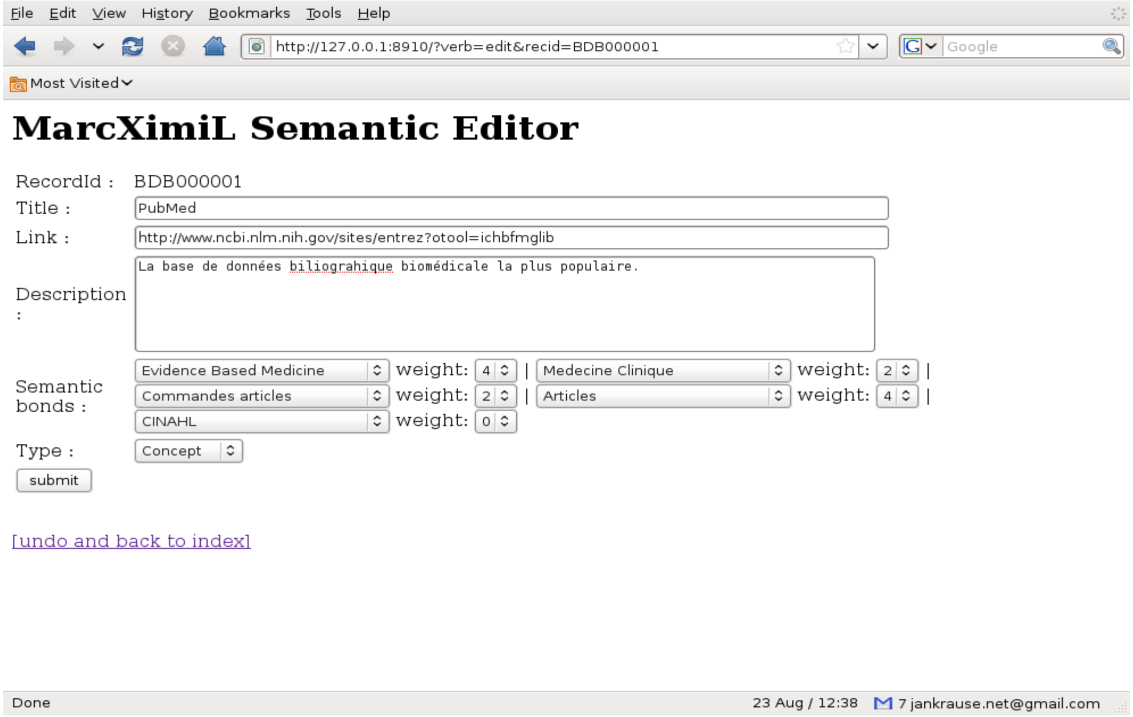
[edit] - [marc] - [delete] --> Theses.

[edit] - [marc] - [delete] --> Maladies.

[edit] - [marc] - [delete] --> Commandes articles.

Figure 7: The semantic.py interface – main window

Editing a record:


File Edit View History Bookmarks Tools Help
http://127.0.0.1:8910/?verb=edit&recid=BDB000001
Google
Most Visited
Done
23 Aug / 12:38
7 jankrause.net@gmail.com

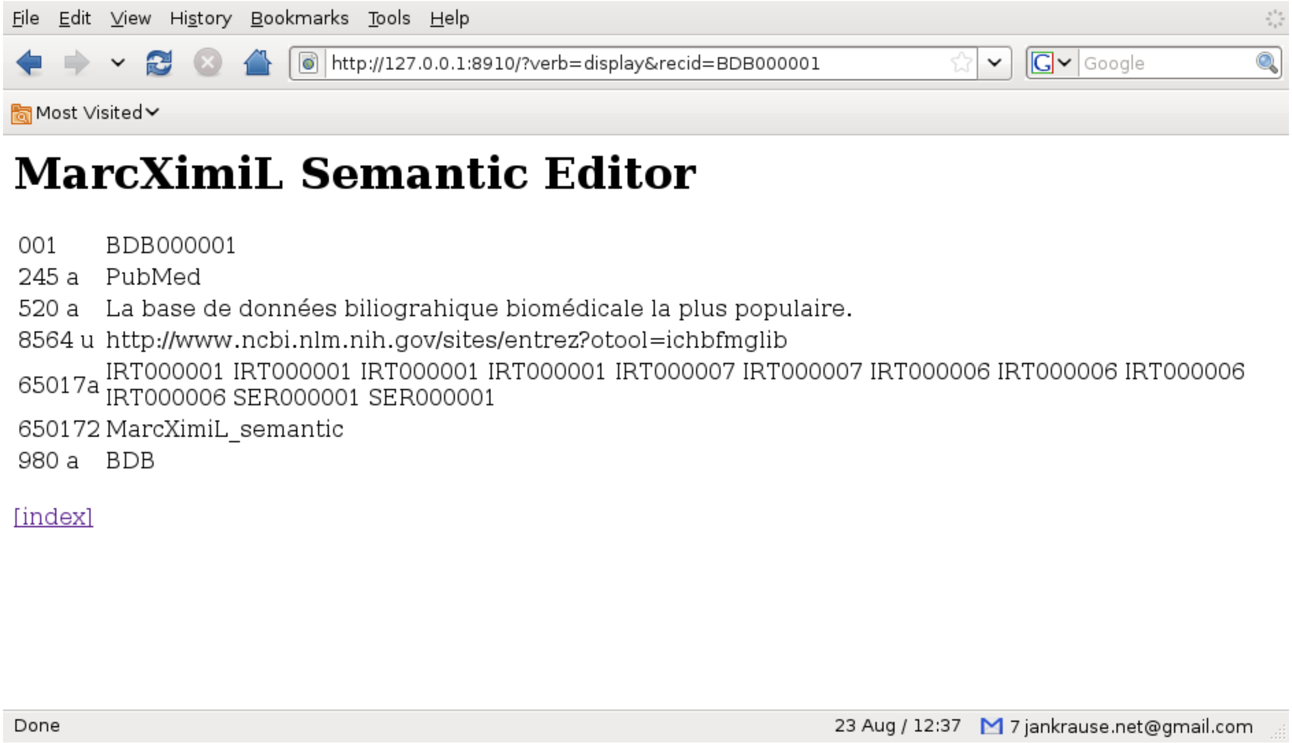
MarcXimiL Semantic Editor

RecordId : BDB000001
Title : PubMed
Link : http://www.ncbi.nlm.nih.gov/sites/entrez?otool=ichbfmglib
Description : La base de données biliograhique biomédicale la plus populaire.
Semantic bonds : Evidence Based Medicine weight: 4 | Medecine Clinique weight: 2 |
Commandes articles weight: 2 | Articles weight: 4 |
CINAHL weight: 0
Type : Concept
submit

[undo and back to index]

Figure 8: The semantic.py interface - editing semantic bonds

Displaying the same record in MARC format:


File Edit View History Bookmarks Tools Help
http://127.0.0.1:8910/?verb=display&recid=BDB000001
Google
Most Visited
Done
23 Aug / 12:37
7 jankrause.net@gmail.com

MarcXimiL Semantic Editor

001	BDB000001
245 a	PubMed
520 a	La base de données biliograhique biomédicale la plus populaire.
8564 u	http://www.ncbi.nlm.nih.gov/sites/entrez?otool=ichbfmglib
65017a	IRT000001 IRT000001 IRT000001 IRT000001 IRT000007 IRT000007 IRT000006 IRT000006 IRT000006 IRT000006 SER000001 SER000001
650172	MarcXimiL_semantic
980 a	BDB

[index]

Figure 9: The semantic.py interface - Displaying MARC.

VI.2.4 Plagiarism.py

As mentioned in the introduction, plagiarism detection is fashionable and universities are gearing up to face that problem. They often chose centralized commercial tools. Such tools require much computing power and massive access to data on the web, and therefore often rely on the powerful infrastructures of web search engines. But the search engines are not precisely adapted for that use, and other more direct approaches could possibly be more efficient. In addition, even with this use of search engines, plagiarism checking with this kind of commercial tools takes time and therefore end users (professors, etc.) will use them only if a document seems really suspicious. This is not optimal.

For those reasons, the path followed in a plagiarism study on arXiv (Sorokina et al. 2006) seems most interesting: all full-text documents were directly compared with each other in one domain of knowledge. Such a plagiarism detection scheme could perhaps be developed and optimized thanks to the MarcXimiL framework. In many cases, the detection could be speeded up in studing the potential plagiarized documents only amongst groups of related document determined using the metadata associated to them. Of course, this approach is only valid in the context of specialized publications within fields that possess quite complete thematic repositories (like arXiv or the future Inspire in high energy physics), or at least well structured metadata databases (like in the biomedical domain with PubMed). This strategy would not be applicable to students exercises at a level that may allow to plagiarise websites such as Wikipedia, Ask, Google Answers, Le Guichet du Savoir, and so forth.

Our technical approach differs of the one that was successfully applied in the previously mentioned arXiv study. While they based their analysis on the document's sentences composition, we use whole paragraphs that are compared with the shingling algorithm and a global output threshold of 0.05.

In order to quickly assess this tool's potential, a knowledge base was built with 12 articles downloaded from the English Wikipedia on August 25, 2009: Voltaire, Rousseau, D'Alambert, Bastille, Charles_XII_of_Sweden, Edward_Gibbon, Geneva, Leibniz, Louis_XV, Pierre_Louis_Maupertuis, Poitou, William_Shakespeare. These articles were selected because they have all connections with the Voltaire article (they are all linked from that article). In total 1090 paragraphs were extracted and stored (very short paragraphs were automatically left aside).

The knowledge base having been set up, the Wikipedia article on Rimbaud was downloaded and tested for plagiarism against it. At this point, most paragraphs were not detected as plagiarized candidates. A couple of them were selected (one in the knowledge base and one in the Rimbaud document). Both paragraphs number 8 met the non-detection requirement and were retained. In the Rimbaud document, paragraph 8 is made of 4 sentences, 110 words, and 724 characters. It was then duplicated and modified 16 times. Modifications 1 to 4 were performed as follows:

1. The first sentence was replaced with the first sentence of paragraph 8 from the Voltaire article.
2. The first 2 sentences were replaced with the first 2 sentences of paragraph 8 from the Voltaire article.
3. The first 3 sentences were replaced with the first 3 sentences of paragraph 8 from the Voltaire article.
4. The first 4 sentences were replaced with the first 4 sentences of paragraph 8 from the article.

Modifications 5 through 8 were applied on top of the first 4 modifications: each sentence that was added previously was fractioned in two around its middle and the parts were per mutated.

Modifications 9 to 12 were applied on top of the first 4 modifications: each sentence that was added previously was truncated of one word.

Modifications 13 to 16 were applied on top of the first 4 modifications: each sentence that was added previously was truncated of two words.

The plagiarism check of this altered Rimbaud document returned 45 suspected paragraphs. Among these 45, the 16 altered paragraphs were returned. The results were sorted and Wikipedia specific noise was removed manually: This noise was due to the following paragraphs: « Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. […] », « Hidden categories: All articles with unsourced statements [...] », « .Wikiquote has a collection of quotations related to:[...] ». After this specific noise removal there were 22 possible candidates left containing the 16 records to find.

Even if the knowledge base was quite small, it shows that plagiarism.py works to a certain extent and that the MarcXimiL framework can be used for plagiarism detection. However this tool remains a prototype and some obvious features are missing, like the ability not to report a possible plagiarism if a rightful citation was made.

Some of the modified paragraphs that were successfully detected as plagiarised in the previous test were also submitted in on-line plagiarism detection tools. Plagium, that looks up Wikipedia through

Yahoo, was able to detect some of the plagiarisms but not all even if the Voltaire article was scanned! The modified paragraphs 1, 5, 9, and 13 were submited and the link to the Voltaire page was not detected in the cases 5 and 13. The same paragraphs were then submitted to SeeSources, that failed only on number 13. In this paragraph, the plagiarised sentence to find was originally made of 22 words, and two of them were removed. This simple modification sufficed to fool both on-line tools. Plagiarism.py did not fail. However the comparison is quite unfair, since plagiarism.py's knowledge base was a lot smaller than the others and a scale effect might very well be involved.

Plagiarism.py supports most popular formats: PDF, DOC, ODT, PPT, ODP, XLS, ODS, HTML, among others. Documents may be loaded from the local file system or downloaded directly from the web if an URL is given. Moreover, a range of tests should be done to optimise recall and precision in that context. This tool may be used as follows:

SYNTAX: `plagiarism.py [[-a] <URI>] [-c]`

NB: <URI> may be an URL or local file name.

OPTIONS

-a : will add data in <URI> to the knowledge base used for plagiarism detection.

EXAMPLES

Perform plagiarism detection:

- `plagiarism.py Thesis.doc`

Add documents to plagiarism knowledge base:

- `plagiarism.py -a Article.ods`
- `plagiarism.py -a Article.doc`
- `plagiarism.py -a Article.html`

Clear knowledge base

- `plagiarism.py -c`

Required:

1. The python_uno module (doc, odt, html, etc. importation through OpenOffice) On Ubuntu: `sudo apt-get install python-uno`
2. OpenOffice (soffice.bin) running in server mode on port 2002. On Ubuntu you may execute

the following command to achive this: `/usr/bin/soffice -headless -invisible '-accept=socket,host=localhost,port=2002;urp;'`

3. xpdf (because pdftotext comes with it). On Ubuntu: `sudo apt-get install xpdf`

VI.2.5 Enrich.py

This little tool is designed to enrich a catalogue with references to similar records.

The principle is quite simple and straightforward : a similarity output file is analysed and only the record pairs that have a global similarity higher than a parametrisable threshold are retained. Then, these record pairs are used to generate a new MARCXML collection in which each record that was retained is present and contains only one customisable field that simply lists all the references of the records that are similar to it.

A tool like the CDS Invenio Library Integrated System comes with out of the box capabilities to update its records on that basis. Practically, it is quite straightforward:

```
bibupload -a collection.xml   # just adds the fields to the existing records
bibupload -c collection.xml   # corrects the fields if present
```

In Invenio, it is then easy to set up display templates using the BibFormat web interface in order to propose a « More like this » section to the end users. NB: Invenio already has that functionality (called Similar Records), but it is a bit slow... this would probably be faster. Anyway, this principle can also be applied to other bibliographic repositories.

VII Conclusion

Using the Python programming language, we have developed a flexible, open-source, multiplatform software tool supporting the implementation of multiple strategies for record comparisons. In order to test our program, we have implemented several strategies for the detection of near-duplicate records and applied them to the analysis of several data sets built from real-world collections (CERN, ETH E-Collection, RERO DOC).

Most of our strategies are capable of finding a duplicate record if only one or two fields were altered reasonably. Reasonable alterations are for example: removing up to one third of the authors or of a title, removing a few sentences in an abstract, changing the publication date of up to two years, any word permutations, punctuation modification, or diacritic variation, variations in the author format, even deletion of entire fields.

Among our strategies, the ones we called *ubiquist* and *initals* yield the best precision in the first 10 results. These strategies are handy to perform a quick duplicate detection. On the other hand, maxim, boundaries max, and abstract fallback have a better recall within the 50 first results. They are therefore adapted to do a more exhaustive but longer detection, involving a careful human examination.

The combination of our three best but unrelated strategies (ubiquist, initials and okapigeom) produces outstanding results: for each test near-duplicate record at least one of them was able to place it in the top 10 results in 96% of the cases. Generally, it seems that the combination of complementary strategies is an excellent approach for an efficient duplicate detection. New strategies might be derived from the combination of ubiquist and initials, possibly with the addition of di/trigrams or Soundex field similarity functions.

On that basis a good recipe to set up a deduplication configuration for any collection might be:

1. Develop or fine tune a few efficient strategies based on different underlying principles. A good starting point would be the strategies shipping with MarcXimiL, as they are flexible and varied.
2. Identify the best strategies that are complementary.
3. Run them in batch (MarcXimiL permits to do that easily)

We have shown that MarcXimiL is an good tool for deduplication, but its speed is not yet sufficient for a routine use on medium to large collections. But it has got a lot more potential than that specific application due to its flexibility. Some applications prototypes packaged within the framework demonstrate its capabilities in information monitoring, detection of plagiarism, visualization of collections in the form of graphs, etc.

Several improvements can and will be made to MarcXimiL. The most important are:

- A speed optimisation. The framework is quite slow on collections of more than 10'000 records. There are many ways to tackle this.
- Adding a graphical user interface is generally useful.
- Adding the ability to use the framework directly within other Python applications might be useful as well.

Alphabetical Index

Index of Tables

Index of Figures

Code Sample Index

Bibliography

Apache, 2009. Apache Lucene - Query Parser Syntax. Available at: http://lucene.apache.org/java/2_4_1/queryparsersyntax.html#Fuzzy%20Searches [Accessed August 16, 2009].

ATS, 2008. Université de Zurich: programmes informatiques anti-plagiat - swissinfo. Available at: http://www.swissinfo.ch/fre/suisse/detail/Universite_de_Zurich_programmes_informatiques_anti_plagiat.html?siteSect=113&sid=8760931&cKey=1203505566000&ty=ti&positionT=37 [Accessed August 23, 2009].

Avram, H.D., 1968. *The MARC pilot project*, Library of Congress.

Bargadaà, M., 2008. Internet: Fraude et déontologie selon les enseignants universitaires. Available at: http://responsable.unige.ch/index.php [Accessed August 23, 2009].

Bogadi, F., 2007. L'Université de Genève prépare un plan d'action contre le plagiat. *Le Temps*.

Bookstein, A., 1980. Fuzzy requests: An approach to weighted boolean searches. *Journal of the American Society for Information Science*, 31(4), 240-247.

CERN, 2009. CERN Document Server. Available at: http://cds.cern.ch/ [Accessed August 16, 2009].

Cousins, S.A., 1998. Duplicate detection and record consolidation in large bibliographic databases: the COPAC database experience. *Journal of Information Science*, 24(4), 231.

Crestani, F. et al., 1998. "Is This Document Relevant?... Probably": A Survey of Probabilistic Models in Information Retrieval. *ACM Computing Surveys*, 30(4), 528-552.

De Melo, V.V. & Lopes, A.D.A., 2005. Efficient Identification of Duplicate Bibliographical References. In *Advances in logic based intelligent systems: selected papers of LAPTEC 2005*. Amsterdam: IOS, pp. 169-176.

Elmagarmid, A.K., Ipeirotis, P.G. & Verykios, V.S., 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 1-16.

Errami, M. et al., 2009. Deja vu: a database of highly similar citations in the scientific literature. *Nucleic Acids Research*, 37(suppl_1), D921-924.

ETHZ, 2009. Home - ETH E-Collection. Available at: http://e-collection.ethbib.ethz.ch/ [Accessed August 16, 2009].

Freire, N., Borbinha, J. & Calado, P., 2007. Identification of FRBR Works Within Bibliographic Databases: An Experiment with UNIMARC and Duplicate Detection Techniques. In *Asian Digital Libraries. Looking Back 10 Years and Forging New Frontiers*. Lecture Notes in Computer Science. pp. 267-276.

Friedl, J.E., 2006. *Mastering regular expressions*, Sebastopol: O'Reilly.

Fuhr, N., 1992. Probabilistic models in information retrieval. *The Computer Journal*, 35(3), 243-255.

GNU Linux Magazine, 2009. Explorez les richesses du langage Python. *GNU/Linux Magazine France*, 40(40). Available at: http://www.gnulinuxmag.com/.

Goyal, P., 1987. Duplicate record identification in bibliographic databases. *Information Systems*, 12(3), 239-242.

Greenstone, 2009. Greenstone Digital Library Software. Available at: http://www.greenstone.org/ [Accessed August 16, 2009].

Gupta, V., 2008. Porter Stemming Algorithm. Available at: http://tartarus.org/~martin/PorterStemmer/python.txt [Accessed August 12, 2009].

Hetland, M.L., levenshtein.py. Available at: http://hetland.org/coding/python/levenshtein.py [Accessed August 4, 2009].

Hylton, J.A., 1996. *Identifying and merging related bibliographic records*. Massachusetts Institute of Technology. Available at: http://portal.acm.org/citation.cfm?id=888609.

IFLA, 1998. *Functional requirements for bibliographic records*, München: Saur.

Knuth, D.E., 1998. *Sorting and searching* 2nd ed., Boston: Addison-Wesley.

Lavanchy, A., 1977. *Fragmentation des alcanes en spectrométrie de masse*. EPF Lausanne. Available at: http://library.epfl.ch/theses/?nr=289

Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10, 707-710.

Lewis, J. et al., 2006. Text similarity: an alternative way to search MEDLINE. *Bioinformatics*, 22(18), 2298-2304.

LoC, 2008. MARC 21 Format for Community Information Data: 00X: Control Fields-General Information (Network Development and MARC Standards Office, Library of Congress). Available at: http://www.loc.gov/marc/community/ci00x.html [Accessed August 4, 2009].

LoC, 2009. MARC 21 XML Schema. Available at: http://www.loc.gov/standards/marcxml/ [Accessed August 4, 2009].

Lutz, M., 2008. *Learning Python*, O'Reilly.

Manning, C.D., Raghavan, P. & Schütze, H., 2008. *Introduction to information retrieval*, Cambridge University Press.

Medline, 2007. Deja vu > Browse. Available at: http://spore.swmed.edu/dejavu/duplicate/ [Accessed August 23, 2009].

Navarro, G., 2001. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1), 31-88.

Noreault, T., McGill, M.J. & Koll, M.B., 1981. *A Performance Evaluation of Similarity Measures, Document Term Weighting Schemes and Representations in a Boolean Environment.*, London: Butterworths.

Porter, M.F., 1980. An algorithm for suffix stripping. *Program*, 14(3), 130-137.

RERO, 2009. bibliothèque numérique RERO DOC: Accueil. Available at: http://doc.rero.ch/ [Accessed August 16, 2009].

Ridley, M.J., 1992. An expert system for quality control and duplicate detection in bibliographic databases. *Program: electronic library and information systems*, 26.

Rivier, A., 2007. *Aide-mémoire d'informatique documentaire*, Paris: Ed. du Cercle de la Librairie.

Robertson, S., 1977. The probability ranking principle in IR. *Journal of Documentation*, 33(4), 294-304.

Salton, G., 1991. Developments in Automatic Text Retrieval. *Science*, 253(5023), 974-980.

Salton, G., Wong, A. & Yang, C.S., 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11), 613-620.

Savoy, J., 1997. Ranking schemes in hybrid Boolean systems: A new approach. *Journal of the American Society for Information Science*, 48(3), 235-253.

Sorokina, D. et al., 2006. Plagiarism detection in arXiv. In *Sixth International Conference on Data Mining (ICDM 2006)*. pp. 1070-1075.

Sparck Jones, K., Walker, S. & Robertson, S., 2000a. Probabilistic model of information retrieval: Development and comparative experiments. Part 1. *Information Processing and Management*, 36(6), 779-808.

Sparck Jones, K., Walker, S. & Robertson, S., 2000b. Probabilistic model of information retrieval: Development and comparative experiments. Part 2. *Information Processing and Management*, 36(6), 809-840.

Tecon, P., 1979. *La fragmentation des alcènes en spectrométrie de masse*. EPF Lausanne. Available at: http://library.epfl.ch/theses/?nr=333.

Wong, S.K. et al., 1987. On modeling of information retrieval concepts in vector spaces. *ACM Transactions on Database Systems*, 12(2), 299-321.