Entity deduplication in big data graphs for scholarly communication

Paolo Manghi, Claudio Atzori, Michele De Bonis and Alessia Bardi
Istituto di Scienza e Tecnologie dell'Informazione, National Research Council, Pisa, Italy

Received 4 September 2019
Revised 13 December 2019
Accepted 19 December 2019

Abstract

Purpose – Several online services offer functionalities to access information from "big research graphs" (e.g. Google Scholar, OpenAIRE, Microsoft Academic Graph), which correlate scholarly/scientific communication entities such as publications, authors, datasets, organizations, projects, funders, etc. Depending on the target users, access can vary from search and browse content to the consumption of statistics for monitoring and provision of feedback. Such graphs are populated over time as aggregations of multiple sources and therefore suffer from major entity-duplication problems. Although deduplication of graphs is a known and actual problem, existing solutions are dedicated to specific scenarios, operate on flat collections, local topology-drive challenges and cannot therefore be re-used in other contexts.

Design/methodology/approach – This work presents GDup, an integrated, scalable, general-purpose system that can be customized to address deduplication over arbitrary large information graphs. The paper presents its high-level architecture, its implementation as a service used within the OpenAIRE infrastructure system and reports numbers of real-case experiments.

Findings – GDup provides the functionalities required to deliver a fully-fledged entity deduplication workflow over a generic input graph. The system offers out-of-the-box Ground Truth management, acquisition of feedback from data curators and algorithms for identifying and merging duplicates, to obtain an output disambiguated graph.

Originality/value – To our knowledge GDup is the only system in the literature that offers an integrated and general-purpose solution for the deduplication graphs, while targeting big data scalability issues. GDup is today one of the key modules of the OpenAIRE infrastructure production system, which monitors Open Science trends on behalf of the European Commission, National funders and institutions.

Keywords Deduplication, Information graphs, Big data, Scholarly communication, Scalability, Implementation

Paper type Research paper

1. Introduction

The advent of Open Science has broadened the scope of interest of scholarly/scientific communication beyond scientific literature, so as to include research data and research software and, for monitoring purposes, projects and funders. Researchers, communities, institutions, governments and funders demand integrated and enhanced access to research graphs (Xia *et al.*, 2017), obtained as metadata aggregations of distributed scholarly communication data sources. Their intent differs, ranging from discovery and access of scientific products, to monitoring and evaluating funding efforts or identifying research trends. Examples of research graphs in the scholarly communication domain are the Google

This work was supported by the European Commission [OpenAIRE2020 project (grant 643410, call H2020-EINFRA-2014-1), OpenAIRE-Advance project (grant 777541, call H2020-EINFRA-2017-1)].

Data Technologies and Applications
Vol. 54 No. 4, 2020
pp. 409-435
Emerald Publishing Limited
2514-9288
DOI 10.1108/DTA-09-2019-0163

Scholar graph [1], the Microsoft Academic graph [2] and the OpenAIRE research graph [3]. Such initiatives collect from different kind of data sources (e.g. libraries, publication repositories, publishers, author directories) and assemble graphs whose objects are authors, organizations, publications, etc. having specific customers in mind. In general, due to the high degree of heterogeneity and independence of such sources, which may preserve objects produced by the same authors or affiliated to the same organizations, such graphs suffer from disruptive duplication rates and require adequate solutions to be adopted. System engineers can easily find general-purpose public/commercial tools for the identification of equivalent metadata records in big "flat" collections of objects. Engineers write code to import their collection into the tool of choice, then use the tool to identify duplicates, retrieve the set of object equivalences (typically groups of object IDs) and finally write code to resolve duplication within their collection. If the collection is also "big", that is untreatable without parallel computing solutions, then the task becomes more complex as extra big data skills are required. However, merging of records in flat object collections is a rather simple task compared to graph-like scenarios. Graphs introduce two different problems: (1) multiple entities, hence the handling of multiple entity-specific deduplication scenarios, and (2) the relationships between objects, which burden the deduplication phase to preserve the topology of the graph. For such reasons, the adoption of existing tools cannot be easily adapted to the problem of deduplication of big graphs. To implement a full entity deduplication workflow for big data graphs curators end-up realizing patchwork systems, tailored to their graph data model, often bound to their physical representation of the graph, expensive in terms of design, development and maintenance, and in general not reusable by other practitioners with similar problems in different domains.

This work, building on literature and tools for duplicate identification in big data flat collections, addresses the more complex challenge of *deduplication of entities in big data graphs*. In the following text a *graph* is intended as any digital representation of a set of entity types (structured properties) linked by relationships. Graphs are *big* when duplicate identification over the objects of such entity types require parallel-oriented approaches to scale up and perform in reasonable time. *Entity deduplication* is the combined process of *duplicate identification*, that is efficient identification of pairs of equivalent objects of the same entity type, and *graph disambiguation*, that is removal of duplicates from the graph, while semantically preserving the topology of the graph.

The need of practitioners for implementing a fully-fledged entity deduplication workflow to disambiguate a generic big graph has led us to devise GDup, an integrated, scalable, general-purpose system for this purpose. GDup offers to graph data curators functionalities to manage "ground truths", acquire end-user feedback to improve the process, and customize algorithms for the identification and merge of duplicates to return an output disambiguated graph; such functionalities rely on an underlying parallel computing engineering that ensures scalability to graphs of arbitrary size. Therefore, the novelty of GDup is not about improving accuracy of deduplication algorithms for specific entities or beating existing deduplication systems on deduplication efficiency; but rather offering out-of-the-box tools to data curators, who should focus on modeling and customizing their big graph deduplication solutions rather than facing the technical challenges necessarily implied by such tasks. This work extends initial outcome on GDup (Atzori *et al.*, 2018), where the authors sketched the concept and high-level architecture of GDup and described its adoption in the real-case scenario of the big graph operated by the OpenAIRE infrastructure for Open Science Research in Europe [4]. OpenAIRE infrastructure's services populate a scholarly communication big data graph, namely the OpenAIRE Research Graph [5], whose goal is to support discovery and monitoring of Open Science trends and research impact for funders, institutions and researchers in specific disciplines; services like Scopus, Zenodo.org, European Commission's Open Science monitor, the ResearchGraph Foundation and others are today using the graph

as main source of information. GDup is the core deduplication service for the production system of the OpenAIRE infrastructure, used today to deduplicate publications, datasets, software and organizations records, in order to ensure sensible statistics are delivered. The contribution of this article, not addressed in previous work (Atzori *et al.*, 2018), is to provide a formal definition of the graph deduplication problem, the formal definition of GDup's architecture, that is the internals of the functions modeling the various deduplication phases, and to report experimental results relative to the usage of GDup in the new OpenAIRE Research Graph, which has since expanded to include new kinds of entities (e.g. research software) and triplicated the size of the data.

1.1 Outline

Section 2 describes the OpenAIRE Research Graph use-case to identify the requirements that a big data graph deduplication system should meet and analyses the state-of-the-art to highlight the limits of current solutions. Section 3 formally presents the functional architecture of GDup, while Section 4 describes its technical implementation and gives an example of its usage in the OpenAIRE infrastructure.

2. Deduplication of big data graphs

To describe the problem of big graph deduplication we introduce the OpenAIRE infrastructure system, whose services populate a big graph of scientific publications, datasets, organizations, authors, and other related entities, and whose size and deduplication challenges are representative of this class of problems. The use-case highlighted the lack of solutions in the literature capable of addressing such challenges and provided input for the definition of functional and non-functional requirements leading to the realization of GDup.

2.1 The OpenAIRE research graph

The OpenAIRE infrastructure is an initiative (Manghi *et al.*, 2010) funded by the European Commission (soon to become a Legal Entity) whose purpose is to facilitate, foster and support Open Science in Europe. The infrastructure has been operational for almost a decade and successful in linking people, ideas and resources for the free flow, access, sharing and re-use of research outcomes. On the one hand, OpenAIRE manages and enables an open and participatory network of people willing to identify the commons and forums required to foster and implement Open Science policies and practices in Europe and globally. On the other hand, it supports the technical services required to facilitate and monitor Open Science publishing trends and research impact across geographic and discipline boundaries.

The OpenAIRE service infrastructure consists of metadata aggregation services and information inference services whose purpose is to populate the *OpenAIRE Research Graph* (Manghi *et al.*, 2012a). A high-level view of the graph's data model is depicted in Figure 1. Its main entities are described below, together with other aspects that are not represented in the picture as they are not relevant for this work:

Products are intended as the outcome of research activities and may be related to Projects co-funding the underlying research or Organizations to which product authors are affiliated. OpenAIRE supports four kinds of research outcome: *Publications*, *Datasets*, *Software* and *Other research products*; that is any product that does not fall in the other three categories. As a result of merging equivalent objects collected from separate data sources, a Product object may have several physical manifestations, called *instances*; instances indicate URL(s) of the full text, access rights, and a relationship to the data source that hosts the file; Organizations include companies, research centers or institutions involved as project partners or are responsible for operating data sources or are the affiliations of Product authors; Funders (e.g.

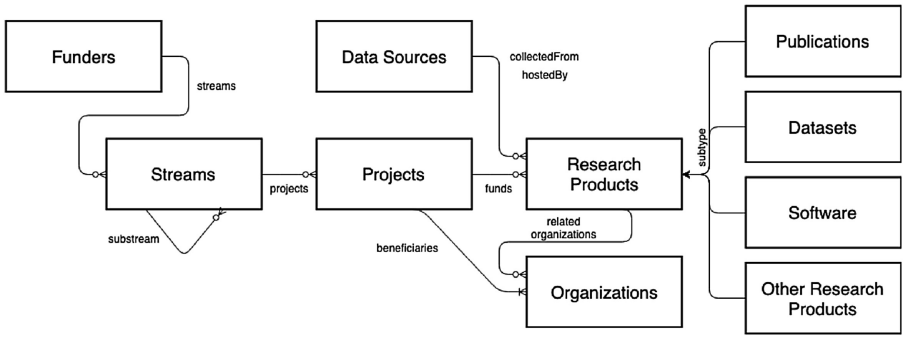

Figure 1.
OpenAIRE data model

European Commission, Wellcome Trust, FCT Portugal, Australian Research Council) are responsible for a list of Funding Streams , that is strands of investments of a Funder (e.g. FP7 and H2020 for the EC). Funding Streams can be nested to form a tree of sub-funding streams (e.g. FP7-IDEAS, FP7-HEALTH); Projects are research projects funded by a Funding Stream of a Funder. Investigations and studies conducted in the context of a Project may lead to one or more Products; Data Sources are the web sources from which OpenAIRE collects the metadata records representing the objects populating the OpenAIRE research graph; examples of sources are institutional and thematic repositories, data repositories, journals, publisher databases, registries such as ORCID, DataCite and CrossRef, etc.. Each object is associated to the data source from which it was collected to track its provenance.

On top of the graph OpenAIRE offers, beyond a search and browse portal, a number of applications, called Dashboards, in support of researchers (Research Community Dashboard [6]), organizations (Institutional Dashboard), funders (Funder Dashboard [7]) and project coordinators (Project Dashboard). The dashboards allow users to access statistics relative to Open Access trends and research impact for organizations, funders, and projects. Deduplication of products and organization is therefore crucial to deliver meaningful statistics to dashboard users.

2.2 Graph entity deduplication

The OpenAIRE Research Graph suffers from heavy duplication phenomena. The main causes are content duplication in the harvested data sources, the low availability of persistent identifiers for the entities involved (e.g. DOI for scientific articles, datasets, and software, ORCID identifiers for authors, ISNI/GRID for organizations, FundRef for funders), but also the fact that products may feature different versions, which are indeed distinct digital objects but to be considered as one in the OpenAIRE context, where reporting and monitoring are key tasks; that is pre-print, post-print, published version of an article cannot be regarded as three different scientific results by a project coordinator, a funder or an institution. The strategy is therefore to (1) generate uniquely identified OpenAIRE objects when harvesting from data sources and (2) subsequently rely in the deduplication process to merge the equal objects. Such identifiers should be "stateless", that is IDs which are identically re-generated for all objects derived from a given metadata record. To this aim, the method relies on the original local identifiers assigned by the data source to such objects or to other unique information within the record. For example, when collecting metadata records of datasets from of a data repository X, the aggregation services extract from such records OpenAIRE objects and relationships to feed the Research Graph: for each record, this process yields one dataset object, the organization to which the dataset is associated, the relationships between the

dataset and the organizations, and possibly relationships from the dataset to other products or projects. The dataset has an OpenAIRE identifier obtained from the unique OpenAIRE identifier for the repository X (assigned at data source registration time) the MD5 hash [8] of the ID of the record as assigned to it by the repository X. In turn, organization objects have an identifier obtained from the OpenAIRE dataset ID as described above, plus the MD5 of the name of the organization or the MD5 of the organization ID if one is available. Interestingly, via an OpenAIRE ID it is always possible to track back the origin of an entity.

Duplication may be of two main kinds: "intra-data source", i.e. duplicates generated by records from one data source, or "cross-data source", i.e. duplicates generated by records from different data sources. In particular, objects of the entity types *Publications*, *Datasets*, *Software*, *Other research products*, and *organization* are affected by both forms of duplication, which in turn lead to specific big data graph deduplication challenges:

Publications Intra-data source publication duplicates are very common in publication aggregators (e.g. NARCIS, CORE-UK, etc.), and in some rare cases also manifest in institutional repositories, due to the lack of curation of the data source managers. Not all aggregators collect from other aggregators, but this is indeed the case for OpenAIRE, which is trying to maximize the number of publications minimizing the number of sources to actively harvest (today around 1,100). Cross-data source duplicates are very common as we can at least expect all co-authors of a publication to deposit in the respective institutional repositories, which will likely be OpenAIRE data sources. Besides, the publication can be further deposited in thematic repositories (e.g. arxiv, PubMed, Repec) or be collected by aggregator services collected by OpenAIRE. For the same reason, as we shall see in Section 4, duplication rate for a publication is generally not high. OpenAIRE collects today around 300Mi individual publication records. Some of these, collected from Unpaywall, Microsoft Acedmics and Crossref, are pre-merged using DOI as pivot to populate the *DOIBoost collection* (La Bruzzo *et al.*, 2019), which returns around 85Mi publication records. This optimization results into a total of around 150Mi publication records to be deduplicated.

Datasets are different in nature as they typically reside in one data archive, database, or institutional repository. As a consequence their cross-data source duplication arises from collecting the same objects from different aggregators, with a very low rate. On the other hand, the same object may have different versions within the same data source. OpenAIRE collects today around 12.5Mi datasets records to be deduplicated.

Software products are not so common into repositories, as most scientists still deposit them into software development repositories, such as GitHub. Recently, the deposition of code as a scientific product is more and more becoming the norm, and platforms such as Zenodo.org or Figshare.org, which provide support for software deposition, metadata and DOIs, are indeed playing a key role on this cultural shift. Duplication of software products has the same features of dataset duplication, but a much lower dimension, OpenAIRE harvesting today around 300K software records.

Other research products (ORPs): ORPs duplication features may match the ones of publication, datasets, and software, as ORPs may in nature be similar to any of the three in terms of cross-data source and, intra-data source duplication. OpenAIRE harvests today around 7.5Mi ORP records to be deduplicated.

Organizations: Organizations are mainly collected from CRIS systems and entity registries, such as project databases from the European Commission (CORDIS) or other funders today included in OpenAIRE (around 30) and the Global Research Identifier Database [9] maintaining an authoritative list of research organizations. Their duplication can be due to both intra-data source and cross-data source issues. Some data sources provide unique identifiers for organizations in the metadata, others provide unique names, but not identifiers and others provide organization names inserted by users, hence potentially different for the same organization within the same data source. To further increase the chaos there is no best-practice

on "granularity" for organization names (i.e. some provide a department, some the principal institution, some provide both), on the language to be used and on the specific structure (e.g. University of Warsaw and Warsaw University). Finally organizations change names over time. OpenAIRE harvests today around 400K organization records to be deduplicated.

From this analysis it is clear that deduplication of the OpenAIRE graph requires a wide variety of functionalities and skills. For publications numbers can be very high, in the order of hundreds of millions, and grow every day generally of a small fraction: this means computational performance is an issue as heuristics alone cannot mitigate the time to compute over such large collection; to cope with evolution, deduplication should count on an incremental approach where deduplication can be executed by adding the new records, without running deduplication on the whole collection; finally, humans must be able to provide feedback to the system in order to permanently fix the inevitable glitches of an automated approach, which leads to false positives or negatives. For organizations cardinality is not an issue as we are targeting about hundreds of thousands, but deduplication suffers from the lack of informative metadata and the lack of best practices mentioned above. Accordingly, humans play an important role in providing feedback to the results of deduplication, and the possibility to count on existing "ground truth" of organizations provided by other organizations, such as isni.org and grid.ac, becomes crucial. Ground truth of organizations are curated by humans and for each organization they keep a number of "aliases" that can be very useful to deduplicate otherwise mismatching organization names or acronyms collected in OpenAIRE. Finally, for all entities, once the groups of similar objects have been identified, specific merging techniques must be adopted that replace groups with one "representative object" for the group and associate the relationships of the merged objects to such representative.

2.3 State-of-the-art and motivations

The deduplication of big data graphs requires a system capable of supporting an end-to-end workflow, which initially focuses on the identification of duplicates for the different entity types and concludes with the construction of a de-duplicated graph. More specifically, a system capable of supporting such workflow should tackle the following high-level challenges:

(1) Graph model: should be able to represent and manage graph-shaped information spaces, hence handle multiple entity types and relationships between them;

(2) General purpose-ness: should be customizable and configurable in order to cope with graphs of any type and whose entities can manifest different deduplication scenarios;

(3) Ground Truth: should be able to import ground truth sets as well as generate ground truth sets from deduplication results;

(4) Scalability: deduplication is a challenging task *per se*, especially in terms of computational cost; in order to process large graphs the system should ground on parallel computing techniques;

(5) Data curators feedback: no machinery will ever replace human ability to judge whether two objects of the same entity are indeed duplicates or should never be regarded as such; to this end such system must allow domain experts to evaluate the results and provide feedback to the system.

Looking up the literature, it is clear that deduplication research has a long history (Fellegi and Sunter, 1969; Köpcke *et al.*, 2010): record linkage, entity resolution, duplicate detection, co-reference resolution, object consolidation, reference reconciliation, fuzzy match, object identification, object consolidation, entity clustering, merge/purge, identity uncertainty and many other research topics may offer different interpretations and solutions to the problem of

deduplication. An analysis of the requirements of the functionalities for deduplication and record linkage systems was proposed by Köpcke, Thor and Rahm in (Köpcke *et al.*, 2010). In fact, as a result of such studies several deduplication tools have been developed (Jurczyk *et al.*, 2008; Manghi *et al.*, 2012b; Christen, 2008; Kang *et al.*, 2008). Our work is orthogonal to most of the large body of work in deduplication since, to our knowledge, no researcher has worked on systems for the deduplication of big data graphs: some approaches address the problem of record linkage or entity resolution for "big data" flat collections, while others tackle disambiguation of "graphs", but none delivers the full workflow of big data graph deduplication as described above.

Among the first category we can mention Dedoop [10] (Kolb and Rahm, 2013) and PACE (Manghi *et al.*, 2012b). Both tools are built on distributed column stores, respectively Hadoop MapReduce and Cassandra. They allow to efficiently process large collections in parallel to identify duplicates and offer flexible configuration of blocking functions (so as to match the Map Reduce processing model they adopt) and general-purpose matching functions.

For the second category, on the side of graphs, approaches have been explored but not effectively implemented, such as (Bhattacharya and Getoor, 2004) and Saeedi *et al.*, 2008) which consider the semantics of graph links as input of deduplication and disambiguation, and Dedupalog (Arasu *et al.*, 2017), which focuses on large-scale graph deduplication.

Dedupalog addresses optimization of clustering in graph deduplication by taking into account constraints identified by relationships between entities, for example "equal publications must have the same related organizations". The underlying algorithm produces a clustering that optimizes the number of matches to be performed in the Reduce phase of optimization. Constraints are expressed using a Datalog-like language, which improves previous approaches in terms of usability and performance. The authors provide numbers relative to an implementation developed on top of an SQL database which performs clustering in 2 min over a "big collection" of 450K publication bibliographic records from the ACM library. Scalability to hundreds of millions of records (300Mi in the case of OpenAIRE for example) would therefore require a different implementation, likely devised keeping parallel computing design in mind. The outcome of Dedupalog is indeed relevant to the topic, but addressing only one of the aspects of deduplication of big graphs identified by the OpenAIRE use-case. Dedupalog does not support disambiguation by merging of objects and relationships, ground truth management, human data curation, scalability to arbitrary numbers of records, etc.

To conclude, practitioners in the need of disambiguate big graphs may reuse existing tools or techniques but then would require to assemble them to form custom deduplication solutions for their graphs. Such ad-hoc solutions are typically complex to engineer, expensive to maintain and hardly reusable in different contexts by others.

3. GDup architecture

We have designed and implemented a system for graph deduplication called GDup (Atzori, 2016) (Atzori *et al.*, 2018) whose aim is to address the general lack of tools capable of addressing a complete graph deduplication workflow, from the graph input phase to the materialisation of the disambiguated graph, enhanced by end user feedback and supported by ground truth. The architecture of system is depicted in Figure 2, whose main functional areas support an end-to-end workflow enabling data curators at:

(1) Graph import: importing their graph in the system;

(2) Candidate identification and matching: configuring for each entity type the relative duplicate identification policy;

(3) Ground Truth injection: generation of ground truth from deduplication results and injection of ground truths in future candidate identification phases;

(4) Duplicates Grouping and Merging: configuring graph disambiguation strategies;

(5) Curators feedback: supporting data curators at providing feedback to the results of deduplication;

(6) Graph export: exporting the output deduplicated graph in a standard exchange format.

In the following we will first describe the type and object language of GDup based on the Property Graph Model (Rodriguez and Neubauer, 2010), then formally introduce the individual areas of the architecture and the relative functionalities.

3.1 Graph data model

The workflow depicted in Figure 2 takes an input "raw" graph $\mathcal{G}_R$ and applies a number of functions that change the topology of the initial graph by adding or removing objects and edges to yield its "deduplicated" version $\mathcal{G}_D$. To semantically describe this workflow, GDup's data model must be able to represent (1) the type schema of the input graphs, since duplicate conditions are formulated at the level of entities and based on the properties of such entities, and (2) different logical views (in the following "overlays") of the graph, since the transition of the graph from one workflow stage to the next will be logical. Among known models for graph representation, the *Property Graph Model* (PGM) (Robinson *et al.*, 2015) has become quite popular in graph databases implementations due to its expressiveness, which subsumes several and simpler forms of graphs types, and its extensions to match specific representation challenges. Of interest to our work are the Extended PGM graphs (EPGM), described as a set of *vertices* and *edges* (Junghanns *et al.*, 2015) where:

(1) *Vertex*: an object that has a unique identifier, a label that denotes its type (i.e. properties) and potentially incoming and outgoing edges;

(2) *Edge*: an object that connects two vertices, may have properties, has a label that denotes the type of relationship between the two vertices, and has a direction, that is head vertex and tail vertex;

(3) *Properties*: are a set of key/value pairs associated to a vertex;

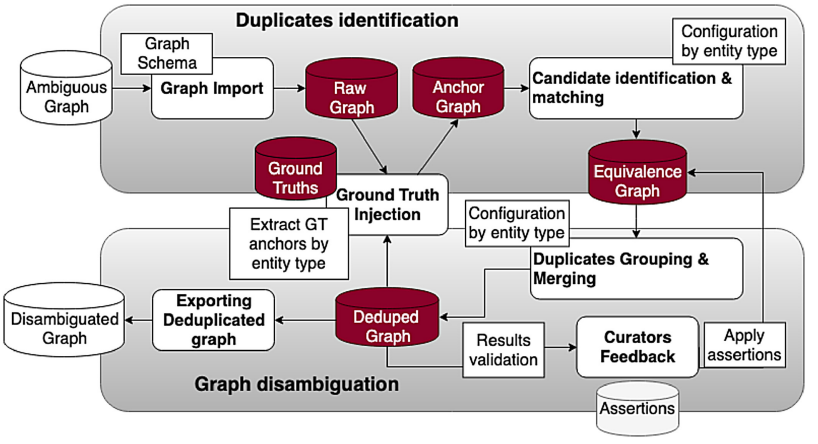

Figure 2.
GDup: architecture of de-duplication workflow

(4) *Overlays*: are labels tagging vertices and edges in order to define "logical graphs", that is subset of vertexes and edges bound together by some logic; the same vertices and edges may belong to different logical graphs.

The concept of *overlay* is of particular interest since it allows the co-existence of several graphs interpretations within the same graph representation. Such logical tagging reflects directly possible underlying representations. In order to introduce a schema for our graphs, that is an expected structure of vertexes and edges, we defined the *Structured Property Graph Model* (SPGM) as an extension of EPGM that includes the notion of *graph schema*. A graph schema is a set of assertions V_T that associate the label of a vertex type in EPGM with a "type structure". More specifically:

(1) *Graph schema*: a set of assertions that constrain the name of a vertex type (i.e. a vertex label in EPGM) with a specific set of structured properties and edge types with a semantic label for each of the directions and a set of structured properties;

(2) *Structured properties*: vertex properties are defined as a set of label-value pairs, where label is the attribute name and the values are primitive types (integer, float, string, . . .), lists of such values, or set of properties;

(3) *Vertex*: an object that has a unique identifier, a label that denotes the entity type it conforms to, the values instantiating its type, and one or more overlay labels;

(4) *Edge*: An object that has a label that denotes the type of relationship between its two vertices, a source and a target vertex and one or more overlay labels.

More specifically, a graph schema is a set of vertex types $[V_{T_1} \ldots V_{T_m}]$ defined as assertions:

$$V_T =< [l_1, \ \ldots, \ l_n], \ \{ \rightarrow^{s_1} V_{T_1} \ldots \rightarrow^{s_m} V_{T_m} \} \rangle$$

where V_T is the name of the type, the l_i's are the property labels of the type V_T, while $\leftrightarrow^{s_i} V_{T_i}$'s indicates a relationship type between V_T and V_{T_i} whose semantics is s_i. Accordingly, an SPGM graph is an EPGM graph whose nodes conform, that is respect the structure, of a given graph schema. Figure 3 illustrates an example of an SPGM graph instance with *Raw graph* overlay, which includes only one type of objects *Article*, whose objects are described by three properties *title*, *authors*, and *issn* and a self-relationship *cites*:

$$Article =< [title, \ authors, \ issn], \ \{ \rightarrow^{cites} Article \} >$$

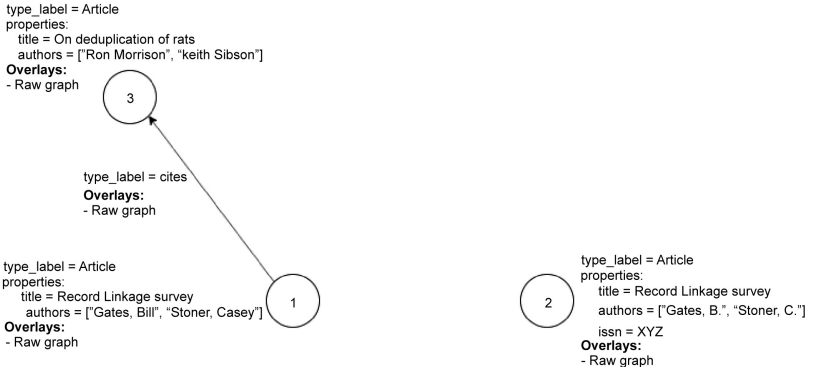

Figure 3.
Structured Property Graph: raw graph

3.2 Deduplication workflow

In this section we illustrate more in detail the functional architecture of GDup by presenting the functional breakdown of the deduplication workflow it implements. More specifically, it is composed of the following phases, as illustrated in Figure 2:

3.2.1 Graph import. The import phase is responsible of importing a graph expressed according to known standards, such as RDFG and JSON-LD, into the graph database of GDup, defined according to a given graph schema. Data curators are responsible for mapping standard exchange format for graphs onto their SPGM representation (mappings onto the PGM model were proposed in (Rodriguez and Neubauer, 2010; Hartig, 2014)).

3.2.2 Candidate identification and matching. This phase operates over an *anchor graph* $\mathcal{G}_A$, obtained by "injecting" into the raw graph $\mathcal{G}_R$ the ground truth for each type V_T, to identify pairs of equivalent objects. For simplicity, we will assume that $\mathcal{G}_A = \mathcal{G}_R$ as if no ground truth was injected, and introduce the benefits of ground truth later on. Data curators initiate this phase by selecting/defining, for each type V_T, a configuration for candidate detection and for candidate matching of the objects in V_T. Candidate identification consists of a set of *duplicate identification configurations*, while candidate matching fine tunes the *similarity function* to be applied to determine equivalence of two objects in terms of their properties. Precision and recall of a solution depend on the ability of data curators to fine-tune such configurations.

Candidate identification Deduplication systems often embed techniques to reduce the inefficiency of quadratic complexity derived by the pairwise comparisons between the objects. To this aim GDup provides a selection of clustering hash functions, which group objects into "blocks" of likely similar objects. The purpose of such functions h is to associate objects $o \in V_T$ to the same "block" $B_{h(o)}$ (or *canopy* (Kolb *et al.*, 2010; McCallum, *et al.*,2000)) if the objects are potential duplicates, in order to limit the comparisons to the worthy ones and achieving logarithmic complexity. The main challenge is therefore to define a function h that $\forall o, o' \in V_T$ analyses their properties and sends them to the same block B_k (namely when $h(o) = h(o') = k$) maximizing the chances that o and o' are equivalent; h should be tailored to the semantic features of V_T, in order to minimize false positives, that is objects that are not equivalent and associated to the same block B_k, and false negatives, that is equivalent objects associated to different blocks. GDup allows to add multiple clustering functions within the same configuration, causing the same object to be included in more than one block, hence minimizing false negatives. Examples of h are functions calculating *ngrams*, fetching *prefixes of words*, etc.

Listing 1: GDup clustering functions configuration for the type V_T = Article

```
1  VertexType = Article,
2  clusteringFunction[ \\ from each title, generates 3 ngrams of size 3 from all
          words of minimal length 5; each title may generate more than one
          clustering key
3   name = ngram
4   params = [minWordLen = 5, ngramLen = 3, ngramNum = 3]
5   property = [title]
6  ],
7  clustering Function [ \\ for all words of minimal length 4 generates a string
          combining 1 char from the head and 1from the tail of the word,then
          appends the results to generate one string; each title generates one
          clustering key
8   name = suffix − prefix − synthesis
9   params = [minWordLen = 4, headLen = 1, tailLen = 1]
10  property = [title]
11 ]
```

As shown in listing 1, GDup allows to add multiple clustering functions within the same configuration, causing the same object to be included in more than one block, hence minimizing false negatives. Each hash function can receive as input one or more of the object property values defined in a given V_T, and return one or more clustering keys, depending on the function specific parameters.

Candidate matching Candidate matching is the phase in the deduplication workflow that performs the comparison between object pairs. The object matching operation in GDup is defined as the computation of a similarity measure F_{sim} between two objects, mapped in the range $[0 \ldots 1]$, where 1 indicates perfect match. A match between a pair of objects is considered positive, and the objects equivalent, when the score obtained by a given similarity function reaches a given configurable threshold $Th \in [0 \ldots 1]$. GDup supports fine tuning of candidate matching configuration, by setting up a *similarity function* with the relative *block sliding window* and configuration of *matching pre-conditions*.

Similarity function The properties of an object can contribute to the equivalence match in different ways. For example, when matching publications, the *title* can be considered as more relevant than the *publisher*. Hence, GDup can be configured to associate to each property l_i a similarity function f_i with a *weight* w_i. Given two objects o, o' belonging to a given Entity Type V_T, the system calculates the similarity $F_{sim}(o, o')$ as the weighted mean of the contributes from the different similarity functions, as defined in the configuration:

$$F_{sim}(o, o') = \frac{\sum_{i=1}^{n} (w_i f_i(o.l_i, o'.l_i))}{\sum_{i=1}^{n} w_i}$$

Where $\sum_{i=1}^{n} w_i = 1$ and $0 <= f_i <= 1$ are respectively the weights and the similarity functions w.r.t. to each object property l_i. In order to adapt to different application domains, GDup supports a predefined set of general purpose and established similarity functions f, and a mechanism to easily include new ones. The selection of such functions was inspired by the work of Cohen, Ravikumar and Fienberg in (Cohen *et al.*, 2003) and the most relevant ones are: *ExactMatch*, *JaroWinkler*, *Level2JaroWinkler*, *Level2Levenstein*, *Levenstein*, *AuthorSurnamesDistance*, *SortedJaroWinkler*, *SubStringLevenstein*. Others have been added and obtained as "specializations of known similarity functions to adapt to special cases: for example *LevensteinTitle* preprocesses the title strings to normalize" them before applying Levenstein distance.

Moreover, GDup provides a selection of functions *exp* used to equally harmonize input values before the similarity functions are applied. Such functions are associated to each l_i and are applied to both $o.l_i$ and $o'.l_i$ before f_i is computed. Examples are *RemoveBlankSpaces*, *removeCapitalLetters*, *extractNumbers*, *removeStopwords*, etc. which can of course be combined. An example of such configuration is shown in listing 2.

Listing 2: GDup similarity function for V_T = Article, based on properties *title* and *authors*

```
1  EntityType = Article,
2  Threshold = 0.98
3  similarity Function[
4        name = LevenshteinDistance
5        weight = 0.8
6        exp = [remove Stop words, remove Capital Letters]
7        property = [title]
8  ],
9  similarity Function [
10       name = PersonDistance
11       weight = 0.2
12       property = [authors]
13 ]
```

Block sliding window Once the configuration of the similarity functions for all interested types V_T is set, GDup runs pair-wise comparisons in all blocks B to identify equivalent objects. Since the number of objects in B's maybe very high, as well as the number of blocks B generated for V_T, GDup allows to fine-grain configure a sliding window heuristic to further reduce the number of matches (Wang *et al.*, 2016). The heuristics sorts the objects in B using a sorting function *sort* then, given a "window size" w, applies F_{sim} to the first element of B with the following w objects; once finished, the algorithm moves to the second element of B and repeats the matching phase for another w objects. The algorithm ends when it reaches the last element in (B. Users can configure GDup to specify which sorting function to use among a list of functions (e.g. DESC, ASC, although custom functions can be added) and to which property l_i it must be applied.

An example of such configuration is shown in listing 3.

Listing 3: GDup sliding window configuration for V_T = Article

```
1  Entity Type = Article,
2  sliding Window[
3        property = title
4        sort = DESC
5        windowLen = 100
6        exp = remove Stopwords, remove CapitalLetters, Remove BlankSpaces
7        maxBlockLength = 2000 \\ the elements of a block can be limited to the
               first 2000 sorted elements, the rest is disregarded by the sliding
               window
8  ]
```

Matching pre-conditions In order to further optimize the candidate matching phase, GDup supports the definition of *pre-conditions* that, given a pair of objects o and o' in B, allow to determine whether the objects are equivalent or distinct (Arasu *et al.*, 2017). Pre-conditions are first-order logic predicates $P(o, o')$ over properties of the two objects, to be evaluated before the similarity functions. If *true*, positive preconditions set $f(o, o') = 1$ while negative preconditions set $f(o, o') = 0$. If none of the predicates are matched the similarity function is applied.

An example of preconditions is shown in listing 4.

Listing 4: GDup preconditions for V_T = Article

```
1  EntityType = Article,
2  precondition[ \\ if the two articles have the same identifier, they are the
         same article
3        class = positive
4        predicate = ExactMatch
5        exp = normalizePID
6        property = identifier
7  ]
8  precondition[ \\ if the titles contain different numbers as characters they
         are relative to different articles
9        class = negative
10       predicate = Mismatch
11       exp = normalizePID
12       property = title
13 ]
```

The duplicate identification phases generate a set of pairs of equivalent objects, which in turn constitute the equivalence graph $\mathcal{G}_E$. As a result of the matching, whenever the distance between two objects o and o' successfully passes the given threshold, then the following

actions are performed: (1) the relationship *equalsTo* between the two is added to the graph and attached to the overlay *equivalence graph*, and (2) both objects are attached to the overlay *equivalence graph*–by "adding the object/relationship to a graph" we imply the object/relationship is decorated with the overlay of the target graph. Figure 4 shows the graph $\mathcal{G}_E$ obtained for our example, after identifying the equivalence between objects 1 and 2.

Before moving to the next phase of duplicates grouping and merging, the graph $\mathcal{G}_E$ is further cleaned by applying the data curator feedback assertions. As described in section 3.2.5 the assertions refine the de-duplication process by adding further relationships or removing relationships from the equivalence graph $\mathcal{G}_E$ as indicated by expert users.

3.2.3 Duplicate grouping and merging. Graph disambiguation consists of two distinct phases. The first phase is *duplicate grouping* and is in charge of identifying all connected components (*CC*) in the equivalence graph $\mathcal{G}_E$ with overlay *equivalence graph*. The second phase is *duplicate merge* and is responsible of, given all connected components, generating a representative object and distributing the relationships of the merged objects to keep the graph topology coherent with the newly created representative object.

Duplicates grouping Duplicate grouping consists in identifying the connected components in $\mathcal{G}_E$ by exploiting the transitivity of the equivalence's relationship *equalsTo*. For example, if $< o_1, equalsTo, o_2 >$ and $< o_2, equalsTo, o_3 >$ we can conclude that $< o_1, equalsTo, o_3 >$. If no other object in $\mathcal{G}_E$ is reached via a relationship *equalsTo* from o_1, o_2, or o_3, then the group of objects o_1, o_2, o_3 is a connected component and represents a set of equivalent objects, ready to be merged into one object. The grouping phase distributes equivalence relationships and adds them to the overlay of $\mathcal{G}_E$ until all its connected components are found. The problem is typically solved using heuristics (Tomaszuk and PÆ'Ä-k, 2018).

Duplicates merging Once duplicate grouping is completed the deduplication workflow proceeds with the action of merging the objects in each group. For each connected component this phase builds a *representative object*, elected to virtually replace all duplicates in the group. To this aim, two issues must be tackled: *representative object election* and *distribution of relationships*.

Representative object election In this phase, for each connected component in $\mathcal{G}_E$, GDup builds a representative object out of the information provided by the objects therein. Important aspects involved in the process are (1) generating a stateless identifier for the representative object (the same identifier is produced from the same group of duplicates), and (2) merging strategy of the duplicate objects' properties. To this aim, GDup first elects a *pivot*

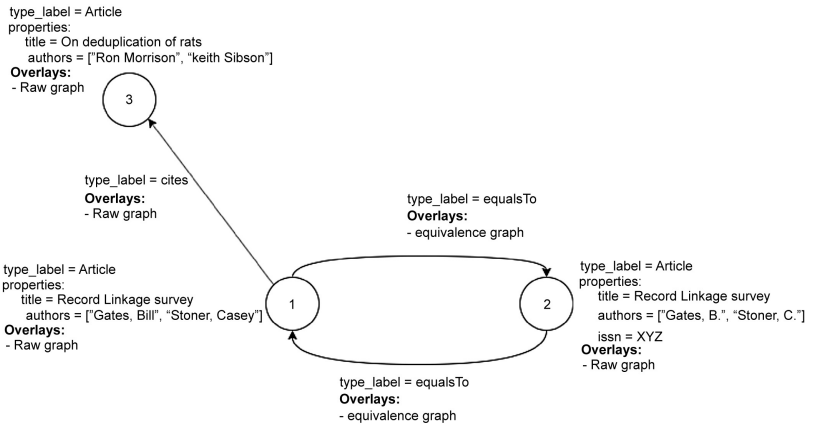

Figure 4.
Structured Property Graph: equivalence graph

object, which is the object with the "smallest" identifier in the group, when sorted in lexicographic ascending order. The representative object identifier is generated by appending to a prefix *dedup_* to the identifier of the pivot object. The properties of the representative object are then generated starting from the pivot object and, for each other object in the group, following the shadowing strategy as indicated by the user for each property: *ifMissing*, if the pivot object does not have a value for a property (e.g. date property), the merge adds the first such property found in the merged objects; *enrich*, the pivot object is enriched with all values found in the merged objects (e.g. subject properties).

Listing 5: GDup shadowing strategy: building a representative objects for V_T = Article

```
1  EntityType = Article,
2  representativeObject[
3      properties [
4          [ property = identifier
5              condition = ifMissing \\ add identifier value if none is
                   available in the object
6          ]
7          [   property = title
8              condition = ifMissing \\ add title value if none is available
                   in the object
9          ]
10         [   property = issn
11             condition = enrich \\ add issn value if
                   is not already available in the object
12         ]
13     relationships [
14         ByType = {< cites, Article >}
15         ByPivot = { }
16     ]
```

Distribution of relationships The creation of a representative object implies the dismissal of the objects it merges, hence modifies the graph's topology. To preserve the consistency of the graph, relationships engaging merged objects must be propagated to the representative object. GDup allows users to pick one of the following relationship distribution policies: *ByType*, that is all relationships with objects of the type listed are to be redistributed to the representative object, and *ByPivot*, that is for the given types, only the relationships associated to the pivot object are kept and the ones of the other objects in the group disregarded. In the example, only the relationships to the authors of the pivot object are regarded. This strategy is conservative as author object deduplication is highly subject to errors and the very same author may be represented multiple times in the graph, as different objects. Applying a *ByType* technique for the Author type would likely create article representative objects related with redundant authors.

The combination of duplicate grouping and duplicate merging generates a new dedup graph $\mathcal{G}_D$ identified by the overlay *dedupedGraph*. $\mathcal{G}_D$ is created as follows:

(1) Adding the new representative objects *ro* obtained from each connected component *CC* (i.e. group of duplicates) as well as new relationships $< ro, merges, o >$ for each $o \in CC$; such relationships are part of the deduplicated graph, in order to provide evidence and "provenance" of the deduplication process;

(2) Adding the relationships created from and to the representative object as specified in the relationships distribution configuration;

(3) Adding all objects in $\mathcal{G}_R$ that are not in $\mathcal{G}_E$; the deduplicated graph should of course include the objects that are not subject to deduplication;

(4) Adding all relationships in $\mathcal{G}_R$ that are not incoming or outgoing an object in $\mathcal{G}_E$: all such relationships were replaced by new and corresponding relationships to the representative objects.

Figure 5 shows the $\mathcal{G}_D$ graph obtained from the $\mathcal{G}_E$ in Figure 3. The graph consists of one representative object *dedup*_1 obtained from merging the nodes 1 and 2, which do not belong to $\mathcal{G}_D$. *dedup*_1 inherits all properties of the two objects and their relationships to 3.

3.2.4 Ground truth injection. A Ground Truth is a graph gt_{V_T} that includes a set of representative objects, and relative merged objects in $\mathcal{G}_D$, considered to be of high quality and trust. The introduction of a Ground Truth approach changes the perspective of the deduplication process, since gt_{V_T}'s can used to pre-process the input graph $\mathcal{G}_R$ in order to replace sets of merged objects with the relative representative object before the deduplication phase. This optimizes performance of the process but, most importantly, leads to a stable graph, where identifiers and object converge to a highly-trusted and stable graph.

A Ground Truth gt_{V_T} is a set of equivalence assertions represented as "trees" rooted in representative objects whose leaves are the merged objects, reached by relationships *merges*. Due to their high level of trust, such representative objects are called *anchor objects*. Figure 6 shows a possible ground truth $gt_{Article}$ consisting of one only assertion. The assertion contains the anchor object $dedup_1$ and the edges $< dedup_1, merges, 1 >$ and $< dedup_1, merges, 2 >$.

As a good practice users may, for example, run deduplication using highly successful configurations (i.e. the "obvious" matches) to generate results (representative objects) which are then stored as ground truths for each type V_T. This way, in subsequent rounds of deduplication, the input graph $\mathcal{G}_R$ can be injected with the "obvious" ground truth matches and users can focus on more experimental and complex levels of deduplication configuration. This pre-processing phase is also useful for two other reasons: (1) reducing the computational time of entity deduplication by limiting the identification of duplicates to the objects that are not yet merged by the Ground Truth, and (2) progressively converging to a stable deduplicated graph by applying deduplication refinements with different deduplication configurations, each based on the results of previous ground truths.

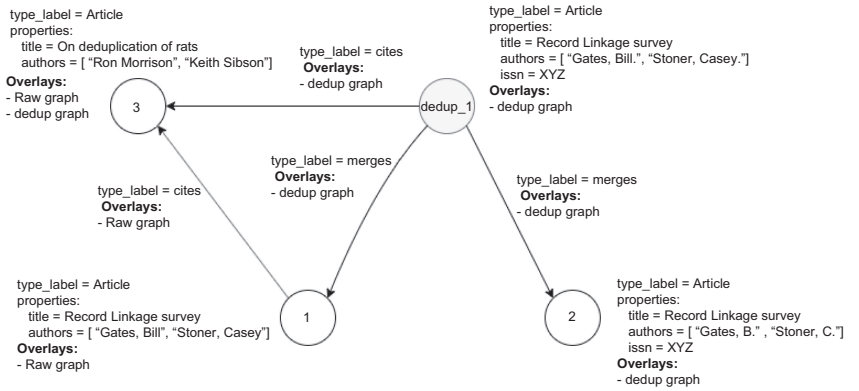

Figure 5.
From raw graph to equivalence graph and to deduplicated graph

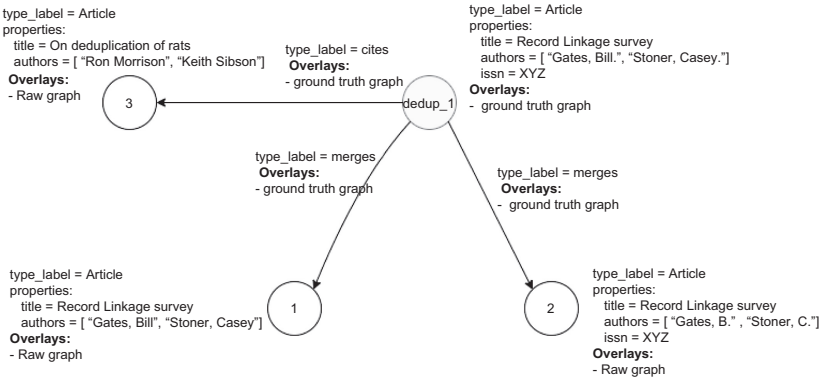

Figure 6.
Example of ground truth assertion

More specifically, given a set of ground truths gt_{V_T}'s selected by the data curator, GDup obtains from the raw graph $\mathcal{G}_R$ a normalized anchor graph $\mathcal{G}_A$ by:

(1) Adding all objects and relationships in $\mathcal{G}_R$ to $\mathcal{G}_A$;

(2) Adding all anchor objects in gt_{V_T}'s to $\mathcal{G}_A$;

(3) Adding all relationships incoming and outgoing the anchor objects to $\mathcal{G}_A$;

(4) Removing from $\mathcal{G}_A$ all objects in $\mathcal{G}_R$ that are merged by anchor objects, as well as their relationships to other objects.

Figure 7 shows an example of the anchor graph $\mathcal{G}_A$ resulting from the injection of the ground truth gt_{Article}. The application of the gt_{Article} to the raw graph with objects 1, 2 and 3 generates an anchor graph $\mathcal{G}_A$. Given the initial raw graph $\mathcal{G}_R$, a further deduplication process would deliver no changes to the graph and deliver a deduplicated graph $\mathcal{G}_D$ that matches exactly the anchor graph $\mathcal{G}_A$ (see Figure 8).

3.2.5 Data curators feedback. GDup allows experienced users to supervise the deduplication workflow, refine the results, and then exploit such feedback in subsequent applications of the process to refine the candidate matching phase. Data curators are provided with tools with which they can search, browse and visualize the objects of any entity type in the $\mathcal{G}_D$, in order to "repair" representative objects that were not correctly created or create groups of duplicates that were overlooked by the process. Such feedback results in a set of assertions for each entity type, which can be of two kinds:

(1) *Equality assertion:* an equality assertion is a group of objects $\{o_1, \ldots o_n\}$ of type V_T in the graph $\mathcal{G}_D$, where the objects can be representative objects or raw objects in the graph. The assertion claims that all raw objects in the input graph $\mathcal{G}_R$, directly or indirectly (via a representative object) involved in the set, are equal.

(2) *Diversity assertion:* a diversity assertion states that two objects o and o' are distinct: the claim is represented as a relationship $< o, \mathit{differsFrom}, o' >$. If either or both of

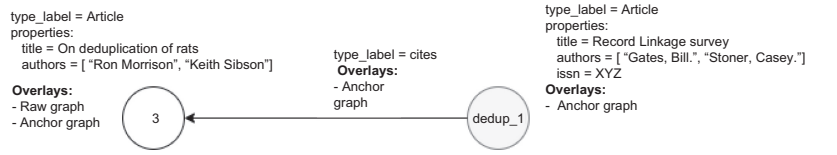

Figure 7.
From raw graph to anchor graph via ground truth gt_{Article}

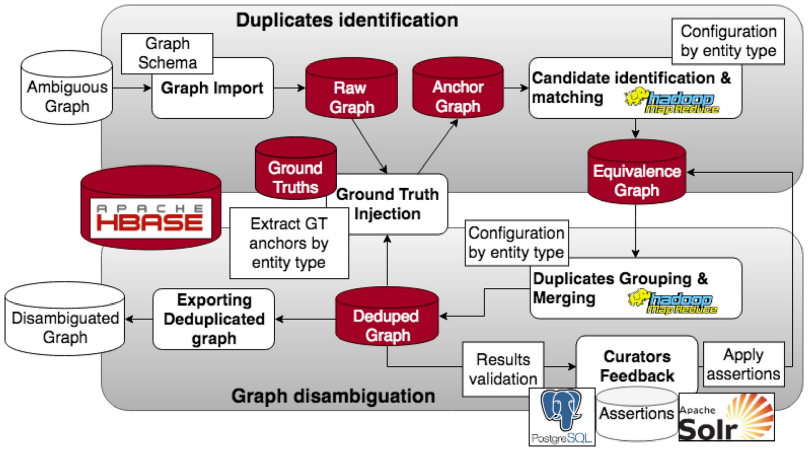

Figure 8.
GDup deduplication workflow: supporting technologies

the objects are representative objects, the claim naturally extends to all objects merged by the representative ones.

(3) These relationships populate a feedback graph $\mathcal{G}_F$, which keep the history of such assertions over time. Once the equivalence graph $\mathcal{G}_E$ is generated, $\mathcal{G}_F$ is used to update its content as follows:

(4) Adding to $\mathcal{G}_E$ all relationships $< o, equalsTo, o' >$ in $\mathcal{G}_F$;

(5) Removing from $\mathcal{G}_E$ all relationships $< o, equalsTo, o' >$ or $< o', equalsTo, o >$ for which a relationship $< o, differsFrom, o' >$ exist in $\mathcal{G}_F$.

3.2.6 Graph export. Any time, data curators can opt to export any of the overlay graphs populated in this process. The export can follow, as for the input, a mapping from the internal graph schema provided by data curators and one of the standards supported by the GDup.

4. GDup implementation

The implementation of GDup realizes the functionalities described in the previous section by assembling known Open Source technologies (GDup Release 1.0 (Atzori and Manghi, 2017)), as shown in Figure 5. GDup is today used in the production system of OpenAIRE to deduplicate publications, datasets, software, and organizations entities in the information graph. In the following we give an high-level description of the implementation and also report on numbers and performances in the adoption of the tool.

4.1 Graph database

To achieve the intended objectives, GDup 's graph database should support, scalability of size, parallel processing, flexibility of models and efficient bulk read and write operations. Such conditions exclude the adoption of classic graph databases, oriented to efficient graph traversal functionalities (Rodriguez, Neubauer). Since the beginning of the project, back in 2010, the OpenAIRE infrastructure based its solution and services on HBase technology [11] (George, 2011). Hbase is the open source version of BigTable (Chang *et al.*, 2008), the large volume distributed storage system developed by Google: "a distributed storage system for

managing structured data that is designed to scale to [. . .] petabytes of data across thousands of commodity servers". Hbase data model consists of a sparse, distributed, persistent multi-dimensional *sorted map*. Such map is indexed by a row key, column key, and a timestamp, and each value in the map is an uninterpreted array of bytes. Hbase is based on the Hadoop framework [12], enabling large scale distributed data processing and analytics based on Map Reduce programming model (Dean and Ghemawat, 2008; Lee *et al.*, 2012).

Hbase represented the optimal candidate for bulk integrating, populating, storing, processing, deduplicating graphs while addressing all non-functional requirements above; since the average real-case we were confronted with is characterized by low-density graphs, a representation of graphs as adjacency lists was adopted. For the future, as part of the OpenAIRE services road map, a solution based on Apache Spark [13] is to be designed, covering all the deduplication theory and methods described in this paper, but also meeting the requirements of frequent metadata aggregation and integration of graphs demanded by OpenAIRE services.

The GDup graph is represented in HBase by associating each object in the graph to an HBase row. Each row contains the following columns: (1) the identifier of the entity, (2) the *type* of entity, (3) the body of the entity (its properties and values) and (4) one column for each relationship, where the name of the columns is constructed from the relationship semantics (e.g. cites), and the ID of the target row. This representation of the graph does not explicitly encode SPGM graph overlays. Starting from the raw graph, the anchor graph and the deduped graph are incrementally constructed by adding representative object rows and "virtually deleting" the rows/edges merged by these. This is modeled by adding a *deleted* column to the row and a *deleted* flag in the cell of the relationships to be removed. The equivalence graph is represented by explicit relationships *equalsTo* between the rows. With respect to the deduplicated graph in Figure 5, the objects *dedup*_1, 3, and 2 would be represented as depicted in Figure 9. Since the deduplicated graph is obtained by modifying the raw graph, GDup runs a "reset" Map job before a new deduplication workflow is run, which restores the original raw graph in HBase.

4.2 Deduplication workflow: implementation

As suggested in Figure 5 the core phases of the deduplication workflow manipulate the graph in HBase via Hadoop MapReduce processing jobs. Other technologies, such as Solr and PostgresDB are used to implement the data curator tools required to explore the deduplicated graph, to store assertions and ground truths. The following sections provide insights on the implementation of GDup for each phase of the workflow.

4.2.1 Candidate identification and matching. The candidate identification phase is implemented as a MapReduce job that (1) in the Map phase applies the clustering function to all rows of a given type to generated blocks of candidate objects, and (2) in the Reduce phase

	Type	Body	RecI:cites:3	RecI:cites:1	RecI:cites:2
dedup_1	Article	indentifier = 10.3405/6574.3 title = Record Linkage survey subjects = computer science...			

	Type	Body	RecI:cites:3	deleted
1	Article	indentifier = 10.3405/6574.3 title = Record Linkage survey authors = ["Gates, B." "Stoner..	deleted	

	Type	Body	RelI:isCited:2	RelI:isCited:dedup_1
3	Article	indentifier = 10.0007/Jom435 title = On deduplication of rats authors = ["Ron Morrison", ...	deleted	deleted

Figure 9.
Graph object representation in HBase

applies the candidate matching function, while sorting of the objects in a block is performed by exploiting the very same functionality as offered by Hadoop. Blocks are independent and the Reduce phase is therefore processed in parallel to perform pair-wise matching over a sliding window. Overall, the parallel execution in combination with heuristics significantly optimizes the execution time.

Precision, recall, and performance depend on configuration parameters and must be fine-tuned by data curators. The current GDup implementation does not yet implement user admin interfaces for the configuration of the workflow phases. Data curators need to edit the files containing JSON representations of such configurations which reflect the ones exemplified in the listings in the previous section. Users can add new clustering functions, matching functions, etc. to the system as instances of respective Java abstract classes.

4.2.2 Duplicate grouping and merging. Duplicate grouping requires an algorithm for the identification of connected components (Tomaszuk and PÆ'A-k, 2018). As explained in the previous section, GDup assigns to each connected component a representative object whose ID is the lexicographic lowest ID. GDup adopts an algorithm based on the "message passing" technique (Lin and Schatz, 2010). The technique recursively applies MapReduce jobs to propagate the minimum object identifier to related objects until no more propagation can be applied. The process ensures that all objects in a connected component of the graph are aware of the minimum ID in the component. The final Mapper can therefore send pairs $< dedup\_minimumID, body >$ to the Reducers, which can create the representative objects by merging all object bodies, following the directives in the respective configuration file, as exemplified in Listing 5.

4.2.3 Ground truth injection. Ground truths are generated by data curators, from the deduplicated graph, by indicating an entity type V_T whose representative objects are particularly reliable and must be preserved as a ground truth. A ground truth is stored in a separate Hadoop HBase table, which stores a copy of the deduplicated graph rows for representative objects (body and relationships) and includes columns of type *merges* :: *objectID* pointing to all objects "merged" by the representative one. Since no user interface is yet available, the creation of a ground truth can only be activated manually by data curators via shell scripts.

The injection of a ground truth in the graph is performed before the deduplication process is fired. This can happen manually via shell scripts or via APIs (as in the case of OpenAIRE, where data processing activities are managed by D-NET orchestration workflows (Manghi *et al.*, 2014)). The process is performed by Map jobs that scan the HBase table for the given ground truth and for each row create a representative row in the HBase graph table, virtually delete rows and columns deprecated by the merging process and adds the inverse relationships towards representative objects.

4.2.4 Data curators feedback. The deduplicated graph is indexed in a Solr 7.5.0 installation which creates a single shard collection for each deduplicated type V_T of the graph. Data curators can, through a GUI, explore the representative objects generated by the process and create assertions to refine the quality of the graph. Assertions are stored in a PostgreSQL database and are applied, on request of the data curator, after the duplicate identification. The process is performed with two separate writing/delete phases, respectively applying equality assertions, hence adding relationships *equalTo*, and diversity assertions, hence removing such relationships.

4.3 GDup experiments in OpenAIRE

The major production instance of GDup is deployed as a key service in the OpenAIRE infrastructure over an Hadoop cluster featuring 16 worker nodes, totalling 160 CPU cores, 480 GB of RAM and 21TB of allocated HDFS disk space–the cluster is used also for heavy full-text mining tasks. GDup is used to perform deduplication of publication, research data,

software, other research products and organization entities in the OpenAIRE Research Graph. The graph is the result of deduplicating around 320Mi metadata records collected from around 1,000+ data sources. Today OpenAIRE APIs offer access to the graph to service like Scopus, Zenodo.org, Open Science monitor of the Commission, and several other funders via OpenAIRE MONITOR [14]. In the following text, we shall describe the two phases of *candidate identification and matching* and *duplicates grouping and merging*, by providing configurations and numbers.

4.3.1 Deduplication of scientific products. Scientific products are characterized by different candidate identification and matching, due to the specific features of the entity records involved, and a common duplicates grouping and merging phase.

4.3.1.1 Publications: candidate identification and matching. Candidate identification generates blocks of publications using three functions: (1) if available, the normalized DOI string, (2) the string obtained by combining 3 characters of the first 2 longest words in the title, (3) the string obtained by combining the last 3 characters of the 2 longest word in the title. While the DOI captures the unique nature of the objects, the latter two are both a clear indicator of uniqueness of the publications while leaving flexibility towards minor differences (e.g. typos) in the titles. Candidate matching sorts publications with the title normalized by removing punctuation, stopwords, blanks, and numbers; it uses a sliding window of size 100 and a max number of elements of 2000. The matching function includes as positive preconditions the equality of DOIs ("if objects have the same DOI they are equivalent") and as negative preconditions the combination of two predicates: "cardinality of authors" and "minimal best equivalence threshold for author names". These two ad-hoc functions capture key preconditions of equality and are applied to the deduplication of publications, datasets, software, and ORPs: the former counts the number of authors for the two publications, which must be the same for record equivalence; the latter verifies that author strings are similar (in their best score pair-wise matches) above a minimal threshold of equivalence set to 0.6, which is considered well below the average threshold of the same function applied to equivalent records in a gold set. If preconditions are verified, candidate matching applies a similarity function, with a threshold of 0.98, matching the title of the articles. Dates are not regarded by the function, as for the OpenAIRE purposes different versions of the same publication should be merged into one record, and these may well bear different publication dates. As shown in Table 1, out of the 146Mi publications aggregated in OpenAIRE (November 2019) this configuration creates 14Mi blocks, producing 92,4Mi of *equalsTo* relationships out of 24.2Bi matches, in one day and half time. Subsequently, GDup identifies 56.8M duplicates, merges them in 20.9M representative records, to deliver a total of 110Mi records visible today at http://beta.explore.openaire.eu, for a duplication ratio of around 39%. Figure 10 shows the number Y of representative objects that group X records: interestingly, more than 90% of representatives merges 2–4 duplicates, and the number of representative objects with more than 8 records drops below 1, 000.

Datasets: candidate identification and matching

Datasets are deduplicated adopting the same configuration used for publications. As shown in Table 2, out of the 12.5Mi dataset aggregated in OpenAIRE (November 2019) this

Table 1. Publications deduplication statistics (November 2019)

Phase	Input	Execution time	Output
Candidate identification publications	146M		14M blocks
Candidate matching publications	14M blocks	~ 38*h*42′	24,2B comparisons & 92,4M *equalsTo* relationships

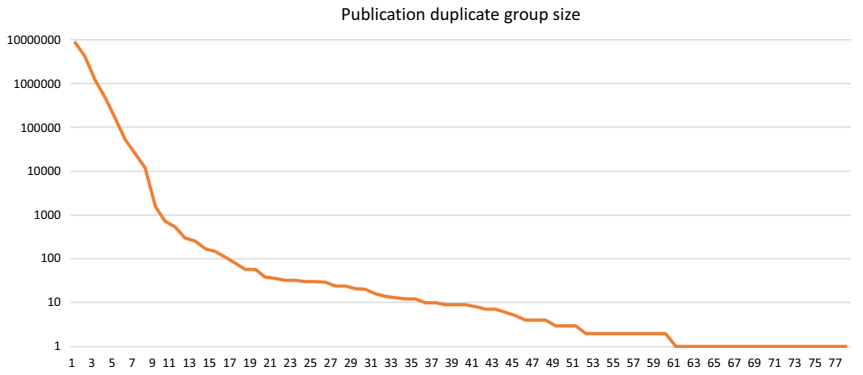

Figure 10.
Publication duplicate groups: distribution of group sizes

Phase	Input	Execution time	Output
Candidate identification dataset	12.5M		1.98M blocks
Candidate matching dataset	1.98M blocks	$\sim 1h8'$	421M comparisons & 4.2M *equalsTo* relationships

Table 2.
Dataset deduplication statistics (November 2019)

configuration creates 1.98Mi blocks, producing 4.2Mi of *equalsTo* relationships out of 421Mi matches, in around one hour. Subsequently, gdup identifies 4.5M duplicates, merges them in 1.9M representative records, to deliver a total of 8.5M records visible today at http://beta.explore.openaire.eu, for a duplication ratio of around 36%. Figure 11 shows the number Y of representative objects that group X records: interestingly, the distribution is not very different from publications, the representatives merging 2–3 duplicates touching 90%, and the number of representative objects with more than 8 records dropping below 1,000.

Software: candidate identification and matching

Software are deduplicated adopting a dedicated deduplication configuration, where software product titles are considered equal independently of the version numbering they

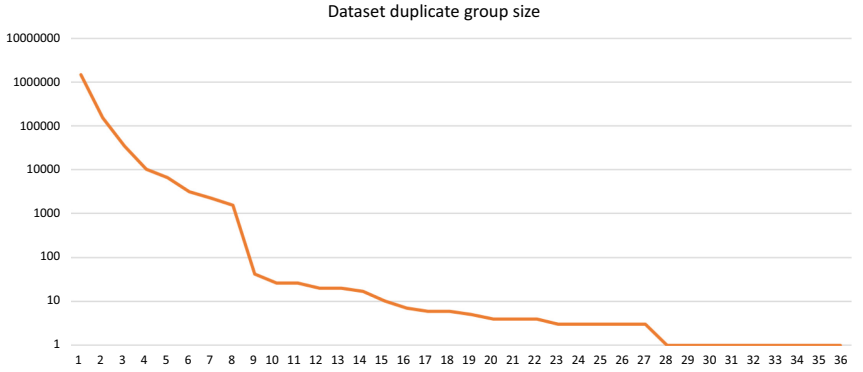

Figure 11.
Dataset duplicate groups: distribution of group sizes

feature. This is to make sure that the practice of publishing different versions of the same software as different products will not affect the statistics in OpenAIRE. Moreover, software repository URLs are also regarded as indicators of equality in the pre-conditions. As shown in Table 3 GDup is fed with 294K software records (November 2019) to create 181.2K blocks, processes 3.3M matches, and generates 402K *equalsTo* relationships in 17 minutes. As a result, the process delivers 181K software products visible today at http://beta.explore.openaire.eu. The Figure 12 shows the number Y of representative objects that group X records: the distribution shows that the representatives merging 2-3 duplicates touch 90%, and the number of representative objects with more than 4 records drops below 1,000.

Other research products: candidate identification and matching

ORPs are deduplicated adopting the deduplication configuration of publications and datasets. Table 4 shows that GDup is fed with 7.4M ORP records (November 2019) to create 825K blocks, processes 431M matches, and generates 2.6M *equalsTo* relationships in one hour and 22 minutes. GDup identifies 1.1M duplicates, merges them in 424K representative records, to deliver a total of 6.7M records visible today at http://beta.explore.openaire.eu. Figure 13 shows the number Y of representative objects that group X records: the distribution shows that the representatives merging 2-4 duplicates touch 90%, and the number of representative objects with more than 7 records drops below 1,000.

Table 3.
Software deduplication statistics (November 2019)

Phase	Input	Execution time	Output
Candidate identification software	294K		181.2K blocks
Candidate matching software	181.2K blocks	~17′	3.3M comparisons & 402K *equalsTo* relationships

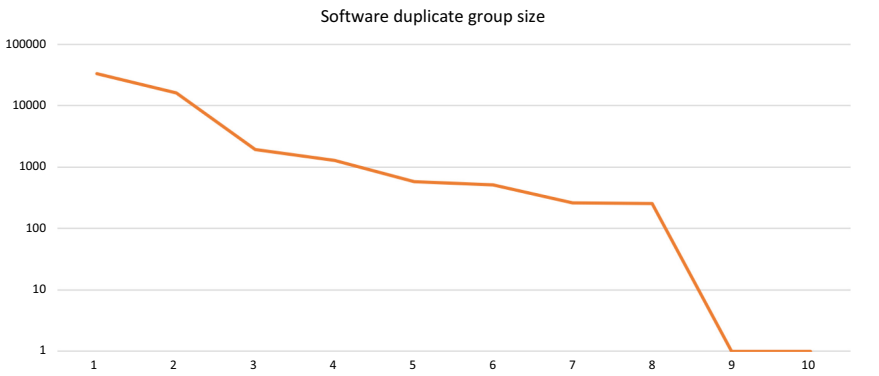

Figure 12.
Software duplicate groups: distribution of group sizes

Table 4.
Other research product deduplication statistics (November 2019)

Phase	Input	Execution time	Output
Candidate identification ORP	7.4M		825K blocks
Candidate matching ORP	825K blocks	~1*h*22′	424M comparisons & 2,6M *equalsTo* relationships

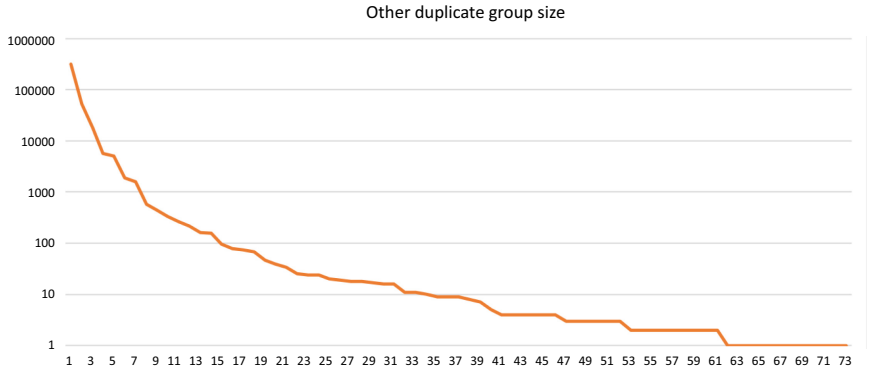

Figure 13.
Other duplicate groups: distribution of group sizes

4.3.2 Scientific products: duplicates grouping and merging. The phase of duplicate grouping and merging is performed in GDup for all scientific products, collecting as input the *equalsTo* relationships produced for publications, datasets, software and ORPs. Table 5 shows the process runs in around 5 hours and half, identifies 23.3Mi connected components/representative objects merging 62.7M duplicate records. Table 6 reports interesting numbers on the minimal, maximum, average number of equivalent records per connected components; numbers show that except from software, all entities feature connected components up to 100 records (limit set by configuration of GDup) and with an average of 2-3 records for scientific products and 3 for organizations.

4.3.3 Deduplication of Organizations. Deduplication of organizations is a non-trivial process. The metadata fields provided for organizations are mainly titles, acronyms, and country, where only the latter, when provided, obeys to a vocabulary of terms. Preconditions for diversity of records use the country field, as indeed different country values exclude the similarity of two organizations. Similarity match is left to the remaining fields, where, however, standard string similarity functions cannot be considered optimal. Organization names can take different forms in the same language (e.g. "University of Pisa" and "Pisa University") or in different languages (e.g. "Universitá di Pisa"). To this aim, GDup exploits dedicated semantic functions, where such similarities are normalized via "bags of words"

Phase	Input	Execution time	Output
Connected components	99.6M *eq.* rels	$\sim 4h21'$	23.3M connected components
Repr. object election & Rels distribution	23.3M conn. comp.	$\sim 1h15'$	23.3M repr. objects & 62.7M duplicates

Table 5.
Result deduplication graph statistics (November 2019)

Type	Min size	Max size	Mean size	Stddev
Publication	2	100	2,633	0,988
Dataset	2	100	2,193	0,664
Software	2	14	2,589	1.058
Other	2	100	2,466	1,881
Organization	2	100	3,172	3,853

Table 6.
Statistics on duplicate groups (November 2019)

identifying semantic equivalence across different languages for institution names, place names, and disciplines names. Table 7 shows that GDup is fed with 385K organization records (November 2019) to create 372K blocks, processes 15.5M matches and generates 341K *equalsTo* relationships in 23 minutes. This process identifies 250K duplicates, merges them in 56K representative records, to deliver a total of 135K records visible today at http://beta.explore.openaire.eu. Figure 14 shows the number Y of representative objects that group X records: the distribution shows that the representatives merging 2-6 duplicates touch 90%, and the number of representative objects with more than five records drops below 1,000.

5. Conclusion

The problem of deduplication applied to Big Data graphs opens actual and interesting challenges. Semi-automatic approaches, based in common general-purpose tools, are needed to reduce as much as possible the manual effort and to assist data curators in solving the domain specific disambiguation issues, repeating experiments, and sharing results with other scientists. This work describes architecture and implementation as a production-ready system of GDup, an integrated and general-purpose system for deduplication of big graphs. GDup is used in the production system of the OpenAIRE infrastructure to ensure the population of a non-ambiguous graph on top of which scientific production and Open Science statistics can be generated. Future development include (*i*) improvement of end-user interfaces, to make it a more user-friendly product, and (*ii*) the development of a crowd-sourcing toolkit allowing groups of experts (entitled by delegation) to manually add, remove, validate equality or distinction assertions so as to clean deduplication results and maintain re-usable ground truths.

Table 7.
Organization deduplication statistics (November 2019)

Phase	Input	Execution time	Output
Candidate identification organization	385K		372K blocks
Candidate matching organization	372K blocks	~9′	15.5M comparisons & 341K *equalsTo* rels.
Connected components Repr. object election & Rels distribution	341K *equalsTo* rels.	~8′	56K connected components
	56K connected components	~4′	56K repr. objects & 185K duplicates

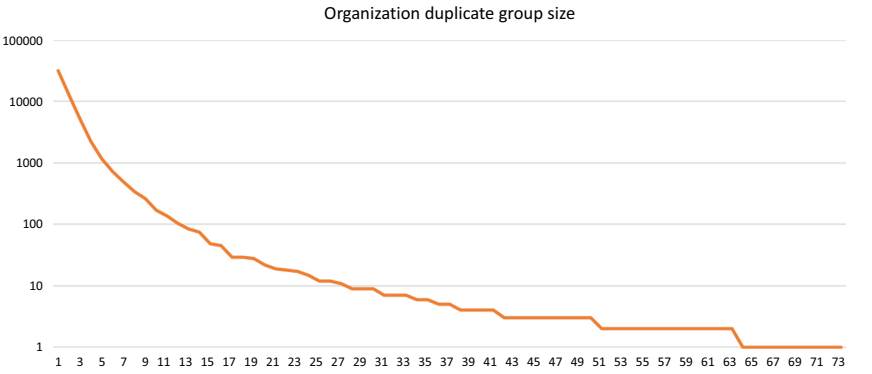

Figure 14.
Organization duplicate groups: distribution of group sizes

Notes

1. Google Scholar: http://scholar.google.com.
2. Microsoft Academic Graph https://academic.microsoft.com.
3. OpenAIRE scholarly communication graph, http://api.openaire.eu.
4. OpenAIRE infrastructure, http://www.openaire.eu.
5. OpenAIRE Research Graph, OpenAIRE Blog, https://www.openaire.eu/blogs/the-openaire-research-graph.
6. Funder Dashboard, http://provide.openaire.eu.
7. Funder Dashboard, http://monitor.openaire.eu.
8. https://tools.ietf.org/html/rfc1321.
9. *GRID database*, http://www.grid.ac.
10. Dedoop http://dbs.uni-leipzig.de/dedoop.
11. HBase - https://hbase.apache.org.
12. Apache Hadoop, http://hadoop.apache.org.
13. Apache Spark, https://spark.apache.org.
14. OpenAIRE Monitoring Dashboard for Funders, http://monitor.openaire.eu.

References

Arasu, A., Ré, C. and Suciu, D. (2017), "Large-scale deduplication with constraints using dedupalog", in *2009 IEEE 25th International Conference on Data Engineering*, pp. 952-963, doi: 10.1109/ICDE.2009.43.

Atzori, C. and Manghi, P. (2017), "gdup: a big graph entity deduplication system - Release 1", doi: 10.5281/zenodo.292980.

Atzori, C., Manghi, P. and Bardi, A. (2018), "Gdup: De-duplication of scholarly communication big graphs", in *IEEE/ACM 5th International Conference on Big Data Computing Applications and Technologies (BDCAT)*, pp. 142-151, doi: 10.1109/BDCAT.2018.00025.

Atzori, C. (2016), "gDup: an integrated and scalable graph deduplication system", PhD Thesis, University of Pisa, Ph.D. Program in Information Engineering of the Engineering Ph.D. School "Leonardo da Vinci", doi: 10.5281/zenodo.1454880.

Bhattacharya, I. and Getoor, L. (2004), "Deduplication and group detection using links", *Proceedings of the 2004 ACM SIGKDD Workshop on Link Analysis and Group Detection*.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. (2008), "Bigtable: A distributed storage system for structured data", *ACM Transactions on Computer Systems (TOCS)*, Vol. 26, p. 4.

Christen, P. (2008), "Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface", *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 1065-1068.

Cohen, W., Ravikumar, P. and Fienberg, S. (2003), "A comparison of string metrics for matching names and records", in *Kdd Workshop on Data Cleaning and Object Consolidation*, Vol. 3, pp. 73-78.

Dean, J. and Ghemawat, S. (2008), "Mapreduce: simplified data processing on large clusters", *Communications of the ACM*, Vol. 51 No. 1, pp. 107-113.

Fellegi, I.P. and Sunter, A.B. (1969), "A theory for record linkage", *Journal of the American Statistical Association*, Vol. 64 No. 328, pp. 1183-1210.

George, L. (2011), *HBase: The Definitive Guide*, O'Reilly Media, ISBN: 9781449396107.

Hartig, O. (2014), "Reconciliation of rdf* and property graphs", arXiv preprint arXiv:14093288.

Junghanns, M., Petermann, A., Gómez, K. and Rahm, E. (2015) "Gradoop: Scalable graph data management and analytics with hadoop", arXiv preprint arXiv:150600548.

Jurczyk, P., Lu, J.J., Xiong, L., Cragan, J.D. and Correa, A. (2008), "FRIL: A tool for comparative record linkage", in *AMIA annual symposium proceedings*, volume, American Medical Informatics Association, p. 440.

Kang, H., Getoor, L., Shneiderman, B., Bilgic, M. and Licamele, L. (2008), "Interactive entity resolution in relational data: A visual analytic tool and its evaluation", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 14 No. 5, pp. 999-1014.

Kolb, L. and Rahm, E. (2013), "Parallel entity resolution with Dedoop", *Datenbank-Spektrum*, Vol. 13 No. 1, pp. 23-32.

Kolb, L., Thor, A. and Rahm, E. (2010), "Parallel sorted neighborhood blocking with mapreduce", arXiv preprint arXiv:10103053.

Köpcke, H., Thor, A. and Rahm, E. (2010), "Evaluation of entity resolution approaches on real-world match problems", *Proceedings of the VLDB Endowment*, Vol. 3 Nos 1-2, pp. 484-493.

La Bruzzo, S., Manghi, P. and Mannocci, A. (2019), "Openaire's doiboost - boosting crossref for research", in Manghi, P., Candela, L. and Silvello, G. (Eds), *Digital Libraries: Supporting Open Science*, Springer International Publishing, Cham, ISBN 978-3-030-11226-4, pp. 133-143.

Lee, K.H., Lee, Y.J., Choi, H., Chung, Y.D. and Moon, B. (2012), "Parallel data processing with mapreduce: a survey", *AcM sIGMoD Record*, Vol. 40 No. 4, pp. 11-20.

Lin, J. and Schatz, M. "Design patterns for efficient graph algorithms in mapreduce", *Proceedings of the Eighth Workshop on Mining and Learning with Graphs. MLG '10*, New York, NY, USA, ACM, ISBN 978-1-4503-0214-2, pp. 78-85, doi: 10.1145/1830252.1830263.

Manghi, P., Manola, N., Horstmann, W. and Peters, D. (2010), "An infrastructure for managing ec funded research output-the openaire project", *The Grey Journal (TGJ): An International Journal on Grey Literature*, Vol. 6 No. 1, pp. 31-40.

Manghi, P., Houssos, N., Mikulicic, M. and Jörg, B. (2012), "The data model of the openaire scientific communication e-infrastructure", in *Metadata and Semantics Research*, Springer, pp. 168-180.

Manghi, P., Mikulicic, M. and Atzori, C. (2012), "De-duplication of aggregation authority files", *International Journal of Metadata, Semantics and Ontologies*, Vol. 7 No. 2, pp. 114-130.

Manghi, P., Artini, M., Atzori, C., Bardi, A., Mannocci, A., La Bruzzo, S., Candela, L., Castelli, D. and Pagano, P. (2014), "The d-net software toolkit: A framework for the realization, maintenance, and operation of aggregative infrastructures", *Program*, Vol. 48 No. 4, pp. 322-354, doi: 10.1108/PROG-08-2013-0045.

McCallum, A., Nigam, K. and Ungar, L.H. (2000), "Efficient clustering of high-dimensional data sets with application to reference matching", *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 169-178.

Robinson, I., Webber, J. and Eifrem, E. (2015), *Graph Databases: New Opportunities for Connected Data*, O'Reilly Media, ISBN: 1449356265.

Rodriguez, M.A. and Neubauer, P. (2010), "Constructions from dots and lines", *Bulletin of the American Society for Information Science and Technology*, Vol. 36 No. 6, pp. 35-41.

Saeedi, A., Peukert, E. and Rahm, E. (2018), "Using link features for entity clustering in knowledge graphs", in Gangemi, A., Navigli, R., Vidal, M.-E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A. and Alam, M. (Eds), *The Semantic Web*, Springer International Publishing, Cham, ISBN 978-3-319-93417-4, pp. 576-592.

Tomaszuk, D. and PÆ'Ã-k, K. (2018), "Reducing vertices in property graphs", *PloS One*, Vol. 13 No. 2, pp. 1-25, doi: 10.1371/journal.pone.0191917.

Wang, Q., Cui, M. and Liang, H. (2016), "Semantic-aware blocking for entity resolution", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 28 No. 1, pp. 166-180, doi: 10.1109/TKDE.2015.2468711.

Xia, F., Wang, W., Bekele, T.M. and Liu, H. (2017), "Big scholarly data: a survey", *IEEE Transactions on Big Data*, Vol. 3 No. 1, pp. 18-35, doi: 10.1109/TBDATA.2016.2641460.

Corresponding author
Paolo Manghi can be contacted at: paolo.manghi@isti.cnr.it