UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Ph.D. Thesis

GDup : an Integrated, Scalable Big Graph Deduplication System

Autore:

Claudio Atzori

Relatori:

Prof. Cinzia Bernardeschi

Dr. Paolo Manghi

Anno 2016
SSD ING-INF/05

Sommario

Partendo dalle esperienze e soluzioni esistenti nell'ambito della identificazione di oggetti digitali duplicati in collezioni "Big Data", questa tesi affronta il piu ampio e complesso problema della *deduplicazione di entità in grafi di grandi dimensioni*. Per "grafi" si intende ogni rappresentazione di un modello Entità-Relazione, definito da Tipi di Entità (e relative proprietà strutturate) e da relazioni tra esse. Per "grandi dimensioni" si intende la limitazione delle soluzioni e tecnologie tradizionali a supportare il processo di identificazione dei duplicati in collezioni di notevole dimensione (decine di milioni ed oltre). Per "deduplicazione di entità" intendiamo la combinazione dei processi di *identificazione dei duplicati* e di *disambiguazione del grafo*. Il primo ha l'obiettivo di identificare in modo efficiente (ed efficace) coppie di oggetti dello stesso tipo, mentre il secondo mira ad eliminare dal grafo la duplicazione di informazione.

Ad oggi svariate applicazioni popolano e manutengono grafi di grandi dimensioni, tra questi vi sono: (i) collezioni sulle quali non vi è alcun controllo di duplicazione, (ii) aggregazioni di diverse collezioni dati che necessitano di deduplicazione continua o estemporanea. Esempi spaziano dalla duplicazione di record anagrafici, alla duplicazione degli autori in aggregazioni di collezioni bibliografiche (es. Google Scholar, Thomson Reuters, OpenAIRE), alla deduplicazione di cataloghi provenienti da molteplici esercizi commerciali, alla deduplicazione dei risultati di integrazione tra diverse collezioni "Linked Open Data", a qualsiasi sottoinsieme del Web, etc...

Ad oggi gli utenti responsabili per la cura dei dati ("data curator") hanno a disposizione una pletora di strumenti a supporto del processo di identificazione dei duplicati, che possono adottare per processare collezioni contenti oggetti del medesimo tipo. Tuttavia, l'estensione di tali strumenti agli scenari posti dall'avvento di "Big Data" è limitata, cosi come il supporto alla disambiguazione di grafi. Per implementare un workflow completo per la disambiguazione di grafi di grandi dimensioni, gli utenti curatori finiscono per realizzare sistemi di sistemi, costruiti su misura per lo specifico modello dei dati a grafo, spesso vincolato ad una specifica rappresentazione fisica del grafo stesso (sistema di basi di dati per grafi), dispendiosi in termini di progettazione, sviluppo, manutenibilità, ed in generale poco adatti ad essere riutilizzati da altri professionisti che necessitano di

risolvere problematiche analoghe in diversi domini applicativi.

Il primo contributo di questa tesi è costituito da una architettura di riferimento per i "Big Graph Entity Deduplication System" (BGEDS), ossia un sistema integrato, scalabile, generico per la deduplicazione di entità in grafi di grandi dimensioni. Tali sistemi mirano a supportare i data curators con funzionalità per la realizzazione di tutte le fasi previste dal workflow di identificazione dei duplicati e disambiguazione del grafo. L'architettura definisce formalmente il problema, fornendo un linguaggio dei tipi per i grafi e per gli oggetti del grafo, fornisce la specifica per le fasi di deduplicazione delle entità, e di come tali fasi manipolano il grafo iniziale fino alla sua versione disambiguata. Inoltre si specifica il livello di configurazione e di personalizzazione che i data curator possono utilizzare affidandosi a tale sistema.

Il secondo contributo di questa tesi è GDup, una implementazione di BGEDS la cui istanza è oggi utilizzata in regime di produzione nel contesto della infrastruttura tecnologica OpenAIRE, riferimento europeo per l'Open Science e l'Open Access. GDup puo' essere utilizzato per operare su grafi di grandi dimensioni rappresentati utilizzando standard come RFD o JSON-LD, e conformi a qualsiasi schema per grafi. Il sistema supporta fasi altamente configurabili di identificazione dei duplicati e di disambiguazione del grafo, consentendo ai data curator di adattare le funzioni di confronto alle caratteristiche degli oggetti nel grafo, nonchè di definire le strategie di fusione degli oggetti duplicati. GDup inoltre consente di gestire in modo semi automatico un Ground-Truth, ossia un insieme di asserzioni di equivalenza tra gli oggetti del grafo, le quali possono essere utilizzate per pre-processare le entità di un certo tipo, al fine di ridurre i tempi di computazione. Il sistema è concepito per poter essere esteso con ulteriori, e possibilmente nuove, metodologie proprie della deduplicazione (ad esempio funzioni di clustering e di similarità), e per supportare scalabilità orizzontale nel processamento di grafi di grandi dimensioni sfruttando il framework Hadoop Map Reduce, ed HBase.

Abstract

In this thesis we start from the experiences and solutions for duplicate identification in Big Data collections and address the broader and more complex problem of *Entity Deduplication over Big Graphs*. By “Graph” we mean any digital representation of an Entity Relationship model, hence entity types (structured properties) and relationships between them. By “Big” we mean that duplicate identification over the objects of such entity types cannot be handled with traditional back-ends and solutions, e.g. ranging from tens of millions of objects to any higher number. 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 for the same entity type, while graph disambiguation has the goal of removing the duplication anomaly from the graph. A large number of Big Graphs are today being maintained, e.g. collections populated over time with no duplicate controls, aggregations of multiple collections, which need continuous or extemporaneous entity deduplication cleaning. Examples are person deduplication in census records, deduplication of authors on library bibliographical collections (e.g. Google Scholar graph, Thomson Reuters citation graph, OpenAIRE graph), deduplication of catalogues from multiple stores, deduplication of Linked Open Data clouds resulting from integration of multiple clouds, any subset of the Web, etc.. As things stand today, data curators can find a plethora of tools supporting duplicate identification for Big collections of objects, which they can adopt to efficiently process the objects of individual entity type collections. However, the extension of such tools to the Big Data scenario is absent, as well as the support for graph disambiguation. 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 (i.e. graph storage), expensive in terms of design, development, and maintenance, and in general not reusable by other practitioners with similar problems in different domains.

This first contribution of this thesis is a reference architecture for *Big Graph Entity Deduplication Systems (BGEDSs)*, which are integrated, scalable, general-purpose systems for entity deduplication over Big Graphs. BGEDSs are intended to support data curators

with the out-of-the-box functionalities they need to implement all phases of duplicates identification and graph disambiguation. The architecture formally defines the challenge, by providing graph type language and graph object language, defining the specifics of the entity deduplication phases, and explaining how such phases manipulate the initial graph to eventually return the final disambiguated graph. Most importantly, it defines the level of configuration, i.e. customization, that data curators should be able to exploit when relying on BGEDSs to implement entity deduplication.

The second contribution of this thesis is GDup, an implementation of a BGEDS whose instantiation is today used in the real production environment of the OpenAIRE infrastructure, the European e-infrastructure for Open Science and Access. GDup can be used to operate over Big Graphs represented using standards such as RDF-graphs or JSON-LD graphs and conforming to any graph schema. The system supports highly configurable duplicate identification and graph disambiguation settings, allowing data curators to tailor object matching functions by entity type properties and define the strategy of duplicate objects merging that will disambiguate the graph. GDup also provides functionalities to semi-automatically manage a Ground Truth, i.e. a set of trustworthy assertions of equality between objects, that can be used to pre-process objects of the same entity type and reduce computation time. The system is conceived to be extensible with other, possibly new methods in the deduplication domain (e.g. clustering functions, similarity functions) and supports scalability and performance over Big Graphs by exploiting an HBase - Hadoop MapReduce stack.

Dedicated to Elena
$= \Lambda.\Lambda =$

Contents

List of Figures

List of Tables

Listings

List of Algorithms

1

Introduction

1.1 Motivation

In the last decade research in computer science has grown increasing interest in “Big Data”. Big Data is a term used to describe all kinds of data management problems that pose challenges in terms of the three dimensions of *Volume*, *Variety* and *Velocity* (3Vs). Volume refers to the amount of data, which today typically ranges in the order of terabytes and exabytes. Variety refers to the number of types of data characterizing the problem to be addressed, which may range from structured or semi-structured databases, to multimedia, videos, social data, mobile or unstructured. Velocity refers to the speed of data generation and processing, which range from batch, to periodic, near-real time and real time. To get a sense, every hour of social media and email activities on the Internet yields 72 hours of video uploaded to Youtube, 4 million search inquiries on Google, 200 millions sent emails, 2,5 millions shares on Facebook, and 300,000 tweets on Twitter.[1]

The 3Vs are relative concepts, in the sense that their measure and thresholds are constantly pushed beyond extant limits by technological advances and decrease on the cost of hardware. In the recent years Big Data caused a serious groundshift, dividing the database community into a debate between the usage of traditional database technologies, such as Relational Database Management Systems, and more unstructured and parallel solutions introduced by the so-called NoSQL movement. This new paradigm promotes database systems that sacrifice the relational model with highly optimized systems in favour of unstructured models leveraging horizontal scalability and parallel processing. Aided by the decreased cost of commodity hardware and data storage systems, today shifted to cloud solutions (infrastructures as a service), such novel solutions have become increasingly adopted by data practitioners of all sorts in industry and research. As a consequence, research in computer science is today living a period of renaissance. Known and solved problems are acquiring a new and appealing flavour of scalability and

[1] http://www.visualcapitalist.com/order-from-chaos-how-big-data-will-change-the-world (last read on 5th of January, 2016)

performance, while known efficiency and performance open challenges can today find alternative solutions under Big Data paradigms.

Among these renovated challenges, the one of interest and touched by this thesis is the problem of *duplicate identification*, intended as the challenge of identifying pairs of duplicates. Extensive scientific literature has been written on this matter and under different names and variations of the problem. When the goal is that of identifying similar/equivalent objects across independent collections, the problem is referred to as *Record Linkage* (RL) [66, 23] or *Entity Resolution* (ER) [89, 10]. Examples of ER scenarios are linking of census records, linking of publications and citations (Thomson Reuters citation index [37]), identifying duplicate customers in enterprise databases, comparison shopping, etc.. When objects belong to the same collection, the problem is known as *Entity Deduplication* (ED) as the goal is about disambiguating the collection by identifying duplicate objects. In general, the outcome of duplicate identification is a list of pairs of equivalent objects, obtained by matching all possible pairs of objects with a similarity function. Theoretically this operation requires pairwise object comparisons, hence a Cartesian product with quadratic complexity $O(n^2)$; therefore, even in cases where the number of objects is not considerable, the complexity can reach intractable dimensions. To overcome this issue, solutions typically consists of two phases, *candidate identification* and *candidate matching*. Candidate identification aims at tackling efficiency issues by implementing heuristics (i.e. blocking, nearest neighborhood [64]) capable of skimming out pairs of objects that are unlikely identical. Candidate matching is the phase where every pair of objects identified in the candidate identification phase are matched to return a measure of similarity in between 0 and 1 [17]. Pairwise similarity is expressed by distance functions in charge of replacing human judgement, often based on (combinations of) string matching distance functions. Before Big Data technologies came into place, known duplicate identification tools were typically based on RDBMS and would take from 3-4 hours (FRIL [40]) to 19-20 hours (LinkageWiz[2]in order to identify duplicates in a collection of "only" 10 million objects [61]. Big Data scenarios, which may count 10s of Millions of objects, are computationally intractable for "old school" solutions. In this context, new advanced solutions have been proposed, which revise existing techniques in order to fit scalable back-ends and parallel techniques, such as Hadoop MapReduce, HDFS, MongoDB, etc. Examples are Dedoop [47], a tool for MapReduce-based entity resolution for large datasets built on top of Apache Hadoop, PACE [61], an authority control tool conceived to maintain "aggregation authority files" built on top of a Cassandra storage system [53].

In this thesis we start from the experiences and solutions for duplicate identification in Big Data collections and address the broader and more complex problem of *Entity Deduplication over Big Graphs*. By "Graph" we mean any digital representation of an Entity Relationship model, hence entity types (structured properties) and relationships between them. By "Big" we mean that duplicate identification over the objects of such entity types cannot be handled with traditional back-ends and solutions, e.g. ranging from tens of millions of objects to any higher number. By "entity deduplication" we mean the combined

[2] http://www.linkagewiz.net

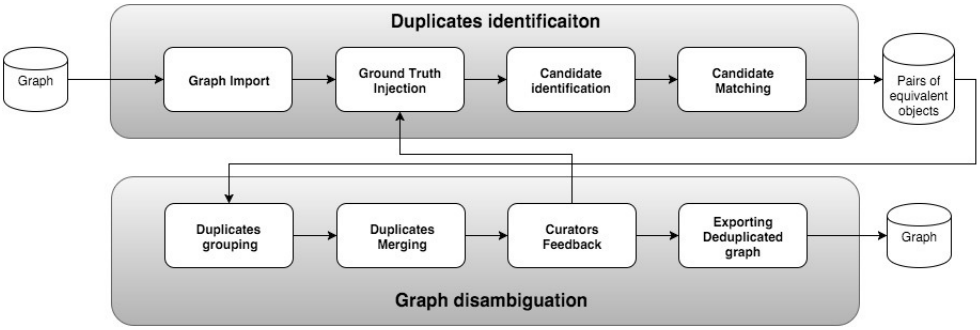


Fig. 1.1: Big Graph deduplication workflow

process of duplicate identification, as described above, and *graph disambiguation*. Duplicate identification can be applied in principle to all collections of objects conforming to one of the entity types in the graph. Disambiguation follows duplicate identification and is the action of adjusting a collection of objects by removing the duplication anomaly. This second phase, although strictly interwoven with duplicate identification, is generally not addressed as a general-purpose mechanism by deduplication tools but rather left as an issue to be solved by data curators in their own information systems, where the original collection is stored, taking advantage of the identified pairs of duplicates. Graph disambiguation is not trivial for "flat" collections and in generally very tedious over graphs. As depicted in Figure 1.1, it consists of two phases, *duplicates grouping* and *duplicates merging*, and may include support for *data curators feedback* and *Ground Truth injection* feeding back the phase of duplicate identification. Duplicates grouping applies to each entity type collection over which duplicate identification was run. It is the process of identifying groups of equivalent objects starting from the set of pairs of equivalent objects, which correspond to the problem of deriving maximum cliques in a graph by transitive closure [65]. Cliques in a graph are sets of elements where each pair of elements is connected, and listing all the maximum cliques is an NP-complete problem [42], whose tractability in Big Graphs scenarios becomes a serious challenge. Duplicates merging is the process of effectively cleaning up an entity type collection by removing duplicates and managing the consequences of such removal. Removing duplicates typically consist in selecting or creating a *representative object* for each group of equivalent objects. Consistency strategies are applied to either select one of the objects of the group or instead create a new object out of the objects of the group (e.g. combining their properties). Moreover, since we are disambiguating in a graph, an important constraint to consider are the relationships occurring between the objects, which must be propagated to the representative objects, again according to given consistency strategies. An important outcome of duplicates merging is the ability to construct and curate a *Ground Truth* (GT), intended as a trusted outcome of deduplication. A GT incarnates a map between "raw" objects and "representative objects" that can be used to pre-process the collection at hand in order to disambiguate it before

a new deduplication round, this time limited to the new objects in the collection, is performed. Adopting GT approaches is not an obvious choice, as it depends on the trade-off between GT maintenance and time-to-process. The challenge is regarded as a problem of preparation of the collection to be deduplicated and deduplication tools do not provide general-purpose support for it. Finally, an important element of graph disambiguation is the ability of supporting *data curators* with the possibility to manage a set of manual assertions ("A is duplicate of B", "A is not a duplicate of B") in order to correct the inevitable errors and exceptions introduced by automated deduplication. Assertions are used by the system as feedbacks to apply ex-post refinement of the deduplication identification process and are again not trivial and not supported as general-purpose mechanisms by existing tools.

A large number of Big Graphs are today being maintained, e.g. collections populated over time with no duplicate controls, aggregations of multiple collections, which need continuous or extemporaneous entity deduplication cleaning. Examples are person deduplication in census records, deduplication of authors on library bibliographical collections (e.g. Google Scholar graph, Thomson Reuters citation graph, OpenAIRE graph), deduplication of catalogues from multiple stores, deduplication of Linked Open Data clouds resulting from integration of multiple clouds, any subset of the Web, etc.. As things stand today, data curators can find a plethora of tools supporting duplicate identification for Big collections of objects, which they can adopt to efficiently process the objects of the entity type collections in the graph. However, in order to implement the entity deduplication processes described above, data curators end-up realizing patchwork systems, tailored to their graph data model, often bound to their physical representation of the graph (i.e. graph storage), expensive in terms of design, development, and maintenance, and in general not reusable by other practitioners with similar problems in different domains.

1.2 Research contributions

The first contribution of this thesis is a reference architecture for *Big Graph Entity Deduplication Systems (BGEDSs)*, which are integrated, scalable, general-purpose systems for entity deduplication over Big Graphs. BGEDSs are intended to support data curators with the end-user functionalities they need to realize all workflow phases of duplicates identification and graph disambiguation depicted in Figure 1.1. The second contribution is GDup, an implementation of a BGEDS whose instantiation is today used in the real production environment of the OpenAIRE infrastructure [3]. Specifically, GDup supports *data curators* with the following functionalities: (i) import of Big Graphs using standards such as RDF-graphs or JSON-LD graphs[4] (ii) configurable duplicate identification settings at the level of entity types, (iii) support for efficient and scalable duplicates grouping, (iv) configurable duplicate merging strategy for representative object election and redistribution of relationships (v) Ground Truth management at the level of entity types, and (vi)

[3] OpenAIRE, `http://www.openaire.eu`

[4] JSON for Linking Data, `http://json-ld.org`

data curator feedbacks support. The system is conceived to be extensible with other methods in the deduplication domain (e.g. clustering functions, similarity functions) and supports scalability and performance over Big Graphs by exploiting an HBase - Hadoop - MapReduce stack.

1.3 Thesis outline

The remainder of this thesis is organised as follows. In Chapter 2 we present a survey of the existing deduplication tools to compare their entity deduplication functionality equipment with the one provided by GDup and therefore motivate its realization. In Chapter 3 we introduce a real-case scenario, which will serve to describe the criticalities of the entity deduplication and identify in more detail the requirements of an architecture for BGEDSs. The real-case is that of the OpenAIRE infrastructure [58], whose production services are populating a Big Graph of around 30 million objects describing publications, authors, projects, organizations, and other related entities by aggregating information from thousands of potentially overlapping data sources on the Web. In Chapter 4, thanks to the analysis of the OpenAIRE use case, we formalize the core requirements of BGEDSs, to be used in Chapter 5 to define a functional reference architecture for BGEDSs. In Chapter 6 we present an implementation of the BGEDS reference architecture: GDup. It is based on Apache HBase and Hadoop MapReduce paradigm, today instantiated to operate as part of the production environment of the OpenAIRE infrastructure. In Chapter 7 we elaborate on the specific usage of GDup in OpenAIRE and report on the results of its operation after three years since its launch. Finally, in Chapter 8 we draw our final remarks and and discuss the future work.

2

Related work

The literature about deduplication is ironically affected by duplication. There are different common names used to refer to this research topic. Among these we can mention: Record Linkage (RL), Entity Resolution (ER), duplicate detection, coreference 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 were implemented in various deduplication tools over the years. In this Chapter we introduce existing tools for deduplication of object collections (found available on the web, free commercial trials or open source tools), highlighting their ability to address the requirements of a BGEDS (BGEDS), as illustrated in figure 2.1.

- *Graph oriented approach*: graph shaped information spaces should be 1st level citizens of modern deduplication tools. They should support structured records, allow to merge them as well as support strategies to guarantee the consistency of the existing relationships between records;
- *Experiment driven*: users must be supported by tools that enable the possibility to experiment new configurations without affecting trusted results;
- *General purposeness*: tooling should be customizable and configurable in order to allow reuse in different application contexts;
- *Ground Truth*: typically ground truth-based techniques are not integrated into the deduplication system, but rather applied as a pre-processing phase;
- *Scalability*: deduplication is a challenging task per se, especially in terms of computational cost. In order to process large graphs the architecture of deduplication systems should allow to scale out by adding new resources;
- *Data curators feedback*: No machinery will ever replace human ability to judge whether two objects of the same entity are indeed duplicates. To this end deduplication systems must allow domain experts to evaluate the results and provide feedback to the system;

The following sections present existing open source and/or commercial tools and the functionalities they offer, classifying the solutions based on the fact they address or not

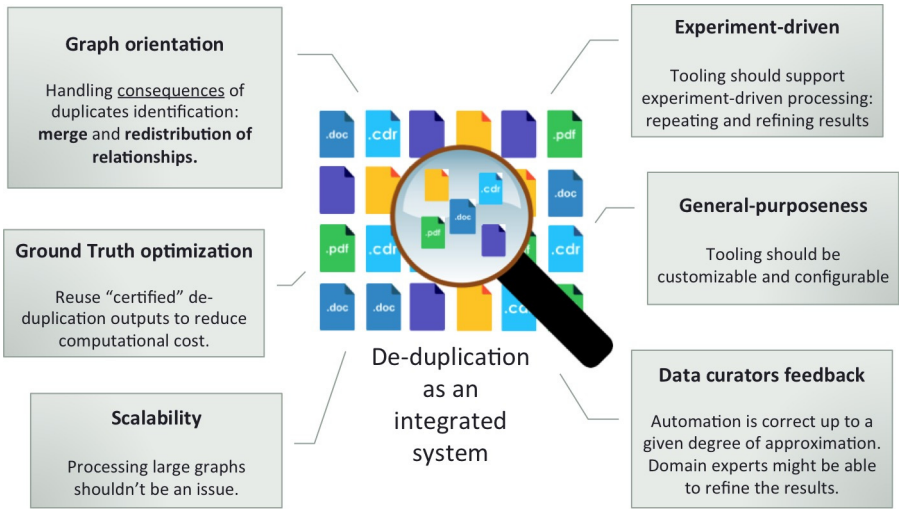


Fig. 2.1: Big Graph Entity Deduplication Systems: tooling requirements

address the non-functional requirement of scalability. Finally, we draw some remarks on the limits of such approaches w.r.t. Big Data deduplication scenario.

2.1 Before Big Data advent

The literature and the market offer a number of tools, which mainly address record linkage between information systems or deduplication of one information system. Record linkage tools focus on the problem of identifying a set of record matches across two or more information systems to be integrated or combined for processing. Typically, candidate identification is not necessarily intended for record merging and deduplication. Strengths and differences of such tools [14] lie in:

- the techniques devised to effectively and efficiently identify candidate record duplicates;
- the level of usability (i.e. user-friendly interfaces versus low-level configuration) and personalisation (i.e. , general-purpose versus domain specific tools [74, 86].

We examined differences and similarities among existing tools, which we believe are relevant and representative with respect to the goals of GDup . The following tools were selected, which target record linkage and deduplication at different degrees: LinkageWiz,[1] FRIL[2] [40], D-Dupe[3] [41], Febrl[4] [13], SRA[5] [70].

[1] LinkageWiz `http://www.linkagewiz.com`
[2] Fine-grained Records Integration and Linkage `http://fril.sourceforge.net`
[3] D-Dupe `http://linqs.umiacs.umd.edu/projects/ddupe`
[4] Freely Extensible Biomedical Record Linkage `https://sourceforge.net/projects/febrl`
[5] Systematic Review Assistant `https://github.com/CREBP/SRA`

In the analysis, we focused on the common data processing phases they support:

- records import;
- candidate identification configuration;
- generation of results.

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

- *Data import functionality*: Febrl, LinkageWiz and FRIL allow for cleansing (normalisation) of imported records, while FRIL also allows for deduplication of data sources to be linked.
- *Data import formats and protocols*: Febrl and FRIL support Excel and CSV Text Files, LinkageWiz supports Excel, CSV, DBASE, FoxPro and MS Access, while D-Dupe works with a proprietary representation of relational data. LinkageWiz and FRIL can also collect data from JDBC connections, while SRA, although providing a tight integration with the Thomson Reuters commercial tool EndNote,[6] supports also importing data from CSV files.

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

- *Clustering methods the users can choose from*: all of them offer traditional blocking (i.e. the string representation of field values is used as blocking key), FRIL, Febrl and D-Dupe support also literature variations of blocking (i.e. , values obtained from elaboration of field values, such as metaphone and soundex), while FRIL and Febrl also provide sorted neighbourhood [16]. In general, the selection of a method requires pre-processing of the records before matching can be applied.
- *Similarity distances the users can choose from*: record similarity function is based on a weighed mean that can be configured based on a list of distance functions, field weights and in some cases, such as in FRIL, introducing conditions. SRA provides a predefined algorithm tailored to identify duplicates in bibliographic references.

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

[6] `http://endnote.com`

2.2 After Big Data advent

In recent years, as the new established paradigm of Infrastructure as a Service facilitated the provision of computing infrastructures, the possibility to develop new systems for Big Data, as well as re-thinking existing solutions led to the emergence of new research and a variety of innovative systems. In particular the Map Reduce programming model has been widely adopted as it enables distributed processing of large data volumes, while abstracting the details of distributed computing and the treatment of hardware failures. Map Reduce permitted to re-implement well known methods for Entity Deduplication, and among these the blocking technique [64] is well suited for a parallel implementation, as the record *blocks* can be processed to compute detailed comparisons independently by parallel processes. This new approach leaded to significant gain in term of process *efficiency*, as the underlying infrastructure can scale out thanks to the addition of new commodity hardware.

One of the most mature tools (as well as available online) that directly address the efficiency issues of deduplication is Dedoop[7] [47]. It exploits the MapReduce programming model to implement several optimized blocking strategies [51], [50] on top of Apache Hadoop clusters[8] to realise efficient entity resolution functionalities:

- a web interface to specify entity resolution strategies for match tasks;
- automatic transformation of the workflow specification into an executable Map Reduce workflow and manages its submission and execution on Hadoop clusters;
- it is designed to serve multiple users that may simultaneously execute multiple workflows on the same or different Hadoop clusters;
- provides load balancing strategies to evenly utilize all nodes of the cluster;
- includes a graphical HDFS and S3 file manager that allows users to upload CSV files;
- supports the repeated execution of jobs within a Map Reduce workflow until a stop condition if fulfilled;
- supports to configure and launch new Hadoop clusters on Amazon EC2;
- can be adapted to configure, schedule, and monitor arbitrary Map Reduce workflows.

A generic framework for declarative entity resolution is Dedupalog [1]. It provides a powerful syntax to express the identification of duplicate entities, i.e. the definition of matching cases that produces equality relationships. However, its scope is limited to a sub-part of our goal.

Another tool that addresses the problem of authority control is PACE (Programmable Authority Control Engine) [61]. Authority control is the combination of software resources and human actions required to maintain authority files, which are lists of authoritative records uniquely and unambiguously describing corresponding sets of entities in an information system. PACE user interfaces deliver a framework where data curators can create and maintain their authority files. Its novelty is that curators can conduct continuous

[7] Dedoop http://dbs.uni-leipzig.de/dedoop
[8] Hadoop https://hadoop.apache.org

activities of population, customisation and evaluation of record matching algorithms. The back end copes with storage and processing scalability of a high number of records and hence, with the potentially high I/O costs implied by sorting, reading and writing million of records, by relying on multi-core parallelism and Cassandra's storage technology [53].

2.3 Remarks

In this thesis we focus on BGEDSs , integrated systems that operate over graphs of interlinked objects to provide tooling for both candidate identification and graph disambiguation.

Overall, existing tools for deduplication fail to meet one or more of the requirements of BGEDSs. As described above the deduplication tools considered in our analysis are mostly focused on resolving record duplicates in datasets (CSV files are the most common type of input), or one information system (typically a relational database). As for highlighting their ability to address the requirements of Big Graph deduplication scenarios illustrated in figure 2.1, we can conclude that:

- *Graph oriented approach*: among the tools considered in our analysis the only one that support graph functionalities is D-Dupe. However, graph data structures are not supported in the input phase, as it allows users to explore the overlay graph of the equivalence relationships produced by a given configuration. This feature however is limited to 300 pairs of equivalent records. LinkageWiz and FRIL support JDBC connections as data source, this means that the objects subject to deduplication can be enriched with the contextual information surrounding them in a given E-R data model (i.e. table joins). Finally, Dedoop only works with CSV files.
- *Experiment driven*: all the analysed tools can be used to run deduplication experiments. Except for D-Dupe with its graphical network visualisation, and PACE with its user interfaces supporting candidate records evaluation and merge, all the other tools lack of an integrated result visualisation/exploration of the results, therefore the results evaluation must be performed by manually checking the output files, or using external tools to visualise them.
- *General purposeness*: most of the analysed tools can be configured to work on different application domains: LinkageWiz was used to link patient records in clinical registries [35] [77], D-Dupe was used to disambiguate author names in bibliographic metadata records [30], Febrl (although the name suggests its use is limited to the biomedical field) contains different techniques for data cleaning, deduplication and record linkage, exposes them on a graphical user interface, and it is fairly easy to integrate new record linkage techniques in it [13]. Dedoop supports a number of general purpose matching techniques that makes it work effectively on hetereogeneous datasets. PACE, thanks to the customisation of the similarity algorithms can be adapted to different application domains that requires to maintain aggregation authority files. Finally, SRA is tool conceived for a very specific application domain, although it

integrates data curation functionalities it is intended to maintain small sized data sets and it does not consider graph data.

- *Ground Truth*: perhaps the only tool in our analysis that allows to maintain a ground truth is PACE. In fact authority files can be seen as a sort of ground truth. However, none of the other tools consider ground truth information, which is left outside the workflow, and eventually needs to be applied as a pre-processing phase;
- *Scalability*: among the tools in our analysis the only one that are designed to process large datasets are Dedoop and PACE. Dedup is built on top of Apache Hadoop, allows to address complex entity resolution workflows by properly sizing the cluster, i.e. by defining the number of nodes and the resources for each of them. The dataset is then distributed across the cluster nodes and whose splits are processed in parallel; PACE exploits the parallelism of multiple cpu cores on a single node for the data processing, and delegates the persistence layer to a scalable NoSQL database.
- *Data curators feedback*: Except for D-Dupe, which is specifically designed as an interactive tool supporting data curators with an iterative and interactive deduplication process [41], the other tools in our analysis are less prone to include the data curator feedback into account in the application workflow.

A general analysis of the requirements of the functionalities for deduplication and record linkage systems was proposed by Köpcke, Thor and Rahm in [52]. However, the premises and constraints of the Entity Disambiguation problem on graphs of interlinked objects are not entirely addressed in the current state of the art, and from the analysis we carried out, none of the tools consider all the requirements we described in this chapter. In particular the most important remark is about the lack of tools that consider the Entity Deduplication workflow from end to end, i.e. from the graph input phase to the materialisation of the disambiguated results, mediated by end user feedbacks. Another important aspect is the lack of support for the import, construction and reuse of ground truth information.

3

Real-case scenario: the OpenAIRE infrastructure

To describe the problem of Big Graph deduplication and motivate the need of Big Graph Entity Deduplication Systems (BGEDSs), in this chapter we introduce the OpenAIRE infrastructure system, whose services populate a very large graph of scientific publications, authors, and other related entities, and whose size and deduplication challenges are representative of this class of problems. The next sections describe the OpenAIRE services and graph data model, then focus on the deduplication issues introduced by this scenario and highlight the functional and non-functional challenges arising when trying to formulate the respective solutions.

3.1 The OpenAIRE Information Space Graph

The OpenAIRE initiative (`http://www.openaire.eu`, [60]) is funded by the European Commission to become the point of reference for Open Access and Open Science in Europe. Its mission is to foster an Open Science e-Infrastructure that links people, ideas and resources for the free flow, access, sharing, and re-use of research outcomes, services and processes for the advancement of research and the dissemination of scientific knowledge. OpenAIRE operates an open, participatory, service-oriented infrastructure that supports:

- The realization of a pan-European network for the definition, promotion and implementation of shared interoperability guidelines and best practices for managing, sharing, re-using, and preserving research outcomes of different typologies;
- The promotion of Open Science policies and practices at all stages of the research life-cycle and across research communities belonging to different application domains and geographical areas;
- The discovery of and access to research outcomes via a centralized entry point, where research outcomes are enriched with contextual information via links to objects relevant to the research life-cycle;
- The measurements of the impact of Open Science and the return of investment of national and international funding agencies.

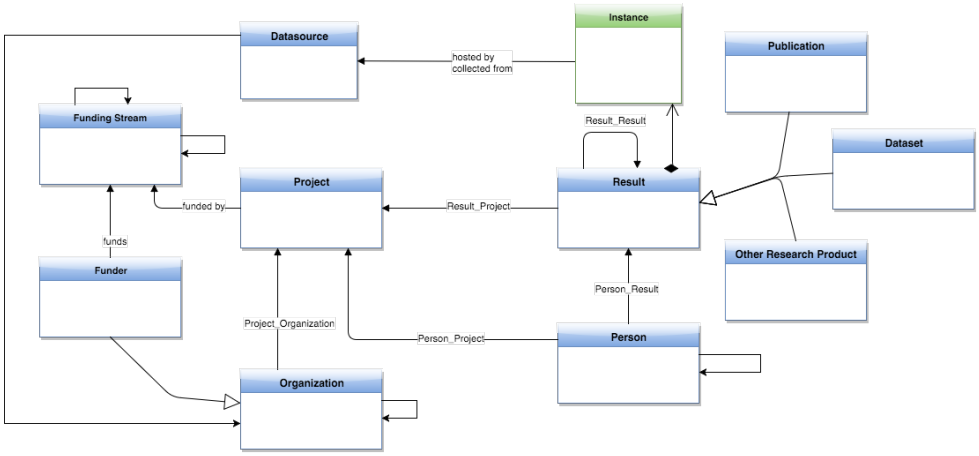


Fig. 3.1: OpenAIRE Data Model

In particular, OpenAIRE operates a technological infrastructure supporting this process. The infrastructure provides aggregation services [60] capable of collecting content from data sources on the web in order to populate the so-called *OpenAIRE Information Space*, a graph-like information space describing the relationships between scientific articles, their authors, the research datasets related with them, their funders, the relative grants and associated beneficiaries. By searching, browsing, and post processing the graph, funders can find the information they require to evaluate research impact (i.e. return of investment) at the level of grants and funding schemes, organized by disciplines and access rights, while scientists can find the Open Access versions of scientific trends of interest. The OpenAIRE Information Space is then made available for programmatic access via several APIs (Search HTTP APIs, OAI-PMH, and soon Linked Open Data) [57], for search, browse and statistics consultation via the OpenAIRE portal (`www.openaire.eu`), and for data sources with Notification Broker Services [2].

The graph data model [59], depicted in Figure 3.1, is inspired by the standards for research data description and research management (e.g. organizations, projects, facilities) description provided by DataCite[1]and CERIF[2], respectively. Its main entities are *Results (datasets and publications), Persons, Organizations, Funders, Funding Streams, Projects*, and *Data Sources*:

Results are intended as the outcome of research activities and may be related to Projects. OpenAIRE supports two kinds of research outcome: *Datasets* (e.g. experimental data) and *Publications* (other research products, such as Patents and Software will be introduced). As a result of merging equivalent objects collected from separate data sources, a Result object may have several physical manifestations,

[1] Data Cite, `https://www.datacite.org`
[2] EuroCRIS, CERIF features, `http://eurocris.org/cerif/main-features-cerif`

called *instances*; instances indicate URL(s) of the payload file, access rights (i.e. open, embargo, restricted, closed), and a relationship to the data source that hosts the file (i.e. provenance).

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 (e.g. European Commission, Wellcome Trust, FCT Portugal, Australian Research Council) are Organizations 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 Results.

Data Sources , e.g. publication repositories, dataset repositories, CRIS systems, journals, publishers, are the sources on the web from which OpenAIRE collects the objects populating the OpenAIRE graph – typologies and numbers of data sources currently included in OpenAIRE are available from `https://www.openaire.eu/search/data-providers`. Each object is associated to the data source from which it was collected. More specifically, in order to give visibility to the contributing data sources, OpenAIRE keeps provenance information about each piece of aggregated information. Since de-duplication merges objects collected from different sources and inference enriches such objects, provenance information is kept at the granularity of the object itself, its properties, and its relationships. Object level provenance tells the origin of the object that is the data sources from which its different manifestations were collected. Property and relationship level provenance tells the origin of a specific property or relationship when inference algorithms derive these, e.g. algorithm name and version.

Objects and relationships in the OpenAIRE graph are extracted from *information packages*, i.e. metadata records, collected from internet/web accessible sources of the following kinds:

Institutional or thematic repositories Information systems where scientists upload the bibliographic metadata and PDFs of their articles, due to obligations from their organization or due to community practices (e.g. ArXiv, EuropePMC);

Open Access Publishers Information system of open access publishers or relative journals, which offer bibliographic metadata and PDFs of their published articles;

CRIS systems Current Research Information Systems are adopted by research and academic organizations to keep track of their research administration records and relative results; examples of CRIS content are articles or datasets funded by projects, their principal investigators, facilities acquired thanks to funding, etc.;

Data archives Information systems where scientists deposit descriptive metadata and files about their research data (also known as scientific data, datasets, etc.); data archives are in some cases supported by research and academic organizations and in some cases supported by research communities and/or associations of publishers;

Entity registries Information systems created with the intent of maintaining authoritative registries of given entities in the scholarly communication, such as OpenDOAR[3] for the institutional repositories or re3data.org[4] for the data repositories;

Aggregator services Information systems that, like OpenAIRE, collect descriptive metadata about publications or datasets from multiple sources in order to enable cross-data source discovery of given research products; aggregators tend to be driven by research community needs or to target the larger audience of researchers across several disciplines; examples are DataCite[5] for all research data with DOIs as persistent identifiers, BASE for scientific publications[6], DOAJ for Open Access journals publications[7]. Aggregation systems however, find relevant applications also in different contexts. For example the project HOPE - Heritage of the People's Europe aggregates digital collections from social history and the history of the labour movement from the late 18th to the beginning of the 21st century [3].

From the data sources mentioned above (over 5000 today), the OpenAIRE aggregation services collect *metadata records* (mainly XML files, but also CSV files, JSONs, structured HTML responses, etc.) and perform two actions: (i) extracting from metadata records OpenAIRE objects and relationships to feed the information space graph and (ii) harmonizing object properties to make them conforming to the OpenAIRE data model types. For example, as shown in Figure 3.2, a Dublin Core bibliographic metadata record describing a scientific article will yield one OpenAIRE publication object and a set of OpenAIRE person objects (one per author) with relationships between them.

The evolution of the OpenAIRE information space has been observed over the last 10 years, starting from the former EC funded DRIVER projects (2006-2009), to the OpenAIRE pilot (2009-2011), OpenAIREplus (2011-2014) and OpenAIRE2020 (2015-2018). Today it counts more than 16 million publications. In order to facilitate interoperability across institutional repositories world-wide, the DRIVER projects first and then OpenAIRE's series developed in collaboration with the community of institutional repository managers – at the global level with COAR, SHARE-US, La Referencia-South America, JISC-UK – the so-called OpenAIRE guidelines for publication metadata records.[8] The guidelines are a specialization of the Dublin Core Metadata specification,[9] an example of which is illustrated in Figure 3.2, and provide instructions on how to express structure,

[3] OpenDOAR - `http://opendoar.org`
[4] Re3data - `http://www.re3data.org`
[5] DataCite - `https://www.datacite.org`
[6] BASE - `https://www.base-search.net`
[7] DOAJ - `https://doaj.org`
[8] OpenAIRE guidelines, `http://guidelines.openaire.eu`
[9] Dublin Core Metadata Element 1.1, `http://dublincore.org/documents/dces`

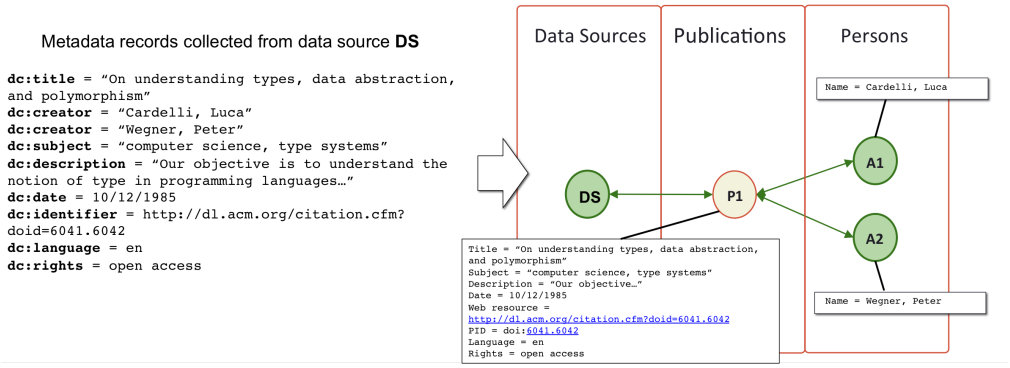


Fig. 3.2: OpenAIRE harmonization process: from metadata records to OpenAIRE objects

semantics, and exchange format when exposing bibliographic metadata records to third-party services. Although the adoption of the guidelines is high for European (and beyond) repositories, some of the records they expose are still subject to heterogeneous semantic interpretations. OpenAIRE aggregator services must therefore support a number of data source-specific metadata transformation mappings, which regard both structure (e.g. mappings between input entities and OpenAIRE entities, mappings between properties and OpenAIRE properties) and semantics (e.g. different vocabularies). Examples of such mappings are:

- Ruling out metadata records: exclude processing of records whose mandatory fields are empty or missing at all (e.g. title, authors);
- Value extraction: e.g. identify a DOI code in dc:identifier, and define it as a dedicated field;
- Vocabulary mapping: mapping local vocabularies onto terms defined in system controlled standard vocabularies, e.g. ISO for countries and languages;
- Format normalization: e.g. mapping date formats onto common standard formats;
- Entity extraction: as shown in Figure 3.2, a metadata record may give rise to multiple interrelated objects in the OpenAIRE graph.

The number of objects of each entity currently included in OpenAIRE are summarized in Table 3.1, up-to-date figures can be found at `http://www.openaire.eu`.

3.2 Deduplication issues

As exemplified in Figure 3.2, OpenAIRE aggregation services collect metadata records from data sources and create the relative OpenAIRE objects and relationships in the OpenAIRE graph. The quality of the resulting graph suffers from serious duplication phenomena. In particular, objects of the entity types *publication*, *person*, and *organization*

Entity Type	Number of objects
Data Source	6,081
Publication	14,632,876
Person	13,727,371
Project	624,392
Organization	62,798

Table 3.1: Number of objects by entity in OpenAIRE (update to date 2016-02)

are affected by different forms of duplication, all providing interesting scenarios which we will address in this section. The main cause of such phenomena is the scarcity of use of PIDs, which are unique and global persistent identifiers for the collected objects. Although the scholarly communication has agreed on some best practices (e.g. DOI for scientific articles, ORCID identifiers for authors) in several cases such PIDs are not made available in the metadata collected from the data sources. As a consequence, OpenAIRE aggregation services generates unique object identifiers for each publication, author, or organization extracted from the metadata, relying on the original object identifiers locally assigned by the data source: OpenAIRE identifier generation is stateless (hence invariant) and is based on the combination of the unique identifier of the data source and the MD5 of the local identifier for the object. More in detail, publications, persons, and organizations suffer of the following duplication issues, caused by intra-data source or cross data source logics:

Publications Intra data source duplicates for publications are in most cases due to the fact that the data source is itself an aggregator service (e.g. NARCIS, CORE-UK, etc), and in some rare cases due to the lack of curation of the data source managers (users in charge of metadata quality). Not all aggregators collect from other aggregators, but this is indeed the case for OpenAIRE, which is trying to maximize the number of publications collected and mitigating data source overlaps by applying deduplication. Cross-data source duplicates are instead common as we at least expect all co-authors of a publication to deposit it in the respective institutional repository of reference, which will likely be OpenAIRE data sources. In turn, the publication can be further deposited in thematic repositories or be collected by aggregator services.

Persons Intra data source duplicates for persons of type authors (see Figure 3.3) are instead very common. All publications of the same author in a repository will generate a new author object in the graph, potentially different from the others. Cross-data source duplicates (see Figure 3.4) are mainly due to the fact that co-authors are depositing in their institutional repositories or in thematic repositories (e.g. ArXiv, EuropePMC), or again, as in the previous case, their publications have been collected by an aggregator data source harvested in OpenAIRE.

Organizations Organizations are mainly collected from CRIS systems and entity registries. Their duplication is mainly due to intra-data source logics, since they are collected from a few data sources where they typically appear as a secondary entity,

not subject to disambiguation, and in some cases they totally lack of any kind of local identifier.

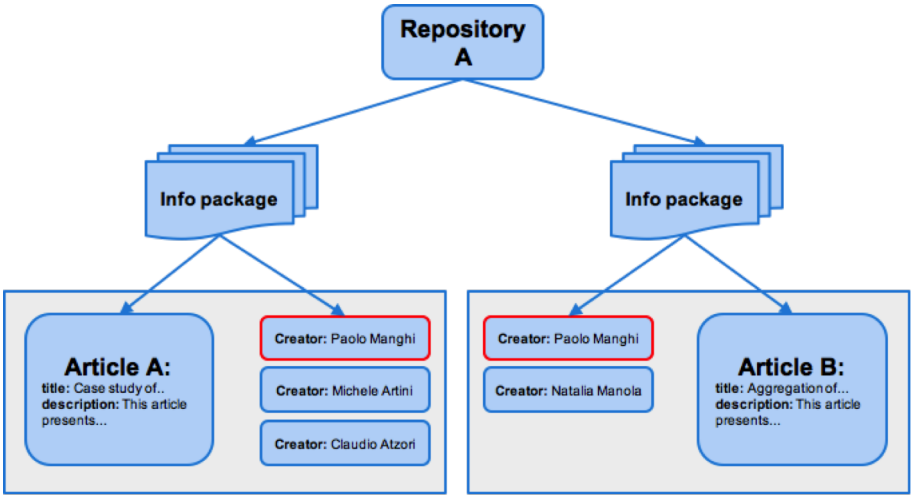


Fig. 3.3: Intra-data source duplicates: repository data sources

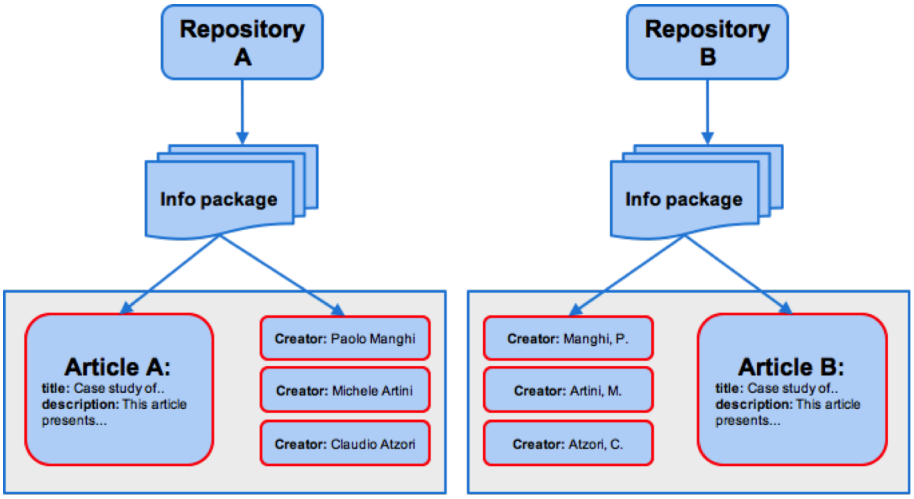


Fig. 3.4: Cross-data source duplicates: repository data sources

In order to present consistent data to its consumers, which are often interested in statistics for example, OpenAIRE requires to deduplicate objects in the graph. As explained

in the introduction, the automation of this process must include two phases, *duplicates identification* and *graph disambiguation*:

Duplicate identification OpenAIRE deduplication requirements are quite standard. Duplicates identification consists of two steps, *candidate identification* and *candidate matching*, which both act at the level of the individual entities of the graph, hence over collections of objects of the same type. Candidate matching is based on the assumption that pairs of objects of a given entity can be compared by means of their properties in order to establish their similarity measure. As mentioned above, OpenAIRE aggregation services clean and harmonize the values of object properties to build uniform entity collections. In theory this process should facilitate both phases above, by suggesting strategies to identify candidate pairs and subsequently identify duplicates, but, as we shall see, the real experience tells us a whole different story and systems supporting this process must very configurable and extensible. In a scenario like OpenAIRE, matching all pairs of objects is by no means intractable. To give an idea of the numbers, today, we count about 15M^2 matches for publications and about 70M^2 for authors. To solve these issues, candidate identification is the phase entitled of providing the heuristics and technological support necessary to avoid this "brute force" solution. Candidate identification is typically solved using clustering techniques [64]. These are based on a *clustering function* that associates to each object, out of its properties, one ore more *key* value to be used for clustering. The idea is that objects whose keys fall in the same cluster are more likely to be similar than across different clusters. This action narrows the pairwise matching of objects within the clusters, thereby reducing the complexity of the problem from a $O(n^2)$ to $O(n \log_n)$.

The definition of a good clustering function for deduplication must respect the following strategy: (i) avoiding false negatives: making sure that obvious duplicates fall in the same cluster, (ii) avoid false positives: making sure that clearly different objects fall in the same cluster, and (iii) the number of matches becomes tractable by the technology at hand. While traditionally the problem has been mainly algorithmic, with the advent of big data technologies, such as the map reduce tooling, an optimal solution can also count on very high performance ratios.

Graph disambiguation Duplicate identification terminates providing, for each entity type, a set of pairs of object duplicates. In order to disambiguate the graph, the requirements of OpenAIRE is that duplicates should be "hidden" to be replaced by a *representative object* which points to the duplicates it represents (and viceversa) and becomes the hub of all incoming and outgoing relationships relative to these objects. As a result, the graph will now be disambiguated but still keep track of its original topography. As a consequence, graph disambiguation consists of two phases: *duplicate grouping* and *duplicates merging*. Grouping duplicates requires the identification of

the connected components[10] formed by the equivalence relationships identified by duplicate identification. Merging the groups of duplicates requires instead the creation of a representative object for each connected component (or group of duplicates) and the propagation towards this new object of all incoming and outgoing relationships of the object it merges. Both actions have serious performance implications, which depend on the topography of the graph (e.g. fragmentation of graph, edge distance of objects in the graph, number of the duplicates). For example, it is generally expected that person duplicates have a higher average ratio than those of publications and organizations. The number of duplicates for one author is at least equal to the number of author's publications in the system. The number of duplicates for one publications depends on the replication of the publication across different data sources, e.g. institutional repository of the author, thematic repository, and a number of aggregators, but it is in general not very high (e.g. co-authors, each depositing in their respective institutional repositories which are in turn harvested by OpenAIRE). Duplicates of the same organizations are in general not many, as not many data sources in OpenAIRE are providing information on organizations. Moreover, the order of magnitude of the collections of different object types varies considerably: 10^7 for publications and person objects, and 10^4 for organizations.

In the following we analyse the challenges arising when finding solutions to duplicate identification and graph disambiguation in the case of publications, persons, and organizations entities in OpenAIRE. As we shall see, the heterogeneity or absence of the information attached to the aggregated objects, as well as their duplication ratio and characteristics, yield a number of challenges that are typical of Big Graph entity deduplication. Such challenges result in a set of requirements that a BGEDS should address in order to support practitioners at facing and solving this class of problems. Finally, as a result if this analysis we illustrate the functional and non-functional requirements for BGEDSs.

3.2.1 Duplicate identification: publication entity

Publication objects, as collected from data sources, are described by the properties depicted in table 3.2. OpenAIRE unique identifiers for publication objects are generated by combining a unique prefix associated to the data source from which the record was collected and the unique identifier of the record in the scope of the data source. This creates a unique and stateless (i.e. the same identifier will be created if and when the record will be collected again) identifier for each publication object in the system.

Candidate matching

In order to define a solid candidate matching function, we need first to identify the properties that can immediately imply the equivalence of two objects, namely global IDs for

[10] In graph theory, a *connected component* of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

Property	Multiplicity	Type	Global ID	Percentage of Null
OpenAIRE_ID	1	String	Yes	0%
Title	1..N	String	No	1%
Author name	0..N	String Template	No	2%
Date of acceptance	0..1	Date	No	5%
Publisher	0..1	String	No	60%
Abstract	0..N	String	No	20%
PID	0..N	String	Yes	70%
Language	0..N	ISO vocabulary	No	15%
Subject	0..N	String	No	60%
URL	1..N	URL	Yes	0%
Access rights	1	Controlled vocabulary	No	0%
Journal codes (e.g. ISSN)	0..1	String Template	Yes	70%

Table 3.2: Property of publications: deduplication support

publications. We can identify two of them, PIDs and URLs. Unfortunately the percentage of Null values for PID is quite high, but still we can rely on URLs to perform exact matches. In both cases, however, global IDs are not enough, since the same publications may have different global IDs, be available without the global ID, or be available at different locations, i.e. different URLs. Accordingly, if matches based on publication global IDs fail, other strategies must be devised. To this aim, we need to identify which properties are most influential in the matching process, that is best contribute to establishing object equivalence by (i) introducing lower computational cost (to be multiplied by the number of matches), (ii) allowing clear cut decisions, i.e. when values are similar chances that relative objects are similar are high, and (iii) are often present in the objects. It turns out that the only reliable property is *Title*, which are present in (almost) all objects and consist of a relatively short text, which can be fast and reliably processed by known string matching functions [17], and are highly selective. In general, if the titles of two publications are not “enough” similar, according to a given threshold, then no other property-to-property match may revise this decision. Of course title equivalence is not enough as one of the following cases may occur:

- Very short titles, composed of few, commonly used words may lead to obvious equivalence; e.g. the title "Report on the project XYZ" may be recurrent, the only difference being the name “XYZ” of the project;
- Recurrent titles; e.g. the title “Introduction” of some chapter is very common and introduces ambiguity in the decision;
- Presence of numbers in titles of different published works; e.g. the title “A Cat’s perspective of the Mouse v2” is likely referring to a publication different from "A Cat’s perspective of the Mouse", but not different from “A Cat perspective of the Mouse";

As a consequence, the decision process must be supported by further matches which may strengthen the final conclusion, possibly based on one ore more of the following publication properties:

Author names The set of authors of an object consists of a set of strings representing author full names. They tend to be present in the majority of the cases and their uniformity of template (i.e. "Surname, N.") has a quite high percentage of use. However, introducing them in the matching process is not trivial for the following reasons:

- Although author names respect the same template their are often written in different manners, e.g. "Turing Alan", "Turing, Alan" "Turing, Mathison Alan", "Turing, M. Alan", "Turing, Alan M."; identifying a string matching function and a threshold capable of identifying pairs of equivalent name strings is rather impossible without introducing false negatives (if we are too strict) and positives (if we are too tolerant);
- Some languages are known to introduce homonyms with a ratio that would make even exact matches prone to errors; e.g. Chinese author names;
- Some publications have hundreds of authors (e.g. physicians) and would therefore introduce a high computational barrier to this process.

De facto, in order to introduce effective name matching we should first deduplicate authors (discussed in following paragraph) and use the relative unique identifiers. As an alternative, the number of authors can be introduced in the process. Author counting adds certainty to the choice and rarely introduces false positiveness or negativeness in the process, when for example authors are wrongly inserted as a sequence of string in the same *author* property or one of the authors is missing;

Date of acceptance This property is often present, but not uniformly used, typical misuses including:

- The lack of a day-grain date, compensated using a value of 01-01-YYYY;
- Typos in date values that are hard to identify hence to map onto a common format;
- Different interpretation of the date, which is not always date of acceptance.
- Different versions of the same work might be described with different dates in some cases, with the same date in others, leading to inconsistent results.

Introducing dates in the matching process has a high risk of introducing false positives.

Abstract The publication abstract, although in principle contains considerable amount of context that may help in publication matching, cannot be used as is for string matching as typical string distance algorithms might be either too costly (e.g. Levenstein [71]), or not suited for such long strings (e.g. Jaro-Winkler [17]).

Language Language cannot be used in the matching process as it is not used uniformly; in most cases it is denoting English and it is not clear if it refers to the abstract of the article or the full-text of the article;

Subject Subject could be extremely relevant, but due to the high degree of heterogeneity of subject vocabularies at original sources, which often use free text, this property is just simple text and cannot therefore be meaningful to deduplication;

Access rights Access rights are always present and obey to a controlled vocabulary but do not impact on deduplication; in fact, different copies of the same object may well

have different access rights (e.g. the pre-print open access version and the publisher journal version);

ISSN As for access rights, ISSN do not impact on deduplication; in fact, different copies of the same object may be the publisher version and the self-archived, open access version of the article, available from the institutional repository.

Therefore, a similarity function, based on the availability of certain properties can take straightforward decisions on equivalence or difference between publications, while in other cases can only come up with a rank of confidence that depends on the availability and weights of the properties above. In some cases, however, the lack of information may lead to identify false positives as duplicates. These errors can be sometime be classified as categories of publications, for example those with very short and common titles (e.g. “Introduction”, e.g. “Bullettin”), or those specific pairs of objects notified by users as different. In both cases, the deduplication process should be able to *blacklist* these scenarios, hence exclude sets of publications that fall in such categories. Similarly, end-users may suggest that publications that have not been identified as identical are instead duplicates (for example in the case of classes of object short titles blacklisted by the system on request of the user). The system should be able to include such duplicates as part of the duplicate identification phase.

Candidate identification

The definition of a good clustering function for deduplication of publications should start from the properties in Table 3.2. From the analysis of the properties in the previous paragraph it seems clear that the only property to be always present and informative enough is *title*. Clustering objects starting from a publication title is not easy and may be done according to different strategies, which avoid or tolerate minor differences in the titles, typically caused by typos or the partial/full presence of words. Some examples are:

1. removing stop words, blank spaces, etc.;
2. lower-casing all words;
3. using combination of prefixes or postfixes of title words;
4. using *ngrams* of relevant words;
5. using hashing functions.

Using any of these strategies has implications that depend on the features of titles in the information space. For example, the heavy presence of short titles, i.e. a few words, may find in the hashing function a better solution than using prefixes of words. On the other hand, the adoption of high performance technology may allow for a more greedy approach, which allows for more matches to be performed hence avoid false negatives. The OpenAIRE information space is very heterogeneous as both data sources and disciplines behind publications are of different kinds. As a consequence the preferred approach is the ones that combines the first letters of words (like an acronym) into a clustering key and the last letters of words into another clustering key. The approach is quite typo-safe

and seems to exclude false negatives, on the other hand it includes false positives, which should be excluded with the subsequent detailed similarity match.

The number of OpenAIRE publications is in the order of tens of millions. As a consequence the requirement is to adopt a big data processing technology, which allows for scalability as well as for efficient parallel processing, and easily supports a clustering-based methodology. However, despite the clustering, the number of clusters and the objects in the clusters is quite high and the number of pairwise matches can reach critical peaks, e.g. long processing times. Today, for publications the clustering functions identify 1,5M clusters with more than 1 object, which feature an average of around 12 publications per cluster and a standard deviation of 105 (with the maximum size of a cluster being of 72,000 publications). This means that from the initial 15M publications leading to $15M^2$ matches, clustering takes the number down 1,5M $\times$ X^2. The decrease is considerable, but still the number of matches is quite high (in the order of Billions if X matches the standard deviation worst case), hence further heuristics can be introduced. An example of such heuristics is the *sliding window* technique [10]. Sliding windows techniques are based on the idea that objects in the cluster are ordered in such a way the chance that similar objects are closer in the ordering is higher. Objects in the cluster are then pairwise matched only if they are part of the same sliding window of length K. Once all objects have been matched, the sliding window is moved to the next element of the ordering and a new set of pairs is matched. Sliding windows introduce false negatives, since they exclude from the match objects in the same cluster, but control performance by giving an upper bound to the number of matches in each cluster. In summary, the realization of a sliding window approach requires providing (i) an *hashing function*, which returns the key based on which an object will be sorted, (ii) a *sorting function*, which will be used to sort the objects based the relative keys, and (iii) the length of the sliding window, which puts a maximum threshold on the number of pairwise matches. Such approach is known in literature as *Sorted Neighbourhood* [31] [91]. Again several strategies may be adopted, the title of the publication being the candidate input for the sorting function; here are some examples of has functions to be sorted as strings:

1. obtaining one string from the title by keeping all relevant lower-cased words in the title;
2. obtaining one string from the title by keeping all relevant lower-cased words in the title but excluding vowels.

In OpenAIRE the current solution is the last one, but algorithms are being refined in order to meet new requirements or solve deduplication failures.

3.2.2 Duplicate identification: person entity

Person objects, mainly authors of articles in OpenAIRE, are described by the properties in Table 3.3. Authors are created from publication bibliographic records out of author names strings (see Table 3.3, corresponding to the *Original Fullname* property. The OpenAIRE identifier is generated as a combination of the OpenAIRE identifier of the publication

object from which the author is being created and the author name. This makes the author object unique in the system, but generates different persons also when the person is the same author – this strategy is preferred in order to avoid the creation of person objects relative to homonyms. Aggregation services apply transformation rules to convert the string in *Original Fullname* (e.g. “Turing, M. Alan”) onto separate *Surname* (e.g. “Turing”) and *Firstname* (e.g. “M. Alan”) properties. As we shall see, this structure is important when de-duplicating person objects. In some rare cases, data sources also provide a persistent identifier, uniquely identifying the author on the Web (e.g. ORCID identifiers, Google Scholar idenfiers); as such, identifiers are stored as pairs, together with their encoding scheme. Authors also include information about their co-authors, which consist of a list of pairs author full name and relative OpenAIRE identifier in the system.

Property	Multiplicity	Type	Global ID	Percentage of Null	Percentage of conformity to template
OpenAIRE_ID	1	String	Yes	0%	100%
Persistent identifiers	0..N	scheme: String; ID: String	Yes	99 %	n.a.
Firstname	0..1	String	No	35%	n.a.
Surname	0..1	String	No	35%	n.a.
Original fullname	1..1	String	No	0%	n.a.
Co-authors	0..N	OpenAIRE_ID: String; firstname: String; surname: String	No	2%	n.a.
email	0..1	String	Yes	98%	n.a.

Table 3.3: Property of persons: deduplication support

Candidate matching

It is not very common for person objects to feature a global identifier. The reasons is that only recently the research community, along with research and academic organizations, has started to feel the urgent need of such powerful disambiguation mechanisms and started integrating them in their processes and information systems (e.g. ORCID). As such deduplication of person objects must rely on different strategies. Given the properties in Table 3.3, the only ones that appear informative enough to enable matching and draw some conclusions are *Firstname*, *Surname*, and *Co-authors*. Intuitively, by first matching the first name and surname we can identify a pair of candidate for similarity (if these are not available we can match the original fullnames, although reliability of matches is lower). If the match is positive, then the co-author lists can be confronted (using the OpenAIRE identifiers or the person name strings) and if a minimum number of matches is found, the two authors can be identified as duplicates. On the other hand, this decision process introduces two issues. The first is that person objects originated from single-authored publications cannot be de-duplicated: a person object created from a single authored publication does not have co-authors, hence does not have enough

context to take decisions in terms of duplication. The second issue is that person deduplication becomes a circular process. When two person objects are merged into one, a new identifier is created for the resulting person object. As a consequence (i) this object contains now the union of the co-authors of the merged authors, (ii) for consistency all occurrences as co-authors of the two merged authors should now be replaced by the new object identifier, and (iii) this new, richer scenario can fire a new set of possible duplication, hence a new cycle of deduplication. The process should be iterated until it converges (i.e. no more matches are identified, hence co-authors are no longer updated). Due to its circularity, person deduplication cannot be re-executed from scratch every time new person objects are injected in the graph (as it happens in aggregation systems like OpenAIRE). The solution seems therefore to suggest an "incremental" approach, whereas the final result of previous cycles of deduplication become the starting point for deduplicating new person objects. However, the adoption of a fully incremental approach is not realistic, as it assumes that all de-duplication cycles are and will be correct: in other words, matching functions are not prone to errors and that person objects (e.g. the nature of their names) will never introduce new challenges that will require the matching functions to be modified. Accordingly, the best solution is typically that of starting from a so called "Ground Truth", which is a set of deduplicated objects called *anchors* whose quality has been somehow validated based on some criteria of quality acceptable for the community. Anchors are objects that merge *raw* objects in the graph, hence define an authoritative map that can be used to disambiguate the original graph of objects. Typically, the Ground Truth is not only built by means of deduplication tools but a lot of manual or ad-hoc efforts is invested in its construction. For example to solve the problem of person objects with no co-authors special techniques must be devised, capable of identifying co-authors from other sources on the Web. The introduction of a Ground Truth approach changes the perspective of the deduplication process, since part of the deduplication can be made available before candidate identification by mapping raw objects onto anchor objects at graph import time. Such pre-processing can considerably reduce the time to process the whole collection of person objects, limiting the identification of duplicates to the objects that are not yet part of the Ground Truth.

Candidate identification

Identifying a clustering function for person objects in OpenAIRE can only be based on the *fullname*, *firstname*, or *surname* properties, which are the only ones with mandatory content. Different strategise can be adopted, some examples being:

1. lower case all strings;
2. remove known name prefixes, e.g. "Van de", "De"
3. if surname and first name are available as separate properties, e.g. "Turing" and "M. Alan":
   a) using person surnames, e.g. "turing"
   b) using first letter of names combined with surname, e.g. "mturing" and "aturing"

4. if only full name is available, e.g. “M. Alan Turing”:
   a) using all words of length greater than 2, e.g. “alan”, “turing”
   b) using all combinations of first letters of words with other words, e.g. “mturing”, “aturing”, “talan”, “malan”

Depending on the features of the person names in the information space these strategies can be combined. On OpenAIRE the majority of the records has a regular (or is mapped onto a regular) structure of first name and surname. As a consequence the combination of first letters of names with surname seems the most appropriate, in combination, when only the full name is available, with the same techniques extended to all "words" in the person fullname. The approach does not exclude false positives, but excludes false negatives (unless typos are present). False positives are introduced by culstering keys obtained from fullnames, when the unfortunate combination of first letter of surname and firstname, e.g. “talan”, matches an author whose surname matches the given name e.g. “Alan” and the first name starts with the first letter of the given surname, e.g. “Ted”.

As in the case of publications, the potential large number of objects in the same clusters calls for the introduction of a sliding window technique. In this case, the clustering function is based on the fullname of the author, which is the only mandatory field. The clustering function lower cases the fullname string, excludes the stop words, orders the words alphabetically, and then combines them in one string, e.g. ‘M. Alan Turing” becomes ‘alanmturing”.

3.2.3 Duplicate identification: organization entity

Organization objects are collected from entity registry data sources and are described by the set of properties illustrated in Table 3.4. Organization objects identifiers are obtained in a stateless way as for publications, when organizations are first-level objects in the original data source (i.e. they are assigned a local original identifier) or second-level object data sources (i.e. they are extracted from the metadata information of a first-level object in the data source). As a consequence, all objects have a unique identifier in the system, even if they represent the same organization.

Property	Multiplicity	Type	Global ID	Percentage of Null	Percentage of conformity to template
OpenAIRE_ID	1	String	Yes	0%	100%
Name	1..1	String	No	3%	n.a.
Acronym	1..1	String	No	20%	80%
Website	0..1	String	No	15%	n.a.
Contact person	0..1	String	No	80%	n.a.
Country	0..1	Vocabulary (ISO)	No	98%	100%

Table 3.4: Property of organizations: deduplication support

Candidate matching

Organization objects do not have a global persistent identifier, hence their deduplication strategy must be based on their descriptive properties. Interestingly, no property is mandatory across organization objects, but either the *name* or the *acronym* of an organization object is always present. Moreover, both acronym and name are sometime erroneously filled in with the same string value, which can be either the name or the acronym. The *country* field is almost mandatory, as only a few organizations are missing this information.

A matching function should therefore allow the usage of conditional primitives to check if the *name* property is present on both objects, since the full name of an organization can be discriminant to determine a similarity measure between organizations. If so, the names should be also checked against the acronyms, since in several cases these are used erroneously, and keep the best similarity match. Finally, if the name-acronym match gives a reasonable (to be defined) score, the country match becomes crucial to take a final decision. In general, however, organization disambiguation is a complex problem due to several factors:

- Names of organizations are expressed in several languages and in English, e.g. “University of Padua” and “Università di Padova”;
- Names of organizations are expressed in different but permitted names, e.g. “University of Pisa” and “Pisa University”;
- The “granularity” of the organization at the original data source is not generally agreed on or aligned with other data sources, e.g. “University of Pisa” or “Department of Informatics, University of Pisa” or “XYZ Lab, Department of Informatics, University of Pisa”.

Organization name matching therefore requires some pre-processing, for example to translate common words (e.g. University, Research Center, Center, Department, Institute, School) into a common encoding (e.g. 0001 for “University, Université, Università...”) generally is agnostic of the ordering of the words in the name, and it based on prefixes of words, trying to capture common roots.

Another possible strategy is that of matching the acronyms and then relying on the country match to take a final decision; indeed, although different organizations worldwide can have identical acronyms, the same cannot be considered true at the level of the same country.

Lack of information and heterogeneity of names represent an obstacle that will increment its complexity over time, as long as more organizations enter the system, rather than fade away. Hence, although inaccuracies of the results can be sometime overcome by modifying the similarity function or be simply blacklisted, organization disambiguation requires constant manual curation of the results in order to spot the false negatives that inevitably survive the automated process.

Candidate identification

Although the cardinality of organizations does not fall in the category of Big Data, the adoption of a candidate identification phase can reduce the time to process. In this case, it seems that two different clustering functions are to be devised, one for each the matching functions proposed above:

- Generation of keys from *name* property: due to the multi-linguality and structural heterogeneity of names, a good technique is that of generating prefixes of relevant words; if an encoding of key terms is provided, these can be attached to the prefix once being alphabetically ordered; for example “University of Pisa” and "Pisa’s University" would both generate the same key “Pis-0001" and therefore fall in the same cluster;
- Generation of keys from *acronym* property: for acronyms the idea is to send the acronym itself, once “cleaned up” of the possible idiosyncrasies (e.g. lower-case letters, blank spaces, separators)

Since each clustering function has its own similarity matching function, the deduplication system should support a notion of *musical score*, which can orchestrate the executions of these two different cycles and then combine the resulting pairs of duplicates.

3.2.4 Graph disambiguation

Once pairs of duplicates have been identified for publications, persons, and organizations, the OpenAIRE graph must be disambiguated. The first step is that of grouping pairs of duplicates by propagating the similarity between pairs of objects. The principle is that of transitivity of similarity: deduplication identification may have established that $A = B$ and $B = C$, hence we now need to derive that $A = B = C$. This challenge is known as the *connected components* identification problem. In graph theory, “a connected component (or just component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph”.[11] A connected component for pairs of duplicates corresponds to a set of equivalent objects. The underlying system should provide a scalable and efficient implementation of existing algorithmic solutions and return, for each deduplicated entity collection, the list of sets of equivalent objects, i.e. connected components in the graph. The problem is independent from the specific features of entity types, but should take into account worst cases of connected components whose objects can be at tens of edges of distance.

Once duplicates grouping is completed, the specific requirement of OpenAIRE is not that of keeping one object and remove all its duplicates from the information graph, but rather build an extra object, named *representative object*, that replaces all duplicates while giving them visibility. The idea is that the representative object will serve as a high-quality surrogate of the merged objects, inheriting the union of their properties (maximizing completeness of information) and their incoming and outgoing relationships from and

[11] Wikipedia, `https://en.wikipedia.org/wiki/Connected_component_(graph_theory)`

to other objects. Merged objects will therefore be virtually removed from the graph as well as their incoming and outgoing relationships, while keeping a relationships *mergedBy* to the representative object. The construction of a representative object out of the merged objects as well as the distribution of relationships has implications that depend on the specific entity type and are described in the following.

Publications

The properties of a representative object for publications are built by selecting the most relevant object and then enriching its properties when these are missing and provided by other objects. Criteria of selection are the completeness of relevant metadata (e.g. title, authors, date) and the availability of PIDs (e.g. DOI).

For relationships, since some of the publication authors may be result of deduplication (point to an anchor person object) and others may be not (point to a raw person object), the strategy is to identify the merged object that is more complete in terms of relationships to anchor objects an include only its relationships in the representative object.

Persons

By introducing a Ground Truth we have two types of persons, *anchor* and *raw* persons, whose importance is different in terms of selection criteria. In this context a representative object is a new anchor object, which merges other objects of the two kinds. The properties of a representative object for persons are built by selecting the most relevant anchor object in the group of duplicates, if any, and then enriching its properties when these are missing and provided by other objects. Anchor objects with PIDs are certainly the most relevant and should be considered first, otherwise the presence of a first name and a second name as separate fields is the second level criteria.

As specified when introducing the Ground Truth, the relative methodology envisages that part of deduplication is resolved while creating the graph, by using the Ground Truth as a map between raw objects and the corresponding anchor objects, if any. This approach can be extended to any result of deduplication cycles. As we have seen, the application of a Ground Truth is generally followed by a number of cycles of candidate identification, candidate matching, duplicates grouping and representative object election. The resulting set of representative objects is a potential Ground Truth (if properly validated), but can certainly be injected at graph creation time as it happens for Ground Truth. The operation is consistent only if the information space is not altered between deduplication cycles and graph import and can avoid the action of propagating the relationships.

Organizations

The properties of a representative object for organizations are built by selecting the object with most complete metadata properties and then enriching the missing properties with that of other objects, if available. For relationships, all relationships of merged objects are inherited by the representative.

4

Big Graph entity deduplication requirements

In this chapter we analyse the functional and non-functional challenges surfaced in Chapter 3 in order to draw a list of requirements that any Big Graph Entity Deduplication System (BGEDSs) should meet. We organize requirements by the different phases of duplicate identification and graph disambiguation, but also introduce a broader category of requirements which regard the system seen as a combination of these two phases.

4.1 Duplicate identification requirements

Candidate matching requirements spin around the configurability of the similarity function. This function should be adaptable to any set of properties an entity type can include and embed several tools and primitives to set the logic of the match between two objects based on their common properties.

4.1.1 Candidate identification

CI.1 ***Clustering functions***

Clustering functions offer ways to generate a key out of the property values of an object. The system should integrate a number of standard and configurable functions the user may select from to set its candidate identification clustering strategy. At the same time, end-users must be put in the condition to integrate new proprietary functions, therefore devise their own strategy.

CI.2 ***Sliding window (Sorted Neighbourhood)***

As part of the clustering function configuration, end-users should be able to activate a sliding window mechanism. Once the sliding window is active end-users can specify: (i) the length of the sliding window, (ii) the hashing function used to generate from the objects the value according to which they will be sorted, and (iii) the sorting function to be used to order the objects based on such values. The system should embed a number

of hashing and sorting functions for end-user selection and allow the integration of other similar functions for end-user customization.

4.1.2 Candidate matching

CM.1 ***Weighted similarity measure***

The similarity between two objects is generally measured as a value in the range $[0 \dots 1]$. Since all properties of an object can contribute to the match and in different extents, the similarity function should allow (i) the definition of the set of matches to be performed in isolation and (ii) the assignment to each of them of a "weight", to be used to combine the scores of the individual matches and return the overall similarity between the two objects. For examples, in publication matching, "title" properties are more relevant than the "number of authors" values.

CM.2 ***Distance functions***

Matching property values is not an obvious process, especially when it comes to string values. Several string distance functions are known in the literature, each applicable to specific domain of strings. The system should include these functions and allow their usage in the definition of the similarity function. Moreover, specific cases may be handled using ad-hoc similarity functions, targeting specific value types and use cases. The system should therefore be extensible and allow end-users to plug-in their distance functions and use them in the definition of the similarity function.

A Comparison of String Distance Metrics for matching names was proposed in [18], and classifies the functions in:

- *Edit-distance like functions*
- *Token-based distance functions*
- *Hybrid distance functions*

CM.3 ***Manipulation of values***

In several cases property values must be "normalized" to a common domain (e.g. lower-case strings, removal of version numbers from titles, removal of English stop words) in order to enable matches that are as much as possible typo-independent and use-case dependent. Alternatively, values should be mapped onto derived values, which better encode the similarity of objects (e.g. counting the authors of a publication). The system should include a set of such accessory functions, to be easily introduced in the definition of the similarity function when needed, and allow for the introduction of new manipulation functions.

CM.4 ***Preconditions***

Similarity functions should include a pre-condition section, where a set of matches can quickly identify if two objects are duplicates (e.g. they share the same global PIDs)

or are not to be considered duplicates (e.g. they have different PIDs from the same PID scheme). If any of these conditions apply, then there is not need to calculate a similarity function as the result is immediately 0 or 1.

CM.5 ***Conditional statements***

The similarity function should include conditional statements, allowing to consider a set of matches instead of other set of matches based on a precondition, e.g. if “firstname” and “surname” are not empty, then use them for matching, otherwise use “fullname”. Conditional statements can also be used in the precondition section.

CM.6 ***Blacklisting and White-listing***

In order to curate and refine the results of the deduplication process, end-users should be able to explicitly provide assertions on similarity or diversity of object pairs, to be used in the precondition section (see Req. 4.1.2). In both cases, end-users may specify classes of assertions (criteria) or specific pairs of objects.

4.2 Graph disambiguation requirements

4.2.1 Duplicates grouping

DG.1 ***Efficiency and scalability***

Duplicates grouping is a time consuming operation, since it has to be executed over all objects of a given entity type in order to identify all possible connected components. Due to the number of objects, which can arbitrarily scale up over time, this operation cannot be simply executed with “brute force” algorithms, i.e. try all possibilities of navigation while marking already visited objects. Depending on the back-end used to store the graph this read-write action may be extremely slow and time-consuming. It is a strong requirement to identify algorithms that can efficiently and repeatedly execute this phase minimizing execution time. In scenarios such as person deduplication, where deduplication has to be executed in cycles until it converges, or publication deduplication, where deduplication has to be executed every time new publications enter the information space, waiting time may become a criteria to invest or not invest in deduplication for given entity types.

4.2.2 Duplicate merging

As highlighted in Chapter 3, duplicate merging has a twofold objective: (i) creating a representative object and (ii) distributing the relationships of the merged objects.

DM.1 ***Creation of representative object*** The creation of a representative object of a set of equivalent objects requires end-users to specify which strategy to adopt to assemble the object. The first action is that of selecting an initial object, the most relevant object,

from the set. This can be done by a function that sorts the objects, based on the value of their properties. The system can offer a set of such functions (e.g. sorting lexicographically the objects by their object IDs) but end-users can define, integrate and use their own (e.g. select object that has a global identifier property and richest metadata). The second action is that, starting from the relevant object, constructing the representative object. This requires (i) the creation of an identifier and (ii) the adoption of a metadata filling mode. Again the creation of an identifier can be by default by generating a random identifier, but end-users can integrate their functions (e.g. stateless identifier out of a global identifier property of the object). The metadata filling mode is of three kinds: keep relevant object properties only or for each property: (i) add if missing and exist in other objects or (ii) always merge other values of the same property of all other objects.

DM.2 ***Distribution of relationships***

Once the representative object is created, the end-user must specify which policy should be adopted in inheriting incoming and outgoing relationships of the merged objects. For each relationship, end-users can specify the following options: (i) inherit all relationships, (ii) inherit relationships of relevant object, (iii) inherit relationship of object(s) respecting predicate P over the properties of the object. End-users may specify other strategies, the specification of the selection function should return the set of objects from which relationships of the given type must be inherited. The approach does not differentiate between incoming and outgoing relationships.

DM.3 ***End-user feedback***

Once the representative objects relative to an entity type have been generated, end-users should be able to explore the result and apply manual corrections, such as (i) remove false duplicates from a group, (ii) add a duplicate to a group, and (iii) create a new group. Any assertion manually inserted by end-users should be kept and reapplied in further deduplication phases for the same entity.

4.3 General system requirements

4.3.1 Configuration and settings

CS.1 ***Deduplication configuration management***

The system should allow end-users to define, store and manage, for each entity type, multiple *deduplication configurations*, intended as combinations of (i) clustering function, (ii) similarity matching function, (iii) sliding window option and configuration.

CS.2 ***Graph disambiguation***

The system should allow users to define, store and manage, for each entity type, a set of *disambiguation configurations*, intended as combinations of (i) representative object generation options and (ii) relationship distribution options.

CS.3 ***Entity deduplication scores***

For each type, the system should allow to store *deduplication scores* intended as sequences of one ore more deduplication configurations over objects of the same entity type aiming at feeding a common phase of graph disambiguation, identified by a disambiguation configuration.

CS.4 ***Ground Truth*** The systems should support the option of Ground Truth over entity types. End-users enabling this option can manage a set of Ground Truth collections (GTCs) which can be either (i) imported by end-users via API or (ii) be created out of deduplication rounds of the system. End-users can, for each entity type, decide which GTC should be used to filter and clean the objects during the next graph import action.

CS.5 ***Dashboard User interfaces*** We assume that such a system should offer, where possible, a configuration Dashboard allowing end-user to configure all aspects of deduplication as depicted in this Chapter. Through the dashboard, end-users must also be able to execute and monitor the process of graph deduplication and apply their manual corrections (i.e. remove false duplicates) and additions (i.e. add missing duplicates).

4.3.2 Non-functional requirements

NF.1 ***Import of graphs*** The system must offer components to import Big Graphs according to standards such as RDF, JSON-LD, or similar ones. It should be possible to import graphs incrementally, possibly by entity type collection. It is generally intended that the graph to be imported is ready for deduplication, hence that all its objects contain enough contextual information to make deduplication decisions.

NF.2 ***Exports of graphs*** The system must offer components to export the Big Graph resulted from the last deduplication round according to standards such as RDF, JSON-LD, or similar. The exported graph contains the representative objects created after the last rounds of deduplication, the respective re-distributed relationships, but also the objects they merge, marked as “virtually deleted”, together with “mergedBy” relationships between them. Also the relationships relative to merged objects are present and marked as “virtually deleted”.

NF.3 ***Efficiency and scalability*** The system must be able to store Big Graph with in principle no limits of scalability. The same storage technology should therefore make it possible to execute all steps of candidate identification and matching, as well as all steps

of graph disambiguation. Indeed, due to the amount of data involved, it is recommended not to invest on an approach that adopts multiple storage systems for the different challenges.

5

Big Graph Entity Deduplication Systems

Figure 5.1 depicts the main functional areas of *Big Graph Entity Deduplication Systems (BGEDS)*, as derived from the relative requirements in Chapter 4. A BGEDS is intended to provide an end-to-end workflow supporting Big Graph data curators at: (i) importing their graph in the system, (ii) configuring for each entity type the relative duplicate identification scores, (iii) managing Ground Truth generation and injection, (iv) configuring graph disambiguation strategies, (v) supporting data curators at manually fixing the results of deduplication, and eventually (vi) exporting a disambiguated graph (i.e. devoid of duplicate nodes). In this chapter we will first describe the type and object language of BGEDSs, based on the Property Graph Model [73], then introduce the individual areas of the architecture and the relative functionalities.

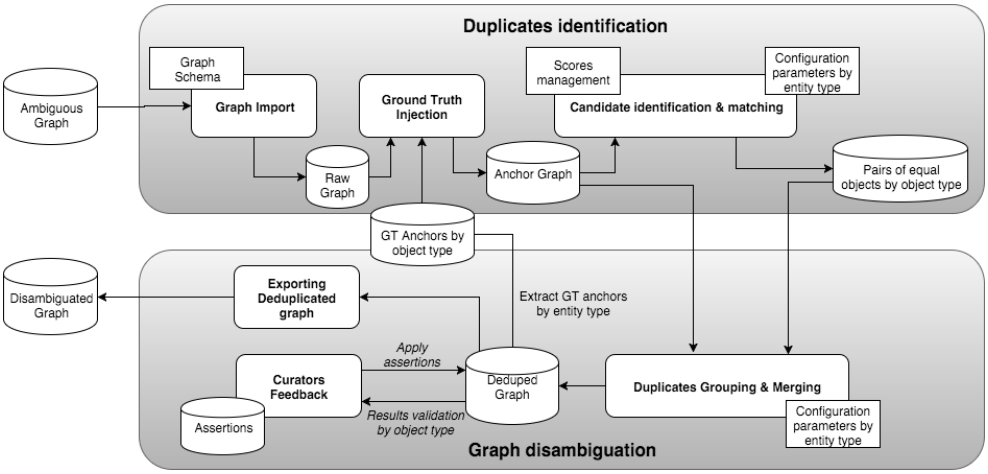


Fig. 5.1: BGEDSs: High-level Architecture

5.1 BGEDSs type language and object language

From the use-cases in Chapter 4 it is clear that a data model for BGEDSs should at minimum support the following requirements:

DM.1 ***Graph schema*** The deduplication workflow processes objects and relationships in the graph according to data curator configurations that assume the graph has an agreed-on statically provided schema structure.

DM.2 ***Structured types*** Objects may describe any real-world entity, hence their such types must be expressive enough to model any standard entity out there; we assume that entity types may be as expressive as XML schema [82], i.e. a tree of possibly nested, mandatory, repeatable properties.

DM.3 ***Relationship types*** Relationships must be directed, i.e. they indicate a source and a target object, and their semantics is expressed with a label.

A Graph Schema is indeed required to enable the configuration of the different phases of the deduplication process described in Figure 5.1. Functions, conditions, criteria, are specified by data curators based on the assumption that the underlying graph conforms to a given structured type. However, a graph data model underlying BGEDSs should also be driven by the kind of processing that needs to be executed on top of it. By observing the different phases of the deduplication workflow we can observe how these are upgrading the topology of the originally imported graph by adding new object and relationships:

- *Raw graph* ($\mathcal{G}_R = importGraph()$): it is the graph as imported by the data curators in the system: it matches a given graph schema and it is ready for processing; objects and relationships in this graph should never be updated or deleted as data curators always need to distinguish between the original graph and its subsequent deduplication-driven manipulations;
- *Anchored graph* ($\mathcal{G}_A = GTinjection(\mathcal{G}_R)$): it is the graph resulting from the Raw Graph after the injection of a Ground Truth GT for one or more of its entities; as such it adds to the Raw Graph the anchor objects, relationships to the merged objects, and new relationships outgoing and incoming the anchor objects as inherited from the objects they merge;
- *Deduped Graph* ($\mathcal{G}_D = entityDeduplication(\mathcal{G}_R, \mathcal{G}_A)$): it is the graph obtained from the *Anchor Graph* by applying candidate identification and graph disambiguation phases and taking into account the assertions from data curators; it reflects the structure of the Anchored Graph described above.

The BGEDS manages a graph database $\mathcal{G}_{db}$ that is updated by the steps above. $\mathcal{G}_{db}$ is instantiated with the original graph $\mathcal{G}_R$ and then updated to manage the graphs $\mathcal{G}_A$ and $\mathcal{G}_D$, together with their intermediate steps. The BGEDSs data model should be able to

model the manipulation of the $\mathcal{G}_{db}$ over the different deduplication phases, allowing, after the last phase, to view $\mathcal{G}_{db}$ from any of the perspectives above. Accordingly, the following requirement adds to the first two:

DM.4 ***Overlay graphs*** The objects and relationships of a graph always belong to one or more *overlay*, indicated with a label. As such an overlay graph is the sub-graph identified by objects and relationships with the corresponding label.

Among known models for graph representation, the *Property Graph Model* (PGM) [73] is not yet standardised, but has been described and adopted in a number of books, tutorials, and resources across the Web. One one of the most representative descriptions is the following by Robinson et al. [72], which describes a PGM graph as a set of vertexes and edges where:

- *Vertex*: an object that has a unique identifier, a label that denotes the its type, properties, and incoming and outgoing edges;
- *Edge*: An object that has properties, a label that denotes the type of relationship between its two vertexes, and a direction, i.e. tail vertex and head vertex;
- *Properties*: are a set of key/value pairs associated to a vertex.

PGM has become quite popular in graph databases implementations due to its expressiveness, which subsumes several and simpler forms of graphs types, e.g. removing edges directionality allows for representing undirected graphs. However, its model doe not match the modeling requirements **DM.1** (hence **DM.2** and **DM.3**) and **DM.4** above since it misses the notion of graph schema and does not provide any explicit tool to model overlay graphs. Figure 5.2 provides a visual example of a property graph which represents the relationships between persons and an article they authored. Unless imposed by the application context, PGM does not impose any dependency between the type of a vertex and its properties or the type of its incoming and outgoing edges.

For similar reasons PGM has been already enriched and specialized in the past. For example, the data model of Gradoop[1], a system for graph analytics, is based on an extension of PGM referred to as *Extended Property Graph Model* (EPGM) [39]. EPGM adds the further concept of *logical graphs* to the formalism, where the PGM graph can be further tagged with labels denoting logical graphs, hence subset of vertexes and edges bound together by some logic. The same vertices and edges may belong to different logical graphs.

[1] Gradoop - http://dbs.uni-leipzig.de/en/research/projects/gradoop

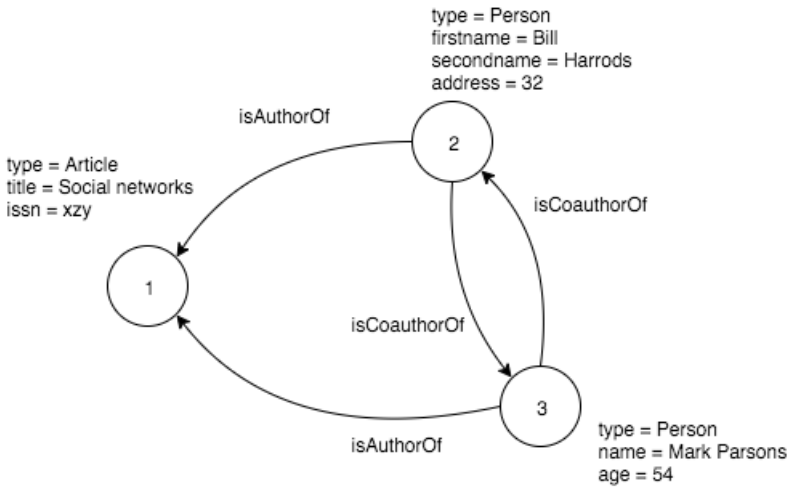


Fig. 5.2: Property Graph Example

Unfortunately, although EPGM matches the requirement **DM.4**, the notion of graph schema is still missing, as well as the possibility of describing a set of structured properties as indicated by **DM.2**. As a consequence, we propose an extension of the EPGM, which we shall refer to as the *Structured Property Graph Model* (SPGM).

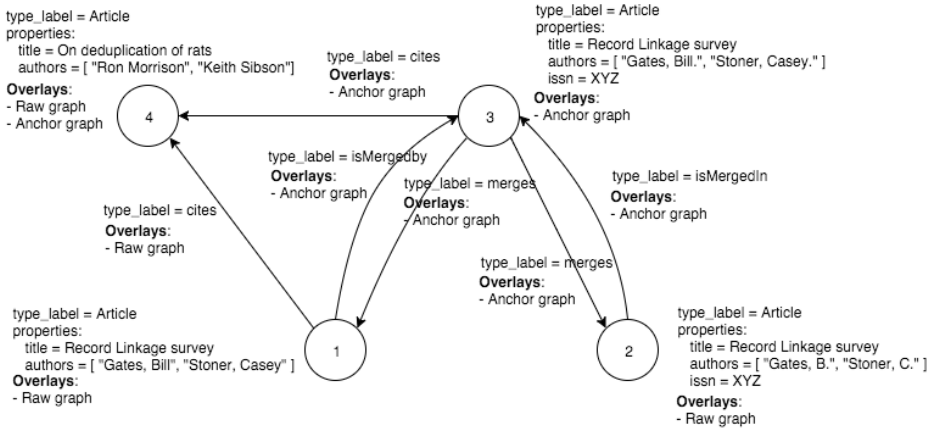


Fig. 5.3: Structured Property Graph Example

We can therefore summarize the SPGM as a EPMG graph endowed with the following notions:

- *Graph schema*: a set of assertions that constrain the name of a vertex type (i.e. a vertex label in PGM) with a specific set of structured properties and an edge type with a semantic label for each of the directions and a set of structured properties;
- *Structured properties*: instead of modelling the vertex properties as a set of key value pairs, where key is the attribute name and the values are primitive types (integer, float, string, ...), value can be themselves structured: another set of properties or a sequence of values;
- *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;
- *Edge*: An object that has a label that denotes the type of relationship between its two vertexes, a source and a target vertex, and one or more overlay labels.

The figure 5.3 provides a visual example of a SPGM graph instance with $\mathcal{G}_R$ and $\mathcal{G}_A$ overlays, describing the result of applying a Ground Truth for the “Article” entity type. Assuming the duplicate identification had generated a GT with an anchor object 3 merging 1 and 2, then the application of the Ground Truth to the Raw Graph has created a new object 3, created a new relationships $cites$ from 3 to 4, as inherited from obejct 1, and created the relationships $merges$ and $isMergedBy$ necessary to keep track of the merging action. The resulting graph can be observed from the point of view of the overlay $RawGraph$, to draw the initial topology of the graph, or the overlay $AnchorGraph$, to select the graph as resulting from Ground Truth injection.

The type language described in Table 5.1 is intended to serve data curators at specifying the structure of the graph in terms of object types, i.e. vertex types, and relationship types, i.e. edge types. The configuration of the different entity deduplication phases, as supported by BGEDSs, are based on annotations over the specific types, properties, and relationships defined in the schema.

T	$::= V_T = [l_1 : U_1, \ldots, l_k : U_k]$	*(entity type declaration)*
	$\mid \; rel[V_T, V_T, l_{st}, l_{ts}]$	*(relation type declaration)*
U	$::= S \mid S? \mid S+ \mid S*$	*(multiplicity and optionality)*
S	$::= [l_1 : U_1, \ldots, l_k : U_k]$	*(property set type)*
	$\mid$ int | string | bool	*(atomic types)*

Table 5.1: BGEDSs Type language

The clause $V_T = [l_1 : U_1, \ldots, l_k : U_k]$ declares a new type named V_T with properties l_i's of type U_i's. The type U may be mandatory (S), optional ($?$), sequence of at least one ($+$), or a possibly empty sequence ($*$) and take one of the following forms: a property set type or an atomic type. The clause $rel[V_{T_s}, V_{T_t}, l_{st}, l_{ts}]$ declares a bi-directional relationship between the entity types V_{T_s} (source) and V_{T_t} (target) whose semantic label from

source to target is l_{st} and from target to source is l_{ts}. For example the type of the SPGM graph in Figure 5.3 is illustrated in Listing 5.1, rows 1-2. Row 4 describes instead the type of an Article as described in Listing 5.2, where authors have a structured property form.

```
1 Article = [ title : string , authors: string+, issn: string?]
2 Rel[ Article , Article , cites , citedBy]
3
4 Article = [
5       title : string ,
6      doi: string?,
7      authors: [firstname: string , surname: string],
8      issn: string?
9 ]
```

Listing 5.1: Example of SPGM graph type for Articles

```
1  [
2     title = Tail Asymptotic Expansions for L−Statistics,
3    doi = 10.1234/xyz,
4    authors = [
5      [name = E., surname = Hashorva],
6      [name = C., surname = Ling],
7      [name = Z., surname = Peng]
8    ],
9    issn = 1234−5678,
10 ]
```

Listing 5.2: Structured properties: Article example

BGEDSs handle a graph database $\mathcal{G}_{db} \subset \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{R})$. Objects and relationships in $\mathcal{G}_{db}$ are managed by a number of definitions and operators described in Table 5.2 to refer to their properties or navigate through the graph, which will be useful in the following to describe the different phases. In Table 5.2, the domain of all objects is $\mathcal{O}$ and the domain of relationships is $\mathcal{R} = \mathcal{O} \times \mathcal{O} \times L$, where L is the domains of labels. Given an object $o \in \mathcal{O}$, $o : V_T$ denotes that o is of type V_T. The object values can be accessed using a standard "dot notation with wildcards": for example, with reference to the object o in Figure 5.2: $o.doi$ returns the value "$10.1234/xyz$", $o.authors[1].surname$ returns the value "$Hashorva$", $o.authors.*.surname$ returns the set of values {"$Hashorva$", "$Ling$", "$Peng$"}.

Table 5.3 defines all overlay graphs, represented as views of the graph database $\mathcal{G}_{db}$ that are produced during the different phases of entity deduplication.

$Overlay_o : L \to \mathcal{P}(\mathcal{O})$	sets of objects in $\mathcal{G}_{db}$ with overlay l
$Overlay_r : L \to \mathcal{P}(\mathcal{R})$	sets of relationships in $\mathcal{G}_{db}$ with overlay l
$Overlay : L \to \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{R})$	the graph in $\mathcal{G}_{db}$ with overlay l
$AddOverlay_o : \mathcal{P}(\mathcal{O}) \times L \to \perp$	adds the overlay l to the set of object O
$AddOverlay_r : \mathcal{P}(\mathcal{R}) \times L \to \perp$	adds the overlay l to the set of relationship R
$R_{in} : \mathcal{O} \to \mathcal{P}(\mathcal{R})$	set of relationships incoming an object
$R_{out} : \mathcal{O} \to \mathcal{P}(\mathcal{R})$	set of relationships outgoing an object
$AddObjects(O)$	adds a set of objects O to the graph
$AddRels(R)$	adds a set of relationship R to the graph

Table 5.2: BGEDSs object operators

$\mathcal{G}_R = Overlay(rawgraph)$	(Raw Graph)
$\mathcal{G}_A = Overlay(anchorgraph)$	(Anchor Graph)
$\mathcal{G}_E = Overlay(equivalencegraph)$	(Equivalence Graph)
$\mathcal{G}_F = Overlay(feedbackgraph)$	(Feedback Graph)
$\mathcal{G}_D = Overlay(dedupgraph)$	(Dedup Graph)

Table 5.3: BGEDSs : Entity Deduplication Graphs

5.2 Architecture overview

In this section we illustrate more in detail the functional architecture of BGEDSs by presenting the functional breakdown of the deduplication workflow it implements. More specifically, we shall describe the following phases:

- *Graph Import and Export* The import phase is responsible of importing a graph expressed according to known standards, such as RDFG and JSON-LD, into the BGEDS. The import is possible thanks to a mapping between the external graph schema and the internal graph schema. As we shall see, this mapping is expressed by mapping the RDF Trutle and JSON-LD schemas onto the SPGM type language, inspired by similar mappings onto the PGM model [73, 29]. This phase creates the Raw Graph $\mathcal{G}_R$ in the $\mathcal{G}_{db}$. The export phase can be fired by data curators to materialize the content of the $\mathcal{G}_{db}$ according to one or more of the overlays.
- *Ground Truth injection* This phase pre-processes the graph $\mathcal{G}_R$ by "injecting" a Ground Truth over each of the entity types V_T defined in the graph schema. Each type V_T has a set of potential Ground Truths $GT(V_T) = \{GT_{V_{T_1}}, \ldots, GT_{V_{T_k}}\}$ and data curators can select which one is to be adopted to generate a new graph $\mathcal{G}_A$.
- *Candidate identification and matching* This phase operates over the graph $\mathcal{G}_A$ to identify a set of pairs of equivalent objects. Data curators can configure this phase by selecting which *score* should be adopted for each of the entity types in place. A score is a sequence of *duplicate identification configurations*, each specifying a different identification strategy: clustering, clustering blocking, and matching functions. The graph $\mathcal{G}_A$ is therefore enriched with set of new relationships which belong to the *equivalence* overlay and form the Equivalence Graph $\mathcal{G}_E$.

- *Duplicate grouping and merging* This phase initially operates over the Equivalence Graph $\mathcal{G}_E$ in order to identify from pairs of equivalent objects the groups of equivalent objects, obtained by transitivity of the equality relationships. For each entity type, data curators can configure the disambiguation strategy to be adopted: how to generate a representative object out of each group, and how to distribute the relationships of the merged objects up to the representative object. The result of this processing phase is a Deduped Graph $\mathcal{G}_D$, obtained from $\mathcal{G}_A$ by including the representative objects and the relative relationships.
- *Data Curators Feedback* Deduplication techniques are based on heuristics and are therefore subject to a margin of error. For this reason, these techniques are typically supervised, allowing experienced users to refine the results and to be able to count on these corrections in subsequent applications of the process. Data curators are provided with tools with which they can search, browse and visualize the objects of any entity type, in order to "repair" representative objects that were not correctly created or create representative objects that were overlooked by the process. Such feedback results in a set of assertions for each entity type, i.e. assertions of equality between objects or difference between objects, which are always kept into account in subsequent runs. Data curators can, once they observe that the collection of objects of an entity type V_T is coherent enough, make it a Ground Truth for the entity type, i.e. add it to $GT(V_T)$.

In the following we shall describe more in details how the different phases interact in order to generate a $\mathcal{G}_{out}$ by updating the $\mathcal{G}_R$ according to data curator configurations.

5.2.1 Configuration Manager

Data curators interact with the BGEDS thanks to a *configuration manager*, depicted in Figure 5.4, that offers a global view of the deduplication workflow and the configuration parameters for all the different phases.

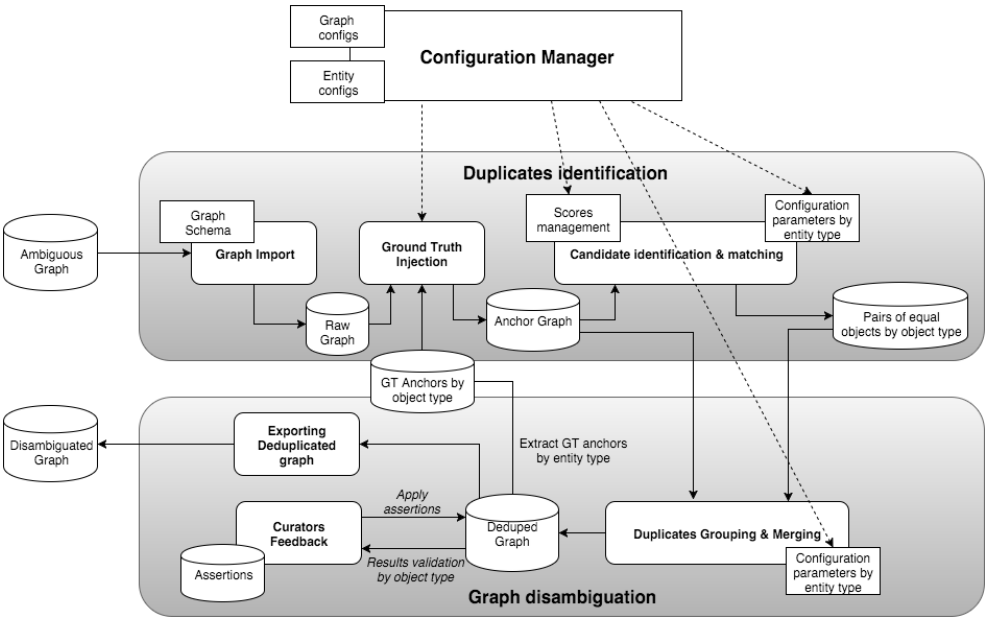


Fig. 5.4: GDup Configuration Manager

The configuration manager allows data curators to manage the following objects, whose correlations are depicted in Figure 5.5:

- *entity type deduplication configurations (entity config)*: an entity config has a unique name and consists of (i) a score for duplicates identification, (ii) a disambiguation strategy configuration, (iii) of a choice of the Ground Truth to be adopted for each entity type, and (iv) the option of applying data curators feedback;
- *graph entity deduplication configurations (graph config)*: a graph configuration has a unique name and consists one entity config for each entity to be deduplicated.

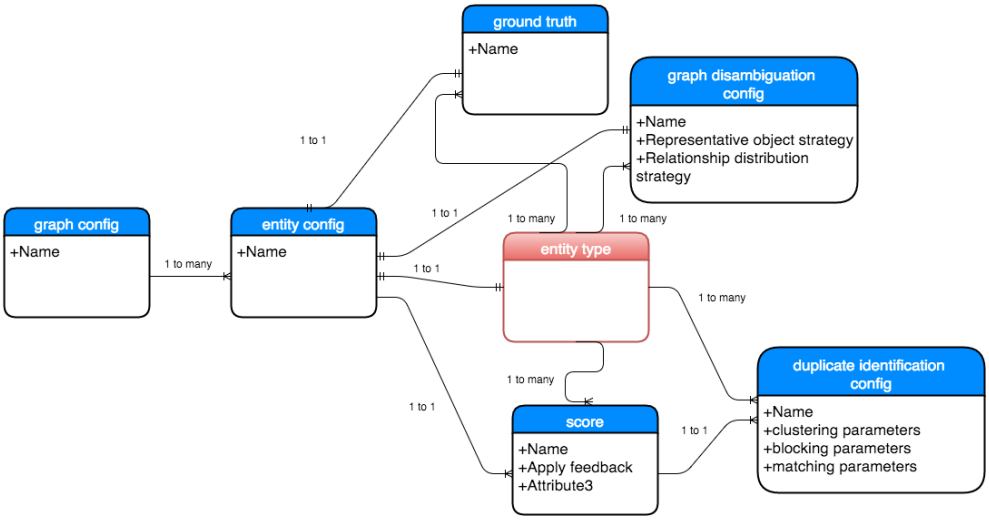


Fig. 5.5: Configuration Manager Data Model

5.2.2 Graph Import and Export

In the import data curators should provide the graph schema according to the specification given in section 5.1 and then import a graph conforming to that schema. Since graphs out there may be stored in any back-end and their graph-nature be described according to any schema language, to facilitate the import procedure the BGEDS must be able to generate a schema, using a standard graph schema language of preference (e.g. RDF Turtle, JSON-LD schema), that corresponds to the BGEDS graph schema. The system should then offer a component that is capable of importing a graph that corresponds to such standard schema and import it in a consistent and controlled way. This approach delegates data curators the issue of mapping their local representations of the graph onto a standard exchange format for graphs.

Any time, data curators can opt to export one or more of the overlay graphs in the graph database $\mathcal{G}_{db}$. 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 BGEDS at hand.

5.2.3 Ground Truth injection

Ground Truth injection is the entity deduplication phase where a Ground Truth for a given entity type is applied to the collection of objects in order to resolve part of the duplicates beforehand. A Ground Truth GT_{V_T} for an entity type V_T is typically a collection of *anchor objects* whose structure is V_T and includes the list of *raw objects* merged by the anchor object:

- *Raw object*: it is a copy of the original object of the graph $\mathcal{G}_R$, hence has type V_T and the same identifier;
- *Anchor object*: it is an object representing a set of merged raw objects; as such it is like a representative object in the graph $\mathcal{G}_D$ whose quality has been somehow validated based on some criteria of quality, endorsed by data curators. Anchors have a Type V_T, a *unique identifier*, and includes the list of raw objects it merges.

The introduction of a Ground Truth approach changes the perspective of the deduplication process, since part of the deduplication can be made available before candidate identification by mapping raw objects onto anchor objects at graph import time. This preprocessing phase is useful for two main reasons: reducing the computational time of entity deduplication as a whole – by limiting the identification of duplicates to the objects that are not yet merged by the Ground Truth – and resolve duplicates based on a Ground Truth knowledge, typically curated and validated by experts, rather than leaving it to a mechanical process, which may be prone to errors. Indeed, Ground Truth information supporting record linkage processes often requires a lot of manual or ad-hoc efforts to be invested in its construction. Data curators may start from the result of a deduplication run for a given entity type in the graph $\mathcal{G}_D$ and then manually curate the entity type object collection until a given quality is reached. Alternatively, a GT_{V_T} may be imported from the file system, in a process similar to the import phase described above. In this case, any external process can be adopted to construct an high-quality GT_{V_T}: for example to solve the problem of person objects with no co-authors special techniques must be devised, capable of identifying co-authors from other sources on the Web, as well as platforms supporting the results validation.

```
Anchor[
 (id = 001)
 properties = [name=Alan, surname=Turing]
 merged = [
  Person[
   (id = 002)
   properties = [name=Alan, surname=Turing]
  ],
  Person[
   (id = 003)
   properties = [name=Alan Mathison, surname=Turing]
  ]
 ]
]
```

Listing 5.3: GT example - Anchor

An example of a Ground Truth entry, describing an anchor person object of type $Person = [name : string, surname = string]$), merging two distinct raw person objects is illustrated in Listing 5.3. The anchor object explicitly defines an entry of a map

between the anchor person with identifier 001 and the raw persons with identifier 002 and 003.

Specifically, a ground truth GT_{V_T} is the set of anchor objects, with the functions:

- $Merged : \mathcal{P}(\mathcal{O}) \times \mathcal{O} \rightarrow \mathcal{P}(\mathcal{O})$: $Merged$ is a function that takes a GT_{V_T} and an anchor object o and returns the set $Merged(o)$ of objects merged by o.
- $Anchor : \mathcal{P}(\mathcal{O}) \times \mathcal{O} \rightarrow \mathcal{O}$: $Merged$ is a function that takes a GT_{V_T} and an object o and returns the anchor object $anchor(o)$ that merges it.

Creation of Anchor Graph

The system supports the management of multiple Ground Truths for the same type V_T. The injection phase takes as input the graph $\mathcal{G}_R$ and a set of G_{V_T}, one for each of the V_T for which a Ground Truth has been selected in the configuration manager. The phase generates a graph $\mathcal{G}_A$ by upgrading the graph $\mathcal{G}_{db}$ according to the pseudo-algorithm described in Listing 5.4, which applies to each GT_{V_T}.

```
1  // adds anchor objects to the graph
2  AddObjects(GT_{V_T})
3
4  // add the overlay anchorGraph to anchor objects in the graph
5  AddOverlay_o(GT_{V_T},' anchorgraph')
6
7  foreach o_a ∈ GT_{V_T} do {
8      MergedObjects = +Merges(o_a)
9  }
10
11 // add the overlay anchorgraph to all objects that are not merged by anchor objects
12 AddOverlay_o(Overlay_o('rawgraph')/MergedObjects,' anchorgraph')
13 foreach o ∈ MergedObjects do {
14     MergedRels = +R_out(o) ∪ R_in(o)
15     foreach < o, o', l >∈ R_out(o) do {
16         AddRels(< o, anchor(o'), l >)
17         AddOverlay_r(< o, anchor(o'), l >,' anchorgraph')
18     }
19     foreach < o', o, l >∈ R_in(o) do {
20         AddRels(< anchor(o'), o, l >)
21         AddOverlay_r(< anchor(o'), o, l >,' anchorgraph')
22     }
23     // adds the overlay anchorGraph to all rels that are not touched by anchor objects
24     AddOverlay_r(Overlay_r('rawgraph')/MergedRels,' anchorgraph')
25 }
```

Listing 5.4: Anchor graph generation pseudo-algorithm

5.2.4 Duplicates identification

One of the main features available in BGEDSs is the possibility to process semantically expressive graphs, providing different entity types linked by relationships of different semantics, responding to a given BGEDS graph schema. Such graphs might be resulting from aggregation processes that integrated heterogeneous information, hence they have high probability to be affected by duplication of information. In order to satisfy the duplicates identification requirements depicted in section 4.1, the system supports management of multiple deduplication configurations, organised by entity type.

For each entity type defined in a given BGEDS graph type, data curators can define and combine multiple configurations of duplicate identification, in order to identify duplicates based in different strategies. A *duplicate identification configuration* is strictly bound an entity type schema V_T for which it defines the parameters of the candidate identification, i.e. a clustering function and a pair blocking-sorting functions, and candidate matching, i.e. a matching function. A *score* is a sequence of candidate identification configurations followed by candidate matching configuration.

For example the configuration manager might hold one configuration score for a given type of entities, whose deduplication results reached a certain degree of acceptance from the data curator. Likewise it can manage a number of configurations that are still being tested, and whose results evaluated for a different entity type of the same graph, or that refer to another graph.

```
1
2  <dedup_score> ::= <V_T> "," <conf> { "," <conf> } "," <merge> "," <rel>
3  <conf>        ::= <ctrlStm> | <condStm> "," <simStm> | <simStm>
4  <ctrlStm>     ::= "IF" <condStm> "THEN" <simStm> "ELSE" <simStm> "ENDIF"
5  <condStm>     ::= "condition [" <b_expr> "]"
6  <b_expr>      ::= "not" <expr> | <expr> [ <relOp> <expr> ]
7  <expr>        ::= ["("] condFunction f(p_{V_T}) → (true|false) [")"]
8  <relOp>       ::= and | or
9  <simStm>      ::= <simF> { "," <simF> }
10 <simF>        ::= f(p_{V_T}) → [0...1] //similarity function
11 <merge>       ::= f(G_{V_T}) → R_{V_T} //attribute merge function
12 <rel>         ::= f(E_ij)              // relationships fixup
```

Listing 5.5: BGEDS configuration grammar

Candidate identification

Deduplication systems often provide different techniques to reduce the search space required by the unpractical quadratic complexity deriving by the pairwise comparisons between the objects. BGEDSs do no exception, and to satisfy requirement 4.1.1 provide a number of predefined functions that allows to select the objects eligible for detailed matching by grouping them into blocks (or *canopies* [62]). Each block will contain all the objects that are much likely to produce a match, thus excluding the need to compute a

large part of true negative matches. To motivate the need to reduce the search space we can think about a naive approach that computes all the pairwise comparisons $|P|^2$ for 16 million publications in OpenAIRE. Assuming each comparison is $1\mu s$, it would take about 8.1 years to perform all the 2.56×10^{14} comparisons. On the other hand, assuming to group publications in blocks by some of their features, where the blocks maximum size is, say 50×10^3, it would take 41.6 minutes to compute all the 2.5×10^9 comparisons. The main challenges in the blocking technique are (i) the identification of the most relevant features in the data at hand, and (ii) the proper functions to extract *hash keys*. In hash based blocking techniques each block B_i is associated with an hash key h_i, such that the property p_i is hashed to the block B_i if $hash(p_i) = B_i$, then all the objects within the same block are compared with each other. Each block is independent from each other, allowing to further reduce the execution time by making the candidate matching phase to be performed by parallel processes. On the other hand, as a certain number of objects will never be compared as they will be included in disjoint blocks it is possible that the overall process recall might be affected negatively, depending on the clustering functions and the data at hand. For this reason the definition of the clustering functions is a crucial aspect that require knowledge on the application domain.

Blocks are defined by grouping the objects responding to a given Entity Type V_T with dedicated *clustering functions*. Clustering functions are characterized by a Vertex Type V_T (identifying the objects they will be applied to), a set of specific parameters, and a list of field values, picked from the given Entity Type instance.

```
1  EntityType = Article ,
2  clusteringFunction[
3   name = ngrampairs
4   params = [max=1, ngramLen=3]
5    fields  = [ title ]
6  ],
7  clusteringFunction[
8   name = suffixprefix
9   params = [max=1, len=3]
10   fields  = [ title ,  authors.surname ]
11 ]
```

Listing 5.6: GDup multiple clustering functions

As shown in listing 5.6, the BGEDS configuration allows to define multiple hash functions within the same configuration score, causing the same object to be included in more than one block (or making the blocks to overlap). Each hash function can receive as input one or more of the object property values defined in a given Entity Type V_T, and return one or more hash keys, depending on the function specific parameters.

Blocks created by the clustering functions might still contain a large number of objects, making the candidate matching phase unpractical. To overcome this issue the configuration allows to further reduce the number of comparisons performed by the candidate matching phase making use of the so called Sorted Neighbourhood Method (SNM),

which can be activated depending on the needs (see requirement 4.1.1). Such method assumes to sort the objects within each block so that similar objects are close to each other, then only compare objects within a small neighborhood, or window of size w that is moved sequentially over the sorted block. Sorting is based on a specific key, which can be obtained from the object properties using the hashing functions available in the system.

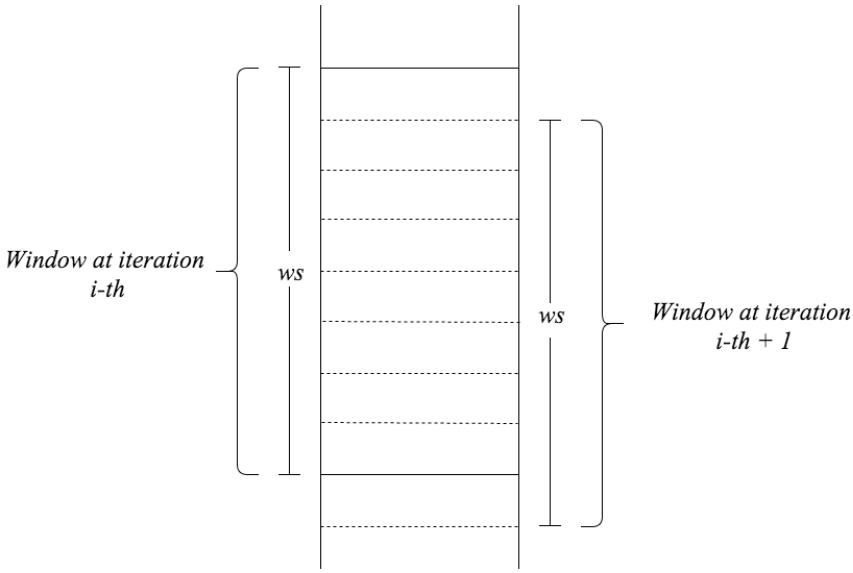


Fig. 5.6: GDup : Sorted Neighbourhood Method

The use of the sliding window limits the number of possible object pair comparisons for each object to $2w - 1$, and the resulting total number of object pair comparisons (assuming a graph with n objects of a given Entity Type) is $O(wn)$. If we consider also the cost of sorting the blocks, the total complexity is $O(wn + n \log n)$.

Depending on the data and the configuration it might happen that the clustering functions produce relatively small blocks. In such cases where the average block size is similar to the sliding window size w, users can configure the system to skip the sorting phase, as all the objects within each block will be compared with each other.

Candidate matching

Similarity function Candidate matching is the phase in the deduplication workflow that actually performs the comparison between object pairs. The object matching operation is defined as the computation of a similarity measure s between two objects. Analogously, distance functions generally map a pair of strings to a real number d, where smaller

values of d indicates greater similarity between the two input strings. Since GDup makes use of both similarity and distance functions, in order to provide a uniform response all the return values are mapped in the range $[0 \dots 1]$, where higher values indicates greater similarity. A match between a pair of objects is considered positive when the score obtained by a given similarity function reaches a given configurable threshold T, event that makes the system to assert the equality between the two objects.

In order to meet the requirement described in 4.1.2, as the properties of an object can contribute to the match in different extents, the system includes in the configuration the possibility to associate weights to each property associated to a similarity function. This makes the most significant properties to be more relevant in driving the matching process. For example, when matching publications, the *title* can be considered as more relevant than the *publisher*. Each similarity function f_i is characterized by a name, is associated to a given data type V_T (might it be an Atomic type, or a Structured type), takes as input a pair of the same property values (or a manipulation of the same) $o.l, o'.l$ from the two objects o, o', and returns the similarity measure $f_i(o, o')$ between the two properties, normalized in the range $[0 \dots 1]$.

Overall, given the 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 form the different n similarity functions defined in the configuration.

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

Where $\sum_{i=1}^{n} w_i = 1$ and $0 <= fi <= 1$ are respectively the weights and the similarity functions w.r.t. to each object property l_i, and exp is the expression needed to de-reference the value from the structured properties of the o and o'.

```
similarityFunction [
    name = LevenshteinDistance
    weight = 0.8
     fields  = [ title ]
],
similarityFunction [
    name = JaroWinklerDistance
    weight = 0.2
     fields  = [publisher]
]
```

Listing 5.7: GDup matching functions

One of the most important capabilities of a general-purpose BGEDS is the possibility to adapt to specific application domains. Therefore the system must (i) support a

predefined set of general purpose and established similarity functions, and (ii) a mechanism to easily include new ones. Special cases of similarity functions can be defined as:

- multi-field: functions that accept structured properties as input; The function will be responsible for the value deserialization, and the internal decision process might benefit from a wider range of information;
- transformed-field: functions that may impact on the selected f_i based on a manipulation of the values of the two inputs to be compared; for example, a function that recognize different variations of a text value by applying a regular expression, or a function that removes the stop words from a text field etc.

BGEDSs provide a predefined set of generic functions typically used in text analysis. The selection of such functions was inspired by the work of Cohen, Ravikumar, and Fienberg in [17], as well as [21], whose evaluation allowed us to define a set of general purpose methods to calculate the similarity between a wide range of application specify cases.

- *Edit distance*: Given a pair of strings a and b on a defined alphabet Σ (e.g. in text it might correspond to the set of ASCII characters), the edit distance $d(a, b)$ is the minimum weight series of edit operations needed to transforms a into b. A simple sets of edit operations was defined by Levenshtein in 1966 [55]: insertion, deletion, substitution of a symbol, each operation associated to a unit cost (except that substitution of a character by itself has zero cost);
- *Jaro distance*: Widely used in record linkage literature, this method is based on the number and order of the common characters between two strings.
- *Jaro-Winkler*: A variation of the Jaro method, which considers also the length of the longest common prefix between the two input strings, typically used to match short strings (e.g. , personal first or last names.).
- *Jaccard similarity*: Among token based similarity functions, that considers the strings S and T as bags of words, the Jaccard similarity is simply expressed as the size of the intersection divided by the size of the union of the same sets $\frac{|S \cap T|}{|S \cup T|}$;

Value manipulation Although well known string matching functions represents a powerful method to determine that two objects are manifestations of the same real world entity, several application domains require to introduce customized constructs to support the decision process. Value manipulation is one of the basic functionalities needed to perform clean-up operations, often needed when dealing with highly noisy data. For example, in case of date values typically users might need to define dedicated cleaning or value extraction functions to compensate different cases of ambiguity, lack of information, and normalization, as it is common to deal with different date formats, or the lack of a day-

grain date, compensated using a value of 01-01-YYYY. Other cases of value manipulation functions supporting the data cleaning are:

- Removing noisy information from text fields;
- Identification of numbers (Arabic or Roman) in text;
- Mapping different representations of characters and symbols, e.g. superscript and underscript to be mapped into ”normal” text;
- Removal of punctuation;
- Normalization of diacritical marks;
- Text encoding normalization.

```
1 similarityFunction [
2     name = LevenshteinDistance
3     weight = 1.0
4     fields  = [titleCleanup( title )]
5 ]
```

Listing 5.8: GDup data cleaning functions

Listing 5.8 describes an example of cleaning functions available to BGEDSs. The property values `result.title` are processed for cleanup by the custom function `titleCleanup` before being used to calculate their similarity with `LevenshteinDistance`. A user-friendly BGEDS, while defining a matching functions, data curators should select the properties to be matched and be promoted with a list of “compatibile” similarity functions, pre-selected based on the type of the selected property.

Pre-conditions In order to enable users to define domain specific decision processes, BGEDSs support the definition of *pre-conditions* (requirement 4.1.2). A group of pre-condition is an optional set of statements defined in GDup configurations that is evaluated before the similarity functions. Each condition is therefore a function from two objects belonging to a given type V_T to a Boolean value b. Overall the return values from the set of conditions can be evaluated using the disjunction operator $\vee$, or the conjunction operator $\wedge$, allowing to express arbitrary domain-specific conditions. When the evaluation of the pre-condition set is $true$, then the system skips the evaluation of the similarity functions, and considers the similarity between the two objects as 1.

```
1  // for Article entity type
2  condition[
3      exactMatch(doi) or exactMatch(PMID)
4  ]
5
6  // for Person entity type
7  condition[
8      exactMatch(email) or
9      (exactMatch(birthdate) and exactMatch(firstname) and exactMatch(surname))
10 ]
```

Listing 5.9: Pre-condition example

The pre-condition evaluation is typically faster than the similarity functions as pre-conditions are intended to identify domain specific cases, and can quickly determine if two objects A, B are duplicates or not. For example, the listing 5.9 shows two different cases where a combination of multiple equivalence assertions can help in the identifying the duplicates. In the first case two publications could be considered as duplicates when they share either a DOI, or a PMID [2] The second case represents an expression of conditions on Person objects, where the most relevant attribute driving the Boolean expression is the email, alternatively in order to satisfy the condition, all the attributes birth date, first name, and surname must match.

Conditional statements Another important aspect provided by the configuration is the possibility to express conditional statements. Conditional statements allows the system to consider a set of matching behavior instead of another depending on conditions to be expressed over the object properties or the evaluation of a function over the object properties. A similar approach was explored by Hall at al.in [28], where is presented a method to model dependencies among object properties.

[2] PubMed Unique Identifier - `https://en.wikipedia.org/wiki/PubMed#PubMed_identifier`

```
IF condition[ isNotNull(firstname) and isNotNull(surname) ] THEN
    similarityFunction [
        name = JaroWinkler
        weight = 0.75
        fields = [surname]
    ],
    similarityFunction [
        name = JaroWinkler
        weight = 0.25
        fields = [firstname]
    ]
ELSE
    similarityFunction [
        name = LevenshteinDistance
        weight = 1.0
        fields = [fullname]
    ]
ENDIF
```

Listing 5.10: GDup conditional statement example

White-listing and blacklisting In order to produce accurate results and allow to customize the system to catch domain specific aspects, end-users can configure assertions based on object properties. Assertions can be defined to exclude records from the candidate identification phase, or using conditional statements provided by GDup configurations, which can include property-driven conditions to enable blacklisting or white-listing of groups of objects. In reference to the person objects depicted in figure 5.12, listing 5.11 illustrates a possible way to define rules that configures the system to consider or not a group of objects as duplicates.

```
condition[
    exactMatch(surname) and
    regexMatch(name, "B.*") and
    regexMatch(birthdate, ".*1983")
]

condition[
    not (
        exactMatch(surname) and
        regexMatch(name, "B.*") and
        regexMatch(birthdate, ".*1983") )
]
```

Listing 5.11: GDup blacklist / whitelist example

Creation of Equivalence Graph

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 actions in Listing 5.12 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 $equivalenceGraph$.

```
1 AddRels({< o, o', equalTo >, < o', o, equalTo >})
2 AddOverlay_r({< o, o', equalTo >, < o', o, equalTo >}, equivalenceGraph)
3 AddOverlay_o({o, o'}, equivalencegraph)
```

Listing 5.12: Populating the Equivalence Graph

Moreover, in order to take into account the *equivalence assertions* provided by data curator feedback, $\mathcal{G}_E$ is further enriched with new edges inherited from the Feedback Graph $\mathcal{G}_F$. As explained in Section 5.2.6 $\mathcal{G}_F$ includes relationships of two kinds $equalTo$ and $differentFrom$. The duplicate identification phase concludes by adding the $equalTo$ relationships in $\mathcal{G}_F$ to the equivalence graph $\mathcal{G}_E$, as explained in Listing 5.13

```
1 equalToRels = {< o, o', equalTo >∈ G_F}
2 AddRels({< o, o', equalTo >})
3 AddOverlay_r({< o, o', equalTo >, < o', o, equalTo >}, equivalenceGraph)
4 AddOverlay_o({o, o'}, equivalencegraph)
```

Listing 5.13: Populating the Equivalence Graph

5.2.5 Graph disambiguation

The Equivalence Graph populated by the phase of duplicate identification contains now the set of pair-wise equivalences identified by the deduplication strategy provided by data curators and by previous manual feedback of data curators. Figure 5.7 illustrates the overlay graph created by the equality edges introduced by the candidates matching phase, where the equivalence relationship between A and B was provided by data curators.

Graph disambiguation consists of two distinct phases. The first phase is duplicate grouping and is in charge of identifying all connected components in the 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.

Duplicate grouping

The challenge of duplicates grouping derive based on the intuition that a relation of equivalence is transitive. For example, with respect to Figure 5.7, since $A = B$ and $B = C$

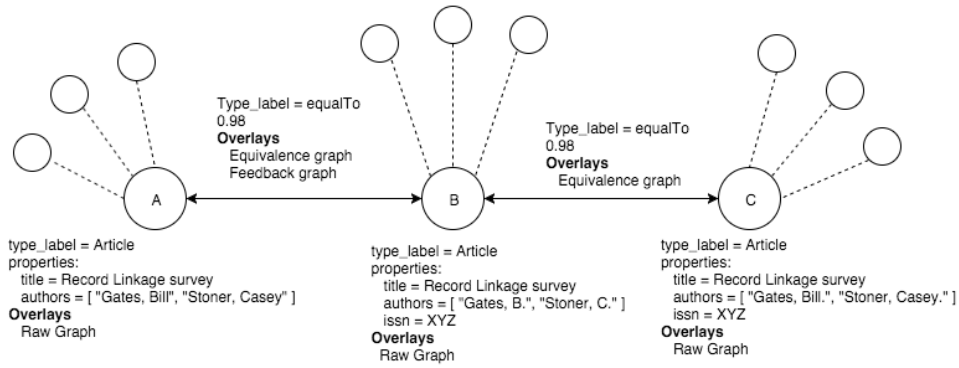


Fig. 5.7: Graph Equality relationships

then we can also conclude that $A = C$. As a consequence, graph disambiguation cannot be based on $\mathcal{G}_E$ alone, i.e. by merging pairs of equivalent objects, but should rather based on groups of equivalent objects. With respect to the example above, if no other object in $\mathcal{G}_E$ is reached via a relationship $equalTo$ from A, B, or C, then the group of objects $\{A, B, C\}$ is a connected component in $\mathcal{G}_E$ and represents a set of equivalent objects, ready to be merged into one object.

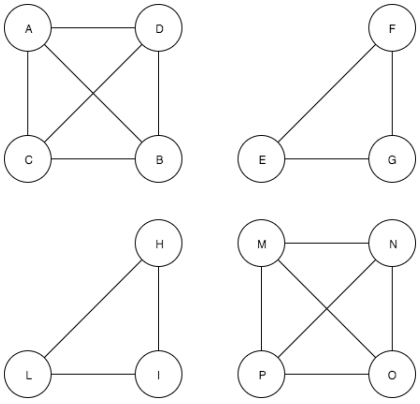


Fig. 5.8: Groups of duplicated objects, graph full mesh

Grouping is required because $\mathcal{G}_E$ is potentially incomplete. On the one hand, duplicate identification adopts heuristics that may generate false negatives, i.e. overlook pairs of equivalent objects. Indeed, in an ideal world where deduplication systems would be able to process duplicate identification in a graph of n objects by computing all the possible n^2 matches in a relatively short amount of time, the resulting overlay graph composed by the

equivalence relationships would correspond to a set of full meshes corresponding to each duplicates group. The figure 5.8 represents an example of four groups (or clusters) of duplicates. However, since the heuristic methods described in paragraph 5.2.4 and 5.2.4 aims at reducing the duplicates search space and therefore the number of matching operations, the overlay graph resulting from the overall duplicates identification phase will be a set of connected components, each corresponding to a group of duplicates. On the other hand, matching strategies or available object properties may not be enough to identify the equivalence of two objects. Only data curators in this case may claim the proper relationships and feed it to the system. A grouping phase distributes equivalence relationships until all its connected components are found.

The problem of connected components has well established linear time solutions (with respect to the numbers of the objects and edges of the graph) using either breadth-first search (BFS) or depth-first search (DFS). The main idea in BFS search algorithm can be described as:

- Start at any source object o and visit,
- All objects at distance $d = 1$,
- Followed by all objects at distance $d = 2$,
- Followed by all objects at distance $d = 3$,
- ...

BFS corresponds to computing *shortest path* distance (number of edges) from o to all other objects. Typically, to control progress of BFS and avoid infinite loops, the relative algorithms "color" the object to mark their visited status:

- *White* before we start;
- *Gray* after we visit the object but before we have visited all its adjacent objects;
- *Black* after we have visited the object and all its adjacent objects (all adjacent objects are gray).

A simple connected components algorithm CCbyBFS is illustrated in Listing 5.14, which computes BFS and calculates individual connected components in $O(|\mathcal{O}| + |Overlay_r(equivalenceGraph)|)$ time. For all objects in the $\mathcal{G}_E$ the algorithm computes the corresponding connected component CC by following a BFS search and then adds it to the set of all connected components $\mathcal{C}$. The process marks the objects it has visited to avoid the same connected component is calculated again in further algorithm rounds and uses a queue Q to remember about all gray objects the algorithm encounters but are still not done with.

```
1  foreach o ∈ 𝒪 do {
2      if color(o) = white then
3          𝒞 = 𝒞 ∪ CCbyBFS(o)
4  }
5
6  CCbyBFS(s) {
7      color[s] = gray
8      CC = {s}
9      ENQUEUE(Q, s)
10     while Q is not empty do {
11         DEQUEUE(Q, o)
12         foreach < o, o', equalTo > ∈ Overlay_r(equivalenceGraph) do {
13             if color[o'] = white then {
14                 color[o'] = gray
15                 CC = CC ∪ {o'}
16                 ENQUEUE(Q, o')
17             }
18             color[o] = black
19         }
20     }
21     return CC
22 }
```

Listing 5.14: BFS Algorithm

After the execution of the algorithm $\mathcal{C}$ contains all connected components in the graph according to the $equalTo$ relationships, namely groups of duplicate objects. Before such groups are passed over to the next step of graph disambiguation, which concerns merging of duplicates, they first need to be validated with respect to the diversity assertions provided by data curators in the feedback phase. Such assertions are provided in the form of relationships $< o, o', differentFrom > \in \mathcal{G}_F$. For each of such triples, all connected components $CC \in \mathcal{C}$ such that $\{o, o'\} \subseteq CC$ shold be removed from $\mathcal{C}$. The intuition is that a relation of difference between two objects of a group has the side effect of transitively propagating to all objects the group.

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 should build a *representative object*, elected to literally represent and replace in the graph all duplicates in the group. This new graph takes for under a different overlay, called $dedupgraph$. Two issues must be tackled: (i) election of a representative object for the group of duplicates and (ii) distribution strategy of the relationships from the objects in the group to the representative object.

Representative object election As we state in this work the purpose of a deduplication system, except for the obvious duplicates (or ambiguity) removal, lies in addressing

a wider spectrum of data quality issues. In fact real world data, other than affected by duplication of information, is often characterized by missing values, issue that becomes more prominent in results of data integration tasks. To tackle such kind of issues, the system exploits the information included in members of a duplicates group to build a representative object characterized by higher quality. The basic idea is that of driving the property fusion process, to obtain a richer representative object. Important aspects involved in the process are (i) generating an identifier for the representative object, and (ii) merging strategy of the duplicate objects' properties.

Starting from the results of our BFS algorithm, the system allows users to define the strategy used to assemble the representative object. Such strategy can be set in the score configuration by picking a predefined or user-defined function from a list. Such functions can be grouped in two categories that drives the system on how to proceed with the the graph disambiguation phase:

- *Pick one:* in this case the system will choose one of the objects in each connected component according to a selection criteria defined by the user in the configuration score. This can be done by a function that sorts the objects, based on the value of objects properties. The system offers a set of such functions (e.g. sorting lexicographically the objects by their object IDs) but end-users can define, integrate and use their own (e.g. select object that has a global identifier property and richest metadata). Once the objects are sorted, the system picks the head of the list, which will become representative for the others, in this case the system will not introduce new objects in the graph.
- *Add new:* The second action is that constructing the representative object, starting from all the objects in each connected component. This requires (i) the creation of an identifier and (ii) the adoption of a metadata filling strategy. Again the creation of an identifier can be by default by generating a random identifier (at the cost of a loss of reproducibility), and also in this case end-users can integrate customized functions (e.g. stateless identifier out of a global identifier property of the object). The objects metadata filling mode can be configured according to different strategies:
  - *Union:* the representative object metadata will contain the union of all the duplicated objects metadata; this approach allows to not lose any information, delegating to the whom will consume the disambiguated graph (service or human) the burden to select the needed information.
  - *Fill by order:* duplicated objects contributing to the representative construction can be sorted according to user-defined functions (e.g. by assigning a priority to the duplicates provenance), and merging the attributes according to a given ordering; This approach allows to "fill the holes" of missing metadata, leading to a general improvement in the model coverage in the disambiguated graph;
  - *Selective:* only select the attributes according to a given user-defined predicate.

Distribution of relationships The other aspect involved in the graph disambiguation process is related to the relationship distribution strategy. As the disambiguation process takes actions that alter the graph structure (i.e. introduction of new representative objects and virtual deletion of merged objects), it is important to consider the effect of such actions on the existing edges, i.e. how the graph disambiguation will affect the relationships between the entities in the graph, and in general its semantic expressivity.

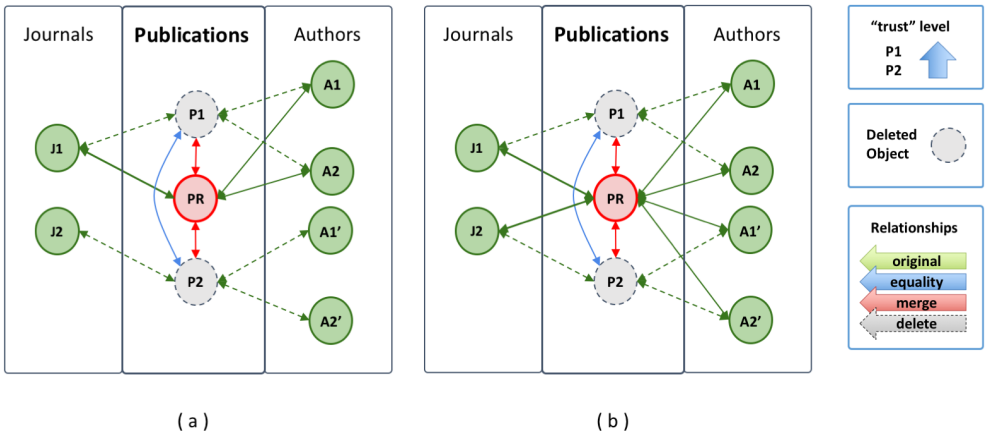


Fig. 5.9: Graph disambiguation example

Figure 5.9 illustrates two cases where a graph representing the publications P_1 and P_2, related to their respective authors A_1, A_2, $A_1^{'}$, $A_1^{'}$, and provided respectively by journals J_1 and J_2, are identified as duplicates and merged into the publication P_R. In case *(a)*, P_1 and P_2 are identified as duplicates, and the system was configured to promote the relationships that originally occurred between P_1 and J_1, and between P_1 and A_1, A_2. On the other hand, the case *(b)* shows the result of a different kind of action taken over the graph. In that case the system was configured to preserve all the relationships, making the representative publication to become an hub.

Overall the system allows users to specify the desired policy for the links management as part of the deduplication score. The user interfaces allows to pick one policy, implemented by dedicated functions and applied as part of the graph disambiguation algorithm. The system provides the following policies:

- *Merge All*: this policy assumes that all the ingoing and outgoing edges occurring between a deleted object and its neighbours gets redirected to the representative object. This allows to reduce to the minimum the information loss derived by the actions performed on the graph, however depending on the case it could introduce further ambiguity. For example, with respect to the figure 5.9 *(b)* the publication P_R will appear as collected from both Journals J_1 and J_2 (fact that could represent an added

value from the end user perspective), but will inherit all the authors of the duplicated publications P_1 and P_2, fact that represent a source of ambiguity until also the person entities are processed for deduplication as depicted in figure 5.10;

- *Include by*: this policy will redirect only the links that satisfy a given predicate, to be expressed in term of the duplicate's neighbour properties, or typology. For example a certain configuration might redirect only the authorship relationships of authors with a given provenance.
- *Exclude by*: this policy will redirect all the existing relationships (ingoing and outgoing edges) occurring between a deleted object and its neighbours, except for those that satisfy a given predicate, to be expressed in term of the duplicate's neighbour properties, or typology.

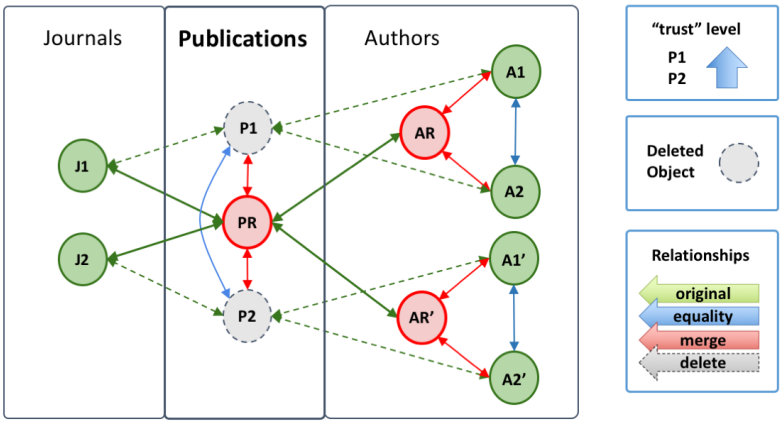


Fig. 5.10: Multiple entity disambiguation

A special case that needs to be treated differently occurs in graphs that include links between entities of same type, e.g. a similarity relationship between two publications. When two (or more) entities gets merged, and they have a link between them, the relationship would be transformed in a self-loop on the representative object. Depending on the application domain this behavior may be needed or not, therefore the system allows to be configured accordingly.

Creation of Dedup Graph

The combination of duplicate grouping and duplicate merging generates a new Dedup Graph $\mathcal{G}_D$, which is obtained, as described in Listing 5.15, by adding to the graph, with overlay $dedupGraph$:

- the representative objects obtained from each group $CC \in \mathcal{C}$ of duplicates, denoted as $repr(CC)$;

- the relationships created from and to the representative object;
- all objects and relationships that were not touched in the equivalence graph.

```
1  reprObjects = {repr(CC)|cc ∈ C}
2
3  // adds representative objects to the graph
4  AddObjects(reprObjects)
5
6  // adds the overlay dedupgraph to representative objects in the graph
7  AddOverlay_o(reprObjects, dedupGraph)
8  foreach o ∈ reprObjects do
9      // the set of objects merged by representative objects
10     MergedObjects = +Merges(o)
11
12 // adds the overlay dedupGraph to all objects that are not merged by representative objects
13 AddOverlay_o(Overlay_o('anchorGraph')/MergedObjects, dedupgraph)
14 foreach o ∈ MergedObjects do {
15     MergedRels = +R_out(o) ∪ R_in(o)
16     foreach < o, o', l >∈ R_out(o) do {
17         AddRels(< o, repr(o'), l >)
18         AddOverlay_r(< o, repr(o'), l >, dedupgraph)
19     }
20     foreach < o', o, l >∈ R_in(o) do {
21         AddRels(< repr(o'), o, l >)
22         AddOverlay_r(repr(o'), o, l >, dedupGraph)
23     }
24
25     // adds the overlay dedupGraph to all rels that are not touched by representative objects
26     AddOverlay_r(Overlay_r(anchorGraph)/MergedRels, deduGraph)
27 }
```

Listing 5.15: Anchor graph generation pseudo-algorithm

Figure 5.11 illustrates three connected components resulting from the equality overlay graph and the relative representative objects, generated according to a given strategy.

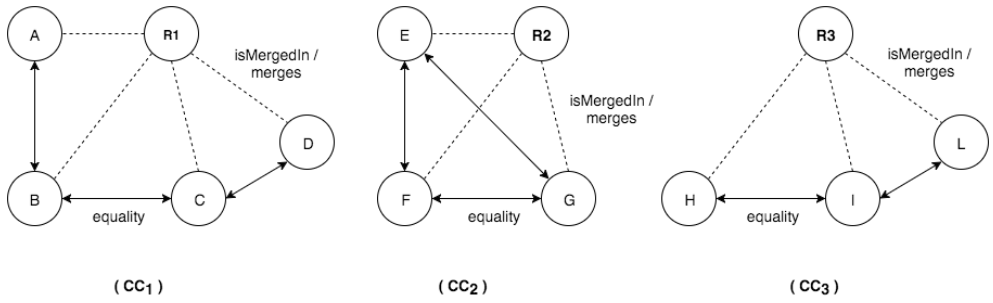


Fig. 5.11: Connected components and relative representatives

5.2.6 Data curators feedback

In order to allow data curators to refine the results of the automatic deduplication process, the system exposes ways to accept assertions about the *equivalence* or *diversity* between objects. Typically curators have access to the collection of objects of an entity type as created in the Dedup Graph $\mathcal{G}_D$. Curators can apply two kinds of actions:

- *Grouping sets of objects to make state their equivalence*: data curators can gather raw or representative objects, in the second case the implicitly insert in the group all objects merged by the representative object; once the group is "committed" a new representative object is created, together with a set of assertions of equality between all pairs of objects in the group;
- *Removing one object from the merged objects of a representative object*: in one scenario the representative object merged two objects, so the removal of one object removes the representative object from the graph and "frees" the two objects; in the other scenario the representative object persists and only the object at hand is removed; in both cases, the action results in a set of assertions of diversity between the removed object and all others that were merged by the representative object.

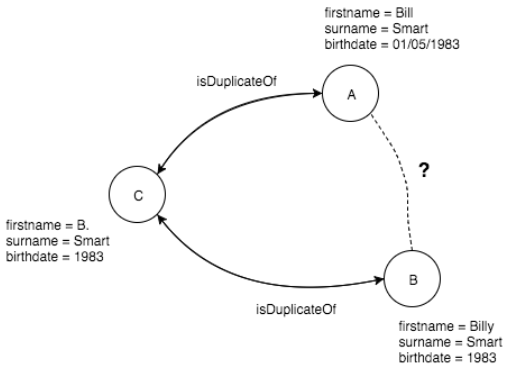


Fig. 5.12: Equivalence transitivity

Each of these assertions nourish a Feedback Graph $\mathcal{G}_F$, where such feedback are made explicit thanks to relationships between objects of kind $equalTo$ and $differentFrom$:

- *Equivalence assertion:* states that two objects B and C are equal. The system will introduce the equivalence relationship by adding the relationships $< B, C, equalTo >$ and $< C, B, equalTo >$ to the graph, marking them with the overlay $feedbackgraph$. After the duplicate identification phase such relationships will be marked with overlay $equivalencegraph$ so that they can be fed to the next grouping phase. Figure 5.13 illustrates two disjoint connected components $[A, B]$ and $[C, D, E]$ that gets connected

by the user assertion above, to be incorporated in the otherwise disjoint connected components that involves them.

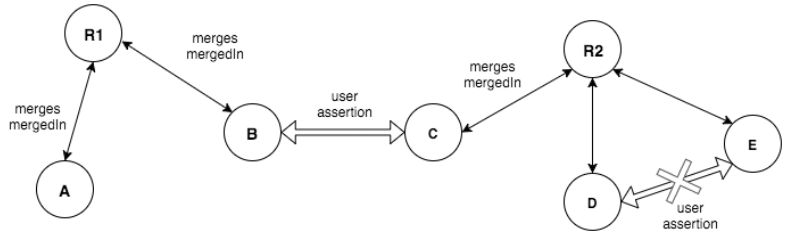


Fig. 5.13: User assertions

- *Diversity assertion:* states that two objects D and E are distinct. The system will introduce the difference relationship by adding the relationships $< D, E, differentFrom >$ and $< E, D, differentFrom >$ to the graph, marking them with the overlay feedback graph $G_{feedback}$. Unlike equivalence relationships, these relationships are not propagated after duplicate identification, but rather after the duplicate grouping phase. This is due to the fact that such statement may be superseded due to transitive closures of erroneous equivalence relatinships. Figure 5.12 shows three equivalence relationships resulting from duplicate identification, and $< A, B, equalsTo >$ and $< B, A, equalsTo >$ are removed by a diversity assertion. By transitivity, the two objects would again be put together in the grouping phase, overlooking the diversity assertion. In order to respect user's statement depicted in figure 5.13 the system applies the transitive property to the diversity assertion as well, such that if D is not equivalent to E, then also the object C must be different from D. This causes the duplicates group to be dissolved, removing the equivalence edge $< E, D, eqaulTo > E_{CD}$ as well as E_{DE}.

6

GDup

In this chapter we describe GDup, a system implementing the BGEDS's architecture described in chapter 5. The section of this chapter will provide the implementation details of the architectural areas identified in the previous chapter in order to meet the requirements presented in chapter 4. In the implementation plan we only considered Open Source tools.

6.1 Import and Storage

In section 5.2.2 we presented the concepts concerning the graph import layer, responsible for acquiring graphs conforming to a given SPGM schema and store them in GDup's storage system. In this section we will proceed in describing the encoding used to store graphs responding to the SPGM and the database system used to implement GDup's persistence layer.

6.1.1 Graph encoding

Graphs find two common representations: adjacency matrix, and adjacency list. Depending on the graph properties at hand the choice might lean towards one representation or another. In particular, an important property to be considered when choosing the most adequate representation is the *density*. In a graph $G = (V, E)$ its *density* can be defined as $D = \frac{|E|}{|V|\,(|V|-1)}$, thus a dense graph is a graph in which the number of edges is close to the maximal number of edges, i.e. $|E| = O(|V|^2)$. Complementarily a graph is *sparse* when its number of edges is close to the minimal number of edges, i.e. $|E| = O(|V|)$.

- Adjacency Matrices (AM) represent a graph as an $n \times n$ square matrix M, where $n = |V|$, and $M_{ij} = 1$ represents the existence of a link from node vertex i to j. Adjacency matrices has the advantage to support mathematical manipulation, and computation on ingoing and outgoing edges correspond to iterations over the rows and columns. On the other hand sparse matrices introduces a lot of zeros in case of sparse graphs, causing considerable waste of storage space;

$$G = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

- Adjacency Lists (AL) can be seen as a compact representation of the adjacency matrix, in which every vertex has associated a list of its outgoing edges. This make it easy to compute over outgoing edges, while makes it much more difficult to support processing over ingoing edges. Moreover adjacency lists are more suited to encode sparse graphs rather than dense graphs.

$$G = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} \longrightarrow \begin{array}{c} a : b, c, d \\ b : a, c \\ c : b, d \\ d : a, b, c \end{array}$$

In general, the level of density is hard to predict as it strictly depends on the nature of the graph to be deduplicated and on its duplication degree. In fact, even if an original graph $\mathcal{G}$ has low density, its corresponding deduplicated graph $\mathcal{G}_d$ may feature higher density, due to the introduction of representative objects and the redirection of the relationships (see section 5.2.5). High duplication rates make representative objects become graph *hubs*, which in graph theory are vertices with an high degree, i.e. a high number of edges, thereby increasing the density of $\mathcal{G}$. In general:

$$D(\mathcal{G}) <= D(\mathcal{G}_d)$$

In GDup we opted for representing graphs as adjacency lists, as the average real-case we were confronted with scenarios that were characterized by low-density graphs. In general, we believe this choice can cope well with a large class of real world scenarios. In case of graphs characterized by an high presence of duplicates, it is expected that the deduplication process will cause a significant increase of density in the deduplicated graph produced as output. Higher number of duplicates typically implies higher load on the processing layer. More specifically this results in (i) the candidate matching phase to produce an higher number of equality relationships between pair of objects, (ii) the connected component identification algorithm to scan larger components, and (iii) the relationship distribution phase to process more data. Likewise the resource usage on the layers dedicated to persist the information will be higher as the HBase rows will store an average higher number of outgoing edges.

In the context of OpenAIRE we had the opportunity to face two cases of entities duplication that well represents the difference between cases of low and high density duplication cases, i.e. publication and person, respectively. In fact, as we will discuss more in details in chapter 7, the average number of duplicated publications in the OpenAIRE graph was found to be between 2 and 3, while in the case of individuals, typically authors of publications, the average number of duplicates is much higher. To cope with this duality, in case of graphs providing high rates of duplicates, GDup can make use of Ground Truth information, activated for a given entity type. The activation of a Ground Truth changes the perspective of the deduplication process, since part of the deduplication results can be made available before the candidate identification is executed, by mapping raw objects onto anchor objects during the graph import phase. Such pre-processing can considerably reduce the time needed to process the whole collection of person objects, limiting the identification of duplicates to the objects that are not yet part of the Ground Truth.

6.1.2 Storage Layer

For the purpose of this work, graphs conforming to the SPGM needs to be stored in a database capable to meet the requirements described in sections 4, more specifically, the non functional requirement 4.3.2 covers an important role for the realization of a deduplication system capable to manage graphs of arbitrary size:

> **NF.3** ***Efficiency and scalability*** The system must be able to store Big Graph with in principle no limits of scalability. The same storage technology should therefore make it possible to execute all steps of candidate identification and matching, as well as all steps of graph disambiguation. Indeed, due to the amount of data involved, it is recommended not to invest on an approach that adopts multiple storage systems for the different challenges.

In fact in GDup we aimed at a database system capable to meet a more elaborate set of different non functional criteria:

- *Persistence of petabytes of data:* The system should be able to store huge amount of data. Scaling up the hardware on single node solutions, other than requiring considerable upfront investment, tend to be unpractical and over expensive beyond certain requirements, therefore we aimed at distributed storage systems;
- *Extendability:* The system should be freely extensible in terms of storage and processing capacity in order to adapt to new application requirements and accommodate changes in workloads. This allows to avoid to re-think the whole system architecture by adding new resources as needed;
- *Distributed processing:* Complementary to the need to store vast amounts of data across a set of nodes, data intensive tasks requires efficient processing techniques that, in case of big data applications, often requires to move the computation as close as possible to the data itself.

Moreover, in strike contrast with typical graph databases that implements efficient graph traversal functionalities, in GDup , like in many other data intensive applications [25], the requirement is shifted towards efficient bulk read and write operations, as well as supporting a tight integration with a distributed data processing system.

The architecture of BGEDS described in chapter 5 is typical of systems realizing an ETL[1] + OLAP[2] chain. Among the database technologies sprouted from the NoSQL movement, column stores are well suited to implement OLAP systems, and we found in HBase[3]a good candidate for implementing arbitrary large Graph Adjacency Lists. A number of projects have been proposed in recent years based on HBase, among these we can mention “Scalable rdf store based on hbase and mapreduce” [80], “HBase and Hypertable for large scale distributed storage systems”[44], “ Jena-HBase: A distributed, scalable and efficient RDF triple store” [43], “Distributed semantic web data management in HBase and MySQL cluster” [24]. HBase is the open source version of BigTable, the distributed storage system developed by Google for the management of large volume of structured data. As stated in the article that presented BigTable [11], it is described as “a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.”, and implements a quite simple, yet powerful data model consisting 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 popular framework Hadoop [88], that in recent years, thanks to the decreasing cost per byte of the storage systems, has been widely adopted in both business and academia, enabling large scale distributed data processing and analytics thanks to the Map Reduce programming model [20, 54]. HBase uses HDFS as primary storage layer [26], the open source implementation of the Google File System presented at the 19th ACM Symposium on Operating Systems Principles in 2003 [27]. The Hadoop Distributed File System (HDFS) is designed to run on commodity hardware, to be highly fault-tolerant, to provide high throughput access to application that manages large data sets [79, 9].

In the GDup implementation plan we considered using one HBase table for each graph received from the mapping layer. Each graph is represented as an Adjacency List, i.e. :

- *Vertices:* every vertex in the graph corresponds to a row. The vertex identifier is used to build row keys, that, in order to support efficient bulk read operations (scan) are built as fixed-length strings, prefixed using a code associated to the Vertex Type (V_T). For example, in case of a graph containing publications and their authors, row keys for the two vertex types will be generated as follows:

[1] ETL - Extract, Transform, Load
[2] OLAP - Online Analytical Processing
[3] HBase - `https://hbase.apache.org`

V_T	V_T code	Vertex Id	Row key
Publication	00	10.1234/xyz	`00|02e0c50bbf0d60f8a378dc611030118e`
Person	01	10.1234/xyz::Turing, A.	`01|509fd4289a8eaf8da90a2e94c601a30e`

Table 6.1: HBase Table - Row Key example

- *Properties:* the properties (metadata) associated to each object are stored in a dedicated column family `metadata`, that will hold the serialization of an object conforming to the properties defined for a given Vertex Type (V_T):
- *Status:* the vertex status (described in section 5.1) is stored in a dedicated column family, and contains information about the virtual deletion of a given vertex, and the provenance of the actions taken by the system on it.

Row key	metadata	status
`00|00`	`{ title : "asd", abstract : "asdasd" }`	`{ deleted:false }`
`01|01`	`{ surname : "Turing", name : "Alan" }`	`{ deleted:false }`

Table 6.2: HBase Table - Object properties

- *Edges:* Edges are stored in labelled column families forming semantically homogeneous groups of relationships. More specifically, an Edge Type E_T defiles a column family, and each qualifier under that column family is an outgoing edge from a given row. The cell value will contain the metadata for a single edge. Overall an HBase table containing a graph that describes articles and authors can be described by table 6.3:

	Row key	metadata		hasAuthor		isAuthor
		article	person	01|01	01|02	00|00
R1	00|00	{ title:"..." }		status:{ }	status:{ }	
R2	01|01		{name:"A1"}			status:{ }
R3	01|02		{name:"A2"}			status:{ }

Table 6.3: HBase table - Vertices and Edges

Deduplication object model

The deduplication configuration defines a binding between the objects (and their properties) of the graph represented in the Structured Property Graph Model (SPGM) and the functions that will process them. To do so, a section of the configuration is dedicated to define the deduplication document model. It consists of a set of triples *name*, *type*, *path*, plus an optional *function* declaration:

- *Name*: represent the property name;
- *Type*: identifies the typology of a property value. Structured values (in particular Lists and Object values) requires custom deserialisation strategies;
- *Path*: declares an xpath-like string used to identify the value in the SPGM to be associated with the given *name*.
- *Function*: declares a function to be applied to the value extracted by the *Path*, used to manipulate the value, typically to perform cleaning and normalization operations.

```
"model" : [
 {
  "name":" title ",
  "type":" String ",
  "path":" result /metadata/ title [ qualifier #classid = {main  title }]/ value"
 },
 {
  "name":"authors",
  "type":" List<Person>",
  "path":" result /authors"
 }
]
```

Listing 6.1: GDup configuration document model

The example in listing 6.1 describes an object composed of a *title* and a sequence of person objects *authors*.

6.2 Workflow pipelines

So far we used several times the expression “workflow” to denote the series of activities that are necessary to complete all the phases of the deduplication task. In order to provide end users with a system capable to configure, execute, and integrate the individual steps depicted in the end-to-end deduplication workflow depicted figure 5.1 GDup required a tool supporting the specifics of such orchestration. In this section we will describe the technological stack supporting the orchestration of the end-to-end deduplication workflow, in particular we will briefly describe the D-Net software toolkit [56], and more specifically its Manager Service and Resource Orchestration component (MS_RO) [4], that was extended and used to compose all the different *workflows* realizing each aspect of the deduplication task.

6.2.1 D-NET Software toolkit

D-NET is a service-oriented framework specifically designed to support developers at constructing custom aggregative infrastructures in a cost-effective way. D-NET offers data management services capable of providing access to different kinds of external data sources, storing and processing information objects of any data models, converting

them into common formats, and exposing information objects to third-party applications through a number of standard access API. Most importantly, D-NET offers infrastructure enabling services that facilitate the construction of domain-specific aggregative infrastructures by selecting and configuring the needed services and easily combining them to form autonomic data processing workflows. The combination of out-of-the box data management services and tools for assembling them into workflows makes the toolkit an appealing starting platform for developers having to face the realization of aggregative infrastructures.

The Enabling Layer contains the Services supporting the application framework. These provide functionalities such as Service registration, discovery, authentication and authorization, subscription and notification and data transfer mechanisms through special ResultSet Services. Most importantly, these Services can be configured to “orchestrate” Services of other layers to fulfill application specific requirements.

D-NET Manager Service

The Manager Service (MS) addresses service orchestration and monitoring, hence “autonomic behavior”. D-Net workflows are resources describing sequences of steps, where each step may consists of business logic (i.e. Java code), remote service invocations, workflow forks (i.e. parallel sub-workflows), and workflow conjunctions (confluence of parallel workflows). Workflows can therefore be defined as a directed graph, where nodes represents *actions* and edges represents the execution path of such actions. Figure 6.1 provides a visual representation of one of the workflow that can be defined in D-NET.

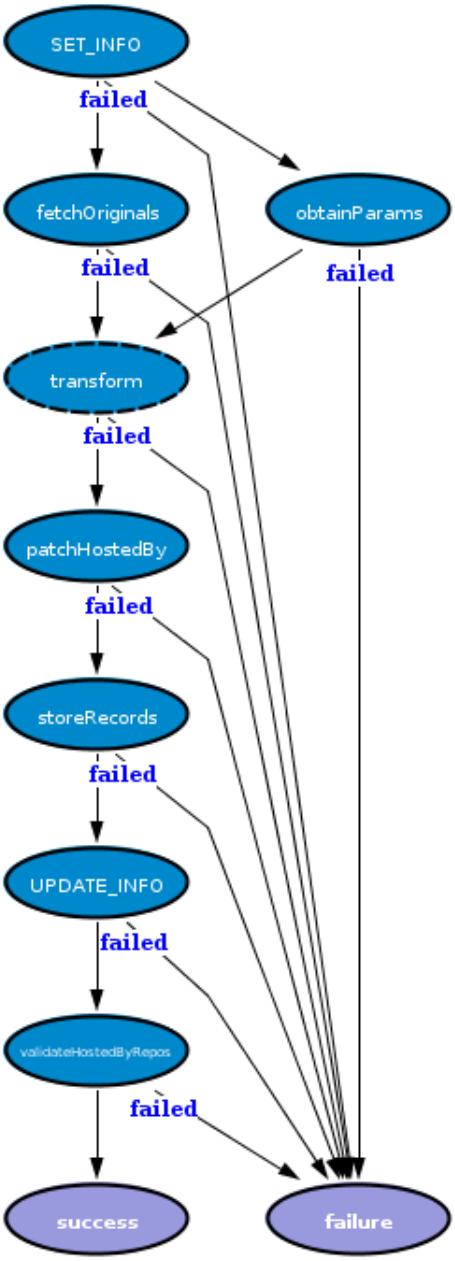


Fig. 6.1: D-NET workflow instance

Typically, service invocations are preceded by a look-up into the Service registry, in order to discover the “best” service of the needed kind and available to execute the call.

Workflows can be fired manually or as a consequence of the notification of a resource-related event from D-NET’s Information System or because of scheduled events. Workflows are commonly used to automatically schedule data collection from data sources, population of information spaces, and synchronization of information space mirrors or information space staging.

Workflows can implement long-term transactions by exploiting subscription and notification of events in the Information System. When a time-consuming step is to be fired (e.g. indexing a large set of metadata objects, or fire a long lasting Map Reduce Job), the invocation is accompanied by a subscription request to the event “conclusion of the step”. The MS waits for the relative notification before moving on to the next step.

Workflows can also be used as monitoring threads, checking for consistency and availability of resources or consistency and Quality of Service of the aggregative infrastructure. For example, aggregative infrastructure policies may require that a given collection of information objects be replicated K times. A monitoring workflow may, at given time intervals, check that this is really the case and possibly take corrective actions, e.g. creating a missing replica. When corrective actions are not possible, warning emails can be sent to administrators.

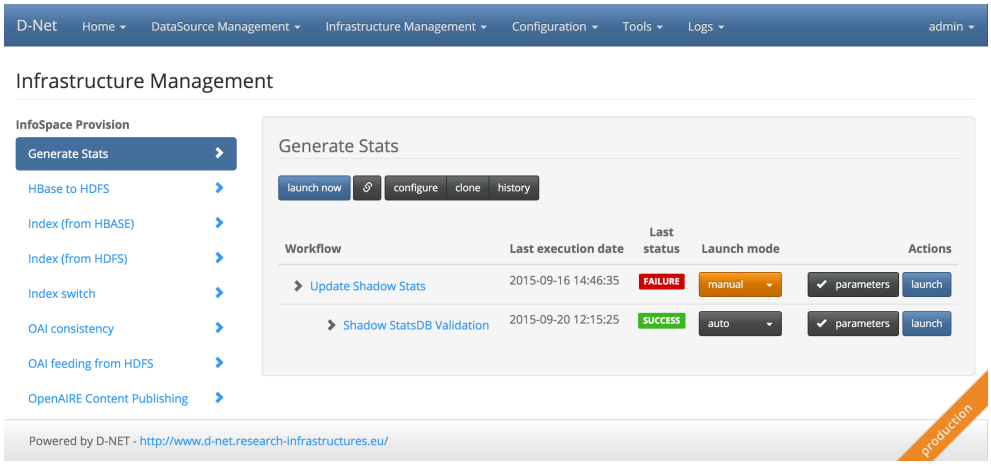


Fig. 6.2: D-NET Manager Service user interface

The MS user interface offers a graphical overview of the running workflow and allows administrators to inspect their execution status. It is possible to manually re-execute them or to redefine their configuration parameters. Administrators can also consult the history of workflow executions, which keep track of successful and failed workflow steps, as well as of values of given input/output parameters for such steps (e.g. endpoint of a service call that did not succeed). Workflows are treated as infrastructure resources,

hence can be shared by different instances of the Manager Service, and are preserved in the Information System.

Hadoop as a D-NET Service

In order to support end users with a single access point to the entire end-to-end deduplication workflow, we developed the D-NET Hadoop Service. It acts as a bridge between the D-NET Manager Service and an arbitrary number of Hadoop clusters, enabling (i) the implementation of smart discovery mechanisms that allows to address different disambiguation tasks on different Hadoop instances, and (ii) a transparent interaction between the Manager Service and the Hadoop cluster capabilities. More specifically the D-NET Hadoop service supports the following features:

- *HDFS:* The Hadoop Distributed File System is made available to all the other services participating in the D-NET infrastructure. Files on HDFS can contain intermediate results of data intensive tasks, or provide support for the applications logging;
- *HBase:* The Manager Service has can access the HBase API. Common operations include the management of tables, which can be created according to arbitrary schemata, read operations to individual rows or subset of rows via scan operations.[4] In order to feed objects to a table, clients must provide an input ResultSet and the proper data transformation rules required to map the objects into rows of the HBase cluster according to the opted physical representation of the data model;
- *Map Reduce:* Most important, in order to realize the data processing chain described in the BGEDS architecture 5.2.4,5.2.5 the D-NET Hadoop Service supports interaction with the Hadoop components responsible for submission and monitoring of Map Reduce jobs (JobTracker on Map Reduce v1, and YARN on Map Reduce v2). The Manager Service can therefore implement complex data processing chains by combining different Map Reduce jobs, hiding the huge parametrization effort required by the deduplication algorithms, possibly exploiting different input and output formats for each job (e.g. input from an HBase table can be processed and the output stored on HDFS files, or vice versa), monitor the job status to allow the workflow to advance in its steps.

Workflow nodes meant to submit Map Reduce jobs will stop the workflow's execution path, while the D-NET Hadoop service starts monitoring the job progress. Typically in case of failures the workflow will finish in an error state (providing also the error cause), while in case of successfully execution it will proceed with the subsequent node. Unlike the dedicated workflow manager provided in the Hadoop ecosystem Oozie[5] that supports only workflows defined as DAGs (Directed Acyclic Graphs), the D-NET Manager Service allows the definition of loops, which makes it possible to implement iterative Map Reduce algorithms. In case of loops the stop condition must be handled explicitly in a workflow node using the parameters found in the workflow environment.

[4] `https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html`
[5] `https://oozie.apache.org`

Each Map Reduce Job managed by the D-NET Hadoop Service has its own configuration profile as a resource persisted in the Information System. It specifies the nature of the input and output formats (e.g. HBase table, SequenceFile, etc. . .), the Java classes implementing the Map and Reduce phases, as well as additional parameters, including number of Map and Reduce tasks. When reading graph objects from a specific HBase table, the configuration specifies also the parameters used to set the scan operation.

6.3 Deduplication on Map Reduce

In this section we illustrate how the Map Reduce paradigm allows to efficiently implement the (i) the candidate identification, (ii) the candidate matching, and (iii) graph disambiguation phases described in the end-to-end deduplication workflow. Although the input of the system is a graph, in this section of the workflow the algorithms implementing *candidate identification* and *candidate matching* phases works on top of vertices of a give type V_T, i.e. a collection of homogeneous objects.

Theoretically duplicate identification requires pairwise object comparisons, hence a Cartesian product with quadratic complexity $O(n^2)$. Candidate identification aims at overcoming the consequent efficiency issues by implementing heuristics (i.e. blocking, nearest neighborhood [64]) capable of skimming out pairs of objects that are unlikely identical. Heuristics efficiently identify sub-collections of objects that are likely to be duplicates and then apply candidate matching only to those. Heuristics improve efficiency but may indeed decrease effectiveness, since some pairs of duplicate objects may be mistakenly left out.

Different implementations were proposed to address object deduplication on large datasets with Map Reduce, each providing a contribution for different aspects of the problem. For example, Kolb at al. analyse the memory bottlenecks and load balancing issues intrinsic in Map Reduce implementations of the Sorted Neighborhood Blocking method [51, 49, 47]. Hsueh at al. further elaborate on the blocking and load balancing techniques in [34], McNeill et al. propose a Dynamic Blocking technique in [63], and Vernica et al. propose a Map Reduce based Set-Similarity Join method [83]. Other approaches are based on the objects semantic similarity [85]. Currently GDup implements the candidate identification phase using a variation of the Sorted Neighborhood Blocking method that uses a configurable set of clustering functions as described in section 5.2.4.

6.3.1 Candidate identification & matching

The candidate identification phase aims to reduce the duplicates search space required by all the possible pairwise comparisons between the properties of the objects belonging to a given Type V_T. GDup implements a number of predefined functions that allows to select the objects in the graph that are eligible for detailed matching by grouping them into blocks. This technique in record linkage literature is known as *Blocking* and a number

of variations have been proposed and evaluated by researchers in the past years [64, 6, 12, 36, 38, 90, 31, 22], as well as non Map Reduce approaches [75]. The general idea of such methods is that each block is populated by selecting approximately similar objects, which are much likely to produce a positive match, while excluding a priori a large part of true negative matches. In GDup we provide an hash based blocking technique, so that each block B_i is associated with an hash key h_i, such that the vertex providing the property p_i is hashed to the block B_i if $hash(p_i) = B_i$. End-users can combine their own hashing functions with a set of domain independent functions provided in GDup that includes:

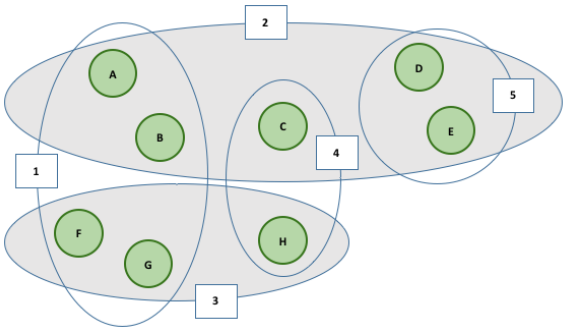
- *Acronyms*: Given a string value, returns a list of strings built by combining the $i-th$ character of each significant word in the input text. Parameters allows to drive (i) the number of hash keys returned, (ii) the minimum and the maximum length of the hash keys. For example the string *"Search for the Standard Model Higgs Boson"* might produce `[ssmh, etoi, aadg, rneg]`;
- *Ngrams*: Given a string value, returns a list of strings built by picking the first n characters from a sliding window of size m that shifts over each significant word in the input text. For example the string *"Search for the Standard Model Higgs Boson"* might produce `[sea, ear, sta, tan, mod, ode, hig, igg]`;
- *NgramPairs*: Given a string value, returns a list of strings built by considering each pair of significant words in the input text, and for each pair build an hash key by combining the first n characters of the two. For example the string *"Search for the Standard Model Higgs Boson"* might produce `[seasta, stamod, modhig]`;
- *Sorted NgramPairs*: Based on the NgramPairs function, sorts the significant words before combining the ngrams. For example the strings *"University of Pisa"* and *"Pisa University"* might both produce `[pisuni]`;
- *Ngrams*: Given a string value, returns a list of strings, each of them built by combining the suffix and the prefix of each pair of significant pair of words in the input text. For example the string *"Search for the Standard Model Higgs Boson"* might produce `[rchsta, ardmod, delhig, ggsbos]`.

Each set of objects grouped in blocks by a common hash h_i can be processed independently from each other. This is where the the Map Reduce programming paradigm provided by the Hadoop framework contributes to allow candidate identification and matching phases to scale out. Objects in the graph of a given type V_T are bulk read from HBase and processed for candidate identification by one map task per HBase RegionServer.[6]

Equivalence matching is the phase where every pair of objects identified in the candidate identification phase are matched to return a measure of similarity in between 0 and 1. Such similarity measure is expressed by functions in charge of replacing human judgement, often based on (combinations of) string matching functions. Similarity functions are hard to define, as they should encode factors that are easy to spot for humans

[6] `http://hbase.apache.org/book.html#splitter`

and less easy for machines, as well as deciding beyond which similarity threshold two objects are to be considered equivalent. Indeed, minimal decimals of difference may lead to consider two distinct objects as equivalent or dissimilar. For example, when matching to two publication objects by their titles “A cat perspective on the logic of mouses” and “A cat perspective on the logic of mouses v2”, the titles are different but may be 0.98 similar. With an equivalence threshold of 0.99 we would rule out the pair of objects, but we would lose the case-match for the publications “A cat perspective on the logic of mouses” and “A cat perspective on the ogic of mouses”, which are mistakenly different due to a typo (the missing “l”).



Vertex	hash keys
A	[1, 2]
B	[1, 2]
C	[2, 4]
D	[2, 5]
E	[2, 5]
F	[1, 3]
G	[1, 3]
H	[2, 4]

Fig. 6.3: Blocking objects by hash keys

The basic idea for the Map Reduce implementation can be described as follows. Each *map task* processes a subset of the objects of a given type V_T, and *emits* a number of *hash keys* h, depending on the hash functions f_H specified in the configuration.

Algorithm 1: Candidate identification (map phase)

input : $conf$: the dedup conf, rk: hbase row key, O: object
output: set of pairs $[h_i, O]$

```
1 Setup(conf)
2 Map(rk, O)
3     for hash_f ∈ conf do
4         for h ⟵ Hash(O, conf) do
5             Emit(h, O)
6         end
7     end
```

On the other hand, each *reduce task* receives the sequence of objects (*objectQueue*) grouped for a given hash value. Then the reduce task calculates the *similarity* between each pair of vertices according to the given configuration, and in case of a similarity value above the configured threshold, emits the equivalence relationships (edges) among the two vertices. In case of large blocks users should activate the sliding window mechan-

ism, which will sort the list of objects received by the reduce task by a configurable object property (or a manipulation of the same), and then perform the comparisons only among the objects that fall within the window bounds. The algorithm implementation considers also configurable thresholds to define the window size, as well as the maximum number of objects to be retained in a block.

Algorithm 2: Candidate matching (reduce phase)

input : $conf$: the dedup conf, h: the grouping hash, $objectQueue$: the objects
output: pairs of equivalence relationships between the objects

```
1  Setup(conf)
2  Reduce()
3      while objectQueue is not empty do
4          pivot ⟵ objectQueue.head()
5          for O ∈ objectQueue do
6              sim ⟵ Similarity(pivot, O, conf)
7              if sim >= conf.threshold then
8                  EmitEquivalence(pivot, O)
9                  EmitEquivalence(O, pivot)
10             end
11         end
12     end
```

Figure 6.4 provides an overview of the Map Reduce implementation of the candidate identification and matching phases.

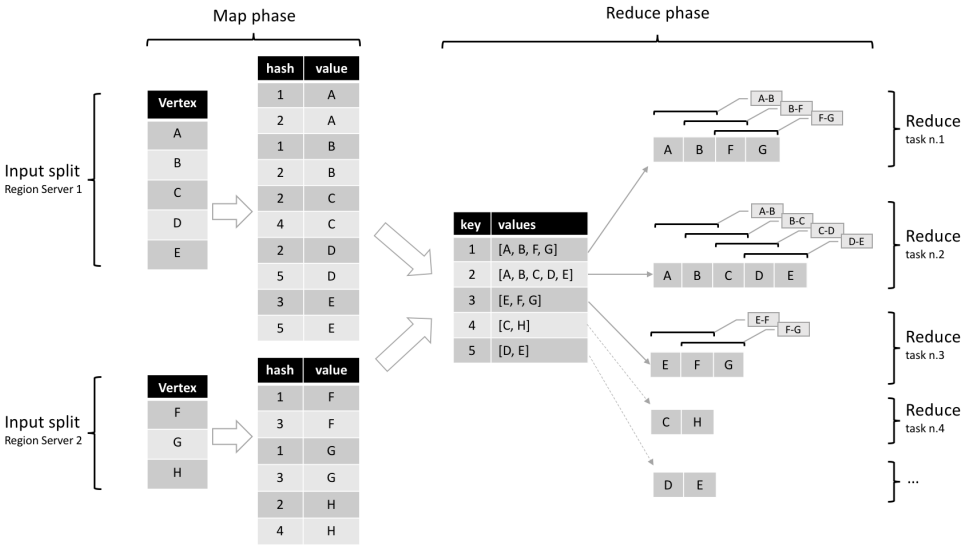


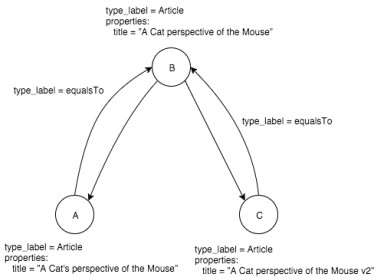
Fig. 6.4: Map Reduce - Blocking with Sliding Window

6.3.2 Graph disambiguation on Map Reduce

Each pairwise comparison whose similarity measure is above the configured threshold produces a new equivalence relationship, which is added to the graph's adjacency list stored on HBase.

	Row Key	metadata		equalsTo		
		article	...	A	B	C
R1	A	title : "A Cat's perspective of the Mouse"			X	
R2	B	title : "A Cat perspective of the Mouse"		X		X
R3	C	title : "A Cat perspective of the Mouse v2"			X	

(a) Equality relationships as Adjacency list



(b) Equality Graph $\mathcal{G}_E$

Fig. 6.5: Equality relationships in HBase

The challenge arising in duplicates grouping is based on the intuition that a relation of equivalence is transitive. For example, with respect to Figure 6.5, since $A = B$ and $B = C$ then we can also conclude that $A = C$. As a consequence, graph disambiguation cannot be based on $\mathcal{G}_E$ alone, i.e. by merging pairs of equivalent objects, but should

rather be based on groups of equivalent objects. With respect to the example above, if no other object in $\mathcal{G}_E$ is reached via a relationship $equalsTo$ from A, B, or C, then the group of objects $\{A, B, C\}$ is a connected component in $\mathcal{G}_E$ and represents a set of equivalent objects, ready to be merged into one object.

In order to identify each connected component cc in the equivalence graph $\mathcal{G}_E$ we need to classify each component to a common key that is unique in $\mathcal{G}_E$. A common solution is to let a component be classified by its lowest id (in our case its row key), e.g. the component $\{A, B, C\}$ has the lowest id A, i.e. the classifier for this component.

Finding the connected components of a graph is a problem that has been studied for a long time. Well known methods perform in linear complexity using depth first (or breadth first) graph traversal search [81] to discover the connected components. In order to address the problem on massive graphs, several algorithms were proposed in [5, 78, 32]. Such methods however, although performing in logarithmic time, were based on shared memory systems, thus not suitable for the Map Reduce programming model, which is based on shared nothing clusters, and designed to batch process independent and disjoint sets of data.

The current GDup implementation of the connected component identification was inspired by the CC-MR method [76, 69], and further analysed by Kolb et al. in [48].

The CC-MR algorithm is based on the assumption that exists a total ordering of the graph vertices (in our case the vertex identifiers are generated as fixed length string, thus they can be ordered lexicographically). It takes as input a graph $G = (V, E)$ and iteratively transform each connected component in a star-like sub-graph, where the vertex having the smallest identifier is the center.

The algorithm 3 illustrates the CC-MR algorithm pseudo-code. It assumes an identity map phase that simply emits the vertices towards the reducers. The described steps are

applied iteratively until no more merges occur.

Algorithm 3: Connected Component on Map Reduce - CC-MR (reduce phase)

input : Graph composed of connected components
output: Graph composed of a set of star-like sub-graphs G_{cc}

```
1  Reduce(Vertex source, Iterator<Vertex> values)
2      locMaxState ⟵ false
3      first ⟵ values.next()
4      if source.id < first.id then
5          locMaxState ⟵ true
6          Emit(source, first)                          // Forward edge
7
8      end
9      lastId ⟵ first.id
10     while values.hasNext() do
11         curr ⟵ values.next()
12         if curr.id = lastId then
13             continue                                 // Remove duplicates
14
15         end
16         if locMaxState then
17             Emit(source, curr)                       // Forward edge
18
19         end
20         Emit(first, curr)                            // Forward edge
21
22         Emit(curr, first)                            // Backward edge
23
24         lastId ⟵ curr.id
25     end
26     if not locMaxState or source.id < lastId then
27         Emit(source, first)                          // Backward edge
28
29     end
```

Given the graph composed of a set of star-like sub-graphs G_{cc}, the identifier of the representative object can be created starting from the smallest identifier of each component, and introducing a new pair of relationships (mergedId/merges) from each member of the connected component and the new center of the star.

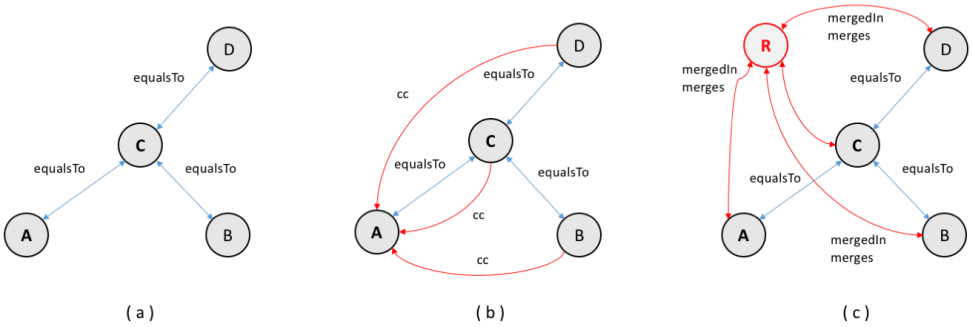


Fig. 6.6: Representative object creation

The representative object R depicted in Figure 6.6 will contain the properties of the objects in the connected component $\{A, B, C, D\}$ merged according to a given configuration. At this stage the graph adjacency list on HBase contains the relationships that links each original duplicated object to its representative (i.e. a link between pairs of row keys). However, the row containing the representative object only contains the "merges" relationships, thus it is needed to process each connected component in order to build the new set of properties according to the configured policy. The Map Reduce implementation assumes that for each object o in the connected components graph G_{cc} the set of properties associated to o is emitted towards the row key in which o is "mergedIn" (map phase). Then the reducer merges the set of properties and persists the result.

Algorithm 4: Property merge on Map Reduce

input : Graph composed of a set of star-like sub-graphs G_{cc}
output: A representative object for each sub-graph

```
1 Setup(conf)
2 Map(row)
3 |   prop ⟵ row.getProperties(conf.entityType)
4 |   Emit(row.mergedIn, prop)
5 Reduce(row.mergedIn, Iterator<Properties> values)
6 |   rep ⟵ Merge(values, conf)
7 |   Emit(row.mergedIn, rep)
```

The other aspect involved in the graph disambiguation process is related to the relationship distribution strategy. As the disambiguation process takes actions that alter the graph structure (i.e. introduction of new representative objects and virtual deletion of merged objects), it is important to consider the effect of such actions on the existing edges, i.e. how the graph disambiguation will affect the relationships between the entities in the graph, and in general its semantic expressivity. To implement the relationship distribution in Map Reduce, we can extend the simple algorithm 4 that merges the object

properties. We can therefore introduce a second version of the algorithm, illustrated in 5, where the map phase emits the object properties and all the relationships belonging to the configured types towards the representative row key identifier. On the reduce phase the properties gets merged according to the configured strategy, while the relationships properly inherited by the new representative object.

Algorithm 5: Property merge and relationship distribution on Map Reduce

input : Graph composed of a set of star-like sub-graphs G_{cc}.
output: A representative object for each sub-graph that inherits the relationships of the objects it merges.

```
1  Setup(conf)
2  Map(row)
3      Emit(row.mergedIn, row.properties)
4      for relType ∈ conf.relTypes do
5          relFamily ⟵ row.getRels(relType)
6          for rel ∈relFamily do
7              Emit(row.isMergedIn, rel)
8          end
9      end
10 Reduce(row.mergedIn, Iterator<Object> values)
11     rep ⟵ ∅
12     for o ∈values do
13         if o.isProperty then
14             rep ⟵ rep ∪ Merge(o.getProperties(), conf)
15         else
16             r ⟵ o.getRel()
               /* emit relation from the representative to the related
                  entity                                              */
17             Write(row.mergedIn, r.getTarget(), r)
               /* emit relation from the related entity to the
                  representative                                      */
18             Write(row.mergedIn, r.getSource(), r)
               /* mark relation from the related entity to the duplicate
                  as deleted                                          */
19             Write(r.getSource(), r.relTarget(), markDeleted(r))
               /* mark relation from the duplicate to the related entity
                  as deleted                                          */
20             Write(r.relTarget(), r.getSource(), markDeleted(r))
21         end
22     end
23     Emit(row.mergedIn, rep)
24 Write(source, targetId, rel)
25     target ⟵ rel.getType()′ :′ targetId
26     Emit(source, target, rel)
```

6.3.3 Data curators feedback

Identify and correct the possible mistakes produced by a given deduplication score is an important aspect that BGEDS must support. In order to support data curators in their activity GDup provides user interfaces that allows to explore the results of the graph disambiguation process. The final step on the graph disambiguation workflow assumes that the collection of objects belonging to the type defined in a deduplication score gets indexed to enable search operations over the whole collection of results.

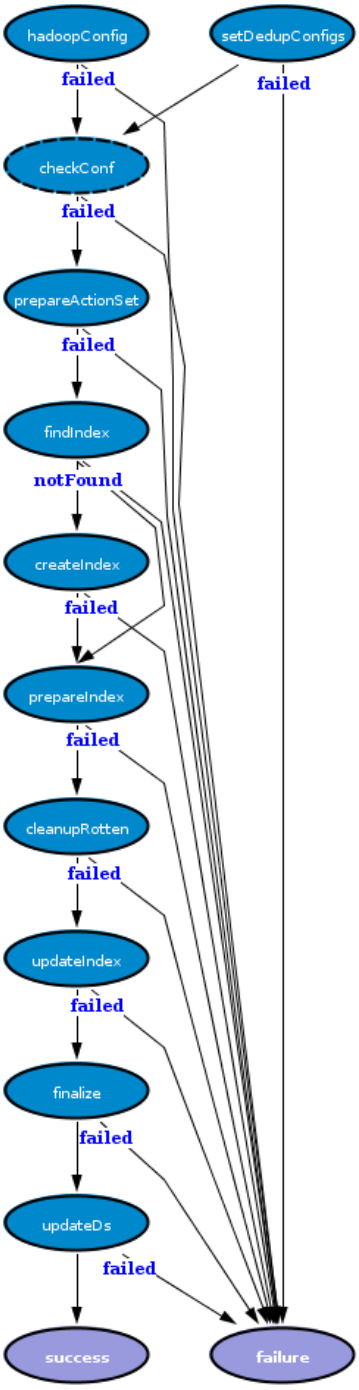


Fig. 6.7: D-NET Deduplication indexing workflow

The indexing process is supported by the D-NET framework, where the Manager Service coordinates the workflow execution illustrated in Figure 6.7. The indexing workflow is composed by a Map Only job tailored to:

- read all the objects of a given type from HBase.
- transform the objects into index-able documents according to an user defined configuration that specifies the relevant objects properties that should be searchable;
- send the document to the index server.

The index contains all the objects in the graph of a given type. More specifically it contains the objects there were not identified as duplicates and the representative objects (enriched with the duplicates they merge), while the duplicates are not indexed as individual entities, but will appear as part of a representative object. An example of a representative object in the index is illustrated in listing 6.2.


```
" entity ": {
  "type": " result ",
  " id ": "50|dedup_wf_001::4104a7740a16a2239e3871a13c6e5d1c",
  " dateofcollection ": "2015−02−06T13:25:33.854Z",
  " result ": {
   "metadata": {
    " title ": "Analysis and Report on Original Documentary Evidence Concerning the Use of
        Opium in India: [Furnished to the "British Medical Journal" by upwards of 100 Indian
        Medical Officers.]",
    "dateofacceptance": "1894−02−17"
    "language": "eng",
    "collectedfrom": "Europe PubMed Central"
   },
   "instance": [{
    "licence": "OPEN", "instancetype": "Article ", "hostedby": "Europe PubMed Central"
    " url ": " http :// europepmc.org/articles/PMC2403858"
   },
   {
    "licence": "OPEN", "instancetype": "Article ", "hostedby": "Europe PubMed Central"
    " url ": " http :// europepmc.org/articles/PMC2403577"
   }]
   },
   " originalId ": [ "oai:europepmc.org:993792", "oai:europepmc.org:993837" ],
   "pid": [{ "value": "PMC2403342", "type": "pmc" },{ "value": "PMC2403793", "type": "pmc" }],
  "merged": [
   {
    "type": " result ",
    " dateofcollection ": "2015−02−06T13:24:43.355Z",
    " id ": "50|od_____908::94111cc8a5601838d924551feeb12894",
    " result ": {
     "metadata": {
      " title ": "Analysis and Report on Original Documentary Evidence Concerning the Use of
          Opium in India: [Furnished to the "British Medical Journal" by upwards of 100 Indian
          Medical Officers.]",
     "dateofacceptance": "1894−02−03",
```

```
32          "language": "eng"
33         },
34         "instance": [{
35          "licence": "OPEN", "instancetype": "Article", "hostedby": "Europe PubMed Central",
36          "url": "http :// europepmc.org/articles/PMC2403858"
37         }],
38        },
39        "originalId": "oai:europepmc.org:993837",
40        "collectedfrom": "Europe PubMed Central"
41        "pid": [{ "value": "PMC2403858", "type": "pmc" }],
42        {
43         "type": "result",
44         "dateofcollection": "2015-02-06T13:20:22.788Z",
45         "id": "50|od_____908::ba4f88f028b9bd3aec61f764fd5080d3",
46         "result": {
47         "metadata": {
48          "title": "Analysis and Report on Original Documentary Evidence Concerning the Use of
                Opium in India: [Furnished to the "British Medical Journal" by upwards of 100 Indian
                Medical Officers.]",
49          "dateofacceptance": "1894-02-10"
50          "language": "eng",
51         },
52         "instance": [{
53          "licence": "OPEN", "instancetype": "Article", "hostedby": "Europe PubMed Central"
54          "url": "http :// europepmc.org/articles/PMC2403577"
55         }],
56         },
57          "originalId": "oai:europepmc.org:993629"
58         "collectedfrom": "Europe PubMed Central"
59         "pid": [{ "value": "PMC2403577", "type": "pmc" }],
60     }
61 }
```

Listing 6.2: GDup index document model

Data curators can explore the deduplication results produced by a given configuration via a dedicated user interface that enables them to spot errors of two different kinds:

- *False positives:* In this category data curators find objects that were mistakenly merged by a given configuration as they are in fact distinct real world objects. Corrective actions can include (i) a revision of deduplication score, e.g. changing the similarity function used to match a certain field, or (ii) introduce an assertion to state that two objects are not the same, or that an object should not be part of a given group. The assertion is persisted on a dedicated database, and can be applied immediately in order to alter the search results on the index, as well as on the graph stored on HBase.
- *False negatives:* A given configuration might underestimate the relevance of a given object property. Data curators exploring the deduplication results can highlight new cases of duplication that were not considered in the first place, or identify side cases

that requires individual corrections. Corrective actions can therefore include (i) a revision of the deduplication score, e.g. adjusting the weights associated to the similarity functions, or (ii) introduce an assertion to state that two objects are duplicates.

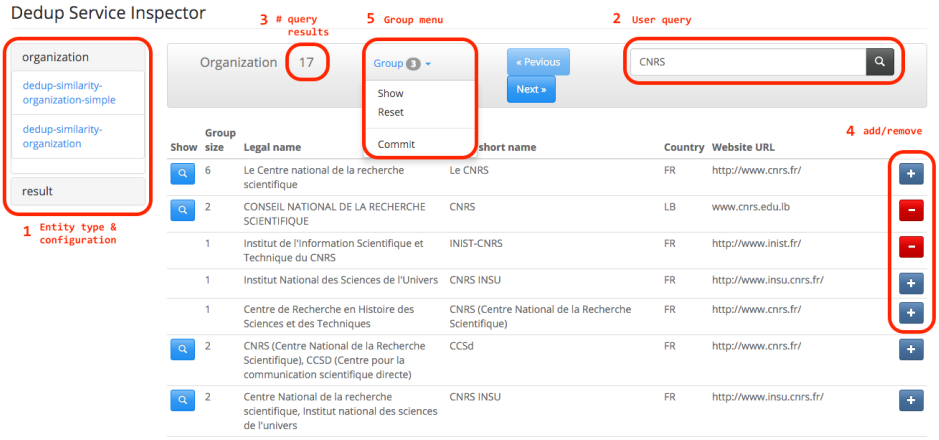


Fig. 6.8: GDup Data Curator user interface

Figure 6.8 illustrates the User Interface where Data Curators can:

1. Select the configuration score. This implicitly selects also the entity type referred by the selected configuration;
2. Queries will be run against the deduplication results produced by the selected configuration. Objects representing groups of duplicates will contain both the representative set of properties, as well as the properties of the individual duplicates;
3. The query results shows which objects were merged into a representative ($groupSize > 1$) or are distinct objects, i.e. objects representing a duplicate will appear within the group they were merged with;
4. Objects (or groups of objects) can be added to a basket-like staging area, in which they can check if the group needs to be extended including any of the other query results, or shrunk by removing any of its members;
5. Finally, the group menu allows to inspect the current "basket", to reset it, or commit the changes.

7

GDup in a real-case scenario

In this chapter we describe the specific usage of GDup in the context of the OpenAIRE technological infrastructure and report on the results of its operation after three years since its launch. More specifically we describe how the OpenAIRE infrastructure implements the end-to-end graph deduplication workflow depicted in Chapter 5, we provide a comment on the results and the execution time of the different workflow sub parts, highlighting benefits and drawbacks of the actual execution environment and implementation.

7.1 Data import

Although BGEDSs are conceived to acquire graphs via explicit mappings between JSON-LD and RDF to the Structured Property Graph Model, objects and relationships in the OpenAIRE graph are extracted from *information packages*, i.e. metadata records represented in various formats, collected from internet/web accessible sources described in 3.1. In order to avoid introducing major overhead in data transformations, in GDup the mapping to the SPGM was realised by implementing an ETL (Extract Transform Load) pipeline conceived to load the graph onto HBase. In this process a Dublin Core [87] bibliographic metadata record describing a scientific article will yield one OpenAIRE publication object and a set of OpenAIRE person objects (one per author) with semantically typed relationships between them. Publications compliant with the OpenAIRE guidelines[1] might optionally provide a reference to a research project. Publications exposing project references conforming the syntax illustrated in listing 7.1, when processed by the mappings used in the Data Import procedures, will yield a semantically typed relationship between the OpenAIRE publication and the OpenAIRE project objects. Research projects, just like publications and datasets, are part of the OpenAIRE information space, as they are aggregated to build an authoritative database of cross funder research projects.

[1] OpenAIRE Guidelines - `https://guidelines.readthedocs.org`

```
info :eu−repo/grantAgreement/ F / FP / PID / [J] / [PN] / [PA]

Where

F   := Funder
FP  := FundingProgram
PID := ProjectID
J   := Jurisdiction (optional)
PN  := Project Name (optional)
PA  := Project Acronym (optional)

Example of a dublin core element embedding a relationship to a project

<dc:relation>
  info :eu−repo/grantAgreement/EC/FP7/244909/EU/Making Capabilities Work/WorkAble
</dc:relation>
```

Listing 7.1: OpenAIRE guideline: Project references

Starting from the information packages collected and harmonized by the aggregation system, the Data Import procedure depicted in figure 7.1 includes all the processes needed to populate the Information Space. All the information needed to build the OpenAIRE objects and relationships is extracted from different services used to persist the aggregation status, transformed according to the dedicated mappings, and loaded to the HBase table dedicated to hold the adjacency list representing the OpenAIRE graph. Such ETL (Extract Transform Load) procedure is run according to a defined schedule and recreates from scratch the information graph. The D-Net orchestration layer executes the workflow that defines the individual steps required by the procedure:

- Connect to the database used by the aggregation system to store Datasources, Organizations and Projects, in order to extract the information needed to construct the relative OpenAIRE objects; The number of objects involved in this part of the process does not pose scalability issues, thus the mapping can be applied sequentially as it does not require a parallel approach.
- Connect to the Metadata Store services used to persist publication and dataset records, copy the content to the hadoop distributed filesystem (HDFS) in dedicated Sequence Files[2] and thanks to the MapReduce framework, apply the transformation provided in the dedicated mapping by parallel tasks, allowing to load on HBase arbitrary large sets of publication and dataset records;
- Apply the Actions stored in a given Ground Truth onto the HBase table. Such information is stored in a dedicated HBase table, thus the MapReduce framework allows to efficiently move its content from one table to another.

[2] Sequence Files - `https://wiki.apache.org/hadoop/SequenceFile`

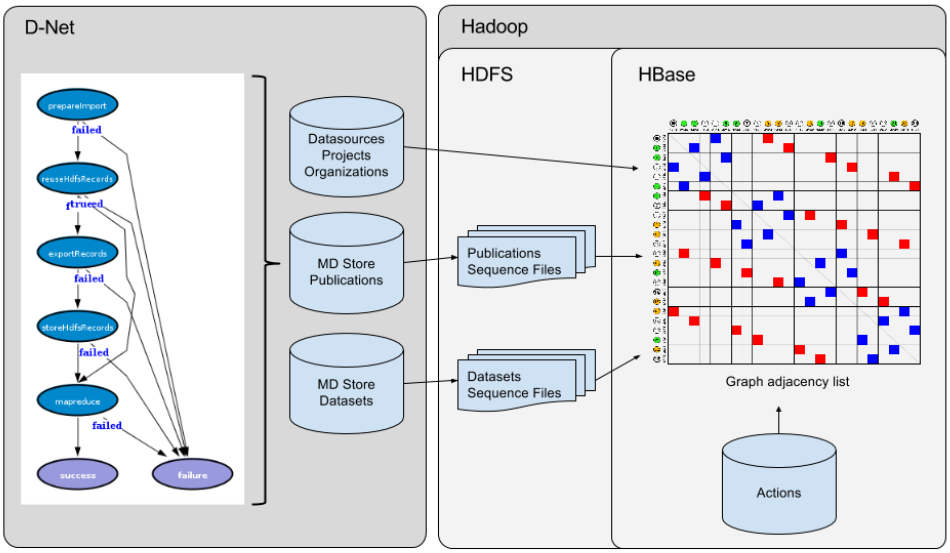


Fig. 7.1: GDup : data import ETL

7.2 The OpenAIRE production environment

The OpenAIRE production environment is composed by a set of virtual machines operated by ICM in Warsaw.[3] It includes nodes to host all the different services depicted in the OpenAIRE Specification and Release Plan.[4] Among such services, the deduplication subsystem architecture depicted in figure 7.2 refers to the BGEDSs presented in chapter 5 and illustrates the technologies used to realize the deduplication workflow phases. The data storage and processing layer is based on a CDH 4.3 Hadoop Cluster[5] composed of 8 worker nodes. Each worker node has 8 cpu cores, 24Gb of RAM, and the total HDFS capacity is about 14.5Tb. The aggregation system runs on dedicated virtual machines, in particular the D-Net Manager Service (*MS*) introduced in 6.2 coordinates and supervises the activities on the Hadoop cluster.

[3] Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego (ICM) - `http://www.icm.edu.pl`

[4] `https://issue.openaire.research-infrastructures.eu/projects/openaire2020-wiki/wiki/D61_OpenAIRE_Specification_and_Release_Plan`

[5] `http://www.cloudera.com/documentation/archive/cdh/4-x/4-3-0.html`

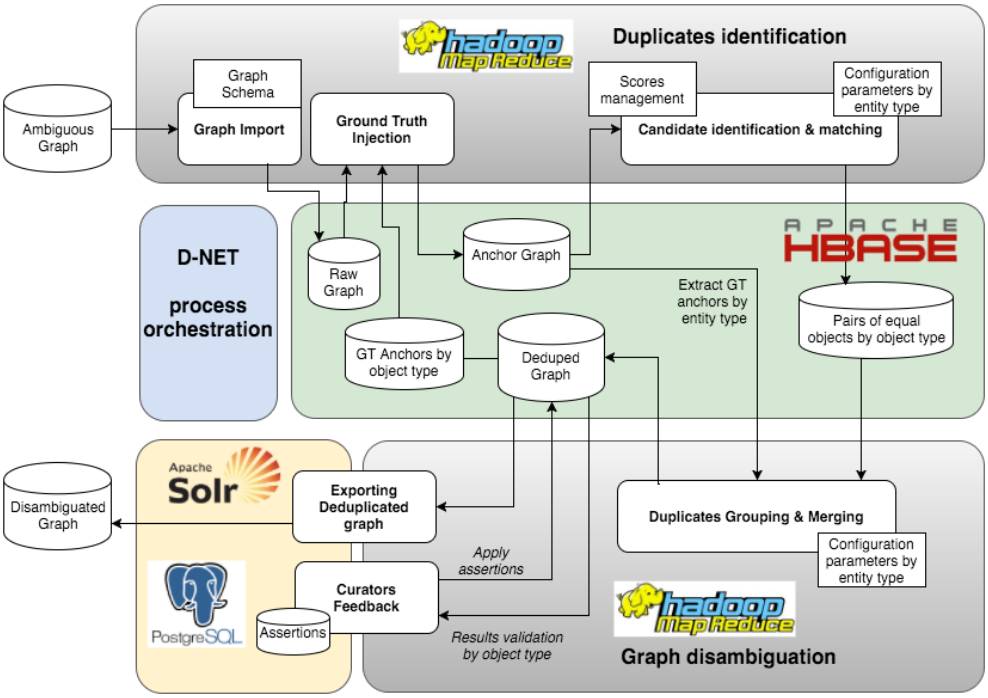


Fig. 7.2: GDup: Technological Architecture

7.3 Graph deduplication: publication entity

The OpenAIRE information space in February 2016 counts more than 16 million publications. These objects are collected from a multitude of data providers of different typologies:

Institutional or thematic repositories Information systems where scientists upload the bibliographic metadata and PDFs of their articles, due to obligations from their organization or due to community practices (e.g. ArXiv, EuropePMC);

Open Access Publishers Information system of open access publishers or relative journals, which offer bibliographic metadata and PDFs of their published articles;

Aggregator services Information systems that, like OpenAIRE, collect descriptive metadata about publications or datasets from multiple sources in order to enable cross-data source discovery of given research products; aggregators tend to be driven by research community needs or to target the larger audience of researchers across several disciplines; examples are DataCite for all research data with DOIs as persistent identifiers, BASE for scientific publications, DOAJ for Open Access journals publications.

As OpenAIRE aims to realize a pan-European network of interoperable publication (any beyond) repositories, by promoting the Open Science policies, it also contributes

to give visibility to the participating repositories. The presence of thematic, Institutional repositories, OA Publishers and aggregators causes a significant overlap in the set of publications that OpenAIRE collects. The presence of duplicates (estimated in the range of 10-15% at the beginning of the project) causes a substantial skew in the aggregated statistics describing the information space, introduces ambiguity to both third party services that need to perform further processing on the OpenAIRE data and, due to the different completeness of the metadata records provided by different data sources, to the users that search for publications on the OpenAIRE portal.[6]

To illustrate the benefits of identifying and resolving the duplicates in the OpenAIRE information space, we provide an example of two statistics that represent an important added value for decision makers. The statistic illustrated in figure 7.3 describes the number of publications linked to FP7 projects in the 2007-2015 time frame, while figure 7.4 describes the distribution of the same publications among the different funding programmes. The presence of duplicates would have caused a considerable skew: in the first case the distribution would have had an higher mean, with a trend that would have depended on the duplicates distribution over the years. In the second case the percentage of FP7 publications in each funding programme would have depended on the duplicates distribution, causing an unpredictable and unrealistic relationship between the funding areas. The deduplication process was therefore necessary to cope with the issues mentioned above.

[6] `https://www.openaire.eu`

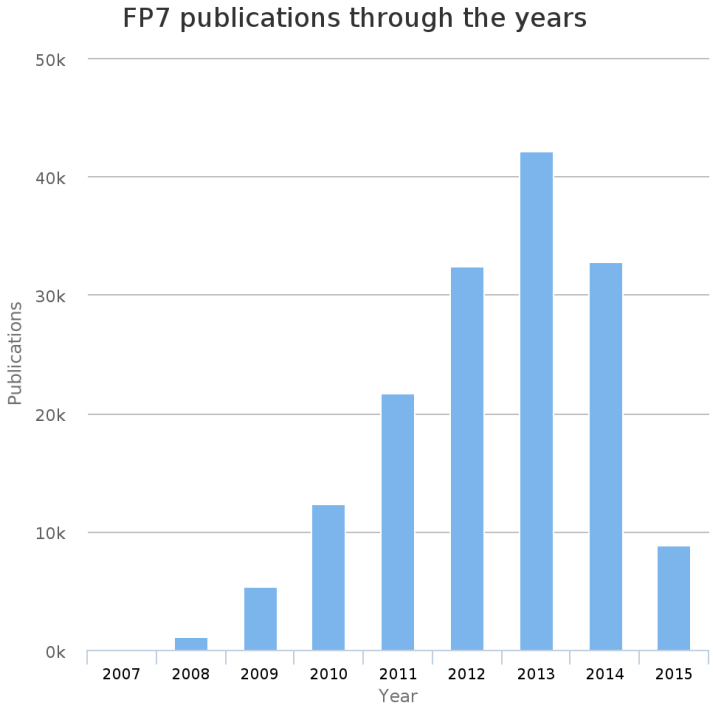


Fig. 7.3: FP7 Publications through the years.

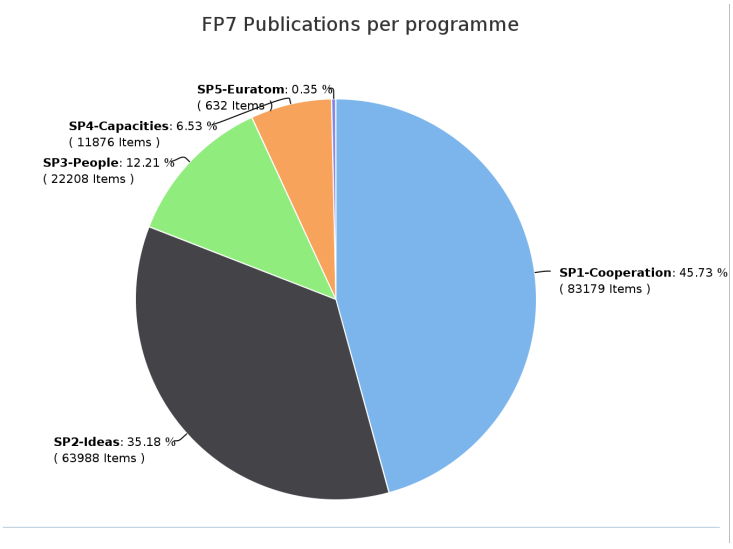


Fig. 7.4: FP7 Publications per programme.

On the bright side the different level of completeness and compliance with the OpenAIRE guidelines of the metadata record enables the object merge procedure to build a richer object, that maintains the original characteristics of the duplicate records, but provides a more complete set of information to the end users. For example in figure 7.5 shows a publication whose Open Access fulltext can be found in three different repositories, meaning that the same metdata record was collected by OpenAIRE from three distinct data sources.

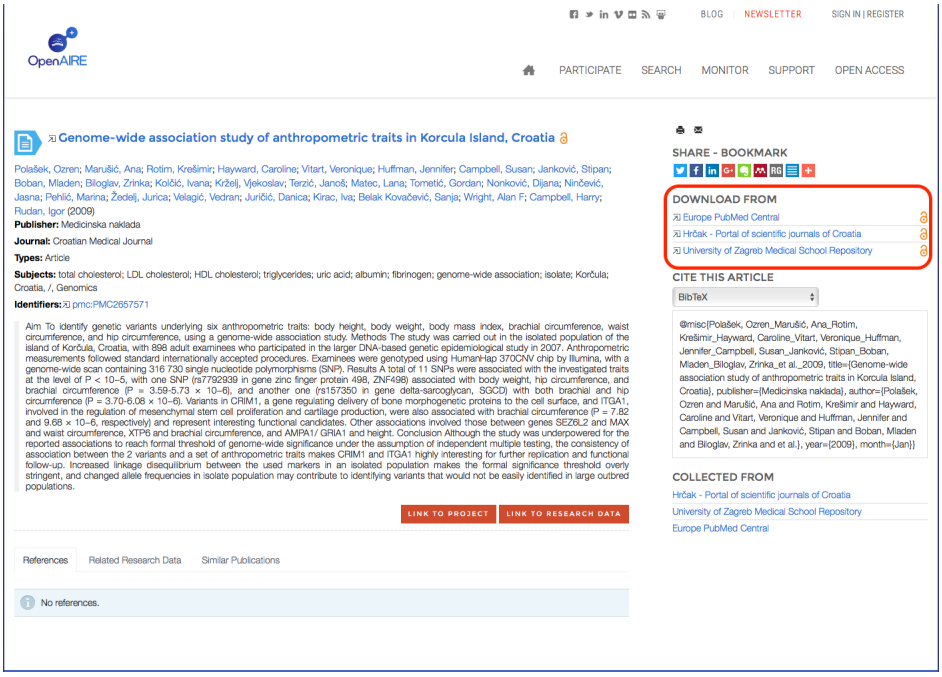


Fig. 7.5: Publication duplicates on the OpenAIRE portal.

The deduplication process run on the 16 million publications aggregated on the production system is summarized in table 7.1. The execution times refer to the processes run on the Hadoop cluster described in section 7.2.

Phase	Execution time	Output
Candidate identification	$\sim 45'$	2.6M clusters 1.5M clusters $(size > 1)$
Candidate matching	$2h$	2.1B comparisons 7M *equalsTo* relationships
Connected components	$\sim 45'$	1.8M connected components
Root construction & Relationships redirection	$\sim 1h15'$	4.2M pubs marked as duplicates

Table 7.1: Publications, deduplication statistics (update to date 2015-12)

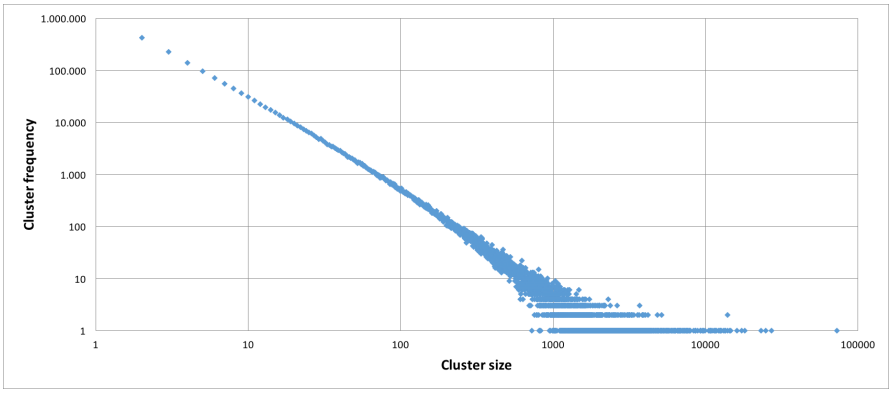


Fig. 7.6: Publication candidates clustering distribution.

Figure 7.6 reports the distribution of the cluster sizes of publications candidates, produced by the clustering functions defined in the configuration used in the production system. Highly frequent clusters typically include few records, with 2.4 million groups of two, while larger groups occur less frequently.

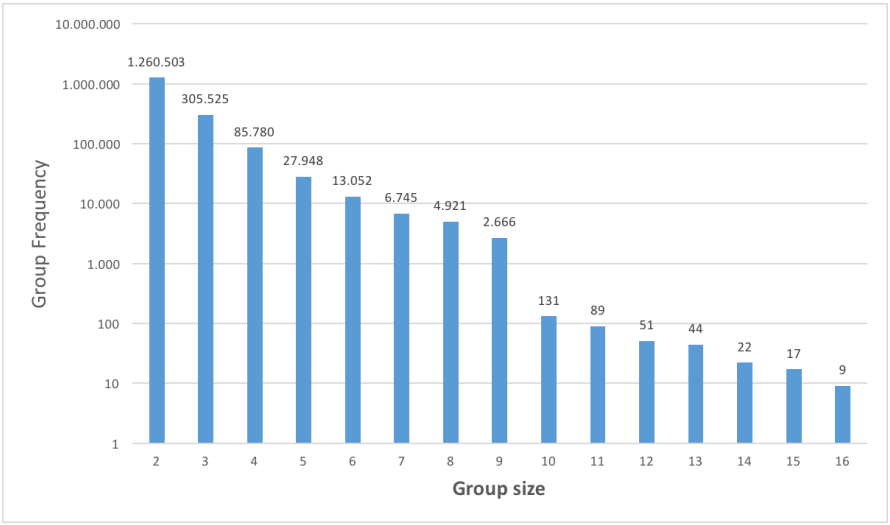


Fig. 7.7: Publication groups distribution

The results of the deduplication workflow is depicted in figure 7.7, which shows the duplicates group distribution. It produces 1.8 million representative publications, grouping 4.2 million duplicates, and the average number of duplicates per group is 2.4. Overall the total execution time for the deduplication workflow on the publication entities makes it possible to execute it on a regular basis (typically once a month), allowing also the possibility to perform extra executions in response to large changes in the aggregation status (typically the addition of a large repository). Therefore it didn't require the introduction of a dedicated Ground Truth.

7.4 Graph deduplication: organization entity

Organizations aggregated in OpenAIRE include companies, institutions or research centers involved as project partners or that are responsible for operating data sources. In February 2016 OpenAIRE counts 68,488 organizations, generally collected from CRIS[7] systems and entity registries. Their duplication is mainly due to intra-data source logics, since they are collected from a few data sources where they typically appear as a secondary entity, not subject to disambiguation, and in some cases they totally lack of any kind of local identifier.

Organization objects identifiers are obtained in a stateless way, when organizations are first-level objects in the original data source (i.e. they are assigned a local original identifier) or second-level object data sources (i.e. they are extracted from the metadata

[7] Current research information system - `https://en.wikipedia.org/wiki/Current_research_information_system`

information of a first-level object in the data source). As a consequence, all objects have a unique identifier in the system, even if they represent the same organization.

For example the organizations participating to the research projects that OpenAIRE collects from the CORDIS[8] have a local identifier:

```
<Project xsi:noNamespaceSchemaLocation="XSD_OpenAire.xsd">
  <ProjectId>322699</ProjectId>
  <ProjectCallWebpageUrl>http://cordis.europa.eu/fp7/home_en.html</
      ProjectCallWebpageUrl>
  <ProjectCallIdentifier>ERC-2012-ADG_20120314</ProjectCallIdentifier>
  <ProjectAcronym>THE FUSION MACHINE</ProjectAcronym>
  <ProjectTitle>The nanomechanical mechanism of exocytotic fusion pore formation</
      ProjectTitle>
  <ProjectStartDate>01APR2013:00:00:00</ProjectStartDate>
  <ProjectFundingScheme>ERC</ProjectFundingScheme>
  <ProjectFundingSchemeDescr>Support for frontier research (ERC)</
      ProjectFundingSchemeDescr>
  <ProjectSubFundingScheme>ERC-AG</ProjectSubFundingScheme>
  <ProjectSubFundingSchemeDescr>ERC Advanced Grant</
      ProjectSubFundingSchemeDescr>
  <ProjectEndDate>31MAR2018:00:00:00</ProjectEndDate>
  <SpecialClause39>N</SpecialClause39>
  <ProjectFramework>FP7</ProjectFramework>
  <ProjectSpecificProgram>SP2</ProjectSpecificProgram>
  <ProjectSpecificProgramDescr>SP2-Ideas</ProjectSpecificProgramDescr>
  <ProjectProgram>ERC</ProjectProgram>
  <ProjectProgramDescr>ERC</ProjectProgramDescr>
  <CoordinatorContactPerson>
    <PersonRole>Contact Person</PersonRole>
    <PersonLastName>Messerschmidt</PersonLastName>
    <PersonFirstName>Manfred</PersonFirstName>
    <ContactEmail>eu-goe@gwdg.de</ContactEmail>
    <ContactFunction>Head Of Administration</ContactFunction>
    <ContactPhone>+49 551 2011221</ContactPhone>
  </CoordinatorContactPerson>
  <OrganisationParticipant>
    <OrganisationPIC>999990267</OrganisationPIC>
    <ParticipantOrder>1</ParticipantOrder>
    <OrganisationShortName>MPG</OrganisationShortName>
    <OrganisationLegalName>MAX PLANCK GESELLSCHAFT ZUR FOERDERUNG DER
        WISSENSCHAFTEN E.V.</OrganisationLegalName>
    <OrganisationWebPage>www.mpg.de</OrganisationWebPage>
    <OrganisationCountry>DE</OrganisationCountry>
     [...]
```

Listing 7.2: Organizations from CORDIS

[8] CORDIS - `http://cordis.europa.eu`

On the other hand, the Directory of Open Access Repositories (OpenDOAR[9]) contributes to OpenAIRE by providing metadata about institutional repositories. In such metadata records, the organization responsible to operate a given repository is provided as a secondary entity, and does not have any identifier associated:

```
<repository rID="610">
  <rName>Fraunhofer−ePrints</rName>
  <rAcronym/>
  <rNamePreferred>Y</rNamePreferred>
  <rUrl>http :// eprints . fraunhofer . de/</rUrl>
  <rOaiBaseUrl>http://publica.fraunhofer.de/jsp/PublicaHarvester</rOaiBaseUrl>
  <uName>Forderung der angewandten Forschung e.V.</uName>
  <uAcronym/>
  <uNamePreferred>Y</uNamePreferred>
  <uUrl/>
  <oName>Fraunhofer−Gesellschaft</oName>
  <oAcronym>FHG</oAcronym>
  <oNamePreferred>Y</oNamePreferred>
  <oUrl>http :// www.fraunhofer.de/</oUrl>
   [...]
```

Listing 7.3: Organizations from OpenDOAR

Reliable deduplication results for the organizations are important in OpenAIRE because of the features provided by the portal to end users. Specifically the portal allows researchers to deposit their works (publications and datasets) in OpenAIRE by guiding them with a dedicated procedure. As depicted in figure 7.8 users are asked to identify their repository of reference by searching the institution they are affiliated with. Identifying the repository associated with an organization that was aggregated by OpenDOAR it's a trivial task because the relationship between the two entities existed in the information package from which the OpenAIRE entities were extracted. On the contrary, identifying the repository associated with an organization that exists in OpenAIRE because it was collected from an entity registry that provides projects is a much more challenging task. Originally such organizations does not provide any relationship that links them to a repository, however when the deduplication system identifies two organizations as duplicates, the merge process enriches the representative record with the attributes and relationship of all the contributing records. In this way, users queries matching an organization collected from CORDIS that was merged with the corresponding organization collected from OpenDOAR, are more likely to produce a result that allows users to locate their repository of reference.

[9] The Directory of Open Access Repositories - `http://www.opendoar.org`

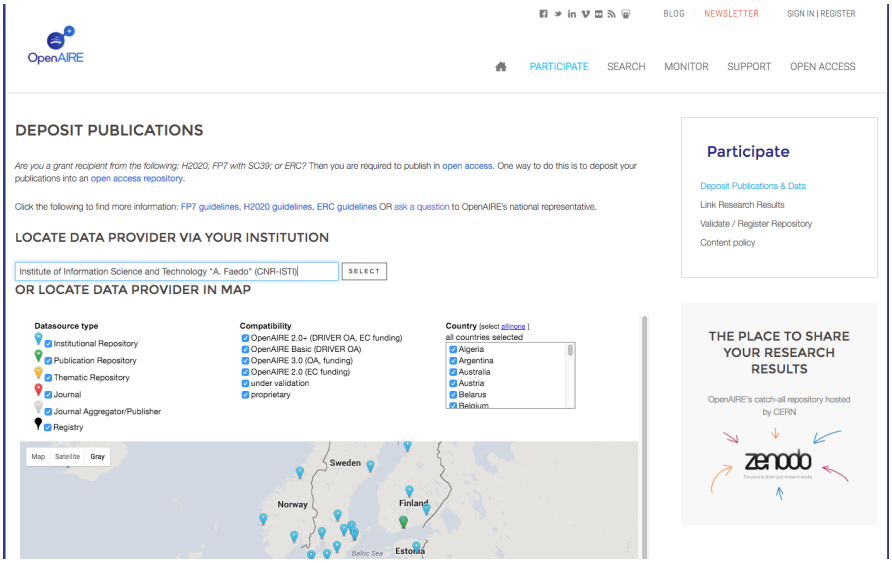


Fig. 7.8: OpenAIRE deposition functionality

The deduplication process run on the 68 thousands organizations aggregated on the production system is summarized in table 7.2. The execution times refer to the processes run on the Hadoop cluster described in section 7.2.

Phase	**Execution time**	**Output**
Candidate identification	$\sim 1'$	134K clusters 26K clusters $(size > 1)$
Candidate matching	$\sim 45''$	393K comparisons 23K *equalsTo* relationships
Connected components	$\sim 45''$	5K connected components
Root construction & Relationships redirection	$\sim 2'30''$	13K orgs marked as duplicates

Table 7.2: Organizations, deduplication statistics (update to date 2015-12)

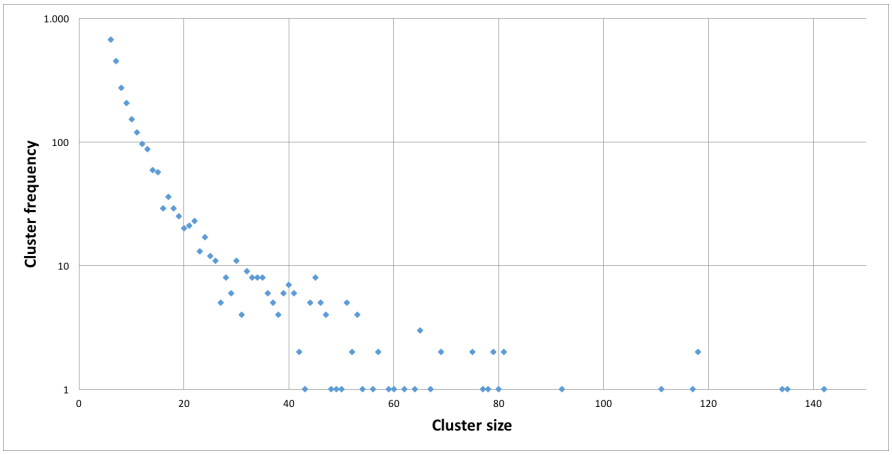


Fig. 7.9: Organization candidates clustering distribution.

Figure 7.9 reports the distribution of the cluster sizes of organizations candidates, produced by the clustering functions defined in the configuration used in the production system. Highly frequent clusters typically include few records, with 672 groups of six records, while larger groups occur less frequently.

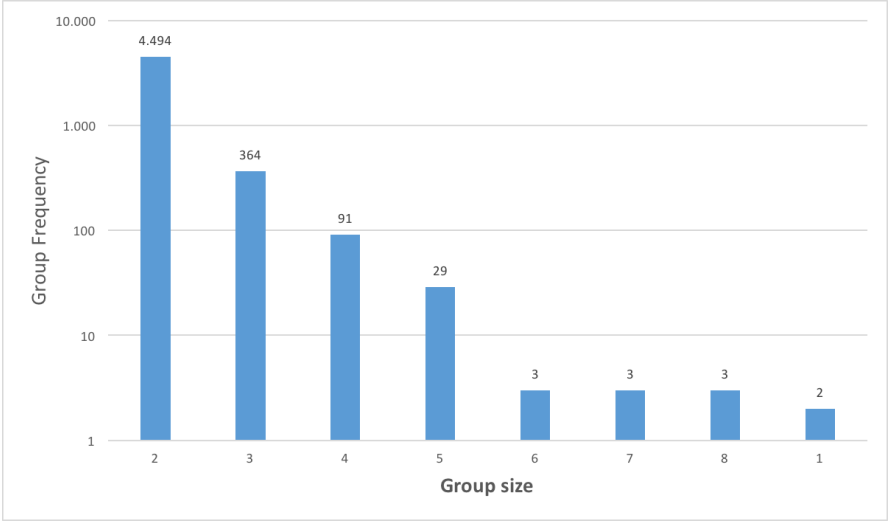


Fig. 7.10: Organization groups distribution

The results of the deduplication workflow is depicted in figure 7.10, which shows the duplicates group distribution. It produces 4989 representative organizations, grouping 10654 duplicates, and the average number of duplicates per group is 2.13. The number of organization entities is considerably lower than publications, which translates in low execution times for deduplication workflow, that makes it possible to execute in response to every change in the aggregation status of the organization entities (typically the addition of a new entity registry, or the update of an existing registry). Therefore it didn't require the introduction of a dedicated Ground Truth.

7.5 Graph deduplication: person entity

Persons in OpenAIRE are individuals that have one (or more) role(s) in the research domain, such as authors of publications or project coordinators. Duplicated person objects in OpenAIRE are mostly caused by two factors:

- the lack of a global persistent identifier provided within the publication metadata records. As for today, only a few repositories include author PIDs;
- the absence of a standard model to represent structured metadata for publication authors in the OpenAIRE guidelines.

As consequence, each authors is associated with a unique local identifier generated by the OpenAIRE aggregation system.

This causes intra data source duplicates for authors to be very common, in fact all publications of the same author in a repository will generate a new author object in the graph, potentially different from the others. On the other hand, cross-data source duplicates are mainly due to the fact that co-authors are depositing in their institutional repositories or in thematic repositories (e.g. ArXiv, EuropePMC), or again, as in the previous case, their publications have been collected by an aggregator data source harvested in OpenAIRE. As result, the set of *raw* persons can be estimated in 56 million objects by considering ~ 3.5 average authors per publication[10] and the 16 million publications aggregated in OpenAIRE.

Differently from publications, where the average number of duplicates is between 2 and 3, in case of persons the average number of duplicates is not a representative statistic, as it may vary considerably, depending on the specific research field or the repository of reference of a given author. Moreover, in general it is common for an author to have a number of papers in his curriculum as a multiple of 10, therefore the groups of duplicates in the set of raw author objects can easily tend to be large, introducing major overhead in the graph disambiguation phase. For this purpose, it was decided to handle the deduplication for person entities by making use of a Ground Truth approach. By introducing a Ground Truth we have two types of persons, *anchor* and *raw* persons, whose importance is different in terms of selection criteria.

[10] https://www.nlm.nih.gov/bsd/authors1.html

As specified when introducing the Ground Truth, the relative methodology envisages that part of deduplication is resolved while creating the graph, by using the Ground Truth as a map between raw objects and the corresponding anchor objects, if any. This approach changes the perspective of the deduplication process, since part of the deduplication can be made available before candidate identification by mapping raw objects onto anchor objects at graph import time. Such pre-processing can considerably reduce the time to process the whole collection of person objects, limiting the identification of duplicates to the objects that are not yet part of the Ground Truth.

```
1  EntityType = Person,
2
3  clusteringFunction[
4   name = personClustering
5   params = [max=5]
6    fields  = [fullname]
7  ]
8
9  similarityFunction [
10       name = anchorsInCommon
11       weight = 0.6
12        fields  = [anchors]
13       params = [n=1]
14 ]
15 similarityFunction [
16       name = coauthorsInCommon
17       weight = 0.4
18        fields  = [coauthors]
19       params = [n=2]
20 ]
```

Listing 7.4: Person match configuration

Listing 7.4 summarizes the idea behind the incremental approach used for person deduplication. Person objects are first matched against the hash values produced by the function `personClustering`, which tries to define a structure for person names by separating first name and surname. In case the heuristic is satisfied, it emits hash values in the form of first character of the firs name, followed by the surname. Otherwise it generate an hash value for each word identified in the first name. Candidates are then matched by evaluating the intersection between the of sets coauthors identified as anchors, and the surnames of the raw coauthors, whose size must match a minimum value.

A key player in this process was identified as the initial Ground Truth, i.e. the first set of *anchor* person objects considered as valid. In order to build such set, we used the results of publication deduplication, considering the assumption that the authors of two publications identified to be same, are as well the same authors. In order to extend

such set we matched it against selected ORCID[11] profiles, obtained by processing the ORCID Public Data file[12], counting more than 1.6 million researcher profiles, each of them associated with a persistent identifier. Although promising, as for today, it contains only 160.000 profiles providing a list of works, together with the relative coauthors. We believe in the next years researchers will continue to curate their profiles by adding their works, as researchers today are facing the need to distinguish their research activities from those of others with similar names.

[11] "ORCID is an open, non-profit, community-driven effort to create and maintain a registry of unique researcher identifiers and a transparent method of linking research activities and outputs to these identifiers" - `http://orcid.org/about/what-is-orcid`

[12] https://orcid.org/content/download-file

8

Conclusions

In this chapter we summarize the main contributions of this thesis and we outline the future directions.

8.1 Summary

The amount of data produced today calls for innovative technologies and solutions to tackle the multitude of challenges arising in several research fields of data management. Among others, also entity deduplication is living a period of Big Data renaissance and extending this problem to large graphs of interlinked objects becomes an interesting and challenging problem. Orthogonally, accurate resolution is important for a variety of reasons ranging from accurate analysis for critical applications and cost-effectiveness, to reduction in data volume.

Semi-automatic approaches are needed to reduce as much as possible the manual effort and to assist data curators in solving the domain specific disambiguation issues. This thesis represents an advancement in the state of the art in deduplication systems by proposing new solutions to address the deduplication issue on large graphs of interconnected entities.

The main contributions can be summarized as follows.

- **Survey of deduplication tools** We provided a survey of the existing tools that support end users facing the deduplication problem, classifying the solutions based on the fact they address or not address the non-functional requirement of scalability posed by large data collections.
- **Big Graph Entity Deduplication Systems** The main contribution is having provided a reference architecture for Big Graph Entity Deduplication Systems, integrated, scalable, general-purpose systems for entity deduplication over Big Graphs. BGEDSs are intended to support data curators with the end-user functionalities they need to realize all workflow phases of duplicates identification and graph disambiguation over Big Graphs. BGEDSs propose a novel workflow to cover all the phases involved in the

graph disambiguation task, from the graph acquisition, through the duplicates identification and matching, to the graph disambiguation and finally to the materialization of the disambiguated graph. The overall process can be supervised by data curators that can configure, evaluate, and refine the results.

- **GDup** We presented an implementation of the BGEDS reference architecture, called GDup. It is based on Apache HBase and Hadoop Map Reduce paradigm, today instantiated to operate as part of the production environment of the OpenAIRE infrastructure to support data curators.

8.2 Future work

Although this work provides a contribution to improve the state of the art in the deduplication field, it also raises several opportunities for improvement and further challenges to be addressed in future work.

We outline some of them in the following.

- **Parallel entity resolution & graph disambiguation** The wide adoption of distributed and parallel processing technologies and techniques has significantly influenced the research of efficient deduplication methods capable to process large collections. The literature shows a spreading interest for such techniques, ranging from the exploitation of multiple cpu cores of a single node [15, 45, 7], to the distribution of the process to multiple nodes [46].

  The the Hadoop Map Reduce framework has proven to be a key enabler for research in object matching, counting a considerable number of contributions [83, 49, 48, 47, 50, 51], moreover the Hadoop ecosystem has grown substantially in recent years, providing powerful tools to support different kinds of data intensive tasks. For example, Apache Hama[1] provides Big Data analytics based on the Bulk Synchronous Parallel (BSP) computing model, or Apache GraphX[2] provides an API for graphs and graph parallel computation. The availability of powerful tools supporting different kinds of data intensive tasks can help to further improve techniques such as collective entity resolution in relational data [8].
- **Linked Open Data & Entity Resolution** Among the major challenges involved in the entity resolution is the need to cope with low quality data, often not expressive enough to be disambiguated effectively. The growing adoption of Linked Open Data, and the consequent development of methods and tools to support the automatic link discovery [33, 68, 84, 19, 67] can contribute to reduce significantly the manual effort needed to enrich incomplete data. Linked Open Data can therefore support the introduction of additional sources of information in the graph disambiguation workflow proposed in this thesis.

[1] `http://hama.apache.org`
[2] `https://spark.apache.org/graphx`

- **Generic framework for graph link discovery** The workflow proposed in this thesis is meant to identify and resolve duplicates objects in a graph, however depending on the configuration, the process could identify other kind of links between the objects in the graph. For example multiple versions of an article (pre-print and post-print), the structure of an organization (laboratory, department, university).

9

Acknowledgements

As I had the pleasure to find out in the last years, pursuing my Ph.D. project has been a both painful and enjoyable experience. Pain, frustration, hardships, encouragement, trust and joy are just a few of the endless moods that accompanied me during this journey. By looking back today I realise that what brought me here was, in the end, teamwork. Without the support of all the people I had the honour to find alongside me on my path I wouldn't have never made it.

First and foremost, I must acknowledge the person who most contributed to this work is my supervisor, Dr. Paolo Manghi. He has been a continuous source of inspiration and precious suggestions, especially when I wasn't so sure I could actually get to the end of this work.

Many thanks goes to my supervisor Prof.ssa Cinzia Bernardeschi for accepting my Ph.D. project, the advices and the support during these years.

I would like to express my greatest gratitude to my colleagues at the Networked Multimedia Information Systems Laboratory of the ISTI - CNR, since I joined I've always felt as part of the group. I'd like to extend my gratitude to Donatella Castelli, for being able to realise an incredible working group, made of tremendous professional and human value.

A special mention goes to Marko Mikulicic, whose skills and passion for computer science have been precious lessons. Next, my gratitude goes to my "roommates": Alessia Bardi, Andrea Mannocci, Sandro La Bruzzo and Michele Artini. Thanks for being there every day. Thanks for making me smile, even when I lost my sense of humor. Thanks for being such a great team. Thanks for all the fish.

Now, best practices suggests to structure acknowledgements from thanking the most formal support to the least formal thanks as, in general – supervisors, other academics, colleagues, and finally family. However, although it might seem to be redundant, at this point I found appropriate to mention again you, Paolo, and this is not because I just want to thank you twice, but I think you can't just stand up there among the formal acknowledgments! I realise that every achievement we get it's because we work as a team, I realise

your capacity to be the glue that keep us together as a working group. I realise that you thought me (and I still have a lot to learn) the importance of teamwork, and I believe this one of the most important lessons I had the honour to learn so far. Thanks for believing in me, I owe you a lot!

I feel in debt with my friends in Pisa, your contribution in preserving my sanity in the few spare time I had during the last years has been priceless.

Of course no acknowledgments would be complete without giving thanks to my family. A huge thanks goes to my parents, Rosanna and Mauro. You have been my first supporters for every choice I made in my life. Mom, your constant dedication through the years represents for me an outstanding model of strength, resilience, and love. I owe you everything I am today.

My warmest gratitude goes also to my sisters Roberta and Carla, and to my life-brother Fabio Sinibaldi. You have the power to give me inspiration and strength in everything I do. I must thank you just for being part of my life!

Last, but certainly not least, I must acknowledge with tremendous and deep thanks my partner, Elena Pinzauti. You (and Bruno) fill my heart with joy every day. I would not be able to get to the end of this journey without your support, and no words can express my gratitude and appreciation for all you've done and been for me. Thank you from my soul and heart.

References

[1] Arvind Arasu, Cristina Re, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 952–963. IEEE, 2009.

[2] Michele Artini, Claudio Atzori, Alessia Bardi, Sandro La Bruzzo, Paolo Manghi, and Andrea Mannocci. The openaire literature broker service for institutional repositories. *D-Lib Magazine*, 21(11):3, 2015.

[3] Michele Artini, Claudio Atzori, Alessia Bardi, Sandro La Bruzzo, Paolo Manghi, Mirko Mikulicic, and Franco Zoppi. The heritage of the people's europe project: an aggregative data infrastructure for cultural heritage. Communications in Computer and Information Science, pages 77 – 80. Springer, 2014. In: Bridging Between Cultural Heritage Institution. pp. 77 - 80. Tiziana Catarci, Nicola Ferro, Antonella Poggi (eds.). (Communications in Computer and Information Science, vol. 385). Berlin: Springer, 2014.

[4] Michele Artini, Claudio Atzori, and Paolo Manghi. Keeping your aggregative infrastructure under control. In *IEEE/ACM Joint Conference on Digital Libraries*, pages 409 – 410. IEEE, 2014. In: JCDL - IEEE/ACM Joint Conference on Digital Libraries (London, 8-12 September 2014). Proceedings, pp. 409 - 410. (IEEE Conference Publications). IEEE, 2014.

[5] Baruch Awerbuch and Yossi Shiloach. New connectivity and msf algorithms for shuffle-exchange network and pram. *Computers, IEEE Transactions on*, 100(10):1258–1263, 1987.

[6] Rohan Baxter, Peter Christen, Tim Churches, et al. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27. Citeseer, 2003.

[7] Omar Benjelloun, Hector Garcia-Molina, Heng Gong, Hideki Kawai, Tait E Larson, David Menestrina, and Sutthipong Thavisomboon. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *Distributed Computing Systems, 2007.*

ICDCS'07. 27th International Conference on, pages 37–37. IEEE, 2007.

[8] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.

[9] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

[10] David Guy Brizan and Abdullah Uz Tansel. A. Survey of Entity Resolution and Record Linkage Methodologies. *Communications of the IIMA*, 6(3):5, 2015.

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26:4, 2008.

[12] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2003.

[13] Peter Christen. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1065–1068. ACM, 2008.

[14] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.

[15] Peter Christen, Tim Churches, and Markus Hegland. Febrl–a parallel open source data linkage system. In *Advances in knowledge discovery and data mining*, pages 638–647. Springer, 2004.

[16] Peter Christen and Karl Goiser. Quality and complexity measures for data linkage and deduplication. In *Quality Measures in Data Mining*, pages 127–151. Springer, 2007.

[17] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[18] William W Cohen, Pradeep D Ravikumar, Stephen E Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, volume 2003, pages 73–78, 2003.

[19] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann De Meer. idmesh: graph-based disambiguation of linked data. In *Proceedings of the 18th international conference on World wide web*, pages 591–600. ACM, 2009.

[20] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[21] Ramesh Dharavath and Abhishek Kumar Singh. Entity resolution-based jaccard similarity coefficient for heterogeneous distributed databases. In *Proceedings of the Second International Conference on Computer and Communication Technologies*, pages 497–507. Springer, 2016.

[22] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

[23] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[24] Craig Franke, Samuel Morin, Artem Chebotko, John Abraham, and Pearl Brazier. Distributed semantic web data management in hbase and mysql cluster. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 105–112. IEEE, 2011.

[25] Borko Furht and Armando Escalante. *Handbook of data intensive computing*. Springer Science & Business Media, 2011.

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

[27] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[28] Rob Hall, Charles Sutton, and Andrew McCallum. Unsupervised deduplication using cross-field dependencies. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 310–317, New York, NY, USA, 2008. ACM.

[29] Olaf Hartig. Reconciliation of rdf* and property graphs. *arXiv preprint arXiv:1409.3288*, 2014.

[30] Nathalie Henry, Howard Goodell, Niklas Elmqvist, and Jean-Daniel Fekete. 20 years of four hci conferences: A visual exploration. *International Journal of Human-Computer Interaction*, 23(3):239–285, 2007.

[31] Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD Record*, volume 24, pages 127–138. ACM, 1995.

[32] Daniel S. Hirschberg, Ashok K. Chandra, and Dilip V. Sarwate. Computing connected components on parallel computers. *Communications of the ACM*, 22(8):461–464, 1979.

[33] Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann. Some entities are more equal than others: statistical methods to consolidate linked data. In *4th International Workshop on New Forms of Reasoning for the Semantic Web:*

Scalable and Dynamic (NeFoRS2010), 2010.

[34] Sue-Chen Hsueh, Ming-Yen Lin, and Yi-Chun Chiu. A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. In *Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing-Volume 152*, pages 3–9. Australian Computer Society, Inc., 2014.

[35] Alice M Jaques, David J Amor, HW Gordon Baker, David L Healy, Obioha C Ukoumunne, Sue Breheny, Claire Garrett, and Jane L Halliday. Adverse obstetric and perinatal outcomes in subfertile women conceiving without assisted reproductive technologies. *Fertility and sterility*, 94(7):2674–2679, 2010.

[36] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[37] Alan Jarvis. Introducing Thomson Reuters Books Citation Index. *Editors' Bulletin*, 6(3):102–103, nov 2010.

[38] Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on*, pages 137–146. IEEE, 2003.

[39] Martin Junghanns, André Petermann, Kevin Gómez, and Erhard Rahm. Gradoop: Scalable graph data management and analytics with hadoop. *arXiv preprint arXiv:1506.00548*, 2015.

[40] Pawel Jurczyk, James J Lu, Li Xiong, Janet D Cragan, and Adolfo Correa. FRIL: A tool for comparative record linkage. In *AMIA annual symposium proceedings*, volume 2008, page 440. American Medical Informatics Association, 2008.

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

[42] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[43] Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani Thuraisingham, and Paolo Castagna. Jena-hbase: A distributed, scalable and efficient rdf triple store. In *Proceedings of the 11th International Semantic Web Conference Posters & Demonstrations Track, ISWC-PD*, volume 12, pages 85–88. Citeseer, 2012.

[44] Ankur Khetrapal and Vinay Ganesh. Hbase and hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*, pages 22–28, 2006.

[45] Hung-sik Kim and Dongwon Lee. Parallel linkage. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 283–292. ACM, 2007.

[46] Toralf Kirsten, Lars Kolb, Michael Hartung, Anika Groß, Hanna Köpcke, and Erhard Rahm. Data partitioning for parallel entity matching. *arXiv preprint arXiv:1006.5309*, 2010.

[47] Lars Kolb and Erhard Rahm. Parallel entity resolution with Dedoop. *Datenbank-Spektrum*, 13(1):23–32, 2013.

[48] Lars Kolb, Ziad Sehili, and Erhard Rahm. Iterative computation of connected graph components with mapreduce. *Datenbank-Spektrum*, 14(2):107–117, 2014.

[49] Lars Kolb, Andreas Thor, and Erhard Rahm. Parallel sorted neighborhood blocking with mapreduce. *arXiv preprint arXiv:1010.3053*, 2010.

[50] Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 618–629. IEEE, 2012.

[51] Lars Kolb, Andreas Thor, and Erhard Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science-Research and Development*, 27(1):45–63, 2012.

[52] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[53] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[54] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD Record*, 40(4):11–20, 2012.

[55] VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, page 707, 1966.

[56] Paolo Manghi, Michele Artini, Claudio Atzori, Alessia Bardi, Andrea Mannocci, Sandro La Bruzzo, Leonardo Candela, Donatella Castelli, and Pasquale Pagano. The d-net software toolkit: A framework for the realization, maintenance, and operation of aggregative infrastructures. *Program*, 48(4):322–354, 2014.

[57] Paolo Manghi, Alessia Bardi, Claudio Atzori, Michele Artini, Andrea Dell'Amico, and Sandro La Bruzzo. Openaireplus - openaire apis for third party services. d8.6. Deliverable, D8.6, 2014. OpenAIREplus, 2nd Generation Open Access Infrastructure for Research in Europe. Deliverable D8.6, 2014.

[58] Paolo Manghi, Lukasz Bolikowski, Natalia Manold, Jochen Schirrwagen, and Tim Smith. OpenAIREplus: the European Scholarly Communication Data Infrastructure. *D-Lib Magazine*, 18(9/10), sep 2012.

[59] Paolo Manghi, Nikos Houssos, Marko Mikulicic, and Brigitte Jörg. The data model of the openaire scientific communication e-infrastructure. In *Metadata and Semantics Research*, pages 168–180. Springer, 2012.

[60] Paolo Manghi, Natalia Manola, Wolfram Horstmann, and Dale Peters. An infrastructure for managing ec funded research output-the openaire project. *The Grey Journal (TGJ): An International Journal on Grey Literature*, 6(1), 2010.

[61] Paolo Manghi, Marko Mikulicic, and Claudio Atzori. De-duplication of aggregation authority files. *International Journal of Metadata, Semantics and Ontologies*, 7(2):114–130, 2012.

[62] Andrew McCallum, Kamal Nigam, and Lyle 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*, pages 169–178. ACM, 2000.

[63] N McNeill, Hakan Kardes, and Andrew Borthwick. Dynamic record blocking: efficient linking of massive databases in mapreduce. In *Proceedings of the 10th International Workshop on Quality in Databases (QDB)*, 2012.

[64] Matthew Michelson and Craig A Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 440. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[65] John W Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.

[66] Howard B Newcombe, James M Kennedy, SJ Axford, and Allison P James. Automatic linkage of vital records computers can be used to extract" follow-up" statistics of families from files of routine records. *Science*, 130(3381):954–959, 1959.

[67] Andriy Nikolov, Victoria Uren, Enrico Motta, and Anne De Roeck. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *The Semantic Web*, pages 332–346. Springer, 2009.

[68] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pages 85–94. ACM, 2011.

[69] Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Akash Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 50–61. IEEE, 2013.

[70] John Rathbone, Matt Carter, Tammy Hoffmann, and Paul Glasziou. Better duplicate detection for systematic reviewers: evaluation of systematic review assistant-

deduplication module. *Systematic Reviews*, 4(1):1–6, 2015.

[71] Eric Sven Ristad and Peter N Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.

[72] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. " O'Reilly Media, Inc.", 2015.

[73] Marko A Rodriguez and Peter Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[74] Alex Rudniy, Min Song, and James Geller. Detecting duplicate biological entities using shortest path edit distance. *International journal of data mining and bioinformatics*, 4(4):395–410, 2010.

[75] W. Santos, T. Teixeira, C. Machado, W. Meira, A.S. Da Silva, D.R. Ferreira, and D. Guedes. A scalable parallel deduplication algorithm. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 79–86, Oct 2007.

[76] Thomas Seidl, Brigitte Boden, and Sergej Fries. Cc-mr–finding connected components in huge graphs with mapreduce. In *Machine Learning and Knowledge Discovery in Databases*, pages 458–473. Springer, 2012.

[77] Sarah L Sheridan, Robert S Ware, Keith Grimwood, and Stephen B Lambert. Febrile seizures in the era of rotavirus vaccine. *Journal of the Pediatric Infectious Diseases Society*, page piu097, 2014.

[78] Yossi Shiloach and Uzi Vishkin. An o (logn) parallel connectivity algorithm. *Journal of Algorithms*, 3(1):57–67, 1982.

[79] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[80] Jianling Sun and Qiang Jin. Scalable rdf store based on hbase and mapreduce. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 1, pages V1–633. IEEE, 2010.

[81] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[82] Henry S Thompson, David Beech, M Maloney, and N Mendelsohn. Xml schema part 1: structures second edition. *W3C Recommendation (28 October 2004) http://www.w3.org/TR*, 2004.

[83] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[84] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. *Discovering and maintaining links on the web of data*. Springer, 2009.

[85] Qing Wang, Mingyuan Cui, and Huizhi Liang. Semantic-aware blocking for entity resolution. *Knowledge and Data Engineering, IEEE Transactions on*, 28(1):166–180, 2016.

[86] Xiaoyi Wang and Suraj M Alexander. Linking medical records: a machine learning approach. *International Journal of Collaborative Enterprise*, 1(3-4):394–406, 2010.

[87] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin core metadata for resource discovery. Technical report, 1998.

[88] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[89] William E Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

[90] William E Winkler. Approximate string comparator search strategies for very large administrative lists. *Statistics*, 2, 2005.

[91] Su Yan, Dongwon Lee, Min-Yen Kan, and Lee C Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 185–194. ACM, 2007.