

Project Report On

UNIQUE ENTITY HEAD EXTRACTION FROM STRUCTURED DATA



Submitted in partial fulfilment for the award of
E-Diploma in Big Data Analytics
From **C-DAC ACTS (Pune)**

Guided by:

Mr. Shivakarthik

Presented By:

Ms. Amruta Argade	PRN: 201940125005
Mr. Omkar Jori	PRN: 201940125020
Ms. Aditi Patil	PRN: 201940125032
Mr. Rohit Gupta	PRN: 201940125039
Mr. Varun Jha	PRN: 201940125052

Centre of Development of Advanced Computing (C-DAC), Pune

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ms. Amruta Argade

Mr. Omkar Jori

Ms. Aditi Patil

Mr. Rohit Gupta

Mr. Varun Jha

Have successfully completed their project on

**UNIQUE ENTITY HEAD EXTRACTION FROM
STRUCTURED DATA**

Under the guidance of **Mr. Shivakarthik**

Program Head

Ms. Risha P.R

Project Supervisor

HOD ACTS

Mr. Sundar Gaur

ACKNOWLEDGEMENT

This project “Unique Entity Head Extraction from Structured Data” was a great learning experience for us and we are submitting this work to Advance Computing Training School (CDAC ACTS).

We all are very glad to mention the name of *Mr. Shiv Karthik* for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to Ms. Risha P.R. (manager (ACTS training Centre), CDAC), for her guidance and support whenever necessary while doing this course Pg-Diploma in *Big Data Analytics (PG-DBDA)* through C-DAC ACTS, Pune.

Our most heartfelt thanks go to *Ms. Seema Sajeevan* (Course Coordinatoe, PG-DBDA) who gave all the required support and kind coordination to provide all the necessities where needed.

From:

Ms. Amruta Argade PRN: 201940125005

Mr. Omkar Jori PRN: 201940125020

Ms. Aditi Patil PRN: 201940125032

Mr. Rohit Gupta PRN: 201940125039

Mr. Varun Jha PRN: 201940125052

TABLE OF CONTENT:

1.	Abstract
2.	Introduction and Overview of project
3.	Information of technology used
4.	Evaluation Metrics
5.	Entity Resolution
6.	Aims and Challenges of Data Matching
7.	Procedure
8.	Pipeline Integration
9.	Example Application Areas
10.	Data Pre-processing
11.	Procedure
12.	Airflow
13.	conclusion
14.	Future Scope
15.	Bibliography

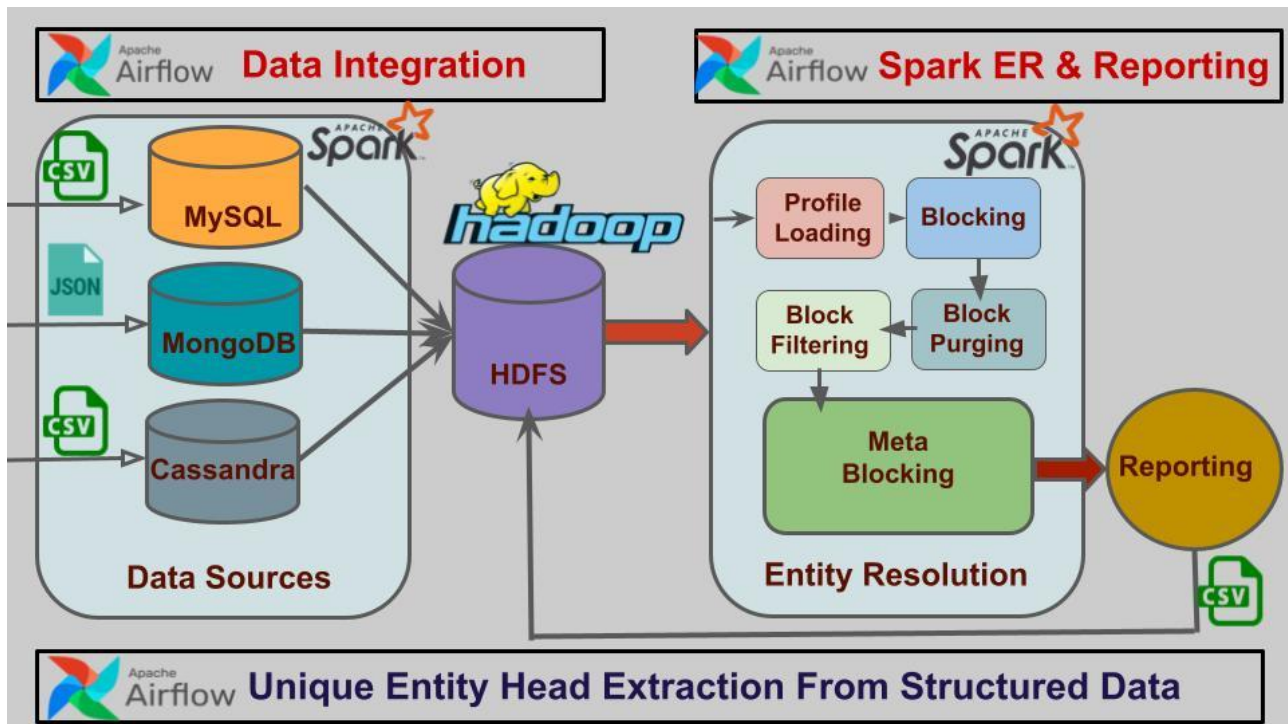
1. Abstract

Real world data contains multiple records belonging to the same customer. These records can be in single or multiple systems and they have variations across fields which makes it hard to combine them together, especially with growing data volumes. Multiple, yet different representations of the same real-world objects in data, duplicates, are one of the most intriguing data quality problems. The effects of such duplicates are detrimental; for instance, bank customers can obtain duplicate identities, inventory levels are monitored incorrectly, catalogs are mailed multiple times to the same household, etc.

Automatically detecting duplicates is difficult: First, duplicate representations are usually not identical but slightly differ in their values. Second, in principle all pairs of records should be compared, which is infeasible for large volumes of data. Two main components to overcome these difficulties: (i) Similarity measures are used to automatically identify duplicates when comparing two records. Well-chosen similarity measures improve the effectiveness of duplicate detection. (ii) Algorithms are developed to perform on very large volumes of data in search for duplicates. Well-designed algorithms improve the efficiency of duplicate detection.

Entity Resolution (ER) is the task of identifying different profiles that pertain to the same real-world entity. ER is a fundamental and expensive task for Data Integration. Comparing all profiles to each other's is impracticable when the data volume increases (e.g. Big Data), thus blocking techniques are employed to cluster similar records and to limit the number of comparisons only among the profiles contained in the same block.

2. Introduction and Overview of Project



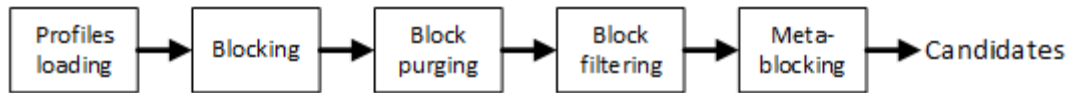
Flow Diagram

Entity Resolution (*ER*) is the task of identifying different records (a.k.a. entity *profiles*) that pertain to the same real-world entity. Comparing all the possible pairs of records in a data set may be very inefficient (quadratic complexity), in particular in the context of Big Data, e.g., when the records to compare are hundreds of millions. To reduce this complexity, usually ER uses different blocking techniques (e.g., token blocking, n-grams, etc.) to create clusters of profiles (called blocks). The goal of this process is to reduce the global number of comparisons, because will be compared only the records that are in the same blocks.

Unfortunately, in the Big Data context the blocking techniques still produces too many comparisons to be managed in a reasonable time, to reduce more the number of comparisons the meta-blocking techniques was introduced. The idea is to create a graph using the information learned from the blocks: the profiles in the blocks represents the nodes of the graph, and the comparisons between them represents the edges. Then is possible to calculate some metrics on the graph and use them to pruning

the less significant edges.

SparkER implements for Spark the Meta-Blocking techniques.



The process is composed by different stages:

1. **Profile loading**: loads the data (supports csv, json and serialized formats) into entity profiles.
2. **Blocking**: performs the blocking, token blocking or Loose Schema Blocking.
3. **Block purging**: removes the biggest blocks that are, usually, stopwords or very common tokens that do not provide significant relations.
4. **Block filtering**: for each entity profile, filters out the biggest blocks
5. **Meta-blocking**: performs the meta-blocking, producing as results the list of candidate's pairs that could be matches.

3. Information on Technologies used

Python: For this program, we will need Python to be installed on the computer. We will be using the libraries pandas, numpy, pyspark, findspark, networkx. The rest already come with the Python interpreter. It doesn't hurt to check that they're up to date though.

MySQL Workbench: MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

MongoDB Atlas Workbench: MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS, Azure, and GCP). MongoDB Atlas is the best way to deploy, run, and scale MongoDB in the cloud.

Cassandra Workbench: Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

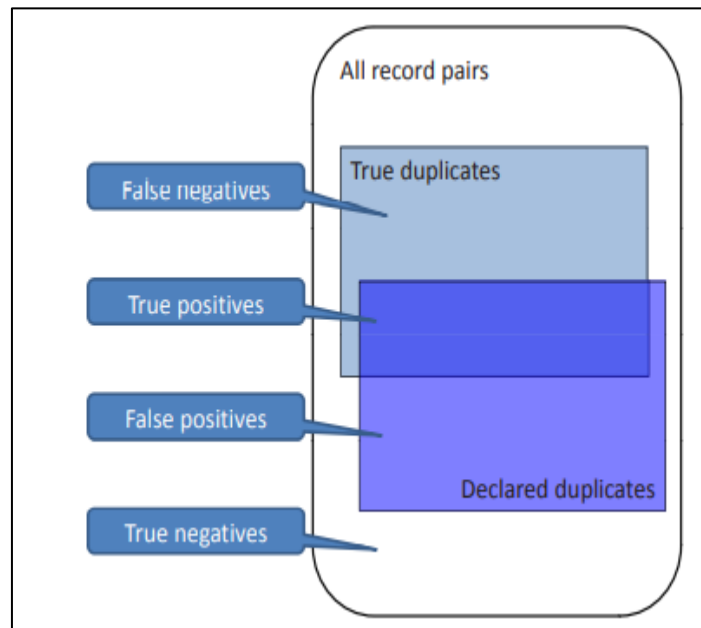
HDFS: Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application. It is cost effective as it uses commodity hardware.

Spark: Spark is a unified analytics engine for large-scale data processing including built-in modules for SQL, streaming, machine learning and graph processing. Apache Spark is an open-source cluster computing framework. Its primary purpose is to handle the real-time generated data. Spark was built on the top of the Hadoop MapReduce. It was optimized to run in memory whereas alternative approaches like Hadoop's MapReduce writes data to and from computer hard drives. So, Spark process the data much quicker than other alternatives.

Airflow: Airflow is a platform to programmatically author, schedule and monitor workflows. Use Airflow to author workflows as Directed Acyclic Graphs (DAGs) of tasks.

4. Evaluation Metrics

Precision, Recall, F1score is one metric for evaluating classification models.



5.1 What is Precision?

Precision is the ratio between the True Positives and all the Positives.

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

- Pairs Quality (PQ) corresponds to precision, assessing the portion of comparisons that involve a non-redundant pair of duplicates.
- In other words, it considers as true positives the matching comparisons and as false positives the superfluous and the redundant ones (given that some of the redundant comparisons involve duplicate profiles, PQ offers a pessimistic estimation of precision).
- More formally, $PQ = |D(B)|/|B|$. PQ takes values in the interval $[0, 1]$, with higher values indicating higher precision for B.
- D (B) stands for the set of co-occurring duplicate profiles and $|D(B)|$ for its

size (i.e., the number of detected duplicates).

5.2 What is Recall?

The recall is the measure of our model correctly identifying True Positives.

$$\text{Recall} = \frac{\text{True Positive}(TP)}{\text{True Positive}(TP) + \text{False Negative}(FN)}$$

- Pairs Completeness (PC) corresponds to recall,
- Assessing the portion of existing duplicates that can be detected in B. A set of blocks B is called block collection.
- More formally, $PC = |D(B)|/|D(E)|$. PC is defined in the interval [0, 1], with higher values indicating higher recall.

5.4 What is F1 score?

F1-score is the Harmonic mean of the Precision and Recall.

$$F1 \text{ Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Groundtruth Data

1. In order to be able to assess the quality of the matched data for a certain data matching project, ground-truth data, also known as ‘gold standard’ data, are required.
2. The characteristics of such ground-truth data must be as close as possible to the characteristics of the data that are to be matched.

Acquiring Ground Truth Data

One possibility is that the results from a previous data matching project in the same domain (ideally an earlier version of the same databases) are available, and that these databases have been manually evaluated with regard to the quality of the previous matching outcomes.

For example, domain experts might have detected wrongly matched as well as missed true matching pairs of records as they have worked with the matched databases. The quality of previously matched data might however not be good enough to be used as training data, especially if a more simpler matching approach was previously employed. Additionally, the manual inspection and possible correction of matches are often not 100 % correct, and it is therefore likely that the databases used as ground-truth data contain mistakes with regard to the match status of certain record pairs.

Another approach to obtain ground-truth data is to manually generate such data by sampling pairs of records from the two databases that are to be matched (or pairs from the single database that is to be deduplicated), and to manually classify these pairs as being either a match or a non-match.

Match: if a record pair has been classified as a match, then the assumption is that both records in the pair refer to the same real-world entity.

Non-match: For a record pair classified as a non-match, on the other hand, the two records in the pair are assumed to refer to two different real-world entities.

if a ground-truth data set with known true matching and non-matching record pairs is available, then similar to other classification problems in machine learning and data mining a variety of measures can be calculated on the outcomes of the classification process

5. Entity Resolution:

What is entity?

In data records often contain observation about real-life people, places, organization as well as things like servers or vessels. Entities can be any noun you care about, that have information in one or more records in your data sources.

What is Resolution?

The act or process of resolving: such as

- a. Analyzing a complex notion to simpler ones
- b. the act of answering: SOLVING
- c. The act of determining

Some of these entities are described differently, but are the same while others are described similarly but are actually different.

In order to integrate multiple knowledge bases or identify similarities between entities, the Entity Resolution (ER) becomes necessary.

The ER is the task that identifies the same real-world object across different entity profiles. It constitutes a quadratic task, since it requires every entity to be compared with all others.

What is Entity Resolution?

Entity Resolution is the task of disambiguating manifestations of real-world entities in various records or mentions by linking and grouping. For example, there could be different ways of addressing the same person in text, different addresses for businesses, or photos of a particular object. This clearly has many applications, particularly in government and public health data, web search, comparison shopping, law enforcement, and more.

Additionally, as the volume and velocity of data grows, inference across networks and semantic relationships between entities becomes a greater challenge. Entity Resolution can reduce the complexity by proposing canonicalized references to particular entities and deduplicating and linking entities.

Generically speaking, we can discuss ER as follows: there exists in the real-world entities, and in the digital world, records and mentions of those entities. The records and mentions may take many forms, but they all refer to only a single real-world entity. We can therefore discuss the ER problem as one involving matching record pairs corresponding to the same entity, and as a graph of related records/mentions to related entities.

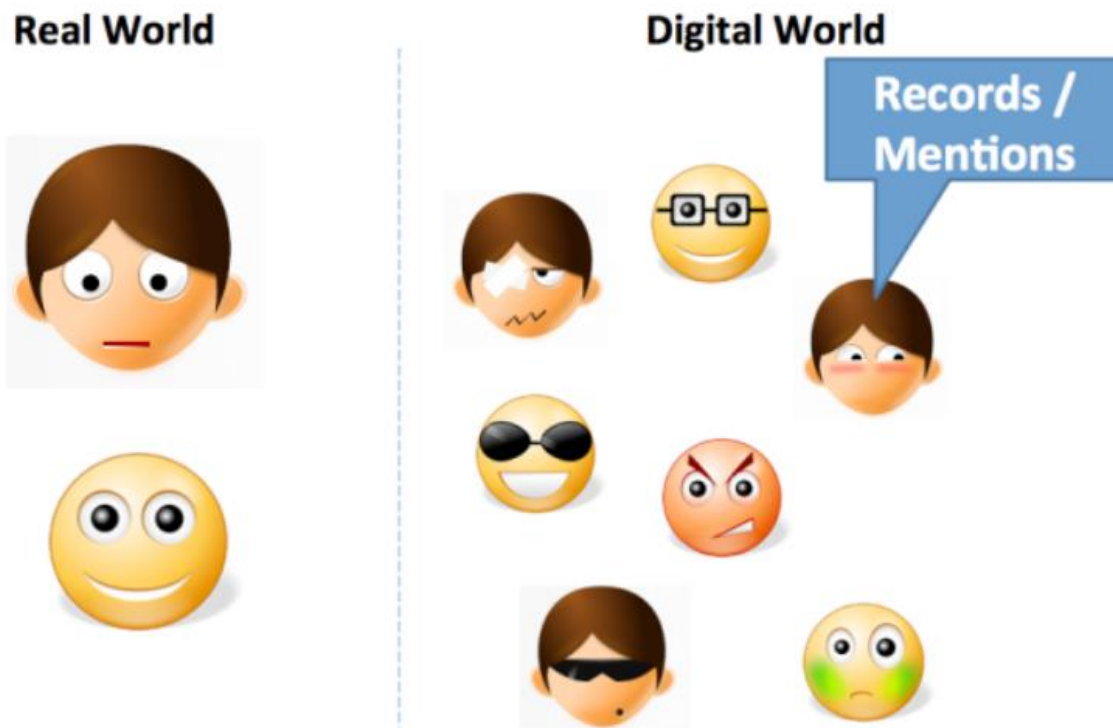


Figure: Entity Resolution

Record Linkage: a slightly different version of the task is to match records from one deduplicated data store to another. This task is proposed in the context of already normalized data, particularly in relational databases. However, in the context of Big Data, a one-to-one comparison of every record is not optimal.

Recognizing when two observation relate to the same entity, despite having been described differently.

Example:

Robert Smith	Robert J. Smith
123 Main Street	123 E Main Street
703.554.1214	RJSasmith.com

Entity Resolution works best when records contain a full name and at least one additional feature such as:

- Address
- Phone
- Email
- Date of Birth
- Other unique identifiers

(Example: Customer ID, passport number, driver's license number, Tax ID)

The more features on each records the better the entity resolution and relationship linking.

Entities, Features & Attributes

Entity <----- Features <----- Attributes

Entity Resolution is About Accurate Counting

- Is it 5 people with 1 bank account or 1 person with 5?
- Is it 20 cases of COVID-19 or 1 case reported 20 times?
- If one cannot count, one cannot estimate direction and speed, making prediction virtually impossible?

5 core steps to Entity Resolutions

1. **Source normalization:** - cleaning data and harmonizing different sources into a common schema of features (columns) that will be used for evaluating potential matches
2. **Featurization and blocking key generation:** -creating features for blocking keys, which are targeted tokens that are likely shared between matching records, to constrain the search space from N^2 to something more computationally manageable
3. **Generate candidate pairs:** - using blocking join keys to create potential candidate pairs. This is essentially a self-join on the blocking key, and is typically implemented using graph data structures, with individual records as nodes, and potential matches as edges
4. **Match scoring and iteration:** - Deciding which candidate pairs actually match via a match scoring function. This can be rules based, but is typically better implemented as a learning algorithm that can adapt and optimize over non-linear decision boundaries
5. **Generate entities:** - Eliminating non-matching edges in the graph, and generating resolved entities and associated mapping to individual source records

7. Aims and Challenges of Data Matching

7.1 Aims and Challenges of Data Matching

Given the ever-increasing amount of data that are being collected, not just by businesses and government organisations but increasingly also by individuals, the past decade has seen strong interest in novel techniques that allow the efficient processing, management and analysis of large data collections. The fields of data warehousing and data mining have gained immense interest in both academia and industry. While data warehousing is concerned with the efficient processing, integration and storage of large amounts of data into clean, consistent and persistent forms that enable basic statistical analysis, data mining is aimed at discovering new and potentially valuable information from such large data collections.

As businesses, public bodies and government agencies are drowning in an ever-increasing deluge of data, the ability to analyse their data in a timely fashion can provide a competitive edge to a commercial enterprise, lead to improved productivity for government agencies and be of vital importance to national security. In many large-scale information systems and data mining projects, data from multiple sources need to be integrated and matched in order to improve data quality, enrich existing data sources or facilitate data mining that is not feasible on a single database. The analysis of data integrated from disparate sources, either within an organisation or between different organisations, can lead to much improved benefits compared to analysing databases in isolation. Integrated data can also allow types of data analyses that are not feasible on individual databases, such as the identification of adverse drug reactions in particular patient groups, or the detection of terrorism suspects through the analysis of certain suspicious patterns of activities.

Integrating data from different sources consists of three tasks.

- The first task is schema matching, which is concerned with identifying database tables, attributes and conceptual structures (such as ontologies, XML schemas and UML diagrams) from disparate databases that contain data that correspond to the same type of information.
- The second task, is data matching, the task of identifying and matching individual records from disparate databases that refer to the same real-world entities or objects. A special case of data matching is duplicate detection, the

task of identifying and matching records that refer to the same entities within a single database.

- The third task, known as data fusion, is the process of merging pairs or groups of records that have been classified as matches (i.e. that are assumed to refer to the same entity) into a clean and consistent record that represents an entity. When applied on one database, this process is called deduplication.

7.2 Data Integration and Link Analysis

Data matching is a commonly required step in the much larger process of data integration. While data matching is concerned with identifying and matching individual records that refer to the same entities from disparate databases, data integration is the overall process of integrating heterogeneous databases, data warehouses or data repositories to provide a unified view of the available data. This process is highly significant, for example, for company mergers, collaborative e-Commerce projects, data mash-ups and scientific collaborations

Data stored in disparate databases are usually heterogeneous not only at the record (instance) level, but also at the structural database (table) level. As illustrated in

Challenges in Data Integration

Data integration has many challenges, while comparing at the database structure and the record level.

When matching schemas, a careful analysis of the names, types of content and other meta-data of the available attributes can help to identify which attributes correspond to each other.

Meta-data can include information such as the number of different values in an attribute and their frequency distribution, descriptions of the sources of the attribute values, or their structure and encoding (such as references to external encoding dictionaries).

Data integration should however not consider schema matching and data matching independently, but rather in an integral fashion. Commonly, data integration is a semi-automated process, where human insight can provide initial information about which

database tables potentially correspond to each other. Such hints can then be used to bootstrap a tool-supported integration process.

The content of attributes can help in the schema matching process. Correlation analysis can help detect attributes that contain related information. Data matching across two databases can identify records that correspond to the same entities, which in turn can help identify which attributes correspond to each other.

A process that exploits information both at the schema and the record level, and iteratively refines the integration process, can therefore lead to efficient and accurate data integration.

Once databases are matched at the schema level, individual records need to be matched to identify which records in two or more databases correspond to the same real-world entities.

The final task in the data integration process, after pairs or groups of records have been identified that refer to the same entities, is to consolidate and merge matching records into a single consistent and clean representation for each entity.

The major challenge of data fusion is how conflicts are resolved when the records that correspond to one entity contain different attribute values. Different aggregation functions can be applied.

7.2 Lack of Unique Entity Identifiers and Data Quality

Generally, the databases to be matched (or deduplicated) do not contain unique entity identifiers or keys. Examples of entity identifiers include unique patient or tax payer numbers, or consumer product codes. If such identifiers are available in all the databases to be matched, then the data matching task becomes a database join that can be implemented efficiently through SQL statements. Even when entity identifiers are available in the databases to be matched, one must be absolutely confident in the accuracy, completeness, robustness and consistency over time of these identifiers, because any error in such an identifier will result in wrongly matched records. As database owner, one must also be confident that there are no duplicate records in a database where different identifiers are used for the same entity. This situation is however common, for example in customer databases, where the same customer can

have several records due to name variations or address changes. If no entity identifiers are available in the databases to be matched, then the matching needs to rely upon the attributes that are common across the databases. If the databases contain information about people, then these common attributes can be names, addresses, dates of birth and other partially identifying personal details. The quality of such information can however be low, as personal details can be wrong, incomplete and they often change over time.

8 Example Application Areas

The following sections describe several example application areas where data matching is an important component of larger information systems, of government and business processes, or of research endeavours. For each area, the unique aspects and challenges encountered are discussed.

8.1 National Census

National census agencies around the world collect data about various aspects of the population, culture, economy and the environment in their respective country. This information is then collated into a diverse range of statistical reports that are used by governments and businesses to plan the allocation of funding and resources. Data matching has been recognised as an important tool for census statistics. It allows the reuse of existing data sources to compile new statistical data sets, and thus reduces the costs and efforts required to conduct large-scale census collections. It also helps to improve data quality and integrity, as matching data from different census collections can help detect and correct conflicting or missing information, or improve estimates of population sizes through capture–recapture techniques. Data matching can also be used to generate longitudinal data sets, by matching census data that have been collected at different instants in time (for example every 5 or 10 years). It is commonly recognised that longitudinal data are an important source of information about how the characteristics of a population change over time. Different countries have different laws and regulations that govern what kind of data matching can be done. In Australia, for example, name and address details collected in national censuses need to be destroyed within 1 year after collection (both the physical paper-based census forms as well as any electronic versions of these data). Such restrictions make it very challenging to create longitudinal data sets, because the matching has to rely upon information such as age, gender, birthplace, religion, highest educational qualification and so on. The US Census Bureau has been one of the early adopters of data matching. Not only has the Bureau applied existing data matching techniques on a regular basis, it has also been at the forefront of data matching research and development over several decades. The size of the data collections the Bureau needs to match can be in the order of several hundred million of records, and therefore it has also been developing large-scale and parallel data matching techniques. The various techniques developed by the US Census Bureau will be described in the corresponding chapters later in this book.

8.2 Online Shopping

As consumers worldwide tend to increasingly use online shopping for consumer products and services, Web sites that provide price comparisons have become popular. These sites allow consumers to either query a certain product or browse product categories, types and brands. A challenge for such comparison-shopping sites is to identify which product descriptions in different online stores do correspond to the same item. While certain types of items have unique identifiers, such as ISBN numbers for books, and systems such as electronic product Code (EPC) that are based on Radio Frequency Identification (RFID) codes are becoming more widespread, the descriptions of the same product are often quite different across several online stores. To allow accurate and comprehensive price comparisons for an individual consumer product, a comparison-shopping site needs to be able to accurately identify which product descriptions refer to the same actual product. Compared to other types of data that are commonly used for data matching (like name and address details of people), the variations in the descriptions of consumer products require different similarity measures. For example, the string similarities between the four descriptions on the other hand, the differences between a description of this camera and its predecessor ('Canon PowerShot G10') might only be the single difference between the '0' and '1' digit in the camera's number. The number of megapixels (10 for this camera) complicates the similarity calculations for this example even further. As this example highlights, data matching is often a very data dependent activity, and techniques such as approximate comparison functions need to be specifically designed for a certain task and data at hand.

9 Data Pre-processing

9.1 Data pre-processing

Database A

RecID	Surname	GivenName	Street	Suburb	Postcode	State	DateOfBirth
a1	Smith	John	42 Miller St	O'Connor	2602	A.C.T.	12-11-1970
a2	Neighan	Joanne	Brown Pl	Dickson	2604	ACT	8 Jan 1968
a3	Meyer	Marie	3/12-14 Hope Cnr	SYDNEY	2050	NSW	01-01-1921
a4	Smithers	Lyn	Browne St	DIXON	2012	N.S.W.	13/07/1970
a5	Nguyen	Ling	1 Milli Rd	Nrth Sydeny	2022	NSW	10/08/1968
a6	Faulkner	Christine	13 John St	Glebe	2037	NSW	02/23/1981
a7	Sandy	Robert	RMB 55/326 West St	Stuart Park	2713	NSW	7/10/1970

Database B

RecID	Name	Address	BYear	BMonth	BDay
b1	Meier, Mary	14 (App 3) Hope Corner, Sydney 2000	1927	4	29
b2	Janice Meyer	Bryan St, O'Connor ACT 2604	1968	11	20
b3	Jonny Smith	47 Miller Street, 2619 Canberra ACT	1970	12	11
b4	Lyng Nguyen	1 Millie Road, 2002 North Sydney, NSW	1968	8	10
b5	Kristina Fawkner	13 St John Street, 2031 Glebe	1981	2	23
b6	Bob Santi	55 East St; Stuart's Point; NSW 2113	1970	12	11
b7	Lynette Cain	6 / 12 Hope Corner, 2020 Sydney N.S.W.	1970	7	13

Figure: Dataset

It is important to make sure that data sourced from different databases have been appropriately cleaned and standardised. The aim of this process is to ensure that the attributes used for the matching have the same structure, and their content follows the same formats. It has been recognised that data cleaning and standardisation are crucial steps to successful data matching.

9.2 Remove unwanted characters and words:

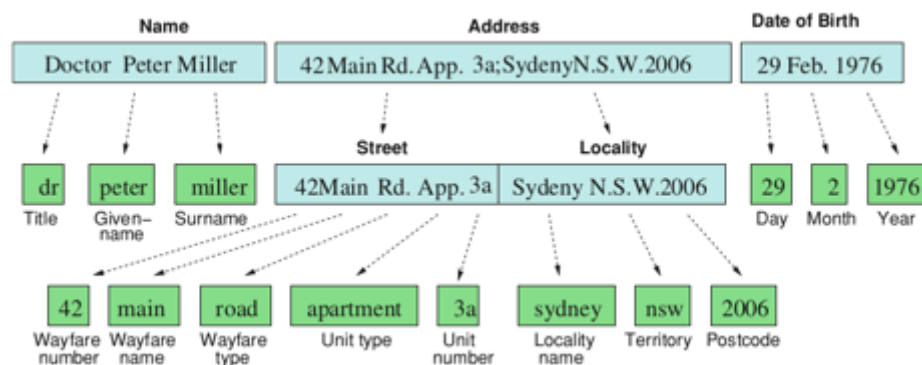
Database A – Cleaned and standardised

RecID	GivenName	Surname	Gender	StrPrefix	StrNum	StrName	StrType	Suburb	Postcode	State	BDay	BMonth	BYear
a1	john	smith	m		42	millier	street	oconnor	2602	act	12	11	1970
a2	joanne	neighan	f			brown	place	dickson	2604	act	8	1	1968
a3	mary	meier	f	3	12-14	hope	corner	sydney	2050	nsw	1	1	1921
a4	lynette	smithers	f			browne	street	dixon	2012	nsw	13	7	1970
a5	ling	nguyen	?		1	milli	road	north sydney	2022	nsw	10	8	1968
a6	christine	faulkner	f		13	john	street	glebe	2037	nsw	23	2	1981
a7	robert	sandy	m	rmb 55	326	west	street	stuart park	2713	nsw	7	10	1970

Database B – Cleaned and standardised

RecID	GivenName	Surname	Gender	StrPrefix	StrNum	StrName	StrType	Suburb	Postcode	State	BDay	BMonth	BYear
b1	mary	meier	f	apt 3	14	hope	comer	sydney	2000	nsw	29	4	1927
b2	janice	meier	f			bryan	street	oconnor	2604	act	20	11	1968
b3	john	smith	m		47	millier	street	canberra	2619	act	11	12	1970
b4	lyng	nguyen	?		1	millie	road	north sydney	2002	nsw	10	8	1968
b5	kristina	fawkner	f		13	saint john	street	glebe	2037	nsw	23	2	1981
b6	robert	santi	m		55	east	street	stuarts point	2113	nsw	11	12	1970
b7	lynette	cain	f	6	12	hope	comer	sydney	2020	nsw	13	7	1970

This step corresponds to an initial cleaning, where characters such as commas, colons, semicolons, periods, hashes, and quotes are removed. In certain applications, some words can also be removed if it is known that they do not contain any information that is of relevance to the data matching process. These words are also known as stop words. Data pre-processing refers to the tasks of converting the raw input data from the databases to be matched or deduplicated into a format that allows efficient and accurate matching.



10. Procedure

10.1 Importing the Data from different Database –

10.1.1 Importing the Data from MySQL-

```
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - root
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- Id: integer (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- displayName: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- Location: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- AboutMe: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- Age: integer (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- phone: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- City: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- State: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-- Country: string (nullable = true)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO -
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO codegen.CodeGenerator: Code generated in 113.222729 ms
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SparkContext: Starting job: showString at NativeMethodAccessorImpl.java:8
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.DAGScheduler: Got job 0 (showString at NativeMethodAccessorImpl.java:8) with 1 output partitions
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (showString at NativeMethodAccessorImpl.java:8)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.DAGScheduler: Parents of final stage: List()
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[2] at showString at NativeMethodAccessorImpl.java:8), which has no missing parents
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO memory.MemoryStore: block broadcast_0 stored as values in memory (estimated size 12.9 KiB, free 366.3 MiB)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO memory.MemoryStore: block broadcast_0_piece0 stored as bytes in memory (estimated size 0.0 KiB, free 366.3 MiB)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.0.115:41717 (size: 6.0 KiB, free: 366.3 MiB)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SparkContext: Created broadcast 0 from broadcast at DAGScheduler.scala:1427
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (MapPartitionsRDD[2] at showString at NativeMethodAccessorImpl.java:8) (first 15 tasks are for partitions Vector(0))
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.TaskSetManager: Adding task set 0.0 with 1 tasks resource profile 0
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0) (192.168.0.115, executor driver, partition 0, PROCESS_LOCAL, 4299 bytes) taskResourcesMap()
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO scheduler.TaskSetManager: Running task 0.0 in stage 0.0 (TID 0)
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO jdbc.JDBCSource: closed connection
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO executor.Executor: Finished task 0.0 in stage 0.0 (TID 0). 7195 bytes result sent to driver
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 184 ms on 192.168.0.115 (executor driver) (1/1)
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.TaskSetManager: Removed taskset 0.0, whose tasks have all completed, from pool
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.DAGScheduler: ResultStage 0 (showString at NativeMethodAccessorImpl.java:8) finished in 0.301 s
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO scheduler.DAGScheduler: Job 0 finished: showString at NativeMethodAccessorImpl.java:8, took 0.323963 s
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:37 UTC INFO codegen.CodeGenerator: Code generated in 21.389883 ms
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - +-----+-----+-----+-----+-----+-----+-----+-----+
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | Id | DisplayName | Location | AboutMe | Age | phone | City | State | Country |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - +-----+-----+-----+-----+-----+-----+-----+-----+
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 0 | Community on the server farm | I'm not reall... | 27 | 605-452-4310 | | | | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 1 | Geoff Dalgas | Corvallis, OR | Dev #2 who helped... | 21 | 440-200-5210 | Corvallis | OR | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 2 | Jin | Raleigh, NC | I design stuff fo... | 20 | 728-871-7285 | Raleigh | NC | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 3 | Stefan Lasinski | Berkeley, Califor... | 31 | 925-363-9079 | Berkeley | CA | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 4 | twaskinger | Canada | Ubuntu member, M... | 35 | 343-335-2102 | Canada | | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 5 | Kevin Montrose | New York, NY, Uni... | 31 | Stack Overflow Va... | 23 | 576-841-6982 | New York | NY | United States |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 6 | Emmett | San Francisco, C... | 40 | 415-526-1780 | San Francisco | CA | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 7 | Toon | Lyon, France | I'm an engineer w... | 28 | 658-533-4045 | Lyon | France | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 8 | Seamus | Leeds, UK | I am a researcher ... | 33 | 557-360-9376 | Leeds | UK | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 9 | Francesco | England, United K... | 41 | 208-682-2651 | England | United Kingdom | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 10 | Bryan Denny | Winston-Salem, NC | Currently web (C#... | 39 | 535-232-0008 | Winston-Salem | NC | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 11 | cengiz | San Francisco Bay... | 32 | 939-726-1904 | San Francisco Bay... | CA | United States |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 12 | Kibbi | New York, NY | Resident of Long ... | 34 | 645-4519 | New York | NY | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 13 | Scharron | Paris, France | Big Data Sergeant... | 37 | 149-642-8599 | Paris | France | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 14 | Matt Simons | Pennsville Twp, NJ | 8+ years administ... | 29 | 249-731-1573 | Pennsville Twp | NJ | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 15 | Scarycat | Northumbria, UK | I say job is well... | 21 | 798-338-7157 | Northumbria | United Kingdom | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 16 | khyrd | United States | All around techno... | 22 | 768-859-6380 | United States | | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 17 | Stephen Jazdzewski | Austin, TX | Getting to be an ... | 39 | 743-775-2900 | Austin | TX | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 18 | mediaman | California | I guess I am a ... | 21 | 878-884-0368 | California | | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - | 19 | xenoterracide | Austin, TX | Former Linux Syst... | 29 | 487-636-6174 | Austin | TX | |
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - +-----+-----+-----+-----+-----+-----+-----+-----+
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - only showing top 20 rows
[2022-04-03, 20:55:37 UTC] [subprocess.py:89] INFO - =====
```

10.1.2 Importing Data from MongoDB:

```
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SparkContext: SUCCESSFULLY STOPPED SPARK CONTEXT
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO util.ShutdownHookManager: Shutdown hook called
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-4ac13ce7-5036-4448-841f-d5cc09ba130
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-1aa8baa-008d-4678-9364-b26dc1093dc
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-4ac13ce7-5036-4448-841f-d5cc09ba130/pyspark-03f399f6-010e-4c63-be3f-84aed7a099aa
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03T20:55:38.415+0530 connected to localhost
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03T20:55:38.443+0530 Imported 38 documents
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC WARN util.Utils: Your hostname, aditi-OMEN-by-HP-Laptop-16-b0xxx resolves to a loopback address: 127.0.0.1; using 192.168.0.115 instead (on interface wlan0)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- loading settings :: uri = jar:file:/home/aditi1/BDMA_HOME/spark-3.2.0-bin-hadoop2/jars/ivy-2.5.0.jar/org/apache/ivy/core/settings/ivysettings.xml
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- ivy default cache set to: /home/aditi1/.ivy2/cache
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- the jars for the packages stored in /home/aditi1/.ivy2/jars
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.sparkmongo-spark-connector_2.12 added as a dependency
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- resolving dependencies :: org.apache.spark:spark-submit-parent-63627d52-ad9b-4a7b-ab9d-ea67539d871c:1.0
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- confs: [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- found org.mongodb.sparkmongo-spark-connector_2.12:3.0.0 in central
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- found org.mongodb.driver-sync:4.0.5 in central
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- found org.mongodb.driver-core:4.0.5 in central
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- resolution report :: resolve 184ms :: artifacts 0 ms
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- modules in use:
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.driver-sync:4.0.5 from central in [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.sparkmongo-spark-connector_2.12:3.0.0 from central in [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.driver-core:4.0.5 from central in [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.driver-sync:4.0.5 from central in [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- org.mongodb.sparkmongo-spark-connector_2.12:3.0.0 from central in [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -----
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - | artifact | | | | | | | | | |
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - | conf | | number | search | dmd | dmd | evicted | | number | dmd |
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - |-----|-----|-----|-----|-----|-----|-----|-----|
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - | default | | 4 | | 0 | | 0 | | 0 | | 4 | | 0 | |
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -----
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- retrieving :: org.apache.spark:spark-submit-parent-63627d52-ad9b-4a7b-ab9d-ea67539d871c
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- confs: [default]
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - -- 0 artifacts copied, 4 already retrieved (8K/5m)
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SparkContext: Running Spark version 3.2.0
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceUtils: =====
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceUtils: No custom resources configured for spark.driver.
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceUtils: =====
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SparkContext: Submitted application: SparkMongoDBApp
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, s
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceProfile: Limiting resource is cpu
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO resource.ResourceProfileManager: Added ResourceProfile id: 0
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SecurityManager: Changing view acls to: aditi
[2022-04-03, 20:55:36 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:36 UTC INFO spark.SecurityManager: Changing modify acls to: aditi
```


10.1.3 Importing data from Cassandra:

```
[2022-04-03, 20:55:47 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:47 UTC WARN auth.PlainTextAuthProviderBase: [ ] /127.0.0.1:9042 did not send an authentication challenge; This is suspicious because the driver expects authentication
[2022-04-03, 20:55:47 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:47 UTC WARN auth.PlainTextAuthProviderBase: [ ] /127.0.0.1:9042 did not send an authentication challenge; This is suspicious because the driver expects authentication
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:48 UTC INFO org.apache.cassandra.connector.Connected: Connected to Cassandra cluster.
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - root
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- id: integer (nullable = false)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- aboutme: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- age: integer (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- city: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- country: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- displayname: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- location: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- phone: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - |-- state: string (nullable = true)
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:48 UTC INFO v2.V2ScanRelationPushDown:
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO - Output: id#, aboutme#, age#, city#, country#, displayname#, location#, phone#, state#
[2022-04-03, 20:55:48 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO codegen.CodeGenerator: Code generated in 107.97427 ms
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO spark.SparkContext: Starting job: showString at NativeMethodAccessorImpl.java:0
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Got job 0 (showString at NativeMethodAccessorImpl.java:0) with 1 output partitions
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (showString at NativeMethodAccessorImpl.java:0)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Parents of final stage: List()
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Missing parents: List()
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[3] at showString at NativeMethodAccessorImpl.java:0), which has no missing parents
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO memory.MemoryStore: Block broadcast_0 stored as values in memory (estimated size 10.0 KiB, free 306.3 MiB)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO memory.MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 9.3 KiB, free 306.3 MiB)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on 192.168.0.115:4475 (size: 9.3 KiB, free: 306.3 MiB)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO spark.SparkContext: Created broadcast_0 from broadcast at DAGScheduler.scala:1427
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (MapPartitionsRDD[3] at showString at NativeMethodAccessorImpl.java:0) (first 15 tasks are for partitions Vec#)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.TaskSchedulerImpl: Adding task set 0.0 with 1 tasks resource profile 0
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.TaskSchedulerImpl: Starting task 0.0 in stage 0.0 (TID 0) (192.168.0.115, executor driver, partition 0, ANR, 18237 bytes) taskResourceAssignments Map()
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO executor.Executor: Running task 0.0 in stage 0.0 (TID 0)
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO executor.Executor: Finished task 0.0 in stage 0.0 (TID 0). 2011 bytes result sent to driver
[2022-04-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 496 ms on 192.168.0.115 (executor driver) (1/1)
```

```
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO scheduler.DAGScheduler: Job 1 finished: showString at NativeMethodAccessorImpl.java:0, took 0.090294 s
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:49 UTC INFO codegen.CodeGenerator: Code generated in 15.62151 ms
4-03, 20:55:49 UTC] {subprocess.py:89} INFO -
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | id | aboutme | age | city | country | displayname | location | phone | state |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - |----|-----|----|-----|-----|-----|-----|-----|-----|
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 20 | Android Engineer... | 39 | Washington | United States | zugaldia | Washington, DC, U... | (202-147-4780) | DC |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 17 | Andrew mostly wor... | 21 | Ireland | theotherreceive | Ireland | (404-406-7087) | null |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 6 | I'm very importan... | 35 | Wheaton | null | Jim Fiorato | Wheaton, Illinois... | (757-448-8875) | Illinois United ... |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 11 | .NET developer, L... | 33 | Illinois | null | vanillaike | Illinois | (516-675-5021) | null |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 15 | Software Develop... | 39 | Madison | null | Larry Wang | Madison, WI | (710-482-5216) | WI |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 22 | wannabe free soft... | 39 | Auckland | null | Piotr Zurek | Auckland, New Zea... | (068-626-3611) | New Zealand |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 1 | Data Engineer | 29 | Wellington | null | Alan Featherstone | Wellington, New Z... | (901-246-5372) | New Zealand |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 10 | Generalist comput... | 31 | Washington | United States | jnoro | Washington, DC, U... | (114-685-9317) | DC |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 2 | Just another Linu... | 38 | United Kingdom | null | jamespo | United Kingdom | (665-081-3271) | null |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - | 5 | I work at Bakpa... | 30 | Raleigh | United States | Benjamin Pollack | Raleigh, NC, Unit... | (820-002-3042) | NC |
4-03, 20:55:49 UTC] {subprocess.py:89} INFO -
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - only showing top 10 rows
4-03, 20:55:49 UTC] {subprocess.py:89} INFO -
4-03, 20:55:49 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO v2.V2ScanRelationPushDown:
4-03, 20:55:50 UTC] {subprocess.py:89} INFO - Output: id#, aboutme#, age#, city#, country#, displayname#, location#, phone#, state#
4-03, 20:55:50 UTC] {subprocess.py:89} INFO -
```

```
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO codegen.CodeGenerator: Code generated in 11.29537 ms
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | id | DisplayName | Location | AboutMe | Age | phone | City | State | Country |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - |----|-----|-----|-----|----|-----|-----|-----|-----|
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 0 | tante | Oldenburg, Germany | Just some random ... | 23 | (507-293-0738) | Oldenburg | Germany | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 1 | Alan Featherstone | Wellington, New Z... | Data Engineer | 29 | (901-246-5372) | Wellington | New Zealand | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 2 | jamespo | United Kingdom | Just another Linu... | 38 | (665-081-3271) | United Kingdom | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 3 | fwaechter | Vienna, VA | I'm feeling rough... | 28 | (923-759-1817) | Vienna | VA | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 4 | Ceppi Marco | Vienna, VA | I'm a Moderator ... | 32 | (375-304-3599) | Vienna | VA | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 5 | Benjamin Pollack | Raleigh, NC, Unit... | I work at Bakpa... | 30 | (820-002-3042) | Raleigh | NC | United States |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 6 | Jim Fiorato | Wheaton, Illinois... | I'm very importan... | 35 | (757-448-8875) | Wheaton | Illinois United ... | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 7 | sajith | Illinois, United ... | Student forever... | 21 | (728-258-7114) | Illinois | United States | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 8 | Jackson | Boston, MA | I am a developer ... | 35 | (408-267-2035) | Boston | MA | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 9 | Dale Ragan | United States | I am Electrica... | 18 | (319-613-6622) | United States | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 10 | Pablo | Illinois | just another tech... | 31 | (616-381-3541) | Illinois | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 11 | vanillaike | Illinois | .NET developer, L... | 33 | (516-675-5021) | Illinois | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 12 | Phil Miller | Illinois | Contact me for he... | 30 | (701-857-3550) | Illinois | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 13 | Avery Payne | Pune, India | As for you, Gilga... | 27 | (357-292-8559) | Pune | India | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 14 | Hemant | Pune, India | null | 21 | (418-639-1377) | Pune | India | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 15 | Larry Wang | Madison, WI | Software Develop... | 39 | (710-482-5216) | Madison | WI | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 16 | Rui Vieira | London, United Ki... | I am currently do... | 25 | (613-128-6350) | London | United Kingdom | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 17 | theotherreceive | Ireland | Andrew mostly wor... | 21 | (484-006-7087) | Ireland | null | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 18 | Sajad Bahmani | Tehran, Iran | Favorite Language... | 19 | (875-803-9408) | Tehran | Iran | null |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - | 19 | jnoro | Washington, DC, U... | Generalist comput... | 31 | (114-685-9317) | Washington | DC | United States |
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - only showing top 20 rows
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - done
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO v2.V2ScanRelationPushDown:
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - Output: id#, aboutme#, age#, city#, country#, displayname#, location#, phone#, state#
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO -
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO output.FileOutputCommitter: skip cleanup_temporary folders under output directory:false, ignore cleanup failures: false
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO datasource.SQLHadoopMapReduceCommitProtocol: Using output committer class org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO spark.SparkContext: Starting job: csv at NativeMethodAccessorImpl.java:0
[2022-04-03, 20:55:50 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:55:50 UTC INFO scheduler.DAGScheduler: Got job 3 (csv at NativeMethodAccessorImpl.java:0) with 4 output partitions
```

10.2 SparkER

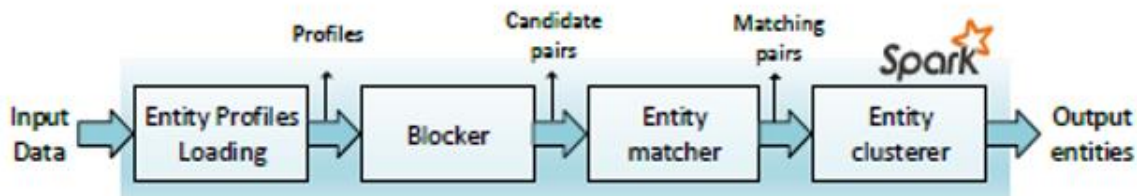


Figure: SparkER Architecture

The process consists of following steps:

1. Profile Loading:

Loads the data (supports csv, json and serialized formats) into entity profiles. The cleaned and standardised database tables (or files) are now ready to be matched. Potentially, each record from one database needs to be compared with all records in the other database to allow the calculation of the detailed similarities between two records. This leads to a total number of record pair comparisons that is quadratic in the size of the databases to be matched. Matching the example databases from For example: - leads to a total of $7 \times 7 = 49$ comparisons (between one record from database A and one record from database B).

- **Clean-Clean ER:** It is also known as Record Linkage, is the process of detecting pairs of matching entities among two heterogeneous, individually clean (i.e., duplicate-free), but overlapping collections of entities. As an example, consider the task of merging individual collections of consumer products that stem from different on-line stores, thus having proprietary identifiers and slightly varying descriptions.
- **Dirty ER:** It is also known as Deduplication, receives as input a single entity collection and aims at detecting the matching profiles that are contained in it. As an example, consider the task of citation matching in the context of a bibliographic database, such as Google Scholar

```

[14]: # Profiles contained in the first dataset
profiles = sparker.CSVWrapper.load_profiles('file:///home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/s
start_id_from=0,
separator="," header=True,
real_id_field = "id",
source_id=0)
  
```

```
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.DAGScheduler: ResultStage 12 (showString at NativeMethodAccessorImpl.java:0) finished in 0.037 s
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.DAGScheduler: Job 12 is finished. Cancelling potential speculative or zombie tasks for this job
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 12: Stage finished
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.DAGScheduler: Job 12 finished: showString at NativeMethodAccessorImpl.java:0, took 0.040208 s
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO codegen.CodeGenerator: Code generated in 6.7228 ms
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - +-----+
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - | attributes|original_id|profile_id|source_id|
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - +-----+
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - |[[[DisplayName, ta...]]| 0| 30| 1|
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Al...]]| 1| 31| 1|
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - |[[[DisplayName, ja...]]| 2| 32| 1|
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - +-----+
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - only showing top 3 rows
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO -
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - None
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:166
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.DAGScheduler: Got job 13 (runJob at PythonRDD.scala:166) with 1 output partitions
[2022-04-03, 20:55:57 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:57 UTC INFO scheduler.DAGScheduler: First stage: ResultStage 13 (runJob at PythonRDD.scala:166)
```

```
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Job 14 is finished. Cancelling potential speculative or zombie tasks for this job
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 14: Stage finished
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Job 14 finished: showString at NativeMethodAccessorImpl.java:0, took 0.030035 s
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - +-----+
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - | attributes|original_id|profile_id|source_id|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - +-----+
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Co...]]| 0| 0| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Ge...]]| 1| 1| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Ji...]]| 2| 2| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, St...]]| 3| 3| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, tx...]]| 4| 4| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Ke...]]| 5| 5| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Es...]]| 6| 6| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, To...]]| 7| 7| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Se...]]| 8| 8| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Fr...]]| 9| 9| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, Br...]]| 10| 10| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, cm...]]| 11| 11| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - |[[[DisplayName, ka...]]| 12| 12| 0|
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - +-----+
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - only showing top 13 rows
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO -
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - None
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:166
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Got job 15 (runJob at PythonRDD.scala:166) with 1 output partitions
1922-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 15 (runJob at PythonRDD.scala:166)
```

2. Blocking / Indexing:

The goal of these methods is to cluster the similar, input entities into blocks such that the matching ones are placed in at least one common block with a high probability.

Why Indexing?

To reduce the possibly very large number of pairs of records that need to be compared, indexing techniques are commonly applied.

What does Indexing does?

These techniques filter out record pairs that are very unlikely to correspond to matches. They generate candidate record pairs that will be compared in more detail in the comparison step of the data matching process to calculate the detailed similarities between two records, as will be described in the following section.

The traditional approach to indexing is called blocking.

The attribute-agnostic functionality ensures that the creation of blocks completely disregards any schema information. The reason is that HHIS entail so high levels of

noise and heterogeneity that attribute names cannot play a reliable role in the creation of blocks.

- **Token blocking:**

Each token ti creates a distinct block bi that contains all entities having ti in the values of their profile regardless of the associated attribute names.

In this way, blocks are built in an attribute-agnostic manner, and every entity is placed in multiple blocks, ensuring a redundancy-positive functionality.

Dirty ER, a token ti is valid (i.e., it creates a block) if it appears in at least two input entity profiles

Clean-Clean ER, it has to appear in entity profiles of both input sets.

```
blocks = sparker.Blocking.create_blocks(profiles)
print("Number of blocks", blocks.count())
```

- **N-Gram Based Indexing:**

For data that are dirty and contain large amounts of errors and variations, both standard blocking approach might not be able to insert records into the same blocks, for example if the beginning of a sorting key value is different for two name variations. N-gram based indexing aims to overcome this drawback by generating variations of each BKV, and to use these variations as the actual index keys for a standard blocking-based indexing approach.

Each record is inserted into several blocks according to the variations generated from its BKV. N-gram based indexing takes each blocking (or sorting) key value and converts it into a list of n-grams. A n-gram (also known as n-gram) is a substring of length q characters or $n = 3$ (called trigrams). A string s that is $c = |s|$ characters long contain $k = c - q + 1$ q -grams. The list of n-grams of a string s is generated using a sliding window approach that extracts n characters from s at any position from 1 to k of the string.

For example, the bigram list that is generated from the string 'christen' is ['ch', 'hr', 'ri', 'is', 'st', 'te', 'en'].

To create variations of a BKV, sub-lists of the n-gram list are generated in a recursive approach. If the original q -gram list contains k n-grams, then in the first step k sub-lists

of length $k - 1$ q-grams are generated. In each of these sub-lists, one n-gram is removed. The process is then applied to each of these sub-lists in a recursive manner.

Advantage of N-gram blocking: The advantage of N-gram based indexing is that it can overcome errors and variations in the BKVs, and therefore records that refer to true matches are more likely inserted into the same index list, even if their BKVs are different from each other. This leads to more true matching records being compared and thus an improved matching quality.

In the following example each token is splitted in ngrams of size 4 that are used for blocking.

```
: blocks = sparker.Blocking.create_blocks(profiles,
                                         blocking_method=sparker.BlockingKeysStrategies.ngrams_blocking,
                                         ngram_size=4)
print("Number of blocks", blocks.count())
```

```
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, from pool
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: ResultStage 26 (reduce at PythonRDD.scala:166) finished in 0.022 s
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Job 23 is finished. Cancelling potential speculative or zombie tasks for this job
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 26: Stage finished
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Job 23 finished: runJob at PythonRDD.scala:166, took 0.03346 s
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - [{"block_id": 0, "profiles": [{"R": 2, "A": 6, "S": 10, "I": 12, "M": 14, "T": 18, "N": 21, "Z": 24, "O": 26, "P": 29}, {"S": 34, "B": 38, "C": 43, "E": 46, "F": 49, "G": 51, "H": 53, "J": 55, "K": 56, "L": 57, "Q": 58, "U": 61}], "entropy": -1.0, "cluster_id": -1, "blocking_key": ""}, {"block_id": 1,
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO spark.SPARKContext: Starting job: zipWithIndex at /home/aditi/BigData/AI/Project/EntityResolution/EntityResolution/Project/ER/Project/File.zip/blockers.py:104
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Registering RDD 306 (groupByKey at /home/aditi/BigData/AI/Project/EntityResolution/EntityResolution/Project/ER/Project/File.zip/blockers.py:93) as input to shuffle 1
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Got job 24 (zipWithIndex at /home/aditi/BigData/AI/Project/EntityResolution/EntityResolution/Project/ER/Project/File.zip/blockers.py:104) with 2 output partitions
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 28 (zipWithIndex at /home/aditi/BigData/AI/Project/EntityResolution/EntityResolution/Project/ER/Project/File.zip/blockers.py:104)
[2022-04-03, 20:55:58 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:55:58 UTC INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 27)
```

3. Block Purging:

The block collection is processed to remove/shrink its largest blocks. Block Purging discards all the blocks that contain more than half of the profiles in the collection, corresponding to highly frequent blocking keys (e.g., stop-words). It considers a smoothing factor to calculate the threshold point. It will Discard all the blocks above the threshold point i.e. (remove the oversized blocks)

These are blocks that contain an excessively high number of comparisons, although they are highly unlikely to contain non-redundant duplicates (i.e., pairs of matching entities that have no other, smaller block in common). Such blocks have a negative impact on efficiency (i.e., they decrease PQ and RR), although they have a negligible contribution to PC. Therefore, the gist of Block Purging is to specify a conservative upper limit on the individual cardinality of the processed blocks so that the oversized ones are discarded without any significant impact on PC. This limit is called purging threshold.

```
j: # Performs the purging
blocks_purged = sparker.BlockPurging.block_purging(blocks, 1.025)
```

4. Block Filtering:

This approach is based on the idea that each block has a different importance for every entity profile it contains.

For example, a block with thousands of profiles is usually superfluous for most of them, but it may contain a couple of matching entity profiles that do not co-occur in another block; for them, this particular block is crucial.

A possible approach to Block Filtering would be to remove every entity profile from the least important of its blocks, i.e., the one with the largest block id.

Block Filtering removes each profile from the largest 20% blocks in which it appears, increasing the precision without affects the recall.

The filtering ratio (r) that determines the maximum number of block assignments per profile. It is defined in the interval $[0\ 1]$ and expresses the portion of blocks that are retained for each profile.

```
126]: performs the cleaning
profile_blocks, profile_blocks_filtered, blocks_after_filtering) = sparker.BlockFiltering.block_filtering_quick(blocks_purged, 0.8)
```

Algorithm 1: Block Filtering.

Input: B the input block collection
Output: B' the restructured block collection

```

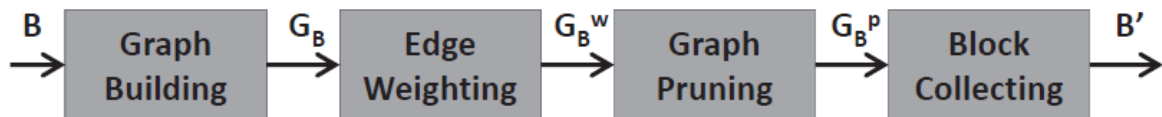
1  $B' \leftarrow \{\}$ ;
2  $\text{counter}[] \leftarrow \{\}$ ; // count blocks per profile
3  $\text{orderBlocks}(B)$ ; // sort in descending importance
4  $\text{maxBlocks}[] \leftarrow \text{getThresholds}(B)$ ; // limit per profile
5 foreach  $b_k \in B$  do // check all blocks
6   foreach  $p_i \in b_k$  do // check all profiles
7     if  $\text{counter}[i] > \text{maxBlocks}[i]$  then
8        $b_k \leftarrow b_k \setminus p_i$ ; // remove profile
9     else
10       $\text{counter}[i]++$ ; // increment counter
11   if  $|b_k| > 1$  then // retain blocks with
12      $B' \leftarrow B' \cup b_k$ ; // at least 2 profiles
13 return  $B'$ ;
```

Block Filtering is outlined in Pseudo-Code

- First, it orders the blocks of the input collection B in descending order of importance (Line 3).
- Then, it determines the maximum number of blocks per entity profile (Line 4).
- This requires an iteration over all blocks in order to count the block assignments per entity profile.
- Subsequently, it iterates over all blocks in the specified order (Line 5) and over all profiles in each block (Line 6).
- The profiles that have more block assignments than their threshold are discarded, while the rest are retained in the current block (Lines 7-10).

In the end, the current block is retained only if it still contains at least two entity profiles (Lines 11-12)

5. Meta Blocking:



At the core lies the blocking graph, a data structure that models the block assignments of the input block collection in an abstract way that decouples meta-blocking from the underlying block building method. Its nodes correspond to the clustered entities and its edges connect every pair of co-occurring entities (i.e., entities that share at least one block).

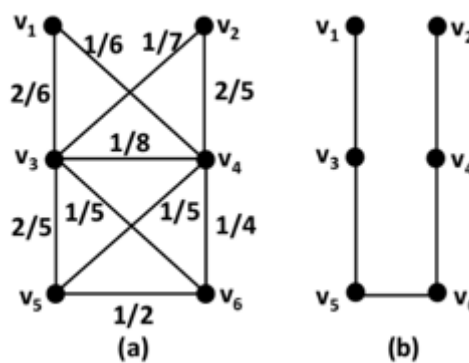
The edges of a blocking graph are naturally undirected and weighted according to a scheme that determines the trade-off between the computational cost and the gain of comparing the adjacent entities.

Meta-blocking involves four successive steps:

1. **Graph Building** receives a block collection B and derives the blocking graph GB from its block assignments.
2. **Edge Weighting** takes as input a blocking graph GB and turns it into the weighted blocking graph (G_b^w) by determining the weights of its edges. There are various techniques for this procedure.
3. **Graph Pruning** receives as input the weighted blocking graph and derives the *pruned blocking graph* (G_b^p) from it, by removing some of its edges.
4. **Block Collecting** is given as input the pruned blocking graph (G_b^p) and extracts from it a new block collection B', which actually constitutes the final output of the entire meta-blocking process.

Meta-blocking restructures a block collection B into a new one B', that contains a significantly lower number of unnecessary comparisons, while detecting almost the same number of duplicates.

The lower the weight of an edge, the more likely it is to connect non-matching entities. Therefore, Meta-blocking discards most superfluous comparisons by pruning the edges with low weights. A possible approach is to discard all edges with a weight lower than the overall mean weight.



Common blocks Scheme (CBS): - A strong indication for the similarity of two entities is provided by the number of blocks they have in common; the more blocks they share, the more likely they are to match. Hence, the weight of an edge connecting entities p_i and p_j is set equal to:

$$e_{i,j}.weight = |\mathcal{B}_{i,j}|.$$

The above weighting scheme rely on the fundamental principle of redundancy-positive blocking methods that the similarity of block assignments provides a good representation of matching probability: the more blocks two enties have in common, the more similar their profiles are expected to be.

Weighting Schemes	Pruning Algorithms
1) Aggregate Reciprocal Comparisons (ARCS)	1) Cardinality Edge Pruning (CEP)
2) Common Blocks (CBS)	2) Cardinality Node Pruning (CNP)
3) Enhanced Common Blocks (ECBS)	3) Weighted Edge Pruning (WEP)
4) Jaccard Similarity (JS)	4) Weighted Node Pruning (WNP)
5) Enhanced Jaccard Similarity (EJS)	

Pruning algorithms:

This process is based on two essential components:

- **Pruning algorithm**, which specifies the procedure that will be followed in the processing of the blocking graph, and
- **Pruning criterion**, which determines the edges to be retained.

The combination of a pruning algorithm with a pruning criterion forms a pruning scheme.

In general, the pruning algorithms can be categorized in two classes:

- The **edge-centric algorithms** iterate over the edges of a blocking graph in order to select the globally best comparisons, by filtering out those that do not satisfy the pruning criterion.
- The **node-centric algorithms** iterate over the nodes of a blocking graph with the aim of selecting the locally best comparisons for each entity (i.e., the adjacent entities with the largest edge weights).

Pruning criteria:

- *weight thresholds*, which specify the minimum weight for the edges to be retained,
- *Cardinality thresholds*, which determine the maximum number of retained edges.

Global thresholds, which define conditions that are applicable to the entire blocking graph (i.e., all the edges of the graph)

Local thresholds, which specify conditions that apply only to a subset of it (i.e. the adjacent edges of a specific node).

Pruning schemes:

1. Weight Edge Pruning (WEP):

This scheme consists of the edge-centric algorithm coupled with a global weight threshold (i.e., the minimum edge weight).

It iterates over all edges and discards those having a weight lower than the input threshold. The remaining edges form the pruned blocking graph of the output. The time complexity of this algorithm is equal to the aggregate cardinality of the original block collection.

The most critical part of this algorithm is the selection of the minimum edge weight w_{min} . Its precise value depends on the underlying weighting scheme and the resulting distribution of edge weights, in particular. In general, though, the matching entities are expected to be connected with edges of higher weights than the non-matching ones. Thus, the goal is to identify the break-even point that distinguishes the former type of edges from the latter one. Experimental evidence with real-world data sets suggests that the average edge weight provides an efficient (i.e., requires just one iteration over all edges) as well as reliable.

Weighted Edge Pruning

```
2132]: results = sparker.WEP.wep(
    profile_blocks_filtered,
    block_index,
    max_profile_id,
    weight_type=sparker.WeightTypes.CBS,
    #groundtruth=gt_broadcast,
    profile_blocks_size_index=profile_blocks_size_index
)
num_edges = results.map(lambda x: x[0]).sum()
num_matches = results.map(lambda x: x[1]).sum()
# print("Recall", num_matches/len(new_gt))
# print("Precision", num_matches/num_edges)
print("Number of comparisons", num_edges)
```

```
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Weighted Edge Pruning
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Starting job: reduce at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wep.py:132
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Got job 42 (reduce at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wep.py:132) with 2 output part
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 87 (reduce at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wep.py:132)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 86)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Missing parents: List()
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting ResultStage 87 (PythonRDD[149] at reduce at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.z
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_73 stored as values in memory (estimated size 14.7 KiB, free 362.8 MiB)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_73_piece0 stored as bytes in memory (estimated size 8.2 KiB, free 362.7 MiB)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.BlockManagerInfo: Added broadcast_73_piece0 in memory on 192.168.0.115:32997 (size: 8.2 KiB, free: 366.0 MiB)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Created broadcast 73 from broadcast at DAGScheduler.scala:1427
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting 2 missing tasks from ResultStage 87 (PythonRDD[149] at reduce at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Proj
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Adding task set 87.0 with 2 tasks resource profile 0
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSetManager: Starting task 0.0 in stage 87.0 (TID 77) (192.168.0.115, executor driver, partition 0, NODE_LOCAL, 4271 bytes) taskResourceAssignments Map()
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO executor.Executor: Running task 0.0 in stage 87.0 (TID 77)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.ShuffleBlockFetcherIterator: Getting 2 (2.2 KiB) local and 0 (0.0 B) host-local and 0 (0.0 B) push-merged-local and 0 (0.0 B)
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Removed Taskset 93.0, whose tasks have all completed, from pool
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: ResultStage 93 (sum at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:264) finished in 0.181 s
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 44 is finished. Cancelling potential speculative or zombie tasks for this job
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 93: Stage finished
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 44 finished: sum at /home/aditi/BigData/AAI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:264, took 0.182496 s
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Recall 1.0
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Precision 0.47619047619047616
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Number of comparisons 21
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - F1 score 0.6451612903225806
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
```

2. Cardinality Edge Pruning (CEP) or Top-K Edges:

This scheme combines the edge-centric pruning algorithm with a global cardinality threshold K that specifies the total number of edges retained in the pruned graph. Its goal is to retain the K edges with the maximum weight. At its core lies a sorted stack that stores the edges in descending order of weight so that the edge with the lowest one is efficiently removed (i.e., pop), when the maximum capacity K is exceeded.

The algorithm iterates over all edges of the input blocking graph twice:

- **First iteration** identifies the top- K edges and stores them in the sorted stack,
- **Second iteration** removes from the graph those edges that are not contained in the sorted stack.

Although this approach maintains the same levels of redundancy (i.e., the same number of block assignments), efficiency is significantly improved; unlike the input block collection, which contains blocks of various sizes, the output one exclusively comprises blocks of minimum size (i.e., two entities per block). This means that *CEP* minimizes the number of pairwise comparisons for a specific level of redundancy.

Cardinality Edge Pruning

```
5]: results = sparker.CEP.cep(
    profile_blocks_filtered,
    block_index,
    max_profile_id,
    weight_type=sparker.WeightTypes.CBS,
    #groundtruth=gt_broadcast,
    profile_blocks_size_index=profile_blocks_size_index
)
num_edges = results.map(lambda x: x[0]).sum()
num_matches = results.map(lambda x: x[1]).sum()
# print("Recall", num_matches/len(new_gt))
# print("Precision", num_matches/num_edges)
print("Number of comparisons", num_edges)
```

```
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Cardinality Edge Pruning
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Starting job: collect at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Registering RDD 153 (groupByKey at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104) as input to shuffle 7
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Got job 45 (collect at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104) with 2 output partitions
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 97 (collect at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 96)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Missing parents: List(ShuffleMapStage 96)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting ShuffleMapStage 96 (PairwiseRDD[153] at groupByKey at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104), which has no
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_76 stored as values in memory (estimated size 16.4 KiB, free 362.7 MiB)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_76_piece0 stored as bytes in memory (estimated size 9.2 KiB, free 362.7 MiB)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.BlockManagerInfo: Added broadcast_76_piece0 in memory on 192.168.0.115:32997 (size: 9.2 KiB, free: 366.0 MiB)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Created broadcast 76 from broadcast at DAGScheduler.scala:1427
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 96 (PairwiseRDD[153] at groupByKey at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/cep.py:104)
2022-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Addition task set 06.0 with 2 tasks resource profile 0

34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO python.PythonRunner: Times: total = 2, boot = -10, init = 20, finish = 1
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO executor.Executor: Finished task 1.0 in stage 106.0 (TID 92). 1569 bytes result sent to driver
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSetManager: Finished task 1.0 in stage 106.0 (TID 92) in 6 ms on 192.168.0.115 (executor driver) (2/2)
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Removed TaskSet 106.0, whose tasks have all completed, from pool
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: ResultStage 106 (sum at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:291) finished in 0.013 s
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 48 is finished. Cancelling potential speculative or zombie tasks for this job
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 106: Stage finished
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 48 finished: sum at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:291, took 0.014839 s
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - Recall 1.0
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - Precision 0.058823529411764705
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - Number of comparisons 170
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - F1 score 0.11111111111111111
34-03, 20:56:00 UTC] [subprocess.py:89] INFO -
34-03, 20:56:00 UTC] [subprocess.py:89] INFO -
34-03, 20:56:00 UTC] [subprocess.py:89] INFO -
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - F1_score_final 0.6451612903225806
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Starting job: count at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:330
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Got job 49 (count at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:330) with 2 output partitions
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 109 (count at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:330)
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 108)
34-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Missing parents: List()
```

3. Weight Node Pruning (WNP):

Reciprocal Weighted Node Pruning

```
]: results = sparker.WNP.wnp(
    profile_blocks_filtered,
    block_index,
    max_profile_id,
    weight_type=sparker.WeightTypes.CBS,
    #groundtruth=gt_broadcast,
    profile_blocks_size_index=profile_blocks_size_index,
    comparison_type=sparker.ComparisonTypes.AND
)
num_edges = results.map(lambda x: x[0]).sum()
num_matches = results.map(lambda x: x[1]).sum()
# print("Recall", num_matches/len(new_gt))
# print("Precision", num_matches/num_edges)
print("Number of comparisons", num_edges)
```

```
1-03, 20:56:00 UTC] [subprocess.py:89] INFO -
1-03, 20:56:00 UTC] [subprocess.py:89] INFO -
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - Reciprocal Weighted Node Pruning
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Starting job: collectASMap at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wnp.py:286
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Got job 39 (collectASMap at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wnp.py:286) with 2 output partitions
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Final stage: ResultStage 78 (collectASMap at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wnp.py:286)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 77)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Missing parents: List()
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting ResultStage 78 (PythonRDD[146] at collectASMap at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/file.zip/wnp.py:286) with 2 partitions
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_51_piece0 on 192.168.0.115:32997 in memory (size: 6.0 KiB, free: 365.9 MiB)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO memory.MemoryStore: Block broadcast_59_piece0 stored as bytes in memory (estimated size 7.9 KiB, free: 362.4 MiB)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.BlockManagerInfo: Added broadcast_69_piece0 in memory on 192.168.0.115:32997 (size: 7.9 KiB, free: 365.9 MiB)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.BlockManagerInfo: Removed broadcast_48_piece0 on 192.168.0.115:32997 in memory (size: 7.9 KiB, free: 365.9 MiB)
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO spark.SparkContext: Created broadcast 69 from broadcast at DAGScheduler.scala:1427
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Submitting 2 missing tasks from ResultStage 78 (PythonRDD[146] at collectASMap at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:236) with 2 tasks resource profile 0
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Adding task set 78.0 with 2 tasks resource profile 0
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSetManager: Starting task 0.0 in stage 78.0 (TID 71) (192.168.0.115, executor driver, partition 0, NODE_LOCAL, 4271 bytes) taskResourceAssignments Map()
1-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO storage.BlockManagerInfo: Removed broadcast_48_piece0 on 192.168.0.115:32997 in memory (size: 7.9 KiB, free: 365.9 MiB)

122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Removed TaskSet 84.0, whose tasks have all completed, from pool
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: ResultStage 84 (sum at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:236) finished in 0.097 s
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 41 is finished. Cancelling potential speculative or zombie tasks for this job
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 84: Stage finished
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 41 finished: sum at /home/aditi/BigData/AI_Project/Entity_Resolution/Entity_Resolution_Project/ER_Project/main.py:236, took 0.099206 s
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Precision 0.2325013953488372
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Recall 1.0
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - Number of comparisons 43
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO - F1 score 0.3773584905663776
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
122-04-03, 20:56:00 UTC] [subprocess.py:89] INFO -
```

Block Collection:

The last step of our meta-blocking approach transforms the pruned blocking graph into the new block collection that is returned as output. This process depends on the type of the blocking graph.

For the undirected pruned blocking graphs, which are produced by the edge-centric pruning algorithms, block collecting is straightforward: every retained edge lays the basis for creating a bilateral block of minimum size that contains the adjacent entities. As a result, the new block collection is redundancy-free (i.e., non-overlapping blocks)

For the directed pruned blocking graphs, which are derived from the node-centric pruning algorithms, block collecting creates a bilateral block for each node v_i .

	P_id	N_id	weight
0	1	8	8
1	2	9	10
2	5	12	7
3	10	14	5
4	3	10	3
5	4	11	13
6	17	18	7
7	6	13	2
8	15	16	9
9	19	20	5

```

2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 117: Stage finished
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO scheduler.DAGScheduler: Job 52 finished: showString at NativeMethodAccessorImpl.java:0, took 0.053303 s
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - +-----+-----+
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - |P_id|N_id|weight|cluster_id|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - +-----+-----+
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - | 24| 4| 56| 0|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - | 28| 8| 15| 1|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - | 26| 6| 16| 2|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - | 25| 5| 45| 3|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - | 29| 9| 28| 4|
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - +-----+-----+
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO -
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO datasources.FileSourceStrategy: Pushed Filters:
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO datasources.FileSourceStrategy: Post-Scan Filters:
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO datasources.FileSourceStrategy: Output Data Schema: struct<id: int, DisplayName: string, Location: string,
2022-04-03, 20:56:00 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:00 UTC INFO codegen.CodeGenerator: Code generated in 13.939207 ms

```

Final Output:

```

022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 129: Stage finished
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO scheduler.DAGScheduler: Job 64 finished: showString at NativeMethodAccessorImpl.java:0, took 0.015520 s
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO codegen.CodeGenerator: Code generated in 5.382096 ms
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - +-----+-----+
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | id| DisplayName| Location| AboutMe|Age| phone| City| State| Country|new_id|cluster_id|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - +-----+-----+
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 0| Ceppi Marco| Vienna, VA|I'm a Moderator ....| 32|375-304-3599| Vienna| VA| null| 4| 0|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 1| Marco Ceppi| Vienna, VA|I'm a Moderator ....| 32|375-304-3599| Vienna| VA| null| 24| 0|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 2| jacksonh| Boston, MA|I am a developer ...| 35|490-267-2635| Boston| MA| null| 8| 1|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 3| jacksonh| Boston, MA|I am a developer ...| 35|490-267-2635| Boston| MA| null| 28| 1|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 4| Jim Fiorato|Wheaton, Illinois...|I'm very importan...| 39|757-448-8075| Wheaton|Illinois United S...| null| 6| 2|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 5| Jim Fiorato|Wheaton, Illinois...|I'm very importan...| 39|757-448-8075| Wheaton|Illinois United S...| null| 26| 2|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 6|Benjamin Pollack|Raleigh, NC, Unit...|I work at Bakpax,...| 30|828-802-3842| Raleigh| NC|United States| 5| 3|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 7|Benjamin Pollack| USA|I work at Bakpax,...| 30|828-802-3842| USA| null| null| 29| 3|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 8| Dale Ragan| United States|I am an Electrica...| 18|319-613-6622|United States| null| null| 9| 4|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - | 9| Dale| USA|I am an Electrica...| 18|319-613-6622| USA| null| null| 29| 4|
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - +-----+-----+
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO -
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - File written successfully on HDFS .
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO spark.SparkContext: Invoking stop() from shutdown hook
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO server.AbstractConnector: Stopped Spark@657778c[HTTP/1.1, (http/1.1)](0.0.0.0:4040)
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO ui.SparkUI: Stopped Spark web UI at http://192.168.0.115:4040
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO memory.MemoryStore: MemoryStore cleared
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO storage.BlockManager: BlockManager stopped
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO storage.BlockManagerMaster: BlockManagerMaster stopped
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO scheduler.OutputCommitCoordinatorOutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO spark.SparkContext: Successfully stopped SparkContext
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO util.ShutdownHookManager: Shutdown hook called
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-d8afacfd-47ed-4051-9548-1906c43f92f
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-d8afacfd-47ed-4051-9548-1906c43f92f/pyspark-dec49e2a-eb56-4536-be31-607d6b0af491
022-04-03, 20:56:01 UTC] {subprocess.py:89} INFO - 2022-04-03, 20:56:01 UTC INFO util.ShutdownHookManager: Deleting directory /tmp/spark-1a37d831-cdbb-4930-b280-afc2439995cc
022-04-03, 20:56:02 UTC] {subprocess.py:93} INFO - Command exited with return code 0
022-04-03, 20:56:02 UTC] {taskinstance.py:1247} INFO - Marking task as SUCCESS. dag_id=ER_Project, task_id=project_commands, execution_date=20220403T152525, start_date=20220403T152528, end_date=20220403T152602
022-04-03, 20:56:02 UTC] [local_task_job.py:154] INFO - Task exited with return code 0
022-04-03, 20:56:02 UTC] [local_task_job.py:264] INFO - 1 downstream tasks scheduled from follow-on schedule check

```

Dirty ER output:

Out[2168]:

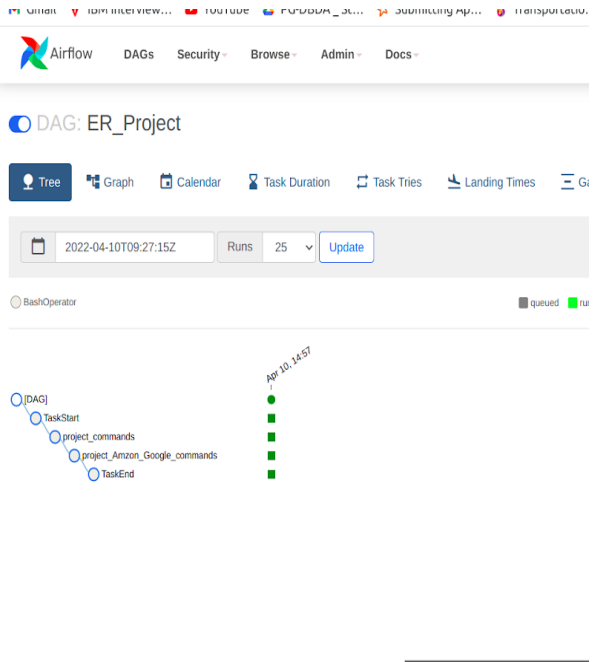
cluster_id	id	name	fathername	mothername	dateofbirth	location	city	state	country	pincode	gender	age	qualification	
0.0	1	Omkar jori	Nathu jori	Aruna jori	1999-01-13	kothrud	Pune	Maharastra	India	411038.0	M	22	BE in mechanical	swin
	8	Omkar nathu jori	Nathu jori	Aruna jori	1999-01-13	Kalashree MD Rivera Bhugaon	Pune	Maharastra	India	411041.0	M	22	BE in mechanical	swin
1.0	2	Mr Gupta	Suresh	Neelam gupta	1994-03-11	None	Orai	Uttar pradesh	India	285001.0	M	27	diploma in CSE	
	9	Rohit gupta	Suresh chandra gupta	Neelam gupta	1994-03-11	tulsi nagar	Orai	Uttar pradesh	India	285001.0	M	27	BE in ECE	
2.0	5	Aditi Patil	Hemant Patil	Yogita bonde	1999-08-22	khanda colony	Panvel	Maharastra	India	410206.0	F	22	BE in EXTC	
	12	Patil Aditi	Hemant Patil	Yogita Patil	1999-08-22	Gurudev Arcade	Panvel	Maharastra	India	410206.0	F	22	BE in EXTC	
4.0	3	Amruta Argade	Diwakar Argade	Sunita Argade	1993-09-14	sankul, Near zeal engineering college	Pune	Maharastra	India	411041.0	F	28	ENTC	de
	10	Argade Amruta Diwakar	Diwakar Argade	Sunita Argade	1993-09-14	Keshav rukmini sankul, Near zeal engineering c...	Pune	Maharastra	India	411041.0	F	30	BE in ENTC	de
	14	Mrs Deshmukh	Diwakar Argade	Sunita Argade	1993-09-14	Keshav rukmini sankul, Near zeal engineering c...	Baner	Maharastra	India	411041.0	F	28	BE in ENTC	de
5.0	4	varun jhaa	Arvind Jha	Urvashi Jha	1998-07-15	M-04,Vaishali Nagar	Jaipur	Rajasthan	India	302012.0	M	24	BE in CSE	sle
	11	varun jha	Arvind Jha	Urvashi Jha	1998-07-15	M-04,Ashiana Mangalam,Vaishali Nagar	Jaipur	Rajasthan	India	302012.0	M	24	BE in CSE	sle
6.0	17	Amruta Bhosale	Pradeep Bhosale	Lata Bhosale	1999-06-19	Sai Krupa housing	Panvel	Maharashtra	India	410206.0	F	23	BE IT	C
	18	Ms Amruta Lokhande	Pradeep Bhosale	Lata Bhosale	1999-06-19	Sector 10, Neelam Arcade	Hyderabad	Telangana	India	512007.0	F	23	BE IT	C Tee
7.0	6	Bhumik Patil	Patil Hemaant	Lalita bonde	2003-09-06	Sai darshan	NewPanvel	Maharashtra	India	400026.0	M	18	Btech EXTC NIT	i
	13	Bhumika Hemant Patil	Hemant Patil	None	2000-09-16	Sai darshan	Navi Mumbai	Maharashtra	India	400046.0	F	21	Btech	sle
8.0	15	Amruta Argade	Anad Argade	Shweta Argade	2003-08-16	Anand Society	Solapur	Maharashtra	India	420036.0	F	18	CSE	Cc
	16	Amruta Anand Argade	Anad Argade	Shweta Argade	2003-08-16	Gandhi nagar	Solapur	Maharashtra	India	420036.0	F	18	Computer science	Cc
9.0	19	Aditee Yadav	Shrikant Yadav	Mamta Yadav	None	Gulmohar Society	Panji	Goa	India	656565.0	F	28	Btech CS	Gi
	20	Aditee Deshmukh	Shrikant Yadav	Mamta Yadav	None	Bakers Street	New York city	New York	US	NaN	F	28	Btech CS	i T

10. Airflow

10.1 Airflow code and output:

```

1 from airflow.models import DAG
2 from airflow.operators.bash import BashOperator
3 from airflow.utils.dates import days_ago
4 args = {'owner': 'Project_Team'}
5 dag = DAG(
6     dag_id='ER_Project_New',
7     default_args=args,
8     start_date=days_ago(1),
9     schedule_interval=None,
10    params={},
11    tags=['edbda', 'spark', 'spark-submit'])
12
13 bashOp1 = BashOperator(
14     task_id='project_commands',
15     start_date=days_ago(1),
16     bash_command='scripts/er_command.sh',
17     params=None,
18     default_args={},
19     dag=dag)
20
21
22 bashOp2 = BashOperator(
23     task_id='project_Amazon_Google_commands',
24     start_date=days_ago(1),
25     bash_command='scripts/Amazon_Google_er_command.sh',
26     params=None,
27     default_args={},
28     dag=dag)
29
30
31 taskStart = BashOperator(task_id="TaskStart",
32     bash_command="echo start_project",
33     cwd="/tmp",
34     dag=dag)
35
36 taskEnd = BashOperator(task_id="TaskEnd",
37     bash_command="echo end_project",
38     cwd="/tmp",
39     dag=dag)
40
41 taskStart >> bashOp1 >> bashOp2 >> taskEnd
        
```



The screenshot shows the Airflow web interface for the DAG 'ER_Project'. It displays a task dependency graph with nodes: [DAG], TaskStart, project_commands, project_Amazon_Google_commands, and TaskEnd. The 'project_commands' task is currently running, indicated by a green dot. The interface also shows the DAG's last run time as 2022-04-10T09:27:15Z and the number of runs as 25.

```

# create dir stage in /ER_Project/stage
hdfs dfs -mkdir /ER_Project
hdfs dfs -mkdir /ER_Project/stage

hdfs dfs -put /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/ground_truth_sample.csv /ER_Project/stage/ground_truth

#Ingest (csv) data to MYSQL
mysql --local-infile=1 -h localhost -u root --password=Amruta@2021 -D ER_Project -e 'source /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/sql_command.sql'
# MYSQL to HDFS (csv)
spark-submit --master local /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/1_SparkSQL.py

#Ingest(json) data into MONGODB
mongoimport --uri mongodb+srv://CDAC:CDAC@cluster0.qqp7g.mongodb.net/ER_Project --collection Sample --type csv --headerline --file /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Pro
#MONGODB to HDFS (json)
spark-submit --packages org.mongodb.spark:mongo-spark-connector_2.12:3.0.0 --master local /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/2_SparkMongo.py

#ingest data to Cassandra
cqlsh -u 'cassandra' -p 'cassandra' -f /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/CQL_command.cql
# Cassandra to HDFS
spark-submit --packages com.datastax.spark:spark-cassandra-connector_2.12:3.1.0 --master local /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/3_SparkCassandra.py

#Spark-ER stage
spark-submit --py-file /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/file.zip --master local /home/gaurav/DBDA_HOME/DBDA_CODE/SPARK/PYTHON_PROJECT1/ER_Project/main.py
    
```

Browse Directory

/ER_Project/stage

Go!

Show

25

entries

Search:

<input type="checkbox"/>		Permission		Owner		Group		Size		Last Modified		Replication		Block Size		Name	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:02		0		0 B		Amazon_Google_Examples	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 14:59		0		0 B		ER_dataset1	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:00		0		0 B		ER_dataset2	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 14:57		0		0 B		dataset1	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 14:57		0		0 B		dataset2	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 14:57		0		0 B		ground_truth	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:01		0		0 B		output	

Showing 1 to 7 of 7 entries

Previous

1

Next

Browse Directory

/ER_Project/stage/Amazon_Google_Examples

Go!

Show

25

entries

Search:

<input type="checkbox"/>		Permission		Owner		Group		Size		Last Modified		Replication		Block Size		Name	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:01		0		0 B		Amazon_data	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:01		0		0 B		Google_data	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:01		0		0 B		ground_truth	
<input type="checkbox"/>		drwxr-xr-x		rohit		supergroup		0 B		Apr 10 15:02		0		0 B		output	

Showing 1 to 4 of 4 entries

Previous

1

Next

Hadoop, 2021.

11. Conclusion

Although ER has been studied for more than three decades in different computer science communities, it still remains an active area of research. The problem has enjoyed a renaissance during recent years, with the avalanche of data-intensive descriptions of real-world entities provided by government, scientific, corporate or even user-crafted data sources. Reconciling different entity descriptions in the Big Data era poses new challenges both at the algorithmic and the system level:

Volume, due to the very high number of entities and data sources,
Variety, due to the extreme schema heterogeneity,
Velocity, due to the continuously increasing volume of data, and Veracity,

Due to the high level of noise and inconsistencies.

In this project, we have focused on how different “sparkER” algorithms performs when used in “Real World” scenario of Entity identification and evaluated their performance using metrics like F-score to identify performance of best algorithm.

Indexing/Blocking is the step to tackle quadratic complexity of record matching and reducing the time taken in comparison and tackle the issue of Volume

Block Purging and Block Filtering is the step to tackle mainly through a schema-agnostic, non-learning functionality. Most matching methods employ a schema-agnostic, collective functionality, which leverages information provided by related entities, in order to address Variety and Veracity. Budget-aware ER methods rely on Blocking and a usually schema-agnostic functionality to simultaneously address Volume and Variety, while Incremental Methods address Volume and Velocity through Blocking, but their schema-aware functionality prevents them from tackling Variety, too. In all cases, massive parallelization, usually through the MapReduce framework, plays an important role in further improving scalability and, thus, addressing Volume. Note, though, that we share the view of ER as an engineering task by nature, and hence, we cannot just keep developing ER algorithms in a vacuum. In the Big Data era, we opt for open-world ER systems that allow to plug-and-play different algorithms and can easily integrate with third-party tools for data exploration, data cleaning or data analytics.

In this paper, we introduced two techniques for boosting the efficiency of Meta-blocking along with two techniques for enhancing its effectiveness.

Our thorough experimental analysis verified that in combination, our methods go well beyond the existing Metablocking techniques in all respects and simplify its configuration, depending on the data and the application at hand.

For efficiency-intensive ER applications,

Reciprocal CNP processes a large heterogeneous dataset with 3 millions entities and 80 billion comparisons within 2 hours even on commodity hardware; it also manages to retain recall and precision above 0.8 and 0.1, respectively.

For effectiveness-intensive ER applications,

Reciprocal WNP processes the same voluminous dataset within 5 hours on commodity hardware, while retaining recall above 0.95 and precision close to 0.01.

In both cases, Block Filtering and Optimized Edge Weighting are indispensable.

In the future, we plan to adapt our techniques for Enhanced Meta-blocking to Incremental Entity Resolution.

12. Future Scope

13. Bibliography

1. <https://github.com/Gaglia88/sparker/tree/master/python>
2. <https://core.ac.uk/download/pdf/304104038.pdf>
3. Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection by Peter Christen [ISBN 978-3-642-31164-2] published by: © Springer- Verlag Berlin Heidelberg 2012
4. Papadakis, George & Papastefanatos, George & Palpanas, Themis & Koubarakis, Manolis. (2016). Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-blocking.
5. Papadakis, George & Ioannou, Ekaterini & Niederee, Claudia & Fankhauser, Peter. (2011). Efficient Entity Resolution for Large Heterogeneous Information Spaces. Proceedings of the 4th ACM International Conference on Web Search and Data Mining, WSDM 2011. 535-544. 10.1145/1935826.1935903.
6. P. Christen(2011).A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication
7. <https://github.com/Gaglia88/sparker>
8. <https://sourceforge.net/projects/sparker/files/datasets/>
9. *An Entity Resolution framework developed in Scala for Apache Spark.*

If use this library, please cite:

```
@inproceedings{sparker,  
  author = {Luca Gagliardelli and  
            Giovanni Simonini and  
            Domenico Beneventano and  
            Sonia Bergamaschi},  
  title = {SparkER: Scaling Entity Resolution in Spark},  
  booktitle = {Advances in Database Technology - 22nd International Conference on  
               Extending Database Technology, {EDBT} 2019, Lisbon, Portugal, March  
               26-29, 2019},  
  pages = {602--605},  
  publisher = {OpenProceedings.org},
```


year = {2019},
doi = {10.5441/002/edbt.2019.66}
}

10. Simonini, G., Bergamaschi, S., & Jagadish, H. V. (2016). BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution. PVLDB, 9(12), 1173–1184.