GDup: De-duplication of Scholarly Communication Big Graphs

Claudio Atzori, Paolo Manghi, Alessia Bardi
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" - CNR
Via Moruzzi 1, Pisa, Italy
Email: {name.surname}@isti.cnr.it

***Abstract*—Today, several online services offer functionalities to access information from big scholarly communication graphs, which interlink entities such as publications, authors, datasets, organizations, etc. Such graphs are often populated over time as aggregations of multiple sources and therefore suffer from entity duplication problems. Although deduplication of graphs is a known and actual problem, solutions tend to be dedicated and address a few of the underlying challenges. In this paper, we propose the GDup system, an integrated, scalable, general-purpose system for entity deduplication over big information graphs. GDup supports practitioners with the functionalities needed to realize a fully-fledged entity deduplication workflow over a generic input graph, inclusive of Ground Truth support, end-user feedback, and strategies for identifying and merging duplicates to obtain an output disambiguated graph. GDup is today one of the core components of the OpenAIRE infrastructure production system, monitoring Open Science trends on behalf of the European Commission.**

***Keywords*-deduplication; information graphs; big data; scholarly communication**

I. Introduction

A large number of online services offer today access to very large information graphs obtained as integration of distributed data sources. Their intent is typically that of providing integrated and enhanced access to such sources by applying a set of harmonisation and enrichment processes to the original data, e.g. harmonization of properties, identification of links between objects, named entity recognition (e.g. author identifiers). Examples in the scholarly communication domain are the Google Scholar graph[1], the Microsoft Academic graph[2], and the OpenAIRE scholarly communication graph[3], which collect from various sources (e.g. libraries, publication repositories, publishers, author directories) and build a graph whose objects are authors, organizations, publications, etc.. Due to the heterogeneity and overlap of the original sources, which naturally keep publications relative to the same authors and organizations, such graphs suffer from disruptive duplication rates and adequate countermeasures must be taken. Data curators can find today several tools supporting duplicate identification for large "flat" collections of objects. Those tools can typically be adapted to efficiently deduplicate objects of specific types in arbitrarily large collections. However, such solutions do not naturally extend to the problem of deduplication of very large graphs. In order to implement a full entity deduplication workflow for "Big Graphs" data 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.

In this paper, starting from the experiences and solutions for duplicate identification in big data collections, we address the broader and more complex problem of *entity deduplication in big information graphs*. By *graph* we denote any digital representation of a set of entity types (structured properties) and relationships between them. By *big* we mean that duplicate identification over the objects of such entity types require parallel-oriented approaches to scale up to arbitrary numbers and still perform in reasonable time. By *entity deduplication* we mean the combined process of *duplicate identification* and *graph disambiguation*. Duplicate identification has the aim of efficiently identifying pairs of equivalent objects of the same entity type; graph disambiguation has the goal of removing the duplication anomaly from the graph, while semantically preserving the topology of the graph.

To support practitioners facing the need of realizing a full entity deduplication workflow on top of a custom big graph, we have realized an integrated, scalable, general-purpose system for entity deduplication over big graphs called GDup. GDup is intended to support data curators with the out-of-the-box functionalities they require to support a fully-fledged entity deduplication workflow over an generic input graph, inclusive of "ground truth" support, end-user feedback, and strategies for identifying and merging duplicates to obtain an output disambiguated graph. As such, GDup is not about better recall/precision for given deduplication problems, but rather about provision of tools enabling data curators to concentrate on modeling and customizing their deduplication solutions without bothering about the extra conceptual and technical challenges that such task necessarily imply.

[1]Google Scholar: http://scholar.google.com
[2]Microsoft Academic Graph https://academic.microsoft.com
[3]OpenAIRE scholarly communication graph, http://api.openaire.eu

This work formally describes the underlying challenges by providing a graph type and object language, defining the functions (semantics and configuration) required to manipulate the input graph through its deduplication phases, and eventually explaining how the resulting conceptual architecture has been implemented in practice into a real production system. GDup is today in use in the production system of the OpenAIRE infrastructure.[4] The infrastructure populates a scholarly communication big graph, whose goal is to support monitoring of Open Science trends and research impact for funders, institutions, and researchers in specific disciplines. GDup is used to deduplicate publications and organizations to ensure sensible statistics are delivered.

Outline: Section II describes the OpenAIRE infrastructure use-case to draw the general requirements that a big graph deduplication system should meet and analyses the state-of-the-art to highlight the limits of current solutions. Section III formally presents the functional architecture of GDup, while Section IV describes its technical implementation and gives an example of its usage in the OpenAIRE infrastructure.

II. Deduplication of Big Information 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.

A. The OpenAIRE Information Space Graph

The OpenAIRE infrastructure is an initiative [1] 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 scholarly communication graph* [2]. Its main entities are *Research Products (datasets, publications, and software), Persons, Organizations, Funders, Funding Streams, Projects*, and *Provenance Data Sources* (from which entity information is collected):

Products are intended as the outcome of research activities and may be related to Projects, Persons or Organizations (when the link to the person is not available). OpenAIRE supports three kinds of research outcome: *Datasets*, *Software*, and *Publications*. As a result of merging equivalent objects collected from separate data sources, a Result object may have several physical manifestations, called *instances*; instances indicate URL(s) of the fulltext, access rights, and a relationship to the data source that hosts the file;

Persons are individuals that have one (or more) role(s) in the research domain, such as authors of a Result or coordinator of a Project;

Organizations include companies, research centers or institutions involved as project partners or that are responsible for operating data sources;

Funders are Organizations (e.g. European Commission, Wellcome Trust, FCT Portugal, Australian Research Council) responsible for a list of *Funding Streams* (e.g. FP7 and H2020 for the EC), which are strands of investments. Funding Streams identify the strands of funding managed by a Funder and 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 of the objects populating the OpenAIRE graph; e.g. publication/dataset/software repositories, journals, publishers. Each object is associated to the data source from which it was collected.

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[5]), organizations (Institutional Dashboard), funders (Funder Dashboard[6]), 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.

B. Object deduplication in OpenAIRE

The OpenAIRE information space graph suffers from heavy duplication phenomena. The main causes are content duplication in the harvested data sources and the low adoption by those of global persistent identifiers for the entities involved. Indeed, although scholarly communication has agreed on some best practices for PIDs (e.g. DOI for

[4]OpenAIRE infrastructure, http://www.openaire.eu

[5]Funder Dashboard, http://provide.openaire.eu

[6]Funder Dashboard, http://monitor.openaire.eu

scientific articles, ORCID identifiers for authors, ISNI for organizations, FundRef for funders) in the majority of the cases either such identifiers are not yet adopted or, when they are, are not made available through the object metadata. As a consequence, OpenAIRE aggregation services generate "stateless" unique object identifiers (i.e. IDs that can always be identically re-generated from the same object) for each publication, author, or organization extracted from the metadata. The strategy is to generate unique OpenAIRE objects when harvesting from data sources and subsequently rely in the deduplication process to merge the equal ones.

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 *publication* and *organization* are affected by both forms of duplication, which in turn lead to specific big graph deduplication challenges:

Publications Intra-data source publication duplicates are very common in aggregators (e.g. NARCIS, CORE-UK, etc.), and in some rare cases also 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 1100). 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. In general duplication rate is not high, since the same publication is expected to be deposited a few times, ideally by the authors.

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 20). 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.

From this analysis it is clear that deduplication of the OpenAIRE graph requires specific deduplication techniques for the types above. For the sake of our investigation we will focus on publications and organizations, whose features cover a wide range of challenges in this field. For publications numbers can be very high, in the order of 100Mi, 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 talking 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, both for publications and organizations, 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.

C. State of the art and motivations

The deduplication of a graph of the size and features of OpenAIRE's 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:

- *Graph orientation*: should be able to represent and manage graph-shaped information spaces, hence handle multiple entity types and relationships between them;
- *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;
- *Ground Truth*: should be able to import ground truth sets as well as generate ground truth sets from deduplication results;
- *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;
- *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 [3], [4], and it is ironically affected by forms of ambiguity as there are different common names used to refer to this research topic. Among these we can mention: 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, etc.. Many techniques developed in such research field resulted into respective deduplication tools over the years [5], [6], [7], [8]. Among existing approaches some address the problem of record linkage or entity resolution for "big" flat collections, some consider specific problems in the disambiguation of "graphs", but to our knowledge none has proposed systems for the deduplication of big graphs. Among the first category we can mention Dedoop[7] [10], PACE [6], and Dedupalog [11]. The first two tools are built on distributed column stores, respectively Hadoop MapReduce and Cassandra, and allow to efficiently process large collections to identify duplicates. The latter focuses on deduplication optmization based on preconditions. However, non of these address graphs deduplication and merging of objects. On the side of graphs, approaches have been explored but not effectively implemented, such as [13] and [14] which include semantics of graph links as input of deduplication and disambiguation. As a consequence, practitioners in the need of disambiguate big graphs end up reusing existing tools or techniques and assemble them to build custom deduplication solutions for their graphs. Such ad-hoc solutions are typically expensive to maintain and hardly reusable in different contexts.

III. GDup architecture

We have designed and implemented a system for graph deduplication called GDup [12] 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 1, whose main functional areas support an end-to-end workflow enabling data curators at:

1) Importing their graph in the system;
2) Configuring for each entity type the relative duplicate identification "configurations";
3) Managing Ground Truth generation and injection;
4) Configuring graph disambiguation strategies;
5) Supporting data curators at manually fixing the results of deduplication;

[7]Dedoop http://dbs.uni-leipzig.de/dedoop

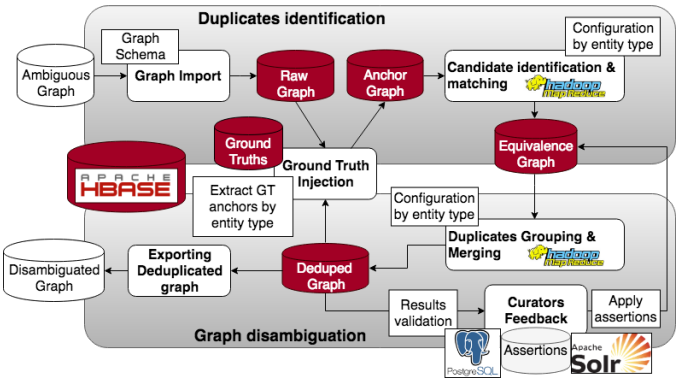

Figure 1. GDup: architecture of de-duplication workflow (and related technologies in current OpenAIRE implementation)

6) Exporting a disambiguated graph, i.e. devoid of duplicate nodes.

In the following we will first describe the type and object language of GDup based on the Property Graph Model [15], then formally introduce the individual areas of the architecture and the relative functionalities.

A. GDup graph model

The workflow depicted in Figure 1 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 (i) 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 (ii) 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) [16] 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* [17] where:

- *Vertex*: an object that has a unique identifier, a label that denotes its type (i.e. properties) and potentially incoming and outgoing edges;
- *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, i.e. head vertex and tail vertex;
- *Properties*: are a set of key/value pairs associated to a vertex;

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

In order to impose an expected structure to vertexes and edges, we defined the *Structured Property Graph Model* (SPGM) as an extension of EPGM which includes the notion of *graph schema*. A graph schema associates a vertex type V_T as defined in EPGM to a given set of properties and to a set of expected edge types:

$$V_T =< [l_1, \ldots, l_n], \{\leftrightarrow V_{T_1}, \ldots \leftrightarrow V_{T_m}\} >$$

Accordingly, an SPGM graph is an EPGM graph whose nodes conform, i.e. respect the structure, of a given graph schema.

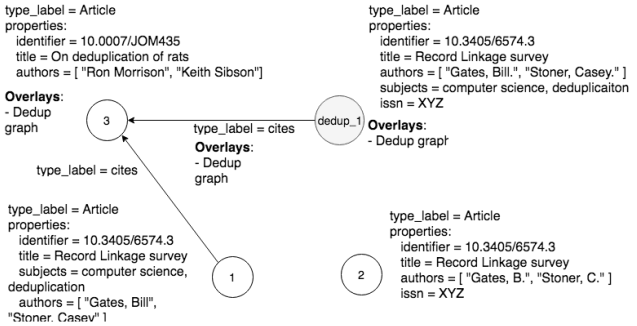

Figure 2. Structured Property Graph: from raw graph to deduplicated graph

Figure 2 illustrates an example of an SPGM graph instance with $\mathcal{G}_R$ and $\mathcal{G}_D$ overlays, which includes objects of type Article:

$$Article = \quad < [identifier, title, authors, date, subjects], \{\leftrightarrow Authors\} >$$

The graph shows how objects 1 and 2 have been detected as equal (they have the same identifier) and merged into one representative object $dedup\_1$. The overlay $dedupGraph$ identifies the nodes in the final deduplicated graph. In the following, by "adding the object/relationship to a graph" we imply the object/relationship is decorated with the overlay of the target graph.

B. Deduplication Workflow Overview

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:

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 [15], [18]).

2) Candidate identification and matching: This phase operates over an *anchor graph* $\mathcal{G}_A$, obtained by "injecting" ground truths for each type V_T to the raw graph $\mathcal{G}_R$, 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 fire this phase by selecting, 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* [9], [19]) 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 for all objects $o, o' \in V_T$ analyses their properties and sends them to the same block B_k (namely when $h(o) = h(o') = k$) maximizing the chances that o_1 and o_2 are equivalent; h should be tailored to the semantic features of V_T, in order to minimize false positives, i.e. objects that are not equivalent and associated to the same block B_k, and false negatives, i.e. 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.

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* and the relative *block sliding window*.

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 [20] 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.

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 [21]. 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.

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: (i) the relationship $equalTo$ between the two is added to the graph and attached to the overlay $equivalenceGraph$, and (ii) both objects are attached to the overlay $equivalenceGraph$. 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 the relative section 5) such assertions refine the de-duplicaiton process by adding further relationships or removing relationships from the equivalence graph $\mathcal{G}_E$ as indicated by expert users.

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 $equivalenceGraph$. 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 $equalTo$. For example, if $< o_1, equalTo, o_2 >$ and $< o_2, equalTo, o_3 >$ we can conlude that $< o_1, equalTo, o_3 >$. If no other object in $\mathcal{G}_E$ is reached via a relationship $equalTo$ 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 [22].

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 (i) generating a stateless identifier for the representative object (the same identifier is produced from the same group of duplicates), and (ii) merging strategy of the duplicate objects' properties. To this aim, GDup first elects a *pivot* 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 the merged objects; $enrich$, the pivot object is enriched with all values found in the merged objects (e.g. subject properties).

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*, i.e. all relationships with objects of the type listed are to be redistributed to the representative object, and *ByPivot*, i.e. 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. Figure 2 shows one representative object $dedup\_1$ obtained from merging the nodes 1 and 2. $dedup\_1$ inherits all properties of the two objects and its relationships. 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:

- Adding the new representative objects obtained from each group of duplicates;
- Adding the relationships created from and to the representative object as specified in the configuration;
- Adding all objects in $\mathcal{G}_R$ that are not in $\mathcal{G}_E$: objects merged into a representative object should not be visible;
- 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.

4) Ground Truth injection: A ground truth is a graph gt_{V_T} representing a set of authority validated assertions about groups of duplicate objects in the input raw graph $\mathcal{G}_R$. Each assertion claims that a given set of objects in $\mathcal{G}_R$ is equivalent and provides the *anchor object* (i.e. a representative object that persists due to its high level of trust) for them, with an identifier, properties and links to other objects. Assertions are trees rooted in anchor objects and whose leaves are the merged objects, reached by relationships *mergedBy*, whose identifiers are those of the objects in $\mathcal{G}_R$. The introduction of a Ground Truth approach changes the perspective of the deduplication process, since part of the deduplication can be pre-processed by replacing sets of merged objects with the relative representative object in the input graph $\mathcal{G}_R$.

In GDup a ground truth gt_{V_T} is typically the subset of the equivalence graph $\mathcal{G}_E$ for a given type V_T under a specific deduplication configuration. 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 harder levels of deduplication configuration. This preprocessing phase is also useful for two other reasons: (i) 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 (ii) resolving duplicates based on a Ground Truth knowledge, typically validated by experts, rather than leaving it solely to a mechanical process.

More specifically, for each ground truth gt_{V_T} selected by the user, the normalized anchor graph $\mathcal{G}_A$ is obtained from the raw graph $\mathcal{G}_R$ by:

- Adding to all objects and relationships in $\mathcal{G}_R$ to $\mathcal{G}_A$;
- Adding all anchor objects in gt_{V_T} to $\mathcal{G}_A$;
- Adding all relationships incoming and outgoing the anchor objects to $\mathcal{G}_A$;
- 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.

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:

- *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. When such assertions are applied after the candidate mathcing phase, the side effect is to add a relationship $< o, equalTo, o' >$ for each pair of objects $o, o' \in \{o_1, \ldots o_n\}$ to the equivalence graph $\mathcal{G}_E$.
- *Diversity assertion:* a diversity assertion states that two objects o and o' are distinct. If either or both of the objects are representative objects, the claim naturally extends to all objects in $\mathcal{G}_R$ merged by the representative ones. When such assertions are applied after the candidate matching phase, the side effect is to remove relationship $< o, equalTo, o' >$ from the equivalence graph $\mathcal{G}_E$ (if they exist) for each pair of objects o, o' that may be produced by the assertion.

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 .

IV. GDup implementation

GDup implements the functionalities described in the previous section by assembling known Open Source technologies (GDup Release 1.0 [23]), as shown in Figure 2. GDup is today used in the production system of OpenAIRE to deduplicate publications and organizations 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.

A. 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 [24]. Since the beginning of the project, back in 2010, the OpenAIRE infrastructure based its solution and services on HBase technology[8] [25]. HBase is the open source version of BigTable [26], 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 is based on the Hadoop framework[9], enabling large scale distributed data processing and analytics based on Map Reduce programming model [27], [28].

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 GraphX [10] 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: (i) the identifier of the entity, (ii) the *type* of entity, (iii) the body of the entity (its properties and values), and (iv) 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 $equalTo$ between the rows. With respect to the deduplicated graph in Figure 2, the objects $dedup\_1$, 3, and 2 would be represented as depicted in Figure 3. 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.

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

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

	Type	Body	Rel:isCited:2	Rel:isCited:dedup_1
3	Article	identifier = 10.0007/JOM435 title = On deduplication of rats authors = ["Ron Morrison", ...	deleted	deleted

Figure 3. Graph object representation in HBase

B. Deduplication workflow: implementation

As suggested in Figure 2 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.

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

2) Duplicate grouping and merging:

[8]HBase - https://hbase.apache.org

[9]Apache Hadoop, http://hadoop.apache.org

[10]Apache Spark GraphX, https://spark.apache.org/graphx

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 [31]). 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) Data Curators Feedback: The deduplicated graph is indexed in a Solr (4.9.0) insallation 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.

C. GDup results 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, 480GB of RAM, and 21TB of allocated HDFS disk space – the cluster is used also for heavy full-text mining tasks. In this context GDup is used to perform deduplication of publication and organization entities in the OpenAIRE scholarly communication graph.

Table I summarizes the results over the 30 million publications aggregated by the OpenAIRE production system from around 1000+ data sources. Candidate identification generates blocks grouping publications by DOI and by hash keys produced from a normalized form of the publication's title. Such keys are generated from the suffix and the prefix of adjacent words and on the concatenation of ngrams produced from adjacent words. From the 30 million publications aggregated in OpenAIRE (June 2018) this configuration produces 8Mi blocks, 5Mi of which contain more than 1 object and will be therefore further processed in subsequent candidate matching phase. Candidate matching sorts publications with the title normalized by removing punctuation, stopwords and numbers; it uses a sliding window of size 100 and a max number of elements of 2000. The matching function takes into account the DOI for equality preconditions, the diversity of numbers in the title for diversity. Eventually, it applies a similarity function, with a threshold of 0.98, matching the title of the articles, number of authors, and an approximate author names match. Dates are not regarded as their meaning is generally ambiguous. GDup identifies 10.5M duplicates, merges them in 4.2M representative records, to deliver a total of 24Mi records visible today at http://www.openaire.eu. Interestingly, the 90% of representatives merges 2 and 3 duplicates.

Phase	**Execution time**	**Output**
Candidate identification	$\sim 1h25'$	8M blocks
Candidate matching	$\sim 3.5h$	5B comparisons & 10M *equalsTo* relationships
Connected components	$\sim 1h20'$	4.2M connected components
Repr. object election & Rels distribution	$\sim 1h45'$	4.2M repr. objects & 10.5M duplicates

Table I
PUBLICATIONS DEDUPLICATION STATISTICS (JUN 2018)

V. CONCLUSION

Among data challenges also entity deduplication is living a period of Big Data renaissance. Extending this trend to large graphs of interlinked objects opens further interesting and actual questions. This paper has presented GDup, which offers a production ready system an integrated and general-purpose system for deduplication of big graphs. GDup is today used in the production system of the OpenAIRE infrastructure. Still, its completion is ongoing to make it a fully user-friendly product, i.e. completion of data curators GUI, and to address further functional scenarios, e.g. deduplication by crowd-sourcing by delegating to a set of experts the addition of assertions to clean deduplication results and build ground truth.

ACKNOWLEDGMENT

This research was co-funded by the EC OpenAIRE2020 project (grant 643410, call H2020-EINFRA-2014-1) and EC OpenAIRE-Advance project (grant 777541, call H2020-EINFRA-2017-1).

REFERENCES

[1] P. Manghi, N. Manola, W. Horstmann, and D. Peters, "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, 2010.

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

[3] I. P. Fellegi and A. B. Sunter, "A theory for record linkage", *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969.

[4] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems", *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 484–493, 2010.

[5] P. Jurczyk, J. J. Lu, L. Xiong, J. D. Cragan, and A. Correa, "FRIL: A tool for comparative record linkage", in *AMIA annual symposium proceedings*, p. 440, 2018.

[6] P. Manghi, M. Mikulicic, and C. Atzori, "De-duplication of aggregation authority files", *International Journal of Metadata, Semantics and Ontologies*, vol. 7, no. 2, pp. 114–130, 2012.

[7] P. 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*. ACM, 2008, pp. 1065–1068.

[8] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele, "Interactive entity resolution in relational data: A visual analytic tool and its evaluation", *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 5, pp. 999–1014, 2008.

[9] L. Kolb, A. Thor, and E. Rahm, "Parallel sorted neighborhood blocking with mapreduce", *arXiv arXiv:1010.3053*, 2010.

[10] L. Kolb and E. Rahm, "Parallel entity resolution with Dedoop", *Datenbank-Spektrum*, vol. 13, no. 1, pp. 23–32, 2013.

[11] A. Arasu, C. R, and D. Suciu, "Large-scale deduplication with constraints using dedupalog", in *2009 IEEE 25th International Conference on Data Engineering*, March 2009, pp. 952–963.

[12] C. Atzori, "gDup: an integrated and scalable graph deduplication system", Ph.D. dissertation, University of Pisa, 2016.

[13] I. Bhattacharya and L. Getoor, "Deduplication and group detection using links", *in Proc. of the 2004 ACM SIGKDD Workshop on Link Analysis and Group Detection*, 2004.

[14] A. Saeedi, E. Peukert, and E. Rahm, "Using link features for entity clustering in knowledge graphs", in *The Semantic Web*, A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, Eds. Cham: Springer International Publishing, 2018, pp. 576–592.

[15] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines", *Bulletin of the American Society for Information Science and Technology*, vol. 36, no. 6, pp. 35–41, 2010.

[16] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases: New Opportunities for Connected Data*. "O'Reilly Media, Inc.", 2015.

[17] M. Junghanns, A. Petermann, K. Gómez, and E. Rahm, "Gradoop: Scalable graph data management and analytics with hadoop", *arXiv arXiv:1506.00548*, 2015.

[18] O. Hartig, "Reconciliation of rdf* and property graphs", *arXiv arXiv:1409.3288*, 2014.

[19] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching", in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 169–178.

[20] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records", in *Kdd workshop on data cleaning and object consolidation*, vol. 3, 2003, pp. 73–78.

[21] Q. Wang, M. Cui, and H. Liang, "Semantic-aware blocking for entity resolution", *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 166–180, Jan 2016.

[22] D. Tomaszuk and K. Pk, "Reducing vertices in property graphs", *PLOS ONE*, vol. 13, no. 2, pp. 1–25, 02 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0191917

[23] C. Atzori and P. Manghi, "gdup: a big graph entity deduplication system - Release 1.0", Feb. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.292980

[24] M. A. Rodriguez and P. Neubauer, "The graph traversal pattern", *arXiv preprint arXiv:1004.1001*, 2010.

[25] L. George, *HBase: the definitive guide*. "O'Reilly Media, Inc.", 2011.

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

[27] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey", *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[29] J. Lin and M. Schatz, "Design patterns for efficient graph algorithms in mapreduce", in *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, ser. MLG '10. New York, NY, USA: ACM, 2010, pp. 78–85. [Online]. Available: http://doi.acm.org/10.1145/1830252.1830263

[30] T. Jungblut, "Graph exploration with apache hadoop and mapreduce" [Blog]

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