Deduplication of Aggregation Authority Files

P. Manghi

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo",
Consiglio Nazionale delle Ricerche, Pisa, Italy,
Fax: +39 050 315-3464,
E-mail: paolo.manghi@isti.cnr.it,
*Corresponding author

M. Mikulicic

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo",
Consiglio Nazionale delle Ricerche, Pisa, Italy,
Fax: +39 050 315-3464,
E-mail: marko.mikulicic@isti.cnr.it

C. Atzori

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo",
Consiglio Nazionale delle Ricerche, Pisa, Italy,
Fax: +39 050 315-3464,
E-mail: claudio.atzori@isti.cnr.it

Abstract:
This paper presents PACE (Programmable Authority Control Engine), an authority control tool conceived to maintain "aggregation authority files". These are obtained as continuous aggregations of records originating from a variable set of information systems with heterogeneous and duplicated content. To facilitate record deduplication in the presence of such heterogeneity and dynamicity, PACE user interfaces enable an iterative curation process, where data curators can: (*i*) configure algorithms for the identification of record duplicates; (*ii*) open *work sessions* where algorithm configurations can be run and evaluated; (*iii*) *merge* the identified record duplicates to disambiguate the authority file, and (*iv*) repeat this cycle several times. PACE supports a tunable probabilistic similarity measure and performs record matching with a customizable variation of the *sorted neighborhood* heuristic. Finally, it addresses the underlying performance and scalability issues by exploiting multi-core parallel processing and Cassandra's storage systems, to support I/O performances that scale up linearly with the number of records.

Keywords: authority control; record deduplication; record merge; record aggregations; sorted neighborhood; candidate identification; performance; scalability; Cassandra.

Biographical notes:
Paolo Manghi received his PhD in the year 2002 from the Dipartimento di Informatica of the University of Pisa, Italy. He is presently working as a researcher at the Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo", Consiglio Nazionale delle Ricerche (CNR) of Pisa, Italy. His research interest include Data Models for Digital Library Management Systems, Types for Compound Objects, data curation in Digital Libraries, service-oriented ICT infrastructures with special focus on data ICT infrastructures.

Marko Mikulicic received his bachelor degree in Computer Science in the year 2002 at the SUPSI, Switzerland. He works as technical research fellow at the Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo", Consiglio Nazionale delle Ricerche (CNR) of Pisa, Italy. His research interest includes data modeling and development of Digital Library Management Systems, system administration of research infrastructures, and design and development of service-oriented architecture middleware for data infrastructures.

Claudio Atzori received his master degree in Tecnologie Informatiche in the year 2009 at the Universitá degli studi di Cagliari, Italy. He works as graduate fellow at the Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI), at the National Research Council. His working scope include Digital Library Management Systems, Service-oriented Data Infrastructures, and NoSQL data models.

1 Introduction

Authority control is the combination of software resources and human actions required to maintain *authority files*, which are lists of *authoritative records* uniquely and unambiguously describing corresponding sets of entities in an information system (Tillett, 2003). In the library world, traditional examples of such lists are authors of publications or publications themselves.

Authority control is essential to provide information systems with certified digital representations for entities and thus leverage exact references, search and browse functionalities. It is a data curation process, whose goal is to addresses two important aspects of information representation: *entity deduplication* and *entity association*. Deduplication is the action of making sure that distinct records, with different identifiers, indeed represent different entities. Association is the action of linking different records to indicate they are "variations" of the same entities, e.g., two author names are pseudonyms of the same person. The function of authority files is more organizational than informational, that is they contain the minimal information required to disambiguate and associate entities and do not generally include information useful for other purposes, such as end-user reading.

Managing authority files is typically a semi-automatic process, which requires significant human effort and technology cost (to the point that the trade-off with the benefits is often a valid debate). It is a continuous process, whose quality is affected by the expertise of the data curators, e.g., librarians for catalogues, as well as by the overall evolution of knowledge in the field of application. The curation process is supported by authority control tools, which typically provide for record matching algorithms that are both *effective*, to maximize precision and recall, and *efficient*, to optimize performance over arbitrary numbers of records. In general, no "best matching algorithm" can be found since different authority files bear structures and semantics which require dedicated solutions. As a consequence, such tools should also allow for an easy-to-use degree of *customization* of the record matching algorithms (e.g., record matching heuristics, similarity functions, and their parameters), in order to be applicable in different contexts.

Today's multidisciplinary and international society is characterized by a strong will to share knowledge and information and is facilitated in this task by a number of standard APIs and formats. Known examples of these are: Linked Data (http://linkeddata.org), which defines an open standard to expose information through RDF graphs, and the Open Archives Initiative protocols: OAI-PMH (Lagoze and Van de Sompel, 2001) (Carl Lagoze and Herbert Van de Sompel, 2003) and OAI-ORE (Lagoze and Van de Sompel), which define APIs and formats to access digital knowledge bulk-wise, as "sets of metadata records", or point-wise, as "web aggregations". In a scenario of accessible data sources, it is often the case that data spread across several information systems needs to be aggregated to become one unambiguous source of information. In this direction, several attempts to construct and maintain in synergy authority files about people or other kinds of entities have been made worldwide (Tillett, 2003). Among the most known initiatives is VIAF (Virtual International Authority File) (Rick et al., 2007), LEAF (Linking and Exploring Authority Files) (Weber, 2003), the Digital Author Identifier (DAI, http://www.narcis.nl), and the projects OCLC-CORC (Connexion, http://www.oclc.org/connexion), NACO and SACO (Cornell University Library, http://www.loc.gov/catdir/pcc/naco/naco.html), which investigated solutions on how to tackle authority control on global scale.

Many of such initiatives need to maintain *aggregation authority files*, obtained by integrating records originated from several information sources. Such files are characterized by sheer number of records (e.g., up to tens of millions), semantic and context heterogeneity of the incoming records, the "short" authoritative values that these generally contain (e.g., name, surname, date of birth for people), frequent bulk records updates, and possibly high rates of duplicates. Frequent updates and heterogeneous record semantics call for an "interactive" and customizable authority control activity made of cycles where data curators can tune and evaluate several record matching strategies, and commit or roll-back deduplication actions. To offer and interactive experience, performance plays a crucial role. Duplicate detection heuristics (e.g., record blocking/clustering techniques) can reduce computational complexity by reducing the number of matches, but may worsen precision and recall due to the presence of false negatives (Elmagarmid et al., 2007). On the other hand, heuristic restrictions may be loosened thanks to the adoption of adequate storage technologies, whose I/O rates and indexing techniques are capable of coping with higher number of records (i.e., disk reads) and duplicates (i.e., disk writes).

Authority tools capable of coping with aggregation authority files must therefore include a front-end supporting data curators with a framework for record matching algorithm customization, and a back-end capable of withstanding the level of performance envisaged. In this paper we present the architecture and the current implementation of PACE (Programmable Authority Control Engine) authority control tool, which extends the one presented in (Manghi and Mikulicic, 2011). PACE is open source and designed to address maintenance of aggregation authority files. Its realization was made necessary in real-case data infrastructures operated by the European Commission projects European Film Gateway (http://www.europeanfilmgateway.eu) and OpenAIRE (http://www.openaire.eu), targeting aggregation of movie data archives and institutional bibliographic repositories. Both infrastructures required part of the

aggregated data to become an authority file (persons and movies for EFG, persons and bibliographic records for OpenAIRE). To this aim, PACE is designed to support a framework where record matching algorithms can be easily customized, similarity functions plugged-in, and authority files can be easily personalized, continuously populated and curated with optimal performance and no development effort.

PACE user interfaces deliver a framework where data curators can create and maintain their authority files. Its novelty is that curators can conduct continuos actitivies of population, customization and evaluation of record matching algorithms, commit of a set *field-grained* record merges, and possibly roll-back to past commits. Record matching is based on an variation of the "sorted neighborhood" technique (Hernández and Stolfo, 1995) aiming at improving recall by reducing the number of false negatives. The complexity introduced by the algorithm is compensated by PACE's back-end. The back end copes with storage and processing scalability of high number of records, hence with the potentially high I/O costs implied by sorting, reading, and writing million of records, by relying on multi-core parallelism and Cassandra's storage technology (Lakshman and Malik, 2010). The back-end also exposes authority files to third-party consuming systems through APIs delivering on request authority file records or history changes.

Outline of the paper. The paper is organized as follows. Section 2 introduces some nomenclature on authority files and defines aggregation authority files. Section 3 details the main features behind authority control tools, with references to existing solutions, pointing out the lack of reusable and open source tools for aggregative authority file control. Section 4 presents functional requirements, data model, and architecture, while Section 5.2 describes the implementation of PACE and provides performance statistics. Section 6 compares PACE with other existing tools, to draw the differences and highlights its peculiarities. Finally, section 7 summarizes the paper outcomes and future issues.

2 Aggregation Authority Files

Nowadays, the strong need of sharing information across independent information systems, makes the construction and maintenance of authority files more and more important (Gorman, 2003). Moreover, such files are not necessarily focused on people names or publication titles as it was in traditional digital library cataloguing settings, but can involve any set of entities of an information system's data model (e.g., journal titles, research organizations, geographic profiles, research subjects (Dalrymple and Young, 1991), biological species), as long as such set requires deduplication to improve search precision and valid references.

Aggregation authority files are a peculiar category of authority files, urging from the multitude of data aggregation initiatives worldwide, such as data infrastructures or archive and institutional repository federations (Artini et al., 2009). They are obtained from the union and continuous integration of records representing entities of the same conceptual class, i.e., with the same "structure", originated from a dynamic set of *information systems* which typically participate to the authority file population in order to disambiguate their corresponding class of entities or/and to align it with others benefiting from the resulting aggregated and deduplicated collection of entities.

In particular, information systems data models are heterogeneous and may not necessarily contain entities which exactly match the authority file entities. Typically, systems feed the authority file with records respecting the authority file structure but are generated from the local database thanks to a structural and semantic mapping from local entities to authority file entities. The action may be straightforward, when a local class of entities "subclasses" the authority file entities, or be more complicated when the information systems have no corresponding class of entities – this is the case when authority file entities are obtained from properties of other entities, for example consider person records obtained from bibliographic records.

The peculiarity of aggregation authority files is therefore due to the variegated characteristics of the feeding information systems:

- *Non-disambiguated records*: records from the same information system may have duplicates;
- *Heterogeneous records*: records from the same information systems are not necessarily semantically uniform;
- *Alive records*: data in the information systems grows in time, hence new records are continuously fed to the authority file;
- *Non-uniform feeding of records*: systems may be able to feed the authority file "incrementally", which means only new records or changes to old ones are transmitted, or by "refresh", which means only the whole record collection can be transmitted;
- *Systems volatility*: systems may join and leave the aggregation any time.

The dynamic and autonomous nature of information systems severely limits the semantic assumptions that can be generally done on the aggregation authority files they feed and therefore affects the process of record deduplication. Contextual information about entities may be used to facilitate deduplication of entities from the same information system or across several systems whose records share similar contextual semantics. Context information may be obtained by

querying an information system to extract information from the entities which are linked (more generally, "reachable from") to the entities to be disambiguated. Since information systems are fully autonomous and not directly accessible from authority control tools, context selection is delegated to the local systems and the notion of context is modeled by enriching the authority file structure with *context fields*. Their purpose is to enrich the record structure with "flattened" encodings of record's context.

An aggregation authority file can be described as a set of authority records conforming to a uniform flat structure $(l_1 : K_1, \ldots, l_n : K_n)$, where l_i is a field label and $K_i = (T_i, M_i, U_i)$ expresses the "kind" of the field in terms of its type T_i (e.g., string, date, number, boolean), its multiplicity M_i (i.e., $[0 \ldots 1]$, $[1]$, $[0 \ldots m]$, $[1 \ldots m]$), and its usage U_i, namely an *identifier*, *heading*, *context*, or *explanation* field:

Identifier field: each record must have one such field, containing a code that uniquely identifies the record within the authority file;

Heading fields: fields describing properties of the entity represented by the record (e.g., for person entities: name and date of birth; for publication entities: language and script);

Context fields: fields capturing information about the context of the entity in a given application scenario (e.g., for person entities: list of own's publication titles; for publication entities: citation references to other publication entities), aimed at enriching the record with local context-disambiguating information; one record may feature context fields relative to different contexts;

Explanation fields: in some cases, for the sake of cataloguing, it can be necessary to introduce fields whose purpose is to hold information, automatically or manually generated, aimed at justifying the current content of the heading fields. For example, why the name of an author is kept in the form "D. Alighieri", rather than "Dante Alighieri"; this may be due to the original source of the name, an article or book citation.

For example, an authority file for people entities may have the following structure:

```
( personID: (number, [1], identifier),
  firstName: (string, [1], heading),
  lastName: (string, [1], heading),
  birthDate: (date, [0...1], heading),
  country: (string, [0...1], heading),
  paperTitles: (string, [0...m], context),
  source: (string, [0...1], explanation)
)
```

An authority record conforming to this structure may be:

```
[ personID: 123456789,
  firstName: "Charlotte",
  lastName: "Summann",
  birthDate: 10/7/1939,
  country: "CH",
  paperTitles:
      {"First order logics of ants",
       "Squirrels: the dawn of mathematics"},
  source: "information system XYZ"
]
```

In some cases, authority control also keeps track of semantic relationships between deduplicated records to determine the "degree of disambiguation" between them. Examples are: "equivalence" relationships, which link an authority record with another record that is not authoritative, but whose content is a valid alternative to the authoritative record (e.g., records relative to pseudonyms of a given author); "deprecation" relationships, which link an authority record with the records that have been deprecated in favor of the former because of merge actions. Depending on the application domain's context, other types of relationships may be introduced.

3 Authority Control Tools for Aggregation Authority

Authority control is an activity operated by a group of expert data curators, whose task is to ensure disambiguation of the set of records by avoiding and resolving *record duplication*, which is when two records represent the same entity. Duplication issues are addressed by *merging* (also referred as *deduping*) actions, whose consequences are to collapse the two records into one. The average cardinality of authority files and the rather machine-only-detectable reasons which may indicate the need for merging two records, make authority control a task impossible for humans without adequate technological support.

Authority control tools are systems supporting a group of data curators with automatic *candidate identification techniques* capable of efficiently spotting record pairs candidate for merging in authority files of arbitrary dimension (Winkler, 2006). Candidate identification techniques are based on two main concepts: (*i*) *similarity functions*, which return a $0 \ldots 1$ similarity distance measure between two records (with 1 they are equal), and (*ii*) *record matching algorithms*, which cope with the optimization of otherwise $O(n^2)$ complexity required to compare all pairs in a collection of n records. Such techniques are delivered to data curators through user interfaces for *managing* authority files (i.e., creating, updating, and deleting authority files and their records) and *drafting* newer and more disambiguated versions (i.e., configuring and running candidate identification processes, record merging, and commit new versions). Finally, authority control tools may support third-party

information systems with APIs for *consuming* a version of an authority file.

In the following we shall present the main functional requirements which authority file tools designed to maintain aggregation authority files should address. In particular, we focus on record candidate identification and management and drafting of authority files.

3.1 Candidate Identification

Candidate record pairs are those who might allegedly entail merge actions. Typically two records are regarded as *candidates for merging* if some "similarity function" $0 \leq F_S \leq 1$ calculated over the fields of the two goes beyond a given threshold; the action is also called *record linkage* (Winkler, 2006), *object identification* (Tejada et al., 2001), and *entity resolution* (Benjelloun et al., 2005).

Designers and implementers of candidate identification techniques must face two main challenges, efficacy and efficiency (Koepcke and Rahm, 2010):

Effectiveness: identification of record matching algorithms and similarity functions which at best capture record pairs candidate for merging, thereby maximizing recall and precision;

Efficiency: adoption of record matching algorithms tackling the inherent $O(n^2)$ complexity of record matching phase and implementing such algorithms based on storage solutions tailored to address the specific I/O issues raised by such algorithms.

Similarity functions. The similarity measure of two records is a function F_S that calculates the "distance" between the two, i.e., the likelihood for these records of being mistakenly used to represent two distinct entities while they are instead representing the same one. Approaches split in "deterministic" ones, i.e., yes/no rule-based detection of duplicates, or "probabilistic" ones, where a "similarity measure" is assigned to each pair of records, quantifying their likelihood of duplication (Elmagarmid et al., 2007).

In the context of aggregation authority files, where field values are generally information-poor, the probabilistic approach is more appropriate. The similarity function of two records is often obtained by combining record field similarity functions associated to the record heading and context fields and depending on their value domains, be them dates, person names, subjects, etc. Examples of record field similarity functions are string-based functions focusing on variants of the "typographical error" problem (e.g., Jaro-Winkler (Winkler, 1990), Cohen's variants of the TFIDF metrics (Cohen et al., 2003), Edit distance, Biagram distance) or the "person name and address matching" problem (Christen and Zhu, 2002) (Churches et al., 2002). Other methods focus on special cases of the problem, such as identifying short texts duplicates (SimFinder (Gong et al., 2008)); when richer textual information is available, near-duplicate functions, typically used for documents and web pages duplicates (SimHash (Charikar, 2002), (Manku et al., 2007)) can be used.

Record matching techniques. The complexity upper bound for the identification of all possible candidate record pairs is $O(n^2)$, where n is the number of records in the authority file. Ideally, the function F_S is calculated over all possible record pairs and the results are then sorted in descending ordering by distance, to let the best hits surface. Due to the amount of records involved in aggregation authority files, possibly scaling up to tens of millions, and the potentially high number of duplicates, the time for such calculation as well as the time for the subsequent storage and sorting of all pairs might be unacceptable and thus hinder a smooth authority file management life-cycle, made of frequent cycles of candidate identification. The technical challenge is dual: (*i*) finding methods and algorithms capable of reducing complexity while identifying candidates with minimal false positive/negatives, and (*ii*) deliver implementations capable of scaling up with the number of records and the I/O costs needed for reading, writing, and sorting by well balancing between RAM and disk operations.

Two main categories of methods and algorithms have been proposed. The first one includes "clustering" techniques, where authority file records are grouped based on some common pattern which may pre-suggest them as candidates for merges. The candidate identification process is then applied to the clusters, thereby reducing the overall complexity. Examples of this are blocking (Jaro, 1989), Sliding Window (also called Sorted Neighborhood) (Hernández and Stolfo, 1995), Canopy Clustering (Cohen and Richman, 2002), Bigram Indexing (Christen and Pudjijono, 2009), and R-Swoosh (Benjelloun et al., 2005). All such approaches decrease the complexity, but may bring in a loss of recall and precision due to the likelihood that true candidate record pairs have been excluded from the matching.

The second category includes approaches based on similarity function properties, assumptions to be made on the domains of the authority file record fields (e.g., vocabularies, common terms), or the data distribution that these may feature. Based on such properties and assumptions, record matching algorithms can avoid classes of string comparisons, thereby reducing the computational cost of the process. Examples are Text Joins (Gravano et al., 2003), Fuzzy Match (Chaudhuri et al., 2003), and the Comparison Store (Gómez-Bao et al., 2009).

Being dependent on application domain preconditions, approaches in the second category tend to be not applicable to handle aggregation authority files where such assumptions are often not possible. Vice versa, approaches in the first category are generally effective, if delivered to data curators together with interfaces for customizing both record matching algorithms and similarity functions. In any case, regardless of the theoretical complexity, implementing

such approaches requires choosing the adequate storage technology, in order to offer input and output rates in line with the expected usage of the authority control activities at hand. Dedicated indexing structures, distributed databases, sorted storage techniques, multi-core parallel processing, are examples of solutions in this context.

3.2 Authority File Management

Aggregation authority files are characterized by frequent bulk record updates of highly semantically heterogeneous records. This is due to the lively and heterogeneous nature of the information systems feeding the file and to the fact that systems may join or leave the aggregation in any moment. This leads to a "chronically" heterogeneous federation, where not only information systems may not share common semantics, but new unpredictable semantics may be introduced whenever new systems join in. As a consequence, the process of candidate identification cannot rely on stable similarity functions nor record matching algorithms. The overall candidate identification process is affected by the semantics of the information systems currently feeding the records and how this relates with the semantics established for the authority file.

To cope with these issues, authority control tools must provide data curators with highly configurable user interfaces easing the process of management and drafting of authority files. On the one hand, such interfaces must provide and intuitive level of abstraction over low-level record matching and similarity functions configuration. On the other hand, they must enable an interactive behavior, with which data curators can experiment with several candidate identification configurations in temporary work sessions before effectively applying and committing record merges. More generally, we can envisage that authority control tools for aggregation authority files should cover the following functionality: *customizability*, *drafting*, and *third-party consumption* – see (Koepcke and Rahm, 2010) for a summary of deduplication and record linkage functionalities.

Customizability. Authority file tools can be used for maintenance of authority files of different structures and semantics, where effectiveness and efficacy require different similarity functions, different parameters for both such functions and the record matching algorithms they adopt. To this aim, tools should provide easy-to-use user interfaces, through which data curators can create their authority files by specifying record structure (i.e., identifier, heading, context, explanation fields) and record semantics (i.e., value domain of each field). Moreover, data curators must be able to personalize their candidate identification process by configuring record matching algorithms and similarity functions with minimal or none development effort.

Drafting. As illustrated in Figure 1, authority file tools enable drafting of new versions of authority files in temporary *work sessions*. Within a work session, as shown in Figure 2, data curators can perform a number of actions, which may include addition, deletion, update, and bulk-feed of records. Most importantly, they can iteratively run candidate identification processes relative to different configurations, to grow a list of candidates under different perspectives, and possibly apply *field-grained* merge actions. Indeed, merges may not be limited to the choice of the predominant record out of a candidate pair, but also offer the possibility of selecting which field-value pairs of the two records should be included in the predominant record. When satisfied, experts may decide to *commit* the work session, that is to make the changes authoritative and generate a new version of the authority file.

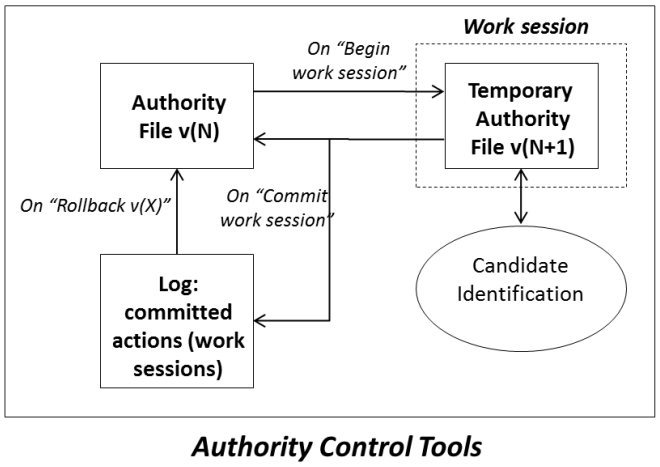

Figure 1 Opening, committing, and rolling-back work sessions

Some tools keep track of the actions performed since the last commit, in order to support *roll-back* of committed authority files, incremental consumption (as described below), and automatic deduplication. The latter is a useful and fundamental optimization, which consists in automatically applying merging actions relative to candidate record pairs which had been previously committed by data curators. This is often the case when information systems can only feed the authority file in "refresh mode" (see Section 2). Without adequate logging support, authority file administrators would be faced, at every information system refresh, with the same candidate list to be merged.

On the other hand, as mentioned in the introduction, some information systems may feed records who do not carry unique identifiers or, more generally, cannot guarantee the persistence of the identifiers they provide. The absence of persistent identifiers makes it impossible to log administrator's merging actions. In such cases, the solution is often to create identifiers from record field values through hash functions and the challenge is to identify sufficient contextual information to

make this identifier effectively unique, hence usable to automatically detect and repeat merge actions.

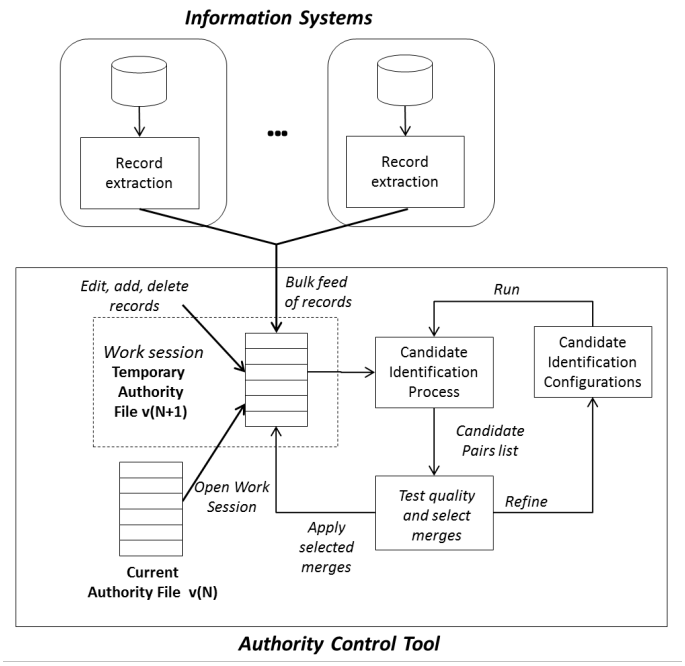

Figure 2 Iterative maintenance process within a work session

Some authority file tools support self-tuning techniques, capable of learning from data curator behavior and from the actual content of records (e.g., training sets) to automatically refine the configuration of record matching algorithms and/or similarity functions. Examples of systems providing for self-tuning techniques are STEM (Self Tuning Entity Matching) (Koepcke et al., 2008) and Febrl (Freely Extensible Biomedical Record Linkage) (Christen, 2008). Due to the heterogeneity and dynamicity of aggregation authority files, such techniques do not apply very well in this context, where assumptions on the record semantics or distribution and typologies of the errors that led to record duplication are hard to make.

Consumption. Aggregation authority files are formed so as to be used by third-party systems to synchronize and align to a common semantics. As such, the relative maintenance tools should implement APIs to support and facilitate third-party system access. Ideally, two access modalities may be envisaged: *in-full*, i.e., fetching the full authority file as available at a given commit time, or *incrementally*, i.e., by fetching the actions executed after a given commit time. Some authority tools may offer on-demand candidate identification functionalities, through which third-party systems may request the list of matching (when logs about committed merges are kept) or candidate authority records relative to an input list of records.

4 PACE (Programmable Authority Control Engine)

In this section we present PACE (Programmable Authority Control Engine), an open source authority control tool designed to offer out-of-the-box and fully-fledged authority file management functionality. The realization of the tool was motivated by the real-case data infrastructures resulting from the European Film Gateway project and the OpenAIRE plus project, where large aggregation authority files of people, movie, and bibliographic records had to be maintained. PACE has been designed to offer the functionalities described in Section 3, as such, it leaves administrators the ability to customize (the structure and semantics of) their authority files, configure their preferred candidate identification settings, and enable an interactive and iterative curation life-cycle. In the following, we shall describe the data model behind PACE, together with the management, drafting, and consumption functionality it implements. Secondly, we shall illustrate the candidate identification process at the core of PACE, that is the record matching algorithms and the similarity function it provides.

4.1 Management of authority files

PACE introduces two levels of drafting, reflected by the conceptual data model depicted in Figure 3. On the first level, data curators (i.e., *owners*) can create one or more authority files and grant to a set of colleagues (i.e., *administrators*) the right to curate such files. In particular, given an authority file, administrators can define one or more *candidate identification configurations* to be used to identify possible record merges.

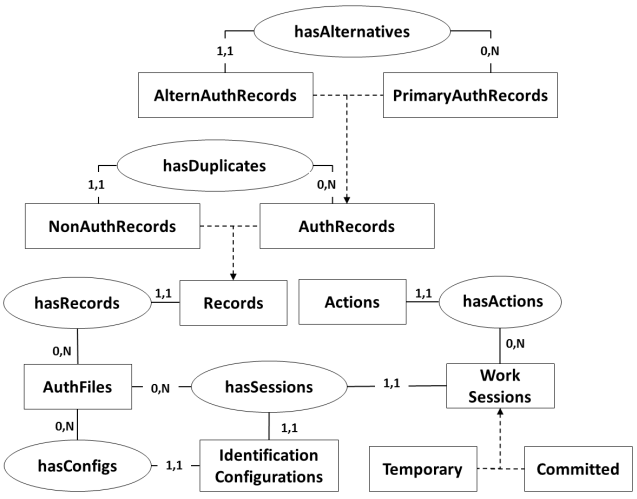

Figure 3 Conceptual data model

On the second level, that is the context of work sessions, administrators of one authority file can apply the actions illustrated in Figure 1 and Figure 2. They

can open multiple *work sessions*, each associated to a given candidate identification configuration. In a work session they can draft a new version of the authority file by adding, deleting, updating and bulk-feeding records. They can run the candidate identification configuration over the authoritative records, visualize the candidate list (Figure 4), and perform merge actions (Figure 5). In PACE, merge actions do not simply consist in identifying which authoritative record of a candidate pair will be deprecated. Precisely, as reflected by the data model in Figure 3, merge actions cause one record of a candidate pair to "prevale" over the other in two possible ways:

- *Duplicate merge*: the records are effectively redundant representations of the same entity, hence one record becomes *authoritative* while the other becomes *non-authoritative*, i.e., deprecated (see Figure 3); a semantic relationship "duplicateOf" is created between the two.
- *Alternative merge*: the records are different and valid representations of the same entity (e.g., the author's pseudonym and the author'd name), hence they are both considered authoritative, but one of the two becomes the *primary* record, while the other the *alternative* record; a semantic relationship "alternativeOf" is created between the two: if the new primary record was alternative of an existing primary record, the relationship is created between the new alternative record and the existing primary record; if the new alternative record was primary of existing alternative records, then these and the new alternative record will point to the new primary record. As a result, "alternativeOf" relationships define clusters of alternative records equivalent to one primary record.

In addition, if the records are to be merged as duplicates, data curators can select which field-value pairs of the non-authoritative record must be added to the authoritative record and which fields-value pairs of the authoritative record are to be removed (see figure 5);

Merge actions cause the authority file to temporarily change and shape up into a new list of primary or alternative authoritative records. When a session is *committed* (*i*) all open sessions are canceled, (*ii*) all actions executed in the session are applied to the last committed authority file, and (*iii*) the actions are persisted into the *authority file log*. Optionally, committing a session may require the approval of a minimal quorum of the data curators in charge of the authority file ("voting mode"). As described in Section 3, action logs keep track of all actions executed on committed sessions and by whom, thereby enabling incremental consumption, automatic re-deduplicaiton, and rolling back the status of a committed authority file to a past committed version. In particular, logs model actions by tracking the following information: *time-stamp* (time of commit), *work session identifier*, *administrator identifier* (responsible of the action), *type of merge* (duplicate or alternative), *prevailing record identifier*, *secondary record identifier*, *identifiers of added field-value pairs*, and *identifiers of removed field-value pairs*. Addition, update, and removal actions are modeled as distinct log entries, which keep track of *time-stamp*, *work session identifier*, *administrator identifier*, *type of action* (add, update, delete), and specific parameters of the action.

Finally, PACE implements and exposes APIs for (authorized) third-party consumers to access authority files of interest. Authority files (referred by a unique identifier) can be accessed in two modalities: *full access*, i.e., the list of records at a given commit time t, and *incremental access*, i.e., the list of actions committed after commit time t, obtained by accessing the relative logs. The list of records can be exported in XML custom format or RDF formats, e.g., Linked Data's (http://linkeddata.org). The list of changes are exported as an XML file or as a text file.

4.2 Candidate identification

PACE implements user interfaces for data curators to create and thus become owners of authority files, by providing a name, a description and providing the flat record structure $(l_1 : K_1, \ldots, l_n : K_n)$, as defined in Section 2. Owners can give the permission to maintain files they own to a number of administrator users, which can manage a set of candidate identification configurations for such files and operate maintenance actions over them. PACE implements a record matching algorithm based on an iterative application of the "sorted neighborhood" (Hernández and Stolfo, 1995) heuristic and adopts a notion of probabilistic record similarity. Configurations consist in the definition of parameters required to configure the record matching algorithm and the parameters needed to define the similarity function.

Similarity function. In PACE, given two records r_1 and r_2 of an authority file with structure $(l_1 : K_1, \ldots, l_n : K_n)$, their similarity function $F_S(r_1, r_2)$ is defined as the weighted mean:

$$F_S(r_1, r_2) = \sum_{i=1}^{n} (f_i(r_1.l_i, r_2.l_i) * w_i) / \sum_{i=1}^{n} w_i$$

where $0 \leq f_i \leq 1$ and $\sum_{i=1}^{n} w_i = 1$ are respectively the field similarity functions and the weights w.r.t. to each l_i.

$F_S(r_1, r_2)$ is therefore the result of combining a set of similarity measures, calculated over one or more record fields and possibly with different weights, assigned by data curators to reflect the impact of one field over the others. For example person names may be regarded as more important than birth dates when trying to identify candidates in a person authority file, but similarity measures for both fields may be considered to establish the overall similarity of two person records.

Figure 4 PACE screenshots: list of candidate record pairs

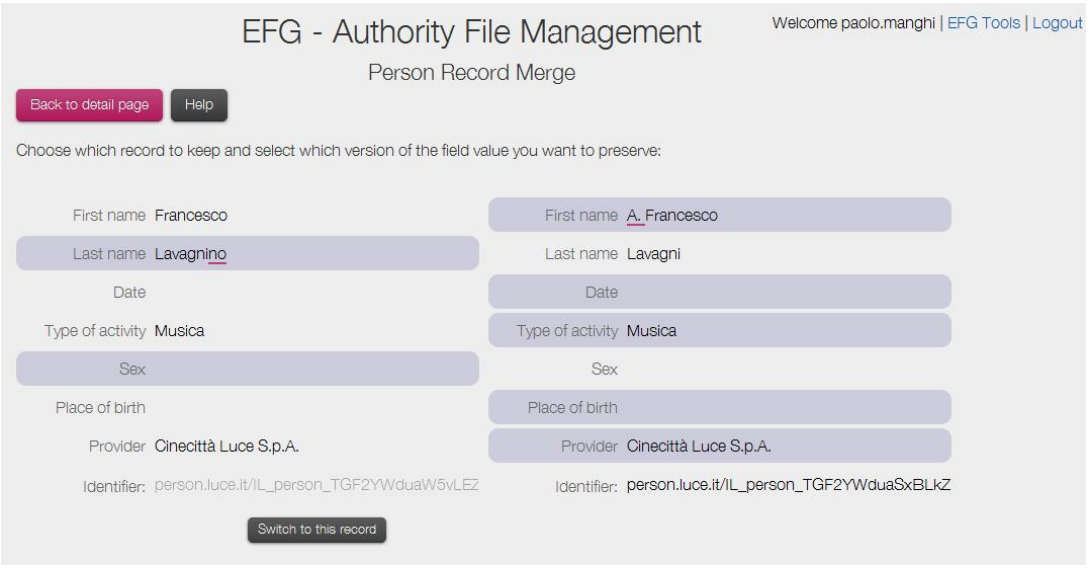

Figure 5 PACE screen shots: merge preparation

New f_i's can be plugged in PACE and be made available through the user interfaces for the selection of data curators. The f_i's are associated to one or more value type domain T of application, in order to be automatically proposed by the interface during similarity function configuration. Special f_i can be defined:

- *multi-field*: functions that accept more than one record field as input; their effect on the formula above is to behave as if the record had one extra "virtual" field l_f;
- *inter-field*: functions that may impact on the selected f_i's and w_i's based on the values of the two records to be compared; for example, a function that if an l_i has empty value removes the field from the distance measurement, a function that increases the weight w_i if the field l_j contains a given value, etc.

Record matching. PACE record matching is based on a variation of the sorted neighborhood process. In its traditional definition, this heuristic takes as input: (i) a data structure $authFile$ which contains the authority file records, (ii) a record similarity function configuration $simConf$, which includes F_S and its parameters, and (iii) a sorted neighborhood configuration $snConf$, which includes the window size K and a clustering function F_C. The clustering function has the purpose of returning, for

each record, the value to be used to sort the records. A run of the algorithm has complexity $O(nlogn) + O(n * K)$ (ordering and pairing) and populates a sorted data structure *clCollector* of records candidate pairs, together with their similarity distance (i.e., triples such as $\langle rec_1, rec_2, F_S(rec_1, rec_2)\rangle$) sorted by the their distance values.

```
  def sortedNeighborhood(
    authFile,       %%% authority file record list
    snConf,         %%% sorted neighborhood config
    simConf,        %%% similarity function config
    clCollector     %%% collector of record pairs
    {
1.  val sortedRecords = sort(authFile, snConf.Fc)
2.  val window = new List()
3.  for(pivot <- sortedRecords.rec) {
4.    for(record <- window) {
        if (pivot <> record)
5.       then {val dist = simConf.Fs(pivot, record)
6.             clCollector.insertSort(pivot,
                                      record,
                                      dist)
            }
7.    window = enqueue(window, pivot, snConf.K)
    }
  }
```

Step 1. generates *sortedRecords*, which is the list of pairs (val, rec) such that $rec \in authFile$ and $val \in F_C(rec)$, ordered by the values val. Steps 3. and 4. collect all possible pairs obtained by combining records that fall in a proximity of K in the list (sliding window algorithm); step 5. applies F_S to each pair (if $pivot \neq record$); and step 6. collects pairs into the sorted data structure *clCollector* (for the sake of simplicity we have not mentioned the distance threshold used to skim the pairs). Record pairs are built by moving a *pivot* along the list *sortedRecords* (step 3.) and pairing it to the K elements *record* which precede *pivot* in the list. This is ensured by step 7., which constructs the list *window* of at most size K of the records in *sortedRecords* that will precede the next *pivot*. Note that by modifying the algorithm to construct the list *window* based on equal F_C values rather than on a constant number K, we would obtain a traditional blocking algorithm.

In the context of aggregation authority files the relatively poor amount of information within the records makes the identification of the functions F_C a real challenge. A misjudged choice, may result in false negatives (i.e., records candidate for merging that are placed at a distance greater than K) and therefore reduce the overall recall rates. Consider an authority file of people, which includes record fields such as 'name' and 'surname'. The intuition may lead to choose a function F_C that returns the field 'surname'. This would yield all people records with similar surnames, hence possible merge candidates, close to each other in the ordering. Unfortunately, simple typing errors, such as exchange of two letters of a name, or a different spellings, may cause true candidates to fall out of the K record window.

To improve recall, PACE extended the sorted neighborhood algorithm introducing two novelties towards a "greedy" approach: *multi-value clustering* and *multi-sort passes*. In PACE F_C is a function which processes a record to return one or more values that will be used to sort the record before record matching. More specifically, a record with many F_C values will appear in the ordering several times, ones for each value. An example may be the function $F_C = ngram(n, m, fields)$: the function concatenates the values of the given $fields$ and extracts m n-grams of size n from the resulting string, starting from the first character (the option $m = *$ extracts all the available n-grams in the string). Clustering functions with multiple values introduce powerful flexibility in the way records can be sorted and therefore be included into the same sorting window. In the case of the *ngram* function for example, duplicate records due to typos of the kind introduced in our benchmark would fall close to each other in the sorting due to a common n-gram.

Moreover, since reasoning based on one $snConf$ may not be enough to capture all possible pairs of candidate records, PACE introduces the possibility to iterate sorted neighborhood several times with different configurations. According to this approach, PACE accepts a set $snConfs$ of sorted neighborhood configurations $snConf$ each indicating a clustering function and and a window size and iteratively applies each of them (step 9.) to the sorted neighborhood algorithm to enrich the same list of record candidates:

```
  def multiSortRecordMatcher(
    authFile,
    snConfs,
    simConf)
    {
8.  val clCollector = new Collector()
9.  for(snConf <- snConfs) {
10.   sortedNeighborhood(authFile,
                         snConf,
                         simConf,
                         clCollector)
    }
  }
```

Note that step 6. of *sortedNeighborhood* is modified to discard record pairs which where already processed on previous runs of the algorithm (using an in-memory hash-set).

Identification configuration parameters. Data curators can define several record matching configurations relative to the same authority file, hence on a given structure $(l_1 : K_1, \ldots, l_n : K_n)$. Configurations consist of the set $snConfs$ of sorted neighborhood strategies, namely $snConf = \langle F_C, K\rangle$'s, and of the similarity function configuration $simConf$, namely the f_i's to be applied to each l_i, together with the weight w_i.

PACE provides administrators with a range of clustering functions F_C. Examples are the functions

$concat(fields)$, which returns the concatenation of the values relative to the given $fields$, and the function $simhash(i, size, fields)$, to cope with fields with large and similar textual content (e.g., context fields). $simhash(i, size, fields)$ calculates a bit SimHash ((Charikar, 2002), (Manku et al., 2007)) of $size$ bits calculated from the concatenation of the values in the given $fields$ of the record and shifts it by i positions. A SimHash function guarantees that records with similar values for such fields have SimHash values with small Hamming distance. However, ordering records using their SimHash would not guarantee that all records with small Hamming distance would fall close to each other. To make sure that this will be the case, the solution is to apply sorted neighborhood $i : 1, \ldots, size$ times, each time applying $F_C = simhash(i, size, fields)$.

Similarly, administrators can tune up their similarity function F_S by selecting distance and weight parameters for each field or by opting for multi-field and inter-field functions. The distance function f_i of a field l_i can be chosen out of a list which is derived by the domain T_i of l_i. Currently PACE supports the following string-string distance functions (Cohen et al., 2003): Jaccard Index, Jaro distance, Winkler distance, Levenshtein distance, Jensen distance, Shannon divergence (Jelinek Mercer and Dirichlet unigram language models), Monge Elkan distance, and Needleman-Wunsch distance.

PACE is designed to facilitate developers at plugging in new clustering functions and new record field distance functions, to be offered for selection to administrators.

5 PACE: implementation issues

PACE v2.0 extends PACE v1.0 (Manghi and Mikulicic, 2011) with the multi-value clustering and multi-sort options for the sorted neighborhood strategy as described above – a demo version of PACE is available for testing at `http://node1.pace.research-infrastructures.eu`. Already included in v1.0, were the implementation of the following functionality described in Section 4 for:

- Authority files management: creation of an authority file specifying a name and a structure;
- Configurations management: creation of candidate identification configurations by providing a name and the parameters required for its settings, selecting F_{Cj}' and f_i's from a list of available functions;
- Work sessions management: opening of a work session, together with a merging configuration of reference, ability to perform merge actions, and commit of work sessions;
- Information systems management: low-level admin interfaces for bulk-feeds of records from a set of information systems compatible with JDBC protocol, i.e., SQL queries over a JDBC data source, or OAI-PMH protocol (Open Archives Initiative - Protocol for Metadata Harvesting) (Carl Lagoze and Herbert Van de Sompel, 2003), which is a simple protocol widely adopted in the library world to expose (OAI) sets of XML metadata records;
- Logs management: merge actions of committed actions to support incremental authority file consumption and "refresh mode" optimization.

Efficient and scalable data management through adequate storage and processing technologies is an important and sometime underestimated issue in authority control, where research is often concentrated on adaptable and configurable frameworks and computational complexity of the underlying algorithms. PACE faces efficiency and scalability as primary goals in order to cope with the large size of aggregation authority files (up to tens of millions) and the multiple sorting neighborhood passes introduced to compensate poorly informative header fields. To this aim, PACE stores authority files on a Cassandra storage and parallelizes record candidate identification over multiple processors.

Efficient storage. PACE stores authority files and the resulting candidate lists on a Cassandra (Lakshman and Malik, 2010) "NoSQL" storage cluster. Cassandra is in use at *Digg*, *Facebook*, *Twitter*, and many other companies which have extremely large and active data sets. Its storage framework manages data replicas over a cluster of servers and offers I/O performance which scales linearly on the number of records through a tunable interaction between RAM buffers, object disc accesses, horizontal partitioning, and degree of robustness by replica. Furthermore, its deferred indexing mechanisms allow for fast sorting at insertion time. As such, its capabilities support PACE record scalability (up to the availability of machines to be added to the cluster) and provide for efficient/parallel disc writes and sorting of the candidate list of records. Consequently, Cassandra outclasses the I/O and sorting performance provided by more traditional storage platforms, such as open source DBMSs (e.g., MySQL, Postgres). In the algorithm $multisortRecordMatcher$, the usage of Cassandra is visible at step 8.:

```
   def multiSortRecordMatcher(
     authFile,
     snConfs,
     simConf)
     {
8.   val clCollector = new CassandraCollector()
9.   for(snConf <- snConfs) {
10.    sortedNeighborhood(authFile,
                          snConf,
                          simConf,
                          clCollector)
     }
   }
```

Parallelism. PACE performs record similarity calculation and Cassandra disc writes in parallel, achieving performances that go well beyond I/O capacities normally supported by other storage systems. This is particularly true in the case of large-size records, where I/O can affect processing and writes; for example in the case the number of heading and context fields reflects standard exchange formats such as MARCXML bibliographic records (http://www.loc.gov/standards/marcxml), custom DIDLXML entity representations (Van de Sompel et al., 2005), RDF Linked Data, etc.

Moreover, to improve performance of the multi-sort passes, the sorting step 1. of the algorithm is executed by a parallel encoding of merge-sort to achieve a complexity of $O(nlogn)/H$, where H is the number of processors.

In the algorithm *sortedNeighborhood*, the effects of these two actions are reflected in steps 1. and $3. - 6.$:

```
  def sortedNeighborhood(
    authFile,       %%% authority file record list
    snConf,         %%% sorted neighborhood config
    simConf,        %%% similarity function config
    clCollector     %%% collector of record pairs)
  {
1.  val sortedRecords =
                  parMergeSort(authFile, snConf.Fc)
2.  val window = new List()
3.  for(pivot <- sortedRecords) {
4.    for(record <- window) {
       par{    %%% begin parallel block
        if (pivot <> record)
5.       then {val dist = simConf.Fs(pivot, record)
6.             clCollector.insertSort(pivot,
                                      record,
                                      dist)
              }
       }       %%% end parallel block
7.    window = enqueue(window, pivot, snConf.K)
    }
    wait;       %%% end of parallel executions
  }
```

In the following we present the benchmark and the experiments we used to show the ability of PACE to achieve improved recall by adopting the multi-value clustering and multi-sort extensions of the sorted neighborhood algorithm, and efficiency in the presence of millions of records.

5.1 Benchmark and Dataset

The experimental dataset consists of an authority file of people entities with the following structure:

```
( personID: (number, [1], identifier),
  firstName: (string, [1], heading),
  lastName: (string, [1], heading),
  birthDate: (date, [0...1], heading),
  country: (string, [0...1], heading)
)
```

Records were generated from a name and surname set of $50,000$ values, the list of world countries. Identifiers were assigned an incremental integer and dates were randomly generated. The dataset counts 10^7 records, with an exact 10% of duplicates. 1 out of 10 records is replicated with a different identifier to generate a *duplicate record* affected by one random error, chosen out of the following list:

- *oneCharMiss*: one missing character (random position) in one of the fields *firstName*, *lastName*, or both;
- *oneCharErr*: one misspelled character (random position) in one of the fields *firstName*, *lastName*, or both;
- *twoCharSwap*: swap two adjacent characters in one of the fields *firstName*, *lastName*, or both;
- *date1Gen*: in the field *birthDate* the value collapse to 01/01/yyyy;
- *dateSwitch*: in the field *birthDate* the day and month value are swapped when the day number is less than 12.

oneCharErr, *oneCharMiss*, *twoCharSwap* errors have an higher probability to be picked, to reflect the trend of introducing typos in entries such as names and surnames.

Records also feature administrative fields to be used to run experiments and measure precision and recall rates, as well as accuracy of the similarity functions. The complete record structure includes the following fields:

- *kind*: the field is used to store the typology of a record, which may range from *unique*, to indicate authoritative-born records, *duplicate*, to indicate the 10% of records obtained from these and to be merged, or *alias*, to indicate the 5% of false positive records;
- *error*: the field indicates, in the case of duplicate records, the typology of error that was applied; its values therefore range in the range: *none*, *oneCharErr*, *oneCharMiss*, *twoCharSwap*, *date1Gen*, *dateSwitch*;
- *relatedTo*: the field is present only in the case of duplicate or alias records and contains the identifier of the record from which the current record was generated.

Here are examples of a unique record and one duplicate record generated from it:

```
[ personID: 0,
  error: "none",
  kind: "unique",
  firstName: "Charlotte",
  lastName: "Summann",
  birthDate: 10/7/1939,
  country: "CH"
]
```

```
[ personID: 1,
  error: "twoCharSwap",
  kind: "duplicate",
  relatedTo: 0,
  firstName: "Charlotte",
  lastName: "Smumann", %%% <-- typo error
  birthDate: 10/7/1939,
  country: "CH"
]
```

5.2 Experiments

As mentioned above, several experiments were run to give evidence of PACE's ability to cope with deduplication of records with poorly informative fields (effectiveness) and to do so with acceptable performance rates, which would enable an interactive deduplication strategy. The benchmark has been constructed in a way that a 10% duplication rate is present and trackable. In addition, authoritative records and their duplicates are generated with typos which may turn into false negatives using traditional blocking or sorted neighborhood techniques.

The initial experiments were run to evaluate the behaviour of a "naive" sorting neighborhood algorithm, where only one clustering function F_C which returns the field *lastName* is used. Three experiments are considered, to measure:

- Precision and recall behaviour on increase of window size (see Figure 7): we fixed a dataset of $10,000,000$ records and 12 CPUs;
- Performance improvement Vs increase of CPUs: we fixed a window of 100 records and a dataset of $10,000,000$ records;
- Performance degradation on increase of dataset size (see Figure 6): we fixed a window of 100 records and run over 12 CPUs;
- Speedup rate on increase of CPUs (see Figure 9): obtained by running several experiments with window sizes of 100, 200, 400, 1000 and dataset size of $1,000,000$ and $10,000,000$.

From these initial experiments we can observe how with our dataset, where duplicates are generated by injecting typos which may jeopardize sensible clustering by *lastName* ordering, traditional sorted neighborhood logic shows low recall and high precision, but with good performances (see related work Section 6). With an increase of window size, performance tends to degrade while precision tends to decrease with only a small gain in recall. This is indeed the expected behaviour, since ordering is not effective in grouping within a reasonable K the real duplicate records in our dataset. As to efficiency, it is clear how, with higher numbers of CPUs, execution time tends to decrease with a near-linear

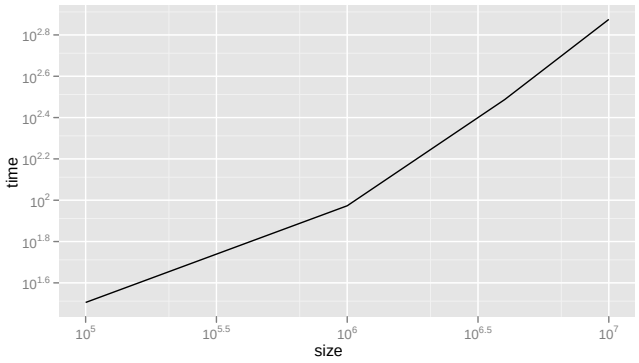

Figure 6 Execution time Vs dataset size (window = 100, 12 CPUs)

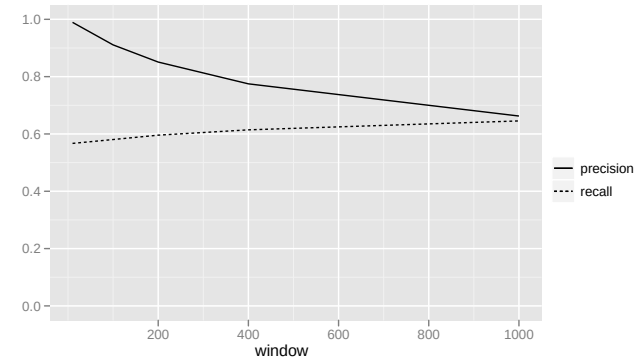

Figure 7 Precision and recall behavior at variation of window size (dataset = $10,000,000$, 12 CPUs)

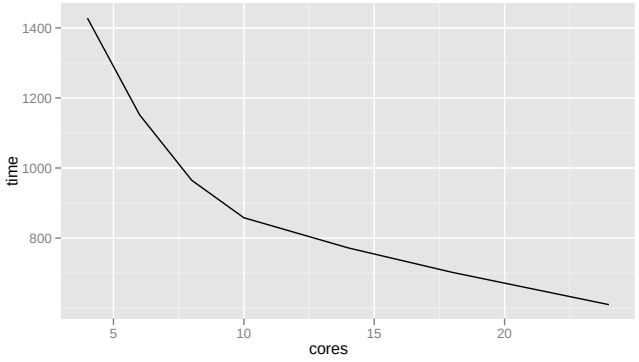

Figure 8 Execution time Vs number of CPUs (window = 100,dataset = $10,000,000$)

speedup which reflects the high-degree of parallelization of PACE.

In the second slot of experiments we identified the best behaviour of PACE for the multi-value clustering function *ngram* and the multi-sort method with clustering function *simhash* (the combination of the two was not considered in this work). The tests, which were run over the largest dataset of $10,000,000$, with 12 CPUs, and an acceptance threshold of 0.9, led to the following best configurations:

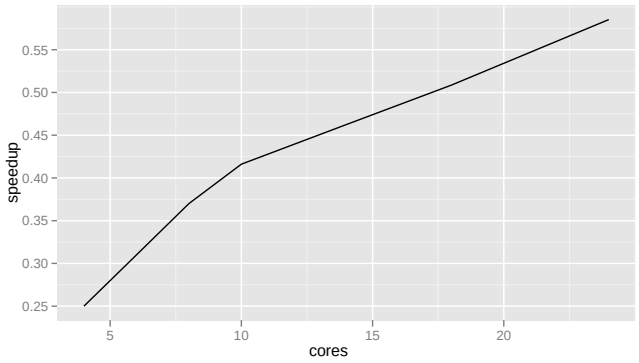

Figure 9 Overall speedup w.r.t. number of CPUs

- $ngram(2, 4, lastName)$: with a window of size 10, applying a 2-gram function for the first 4 characters of $lastName$ values, PACE returns a precision of 0.99 and recall of 0.95 in $0h : 42m$;
- $simhash(i, 32, lastName)$: applying a 32-bit simhash map over $lastName$ values, with with $i : 1, \ldots 8$ passes and a window of size 100, PACE returns a precision of 0.99 and recall of 0.84 in $1h : 37m$ minutes.

Results are optimal since the two clustering functions are designed to reduce the impact of differences in spelling introduced by the typos into record duplicates. By using $ngram$, splitting $lastName$ (or other fields) in smaller substrings ($ngram$) helps matching common substrings in any part of the value (or at least in a significant portion of every token). By using $simhash$, $lastName$ (or other fields) values are mapped onto a metric which is weakly affected by the relative ordering of the features.

5.3 Real-Case Scenarios

PACE is today used in the European Film Gateway European Commission project, whose aim is to operate a data infrastructure aggregating content from an initial set of 20 movie archives in Europe. The infrastructure collects from such archives heterogeneous information about movies, people, companies, and available digital material, including videos, images, and documents, and operate sophisticated data cleaning and normalization actions, to populate a uniform European movies information space. PACE was devised to address the construction of an European aggregation authority file of persons and movies in order to align authority files at the local archives, where these existed, or create valuable references from those archives which are not supporting authority files. As mentioned above the current authority file of people counts around 300,000 records, while the movie authority files counts around 140,000 records. In the first file, context fields (e.g., movies titles in which such people were involved with some role) are used to improve the extremely poor information about persons (i.e., name, surname, year of birth), which sometime does not come with a persistent identifier assigned by the archive (e.g., persons are properties of other entities, such as movies). In the second file, movie records were rich of complete heading fields, which simplified their deduplication process.

PACE is currently under deployment in the OpenAIRE European Commission project whose aim is to operate a data infrastructure capable of collecting publication bibliographic records from a set of European institutional repositories and from manual "claiming", i.e., ingestion, by authors. The purpose of the infrastructure is to offer statistics over the ratio Open Access/licensed scientific publications funded by given European Commission projects. In its continuation through the OpenAIREplus European Commission project, the infrastructure will have to integrate the DRIVER Open Access infrastructure (http://search.driver.research-infrastructures.eu), which today contains around 6,000,000 Dublin Core bibliographic records from around 320 institutional repositories in Europe and beyond. PACE will be used to disambiguate the aggregation authority files of publication records and the relative authors. Author records are obtained from the bibliographic records and therefore consists only of a string inclusive of names and surname and lead to high percentage of duplicates. Authors grow with the number of publications and are today estimated to bring more than 22 Millions records, inclusive of duplicates. To facilitate disambiguation of this set, context fields relative to abstracts of author publications will be introduced.

6 Related work

The literature and the market offer a number of tools, which mainly address record linkage between information systems or deduplication of one information system. Record linkage tools focus on the problem of identifying a set of record matches across two or more information systems to be integrated or combined for processing. Here, candidate identification is not necessarily intended for record merging and deduplication. Deduplication tools focus instead on resolving record duplicates in one information system, typically a relational database. Here, candidate identification is generally followed by application of record merges. Hence, the process can take advantage of contextual information (e.g., relationships with other entities (Bhattacharya and Getoor, 2007) (Kalashnikov and Mehrotra, 2006)) and can exploit logs to enable advanced data curation options (e.g., history information, roll-backs to previous status). Strengths and differences of such tools (Christen, 2011) lay in (i) the techniques devised to effectively and efficiently identify candidate record duplicates, and (ii) the level of usability (i.e., user-friendly interfaces versus low-level configuration) and personalization (i.e., general-purpose

versus domain specific tools (Rudniy et al., 2010; Wang and Alexander, 2010).

We examined differences and similarities between PACE and existing tools, which we believe are relevant and representative with respect to the goals of PACE. The following tools were selected, which target record linkage and deduplication at different degrees: *LinkageWiz* (http://www.linkagewiz.com), *FRIL* (Fine-grained Records Integration and Linkage, http://fril.sourceforge.net) (Jurczyk et al., 2008), *D-Dupe* (http://www.cs.umd.edu/projects/linqs/-ddupe/) (Kang et al., 2008), and *Febrl* (Freely Extensible Biomedical Record Linkage, https://sourceforge.net/projects/febrl/) (Christen, 2008). In the analysis we focused on the common data processing phases they support of (*i*) records import, (*ii*) candidate identification configuration, and (*iii*) generation of results. Finally, to compare effectiveness and efficiency, we run experiments with the dataset presented in Section 5.1.

Data processing phases. The records import phase is characterized by user interfaces to load records from files or to fetch records from data sources. Tools differ by:

- Data import functionality: Febrl, LinkageWiz and FRIL allow for cleansing (normalization) of imported records, while FRIL also allows for deduplication of data sources to be linked.
- Data import formats and protocols: Febrl and FRIL support Excel and CSV Text Files, LinkageWiz supports Excel, CSV, DBASE, FoxPro, and MS Access, while D-Dupe a proprietary representation of relational data. LinkageWix and FRIL can also collect data from JDBC connections.

The candidate identification configuration phase consists in user interfaces for the specification of the parameters for record matching algorithms and probabilistic similarity functions. Tools differ by:

- Clustering methods the users can choose from: all of them offer traditional blocking (i.e., field values), FRIL, Febrl and D-Dupe support also literature variations of blocking (i.e., values obtained from elaboration of field values, such as metaphone and soundex), while FRIL and Febrl also provide sorted neighborhood. In general, the selection of a method requires pre-processing of the records before matching can be applied.
- Similarity distances the users can choose from: record similarity function is based on a weighed mean can be configured based on list of distance functions, field weights, and in some cases, such as in FRIL, introducing conditions.

The generation of result phase generally allows to export the result of record linkage or deduplication as processable text files. Tools differ by the way the output can be handled by users before the final export. In FRIL users can only opt for what should be exported, which is either data source records purged of the duplicates (deduplication mode) or the identified record pairs together with a confidence level (record linkage mode). LinkageWiz allows instead to tune up the thresholds for automated approval or manual approval of record pairs, offering user interfaces to visualize and manually approve or reject record pairs. D-Dupe graphically visualizes record pairs (with a maximum limit of 300) together with the graph of records related with them, to help users at making the right choice. Febrl offers user interfaces to scroll through and evaluate candidate pairs for linking or deduplication (i.e., "clerical evaluation").

Dataset size	FRIL	LinkageWiz	PACE
1,000,000	P: 0.91 R: 0.66 t: 0h:2m	P: 0.85 R: 0.43 t: 1h:24m	P: 0.91 R: 0.59 t: 0h:1m
10,000,000	P: 0.65 R: 0.63 t: 3h:46m	P: 0.74 R: 0.33 t: 19h:20m	P: 0.91 R: 0.58 t:0h:13m

Table 1 Comparative analysis

Efficiency and effectiveness. For the purpose of this paper, we have run the same experiment with $10,000,000$ and $1,000,000$ records taken from our dataset over FRIL and LinkageWiz. Specifically, we used blocking techniques based on $lastName$ and the same similarity function and with an acceptance threshold of 0.9 (where possible, since LinkageWiz offers field templates, no explicit distance functions). The tools executed on the following platform: 2x AMD Opteron 6172 2.48 Ghz (12 CPUs), with 16GB RAM, running Debian 6 Squeeze and Xen 4.0.1, and mounting a Windows Server virtual machine. Table 1 shows the results, where LinkageWiz was run on $4,000,000$ records (its declared upper bound), rather than over 10,000,000, with its default blocking configuration – it must be remarked that by using an "accelerated matching" option in LinkageWiz, execution time decreases considerably: about $0h:30m$ for $1,000,000$ records and $7h:00m$ for $10,000,000$, with the same precision and recall rates. To enable a reasonable comparison between the tools blocking methods and PACE's sorted neighborhood, we considered the worst case experiment of PACE, which sorted records according to the values of $lastName$.

All tools suffer from a lack of precision and performance due to the number of possible duplicates, which increases with the number of records. PACE generally performs better, especially with very large numbers of records, but its configuration delivers lower recall rates due to the issues highlighted in Section 5.2. Indeed, given our dataset, blocking techniques generally behave better than traditional sorted neighborhood since

they can restrict record pair matching to windows of *lastName*-related records only.

Comparison. FRIL, LinkageWiz, and Febrl support effective techniques for record linkage and deduplication, and generally deliver to the users a wider range of blocking techniques compared to PACE. However, unlike PACE they are not concerned with authority control of aggregation authority files. D-Dupe targets deduplication, supports iterative curation of one static authority file, and offers elegant visualization tools to facilitate manual evaluation of potential duplicates. On the other hand, it does not cope with incremental population of the authority file and multiple users curation through work sessions and logs.

With regard to efficiency, the examined tools do not cope with arbitrary scalability and performance. Attempts are made only by FRIL which exploits multi-core parallelism. However, as shown by the experiments, both FRIL and LinkageWiz cannot cope with the performances reached by PACE when operating over 10,000,000 record datasets.

Finally, as to effectiveness, the experiments show that PACE multi-value clustering method with n-gram and the multi-sort "greedy" approach with SimHash manifest precision and recall results which are generally better than other tools. Although this cannot be considered a general statement, as indeed the evaluation depends on the datasets at hand, we do not expect PACE to behave worse than the other tools. Indeed, PACE can offer the same similarity functions and record matching techniques which lead to higher recall and precision with better performance.

7 Conclusions

This paper presented PACE, an open source and general-purpose tool for managing aggregation authority files. PACE offers a framework where administrators can create their authority files, configure their preferred candidate identification configurations and use them to curate authority files life-cycle. PACE aims at offering an ideal environment for maintaining aggregation authority files, whose semantics and content dynamicity requires "fast" deduplication experimentations with different similarity matching functions and ability to scale to arbitrary sizes. Its implementation features high sorting performances due to parallel processing and underlying optimizations and can scale up to arbitrary numbers of records. Future issues include the completion of the front-end functionalities according to the specification in Section 3, the introduction of blocking techniques, and the study and experimentation of logging techniques to cope with the absence of identifiers and the heavy usage of context fields.

Acknowledgments. This work was partially funded by the European Commission projects European Film Gateway (Best Practice Networks project, grant agreement: ECP-517006-EFG, call: FP7 EU eContentplus 2007) and OpenAIRE (grant agreement: 246686, call: FP7-INFRASTRUCTURES-2009-1).

References

Michele Artini, Leonardo Candela, Donatella Castelli, Paolo Manghi, Marko Mikulicic, and Pasquale Pagano. Aggregative Digital Library Systems in the DRIVER Infrastructure. *World Digital Libraries Journal*, 2(2): 113–130, December 2009. ISSN ISSN 0974-567X.

O. Benjelloun, H. Garcia-Molina, Q. Su, and J. Widom. Swoosh: A generic approach to entity resolution. *Stanford University technical report*, March 2005.

Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007.

Carl Lagoze and Herbert Van de Sompel. The making of the open archives initiative protocol for metadata harvesting. *Library Hi Tech*, 21(2):118 – 128, 2003.

M. Charikar. Similarity estimation techniques from rounding algorithms. In *34th Annual Symposium on Theory and Computing*, Montreal, Quebec, Canada, May 2002.

Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 313–324, New York, NY, USA, 2003. ACM. ISBN 1-58113-634-X. doi: http://doi.acm.org/10.1145/872757.872796. URL `http://doi.acm.org/10.1145/872757.872796`.

P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1, 2011. ISSN 1041-4347. doi: 10.1109/TKDE.2011.127.

Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 1065–1068, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4. doi: http://doi.acm.org/10.1145/1401890.1402020. URL `http://doi.acm.org/10.1145/1401890.1402020`.

Peter Christen and Agus Pudjijono. Accurate Synthetic Generation of Realistic Personal Information. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge Discovery and Data Mining*, volume 5476 of *Lecture Notes in Computer Science*, pages 507–514. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-01306-5. URL `http://dx.doi.org/10.1007/978-3-642-01307-2_47`. 10.1007/978-3-642-01307-2_47.

T. Christen, P. Churches and J.X. Zhu. Probabilistic name and address cleaning and standardization. *The Australian Data Mining Workshop*, November 2002.

T. Churches, P. Christen, J. Lu, and J. X. Zhu. Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models. *BioMed Central Medical Informatics and Decision Making*, 2 (9), 2002.

W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string metrics for matching names and addresses. *International Joint Conference on Artificial Intelligence, Proceedings of the Workshop on Information Integration on the Web*, August 2003.

William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 475–480, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775116. URL `http://doi.acm.org/10.1145/775047.775116`.

Prudence W. Dalrymple and Jennifer A. Young. From authority control to informed retrieval: Framing the expanded domain of subject access. *College & Research Libraries*, 52:139 149., 1991. doi: http://hdl.handle.net/1860/3173.

A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, Jan. 2007. ISSN 1041-4347. doi: 10.1109/TKDE.2007.250581. URL `http://www.cs.purdue.edu/homes/ake/pub/-survey2.pdf`.

Jordi Gómez-Bao, Josep-L. Larriba-Pey, and Josepa Ribes Puig. Record linkage performance for large data sets. In *Proceedings of the ACM first international workshop on Privacy and anonymity for very large databases*, PAVLAD '09, pages 9–16, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-804-9. doi: http://doi.acm.org/10.1145/1651449.1651453. URL `http://doi.acm.org/10.1145/1651449.1651453`.

Caichun Gong, Yulan Huang, Xueqi Cheng, and Shuo Bai. Detecting Near-Duplicates in Large-Scale Short Text Databases. In Takashi Washio, Einoshin Suzuki, Kai Ting, and Akihiro Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 877–883. Springer Berlin Heidelberg, 2008. URL `http://dx.doi.org/10.1007/978-3-540-68125-0_87`.

Michael Gorman. Authority control in the context of bibliographic control in the electronic environment. *International Conference Authority Control: Definition and International Experiences, Florence, February 10-12 2003*, 2003. doi: http://hdl.handle.net/10760/4164.

Luis Gravano, Panagiotis G. Ipeirotis, Nick Koudas, and Divesh Srivastava. Text joins in an RDBMS for web data integration. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 90–101, New York, NY, USA, 2003. ACM. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775166. URL `http://doi.acm.org/10.1145/775152.775166`.

Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. *SIGMOD Rec.*, 24:127–138, May 1995. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/568271.223807. URL `http://doi.acm.org/10.1145/568271.223807`.

Matthew A. Jaro. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, June 1989. URL `http://www.jstor.org/stable/2289924`. published by American Statistical Association.

Pawel Jurczyk, James J. Lu, Li Xiong, Janet D. Cragan, and Adolfo Correa. Fine-grained record integration and linkage tool. *Birth Defects Research Part A: Clinical and Molecular Teratology*, 82(11):822–829, 2008. ISSN 1542-0760. doi: 10.1002/bdra.20521. URL `http://dx.doi.org/10.1002/bdra.20521`.

D.V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2): 716–767, 2006.

Hyunmo Kang, Lise Getoor, Ben Shneiderman, Mustafa Bilgic, and Louis Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):999–1014, 2008.

Hanna Koepcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197 – 210, 2010. ISSN 0169-023X. doi: 10.1016/j.datak.2009.10.003. URL `http://www.sciencedirect.com/science/article/-pii/S0169023X09001451`.

Hanna Koepcke, Erhard Rahm, and Erhard Rahm. Training selection for tuning entity matching. In *QDB/MUD*, pages 3–12, 2008.

Carl Lagoze and H. Van de Sompel. The open archives initiative: building a low-barrier interoperability framework. In *Proceedings of the first ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 54–62. ACM Press, 2001. ISBN 1-58113-345-6. doi: http://doi.acm.org/10.1145/379437.379449.

Carl Lagoze and Herbert Van de Sompel. The OAI Protocol for Object Reuse and Exchange. `http://www.openarchives.org/ore/`.

Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1773912.1773922. URL `http://doi.acm.org/10.1145/1773912.1773922`.

Paolo Manghi and Marko Mikulicic. PACE: A General-Purpose Tool for Authority Control. In Elena Garcia-Barriocanal, Zeynel Cebeci, Mehmet C. Okur, and Aydin Ozturk, editors, *Metadata and Semantic Research*, volume 240 of *Communications in Computer and Information Science*, pages 80–92. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24731-6. URL `http://dx.doi.org/10.1007/978-3-642-24731-6_8`. 10.1007/978-3-642-24731-6_8.

G.S. Manku, A. Jain, and Sarma A.D. Detecting near-duplicates for web crawling. In *16th International World Wide Conference*, Banff, Alberta, Canada, May 2007.

Bennet Rick, Christal Hengel-Dittrich, Edward T. ONeill, and Barbara Tillett. Viaf (virtual international authority file): Linking the deutsche nationalbibliothek and library of congress name authority files. In *International Cataloging and Bibliographic Control*, number 1 in 36, pages 12–19, 2007.

Alex Rudniy, Min Song, and James Geller. Detecting duplicate biological entities using shortest path edit distance. *International Journal of Data Mining and Bioinformatics*, 4(4):395 – 410,

2010. doi: 10.1504/IJDMB.2010.034196. URL http://inderscience.metapress.com/content/-TQ3737625VK1R573.

S. Tejada, C. Knoblock, and S. Minton. Learning object identification rules for information extraction. *Information Systems*, 26 (8):607–633, 2001.

Barbara T. Tillett. Authority control: State of the art and new perspectives. In *Authority Control International Conference*, Florence, Italy, 2003.

Herbert Van de Sompel, Jeroen Bekaert, Xiaoming Liu, Luda Balakireva, and Thorsten Schwander. aDORe: a modular, standards-based Digital Object Repository, 2005. URL http://www.citebase.org/abstract?id=oai:-arXiv.org:cs/0502028.

Xiaoyi Wang and Suraj M. Alexander. Linking medical records: a machine learning approach. *International Journal of Collaborative Enterprise*, 1(3):394 – 406, 2010. doi: 10.1504/IJCENT.2010.03836. URL http://inderscience.metapress.com/content/-T93552025P150H36.

Jutta Weber. LEAF. Linking and Exploring Authority Files. *International Conference Authority Control: Definition and International Experiences, Florence, February 10-12 2003*, 2003.

W. E. Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 354–359, 1990.

W. E. Winkler. Overview of record linkage and current research directions. Technical report, Research Report Series, RRS, 2006. URL http://www.census.gov/srd/papers/pdf/-rrs2006-02.pdf.