CORE
Metadata, citation and similar papers at core.ac.uk
Provided by UPCommons. Portal del coneixement obert de la UPC

Informatics Engineering Final Project

Optimal Assignment Problem on Record Linkage

Pablo Rodríguez Fernández

Director: Jordi Nin Guerrero

June 4, 2013

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Universitat Politècnica de Catalunya (UPC)
Barcelona School of Informatics (FIB)
Department of Computer Architecture (AC)

Ever tried. Ever failed.
No matter. Try again.
Fail again. Fail better.

Samuel Beckett
(1906-1989)

Agraïments

De vegades ens oblidem dels privilegis que tenim. Si alguna cosa hem d'aprendre dels temps que corren i les dificultats que hi trobem, com a mínim ha de ser la humilitat i la capacitat de valorar el que tenim i el sacrifici que han de fer i han fet molts per ser on som ara. No vull desaprofitar aquesta ocasió per tractar de reconèixer aquest sacrifici i esforç que molts han fet per a que jo pugui presentar-vos aquest projecte de fi de carrera.

En primer lloc, haig d'agrair tot l'esforç i sacrifici dels meus pares, que han treballat molts anys per a que jo hagi pogut estudiar allò que m'agrada.

En segon lloc, a en Jordi Nin, el meu tutor del projecte, que m'ha ajudat i guiat en tot aquest procés, i ha tingut una gran paciència amb mi i amb la meva particular forma de treballar.

No seria just no esmenar també a aquells professors que al llarg de tota la meva vida acadèmica m'han ensenyat i ho han fet posant-hi tota la seva energia i passió, contagiant-la i fent que sigui molt més fàcil i agradable arribar fins aquí. Alguns d'ells són en Pep, l'Álex Borrás, en Miquel Gallart, en Joan Marqués, en Miquel Vallonesta, en Sergi Ramos, la Dolors, l'Albert Ghanime, en José Luís Galisteo, en José Cabré i molts, molts altres. A tots vosaltres, moltes gràcies; us guardo un gran record dins el cor.

D'altra banda, també haig d'agrair a molts dels companys que m'han acompanyat aquests últims anys, els quals m'han ajudat molt més del que ells es puguin pensar.

Finalment, d'acord amb la teoria del caos, un petit canvi pot acabar provocant un de gran després d'un llarg procés; per aquest motiu, crec que es just acabar donant les gràcies a tots aquells que, directa o indirectament, els hagi conegut o no, han aportat el seu petit gra de sorra per fer que jo sigui avui amb vosaltres, presentant el projecte que dona fi a aquest cúmul d'esforç i treball fet durant tants anys.

I a tu, Carla, gràcies per ser-hi; sense tu, tot això no tindria tant de sentit.

Abstract

Many legislations require statistical agencies to publish data without revealing confidential information about data owners. Statistical Disclosure Control (SDC) is the discipline in charge of ensure data owners privacy and data utility in statistical surveys. Hence, main goal of SDC methods is to minimize *disclosure risk* maintaining at the same time an acceptable *information loss* level. In practice, disclosure risk is usually measured using Record Linkage algorithms, which are database integration methods able to relate different database entries belonging to the same individual. Since most of record linkage algorithms currently used are heuristic for performance reasons, disclosure risk is usually underestimated.

In this project, we present an application of the Hungarian Method, an optimal assignment graph theory algorithm, to record linkage in order to improve the disclosure risk assessment. However, we should note that Hungarian Method has $O(n^3)$ complexity, because of this, three different methods are presented in order to reduce its computational cost.

Contents

List of Tables

List of Figures

1 Introduction

Everyday a huge amount of data is gathered from statistical surveys; to be useful, these data must be treated statistically and properly published by Statistical Agencies. When those data are confidential information of citizens, they must be protected to prevent third parties to disclosure private knowledge about data owners.

Statistical Disclosure Control (SDC) [14] and *Privacy-Preserving Data Mining* (PPDM) [1] have been studied extensively in order to manage this matter. A large number of algorithmic techniques and theoretical models have been designed for privacy-preserving data releasing.

Anonymization methods attempt to solve the privacy problem in statistical surveys. However, when they are applied, a new important challenge arises: the disclosure risk evaluation of these methods. Roughly speaking, disclosure risk evaluates the privacy of data owners against possible malicious uses that third parties could do with the information released. Traditionally, disclosure risk is empirically evaluated as the number of individuals whose identity is revealed when a database is released.

To do that, record linkage methods are applied between the original and the anonymized released database. Since record linkage methods are not optimal, disclosure risk is systematically underestimated having a negative impact on the security of anonymization methods.

1.1 Motivations and objectives

Since social networks irruption like Facebook, Orkut, Tuenti, Twitter, etc., people are increasingly dumping information about themselves; information that, in a first look, may be not sensitive. However, this amount of new information can be combined with the data released by statistical agencies to obtain sensitive information about citizens [18]. Moreover, if disclosure risk of released databases is underestimated, it signifies a serious security risk for data owners. In order to minimize this problem, in this project we will propose and study the use of optimal assignment algorithms for disclosure risk assessment.

Optimal assignment algorithms have been largely studied; nevertheless, as far as we know, this is the first attempt to introduce them into disclosure risk evaluation. The main drawback of optimal assignment algorithms is their high computational complexity. For this reason, in this project we will provide an efficient implementation of the Hungarian Algorithm [11], a well-known assignment method; we will introduce some data management tricks, as data clustering or blocking, to reduce its execution time on very large databases; and, finally, we will deliver an implementation in Hadoop framework, which provides map-reduce methodology, largely used in environments where great amount of data need to be processed.

1.2 Structure of the Document

This document is structured in three parts, which are described as follows.

In the first part, composed by chapters 2 and 3, we expose the basis of the project. First, we describe the protection scenario and the assessment process. Next, we depict main work which this project is based on.

Second part, which contains chapters 4 and 5, describing our contributions and show our results. Chapter 4 does a preliminary comparison between standard record linkage algorithms and the proposed one. Later, in Chapter 5, three methods are proposed in order to ease the large data processing.

Finally, the third part is the closure of this project. It includes Chapter 6, which are related to the project elaboration and its cost, and Chapter 7, which is composed by a short summary of the work done, the conclusions and the suggested future work.

2 Preliminaries

In this section, we present the main techniques or methods used in this project and some related work about them.

First, we introduce the main clustering definitions and its main classification. Next, we explain Hadoop and Map-Recude and how they work. Finally, we describe one of the cores of this project, the Optimal Assignment Problem, and how it is solved.

2.1 Clustering

Clustering is the task of relating similar elements in a dataset to build groups and create general representations (centroids). It is widely used in many fields as data mining, machine learning, image analysis, etc.

Since clustering is a very general concept, there are a great variety of algorithms that apply clustering, focusing, for example, on possible centroids, density, data structures, etc.

In this project, we apply clustering in order to divide datasets into big clusters with those records which are closer, making it more probable that they will reveal correct links.

Clustering algorithms are divided into two different categories: space partitioning, also called top-down methods, and hierarchical methods, known as bottom-up methods as well. Now, we review two well-known clustering algorithms which illustrate these two categories: k-means (a space partitioning method) and the agglomerative hierarchical clustering.

k-means algorithm [21]. It is one of the most commonly used clustering techniques. It is an algorithm to cluster n objects into k partitions ($k < n$). K-means starts by partitioning randomly the input objects into k initial sets. Then, it calculates the centroid of each set. Following, it constructs a new partition by associating each object

with the closest centroid. Finally, the centroids are recalculated for the new clusters. This algorithm is repeated until it convergences, i.e. there is no changes in its centroids. Figure 2.1 shows an example of k-means over a dataset of 100 gaussian random pairs.

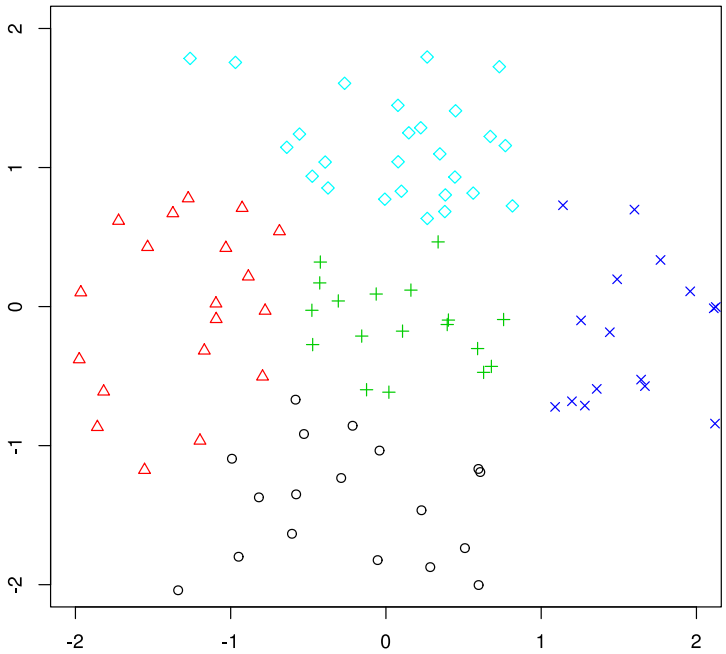


Figure 2.1: Example of clustering using k-means algorithm with $k = 5$

Agglomerative hierarchical clustering [15]. This method builds a hierarchy tree, called dendrogram, from the individual elements by progressively merging clusters. Note that, at the beginning each element is considered as an independent cluster. The algorithm starts computing a distance matrix among all the elements to be clustered, where the distance in the (i, j) position corresponds to the distance between the i-th and j-th elements. Then, when clustering progresses, the corresponding rows and columns have to be also merged. This algorithm does not explicitly builds a number of clusters, instead, we must decide the number of clusters and where we split them within the dendrogram. An example, over the same dataset as Figure 2.1, is illustrated by Figure 2.2, which suggests to cut at height 3 or 4.

2.2 Hadoop

Hadoop is a Java framework that implements Map-Reduce, a distributed cloud-computing model. It is widely used by many companies such as Yahoo, Facebook, Last.fm, among

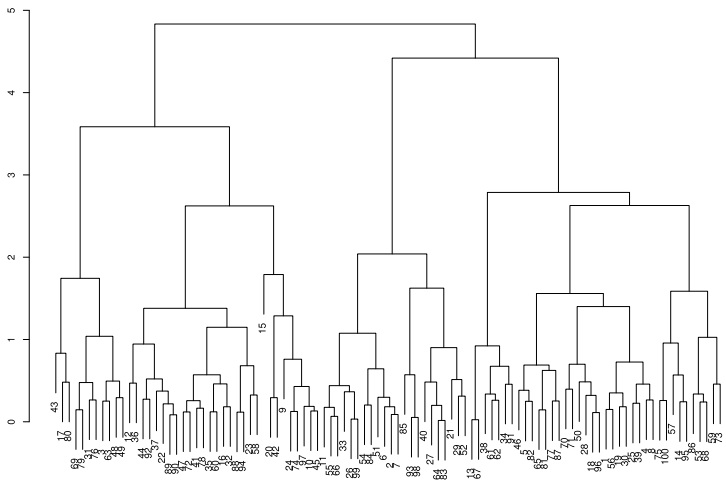


Figure 2.2: Example of dendrogram after applying hierarchical clustering

others[1] [29], and it can be download from the Apache Foundation[2].

Map-Reduce methodology proposes to divide a big dataset into smaller parts to be processed at different nodes in a data center; this step is called a *map* operation. Then data is processed in parallel and results are finally combined using a *reduce* operation to deliver the user the final aggregated result. However, as we will see later on following chapters, although many tasks are expressible in this model, some algorithms cannot be efficiently implemented with map-reduce because they need to perform many central computations with the complete dataset.

One of the key aspects of a distributed environment is the data access and modifications, specially in an environment which has been called Big Data, where Map-Reduces is mainly addressed. Because of that, Hadoop provides a Distributed FileSystem, HDFS, based on the Google FileSystem (GFS) [10]. This filesystem has been designed to run on cheap hardware, with huge files which are more appended than modified, on a distributed environment and to optimize bandwidth. Files in it are split on large blocks (64 MB), whose are replicated and transferred to nodes which executes the tasks queued, called tasktrackers. All these operations are managed by a jobtracker server and a HDFS server, as we explain later on.

2.2.1 How Hadoop Works

Here, we need to deeper describe how Hadoop works in order to easily understand the results depicted in Chapter 5.

[1] A complete list can be found at `http://wiki.apache.org/hadoop/PoweredBy`.
[2] Hadoop website is at `http://hadoop.apache.org`.

First, we must enumerate the different elements that compose a Hadoop execution:

- The client, which submits the job.
- The jobtracker, which coordinates the execution.
- The tasktrackers, which run the tasks that the job has been split into.
- The distributed filesystem, typically HDFS, which is used for sharing job files between the other elements.

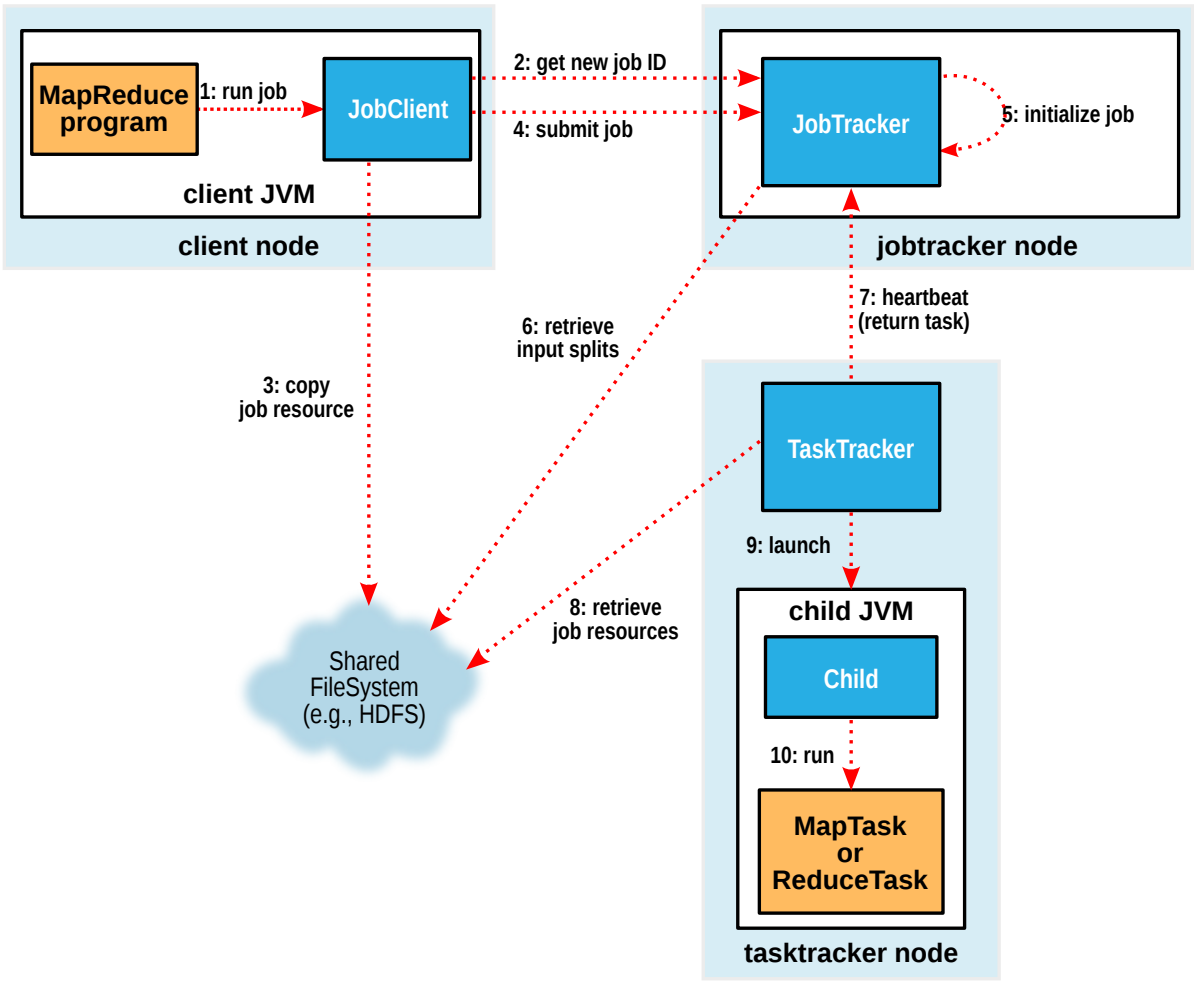


Figure 2.3: Hadoop launch job process

Then, when the client launches a job, Hadoop follows several steps, illustrated in Figure 2.3 extracted from [29], in order to execute it:

1. Client calls *submitJob()* function.
2. Get a job ID from jobtracker.
3. Checks the output directory of the job, computes the input splits and copies the resources needed to run the job, like JAR or configuration files to the jobtracker.
4. Tells the jobtracker that the job is ready for execution.
5. Jobtracker enqueue the job, which will be initialized by the job scheduler.
6. Retrieves input splits and builds the list of tasks to execute, with a mapper for every input split and a number of reduces determined by the client or the default configuration.
7. When Jobtracker receives a heartbeat of a Tasktracker and it indicates that it is ready for the next task, Jobtracker allocates a task for it using the return value of the heartbeat.
8. Once that the tasktracker has been assigned a task, it copies the job JAR to its local filesystem.
9. Launches a new Java Virtual Machine (JVM) to isolate the job execution of tasktracker.
10. Finally, this last JVM runs the task.

Process followed by Hadoop to allocate a job

2.3 Optimal Assignment Problem

Traditionally, Optimal Assignment Problem has been related to the *transportation problem* [24]. It has been studied since 1931 [17], getting new perspective in 1950s with linear programming techniques. Since then, have appeared many algorithms to solve different versions of the same problem on different scenarios.

The main idea behind this problem is to assign a certain number of agents (or workers) to some tasks, maximizing a *benefit* or minimizing a *cost*. It is formally defined as follows [5]:

Definition 1 (Optimal Assignment Problem) *Given a weighted complete bipartite graph $G = (X \cup Y, X \times Y)$, where edge xy, from node $x \in X$ to node $y \in Y$, has weight $w(xy)$, the goal is find a matching M from X to Y with maximum weight.*

This problem can be solved in $O(n^3)$ applying the Hungarian Method described in the following section.

2.3.1 Hungarian Method

Hungarian (also known as Khun-Munkres) Method, was presented in 1955 by Harold W. Kuhn and was proved as polynomial algorithm by James Munkres in 1957 [24]. Its original complexity was $O(n^4)$, but Jack Edmonds and Richard Karp presented a version with complexity $O(n^3)$ in 1972 [9]. We will start from this latter version for our implementations.

In order to explain the version of the algorithm that we used [5, 11], first we should define some concepts:

Definition 2 (Feasible vertex labeling) *Given a weighted complete bipartite graph $G = (X \cup Y, X \times Y)$, where edge xy has weight $w(xy)$, a* feasible vertex labeling *in G is a real-valued function l on $X \cup Y$ such that $\forall x \in X, \forall y \in Y,\ l(x) + l(y) \geq w(xy)$.*

A trivial example of feasible labeling is $\forall x \in X, \forall y \in Y,\ l(y) = 0,\ l(x) = max_{y \in Y}(w(xy))$.

Definition 3 (Equality subgraph) *Given a graph G, and a feasible labeling l, an* equality subgraph *G_l is that which contains only those edges where $l(x) + l(y) = w(xy)$.*

Definition 4 (Neighbourhood) *Given an equality subgraph G_l, the neighbour of a vertex u is defined as $N(u) = \{v : (u, v) \in G_l\}$.*

Definition 5 (Alternating path) *Given an equality subgraph G_l and a matching M of G_l, an* alternating path *is a path which its edges alternates between M and $G_l - M$.*

If an alternating path starts and ends with a free vertex, it is called *augmenting path.*

Definition 6 (Alternating tree) *Given an equality subgraph G_l and a matching M of G_l, an* alternating tree *is a tree which its root is a free vertex and every path within it is an alternating path.*

Considering the previous definitions, the Hungarian algorithm works as follows:

1. Generate initial labeling l and matching M in G_l.
2. If M is *perfect*, then the algorithm is over. Otherwise, pick a free vertex $u \in X$, set it as root of alternating tree and set $S = \{u\}$, $T = \emptyset$.
3. If $N(S) == T$ update labels.
4. Pick $v \in N(S) - T, v \in Y$
   - If v is free, $v - u$ is an augmenting path; augment M and go to step 2.
   - Else, name z the vertex matched with v, augment the alternating tree, set $S = S \cup \{z\}$, $T = T \cup \{v\}$ and go to step 3.

Now, we will describe two key aspects of the Hungarian Method.

Update labels

Updating labels force $N(S) \neq T$, augmenting G_l size. Therefore, given

$$m = min_{x \in S, y \notin T}\{l(x) + l(y) - w(xy)\}$$

update labels as

$$l'(v) = \begin{cases} l(v) - m & \text{if } v \in S \\ l(v) + m & \text{if } v \in T \\ l(v) & \text{otherwise} \end{cases}$$

Implementation details

In order to keep time complexity equal to $O(n^3)$, $\forall y \notin T$ we keep track of

$$slack_y = min_{x \in S}\{l(x) + l(y) - w(xy)\}$$

Keeping this information able us to update labels with linear complexity, being $m = min_{y \in T} slack_y$. Then, we update slack as $\forall y \notin T, slack_y = slack_y - m$.

2.3.2 Example

In this section, given the graph described on Figure 2.5, we will show an example of applying the Hungarian Method in order to solve the Optimal Assignment Problem as follows:

	A	B	C	D
1	2	5	3	4
2	7	7	7	5
3	6	8	5	8
4	6	6	1	3

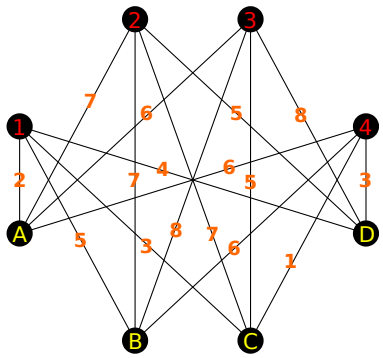
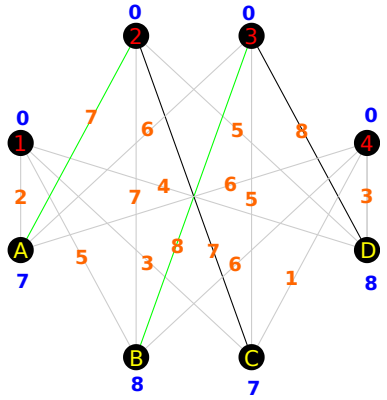


Figure 2.5: Description of the weighted complete bipartite graph used as example.

1. First, do an initial labeling, and then, initial matching is $M = \{(2, A), (3, B)\}$:

	A	B	C	D	L
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*0*
3	6	**8**	5	**8**	*0*
4	6	6	1	3	*0*
L	*7*	*8*	*7*	*8*	L



2. Pick up free vertex and set $S = \{C\}$, $T = \emptyset$, then $N(S) = \{2\}$; also set free vertex as root of alternating tree:

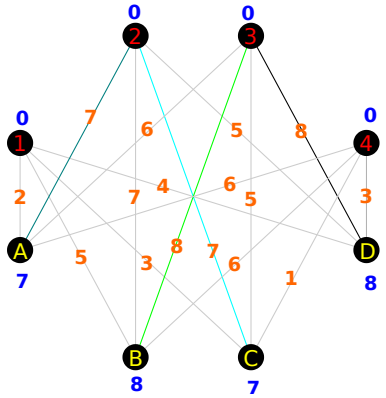
	A	B	C	D	L
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*0*
3	6	**8**	5	**8**	*0*
4	6	6	1	3	*0*
L	*7*	*8*	*7*	*8*	L

3. $N(S) == T$? No.

4. Pick up vertex 2, but it is not free, so we add it to set $T = \{2\}$ and vertex which is connected to set $S = \{C, A\}$. We update $N(S) = \{2\}$ and alternating tree; then,

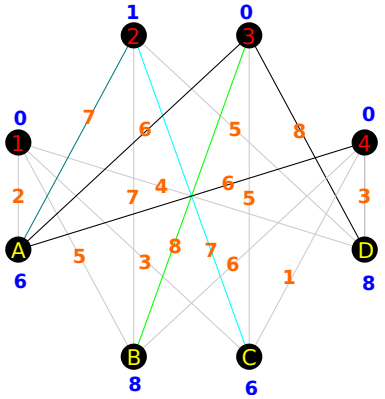
we go to step 3:

	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*0*
3	6	**8**	5	**8**	*0*
4	6	6	1	3	*0*
L	*7*	*8*	*7*	*8*	***L***



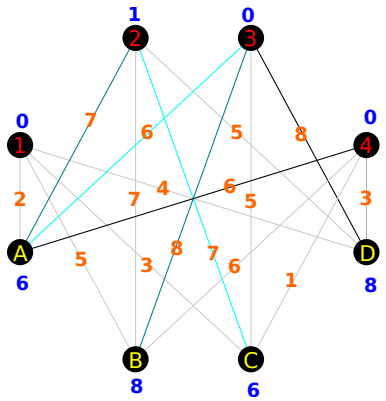
3. $N(S) == T$? Yes. Update labels with $m = 1$, so $N(S) = \{2, 3, 4\}$:

	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*1*
3	**6**	**8**	5	**8**	*0*
4	**6**	6	1	3	*0*
L	*6*	*8*	*6*	*8*	***L***



4. Pick up vertex 3, which is not free, so we add them to sets, $S = \{C, A, B\}$, $T = \{2, 3\}$, then $N(S)$ remains equal:

	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*1*
3	**6**	**8**	5	**8**	*0*
4	**6**	6	1	3	*0*
L	*6*	*8*	*6*	*8*	***L***

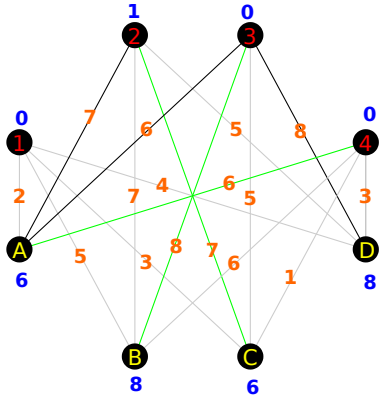


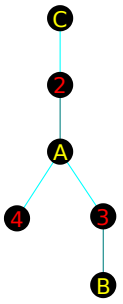
3. $N(S) == T$? No.

4. Pick up vertex 4, which is free, so $C - 4$ is an augmenting path, being $M =$

$\{(2, C), (3, B), (4, A)\}$:

	A	B	C	D	L
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*1*
3	**6**	**8**	5	**8**	*0*
4	**6**	6	1	3	*0*
L	*6*	*8*	*6*	*8*	***L***





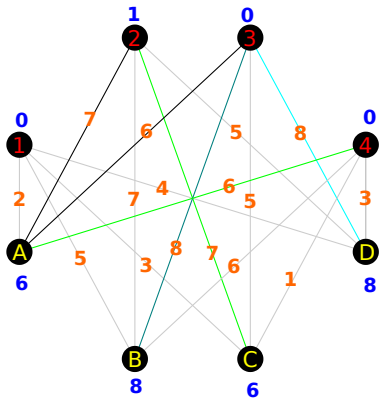
2. Pick up free vertex D, getting sets $S = \{D\}$, $T = \emptyset$, $N(S) = \{3\}$:

	A	B	C	D	L
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*1*
3	**6**	**8**	5	**8**	*0*
4	**6**	6	1	3	*0*
L	*6*	*8*	*6*	*8*	***L***

3. $N(S) == T$? No.

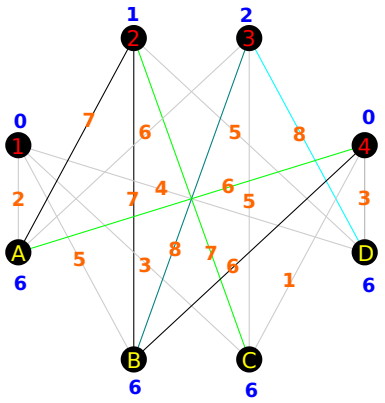
4. Pick up vertex 3, which is not free, and set $S = \{D, B\}$, $T = \{3\}$, $N(S) = \{3\}$:

	A	B	C	D	L
1	2	5	3	4	*0*
2	**7**	7	**7**	5	*1*
3	**6**	**8**	5	**8**	*0*
4	**6**	6	1	3	*0*
L	*6*	*8*	*6*	*8*	***L***



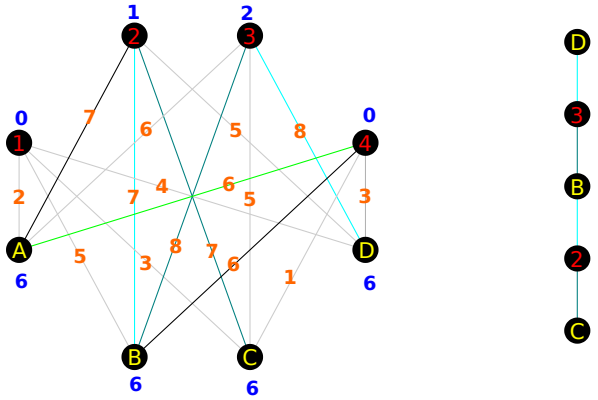
3. $N(S) == T$? Yes. Update labels with $m = 2$, so $N(S) = \{2, 3, 4\}$:

	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	**7**	**7**	5	*1*
3	6	**8**	5	**8**	*2*
4	**6**	**6**	1	3	*0*
L	*6*	*6*	*6*	*6*	***L***



4. Pick up vertex 2, which is not free, setting set $S = \{D, B, C\}$, $T = \{3, 2\}$, $N(S) = \{2, 3, 4\}$:

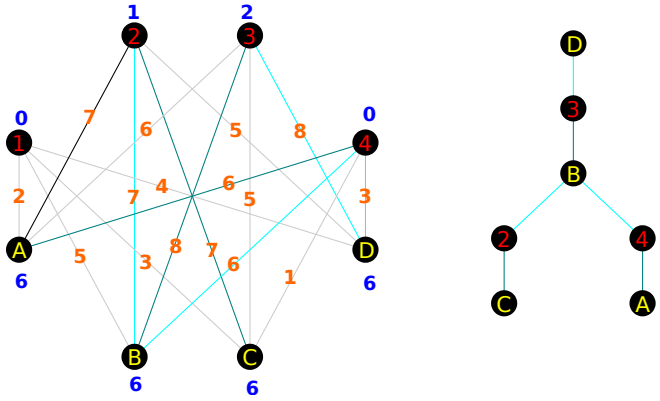
	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	**7**	**7**	5	*1*
3	6	**8**	5	**8**	*2*
4	**6**	**6**	1	3	*0*
L	*6*	*6*	*6*	*6*	***L***



3. $N(S) == T$? No.

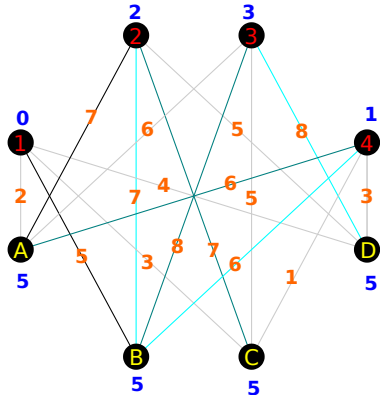
4. Pick up vertex 4, which is not free, setting set $S = \{D, B, C, A\}$, $T = \{3, 2, 4\}$, $N(S) = \{2, 3, 4\}$:

	A	B	C	D	*L*
1	2	5	3	4	*0*
2	**7**	**7**	**7**	5	*1*
3	6	**8**	5	**8**	*2*
4	**6**	**6**	1	3	*0*
L	*6*	*6*	*6*	*6*	***L***



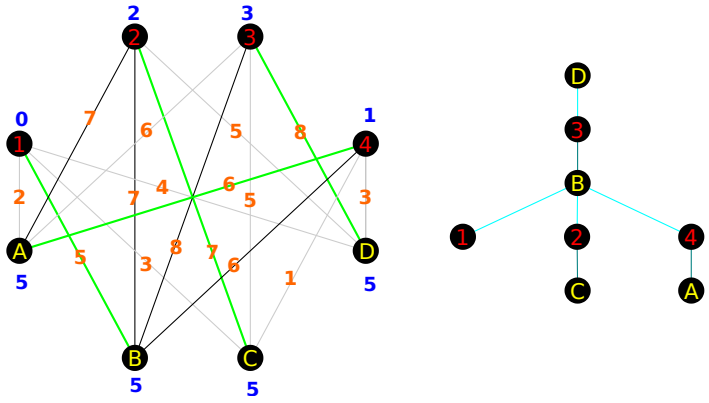
3. $N(S) == T$? Yes. Update labels with $m = 1$, so $N(S) = \{1, 2, 3, 4\}$:

	A	B	C	D	L
1	2	**5**	3	4	*0*
2	**7**	**7**	**7**	5	*2*
3	6	**8**	5	**8**	*3*
4	**6**	**6**	1	3	*1*
L	*5*	*5*	*5*	*5*	L



4. Pick up free vertex 1, thus $D-1$ is an augmenting path, being $M = \{(1, B), (2, C), (3, D), (4, A)\}$:

	A	B	C	D	L
1	2	**5**	3	4	*0*
2	**7**	**7**	**7**	5	*2*
3	6	**8**	5	**8**	*3*
4	**6**	**6**	1	3	*1*
L	*5*	*5*	*5*	*5*	L



2. M is a perfect matching, so we are done.

3 State of the art of Statistical Disclosure Control

As we introduced before, *Statistical Disclosure Control* (SDC) is the discipline concerned with the protection and anonymization of statistical data that has to be published. This discipline has developed methods to protect confidential data and protocols to follow when it is released. A SDC protocol can be found on U.S. Census Bureau [27].

In this chapter, we will discuss the background needed in this project, visiting some concepts in SDC scenario which will help the reader to contextualize it.

3.1 Protection Methods

A dataset consists of some rows, called records, and columns, called attributes. Each record contains the values of the attributes of an individual. Attributes can be classified in three categories:

- **Identifiers.** Attributes whose value uniquely identify an individual, like a Social Security number.
- **Quasi-identifiers.** Attributes which, combined with others, identify an individual, like name, age and address.
- **Confidential.** Attributes which contain sensitive information of an individual, like worship.

Assuming we want to release confidential data for statistics purposes, e.g. distribution of worships in a country, we must drop identifiers and, instead of dropping quasi-identifiers, they are protected, because dropping them will reduce the statistical utility of the data drastically; confidential data, which is what we want to release, is left intact. After that, we evaluate disclosure risk, and then, if it is lower enough, release it. Figure 3.1 illustrate the process.

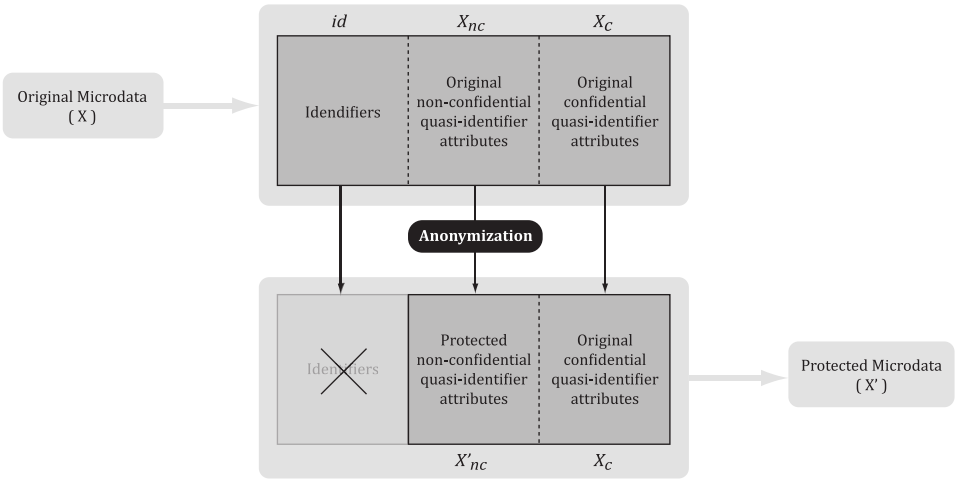


Figure 3.1: Protection process: drop identifiers and protect quasi-identifiers. After disclosure risk evaluation, release protected data.

Protection methods can be classified, in three disjoint categories, depending on how it affects the data:

- **Perturbative.** Data is distorted, introducing noise and making harder the re-identification process. This includes distortions like modifying values, swapping them between records or aggregating them, among others. However, because of that distortion, these methods must ensure that statistical information is preserved.

- **Non-preturbative.** These methods do not distort data, still they do partial suppressions or generalizations, converting combinations of values which identify individuals into more general ones.

- **Synthetic Data Generators.** These methods build new datasets from original data. Such synthesized datasets are generated constrained by the computed model.

In order to offer a deeper understanding of protection methods, three of them are detailed at following sections.

3.1.1 Noise Addition

Noise addition protection method [16] is a simple method which adds gaussian noise based on the attribute standard deviation vector and σ provided. This method is the one we have used in this project.

Noise addition proceeds computing the standard deviation for each attribute; then, a random gaussian value is multiplied by the attribute standard deviation and the σ provided, divided by 100, for each original record, adding the result to it; this is $x' = x + \frac{random \cdot std \cdot \sigma}{100}$. Table 3.1 shows an example of applying noise method.

a_1	a_2	a_3	a'_1	a'_2	a_3
8	9	1	8.575	7.681	1
6	7	10	6.402	8.266	10
10	3	4	10.640	4.217	4
7	1	2	7.235	1.753	2

Table 3.1: Example of Noise Addition method with $\sigma = 25$

We should note that, in case standard deviation is quite low, i.e. less than 0.0001, it is set as 1, in order to ensure that some noise is added.

3.1.2 Rank Swapping

Rank swapping [23] was initially proposed as a protection method addressed to categorical attributes, but it is also considered as a good numerical protection method [25].

This method works sorting values of the attribute to be protected, and then, given a number p, every record at position i, within this attribute, is swapped with record at l chosen randomly from position range $i < l \leq i + p$. After that, the sorting is undone, returning the swapped values to their original position. Table 3.2 shows an example of rank swapping.

a_1	a_2	a_3	a'_1	a'_2	a_3
8	9	1	6	7	1
6	7	10	10	9	10
10	3	4	7	1	4
7	1	2	8	3	2

Table 3.2: Example of Rank Swapping with $p = 2$

3.1.3 Microaggregation

The main idea behind microaggregation is to build small clusters (see Section 2.1) of at least k records and substitute their values by the cluster centroid. We show an example

a_1	a_2	a_3	a'_1	a'_2	a_3
8	9	1	7	8	1
6	7	10	7	8	10
10	3	4	8.5	2	4
7	1	2	8.5	2	2

Table 3.3: Example of basic Microaggregation with $k = 2$

of basic microaggregation in Table 3.3. On it, we can notice that, after apply microaggregation, the probability of identifying the value of the third attribute for the known quasi-identifier $(10, 3)$ is $\frac{1}{k} = 0.5$; this is called *k-anonymity* [6].

However, when the number of attributes is large, the information loss increases. This is due to that distance between original records and centroids is larger.

To solve this drawback, attributes are split in blocks and microaggregation is applied to them separately. Unfortunately, this not preserves the k-anonymity, because records which are assigned to the same cluster in a block, may not be assigned to the same in the other blocks, increasing the disclosure risk (see Section 3.3). When the size of blocks is one attribute, this method is known as *Univariate Microaggregation*; an example is shown in Table 3.4.

a_1	a_2	a_3	a'_1	a'_2	a_3
8	9	1	9	8	1
6	7	10	6.5	8	10
10	3	4	9	2	4
7	1	2	6.5	2	2

Table 3.4: Example of Univariate Microaggregation with $k = 2$

In any case, in order to maintain the statistical utility, the sum of square distances between centroids and original records must be minimum. Nonetheless, while optimal univariate microaggregation has polynomial approaches, optimal multivariate microaggregation is a NP-hard problem; because of this, several heuristic microaggregation methods has been proposed [25].

3.2 Record Linkage

Record Linkage methods were initially created for data cleaning and integration, hence, it can be used to link protected quasi-identifiers with the original ones obtained from another data source. This re-identification process is illustrated in Figure 3.2.

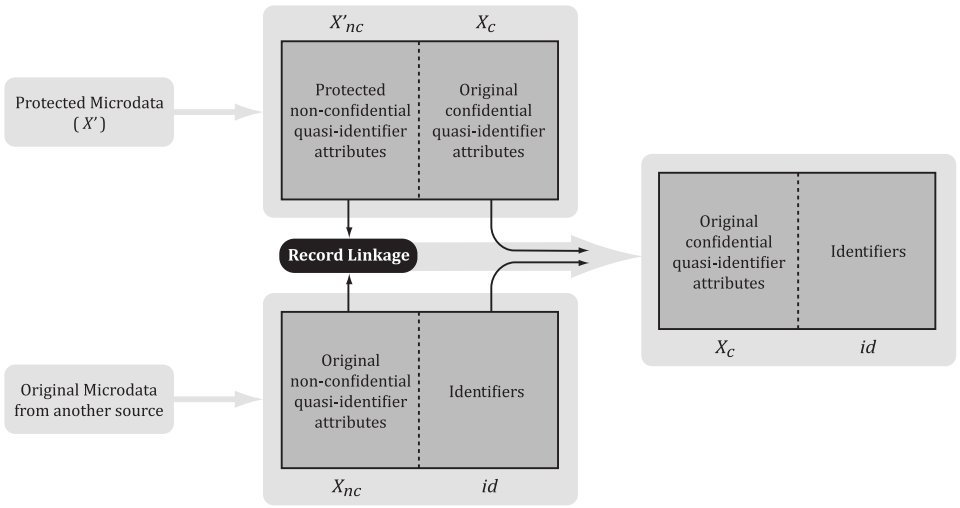


Figure 3.2: Re-identification process: compare common quasi-identifiers to link both datasets and to identify confidential data owners.

Record linkage methods can be divided into two different families: *Distance Based* and *Probabilistic* record linkage.

3.2.1 Distance Based Record Linkage

In this family, a certain distance between the original and protected records is computed; then, the pair of records whose distance is minimum is considered the correct link. An example of applying Distance Based Record Linkage (DBRL) is presented in Table 3.5.

	r'_1	r'_2	r'_3	r'_4
r_1	**0.200**	1.131	1.079	0.977
r_2	1.217	**0.178**	1.345	1.499
r_3	0.954	1.523	**0.195**	0.598
r_4	0.990	1.663	0.899	**0.105**

Table 3.5: Example of DBRL using dataset from Table 3.1 with the Euclidean distance

Distance selection is one of the most important aspect of this method. For instance, in the experiments carried out in this project, we have used the Euclidean distance because all the considered datasets have numerical attributes. For categorical or textual attributes other distances, as Edit [19] distance, can be used. This freedom in the distance selection offers us a great flexibility, because it is possible to change the distance function, combine more than a single distance or even weight attributes at our criterion.

3.2.2 Probabilistic Record Linkage

Probabilistic Record Linkage (PRL) works classifying a pair of records as Linked Pair (LP) or Non-linked Pair (NP) based on a score index.

For each pair (a, b), when a is a record of the original dataset and b of the protected dataset, we define coincidence vector $\gamma(a, b) = (\gamma_1(a, b), ..., \gamma_n(a, b))$, where $\gamma_i(a, b)$ is defined as 1 if $attr(a) == attr(b)$ or 0 otherwise. Score index is computed based on this coincidence vector [3, 25].

Then, using Expectation-Maximization (EM) algorithm, attributes are weighted and scores estimated. Finally, thresholds to classify as LP or NP are computed based on the given percentage of tolerated false positive and false negatives. Since EM algorithm has a high computational cost, in our experiments we will only use DBRL. This is not a very important issue because in [8] it was proven that DBRL outperforms PRL when the considered datasets are numerical.

3.3 Information Loss and Disclosure Risk

The main goal of any protection method is to minimize *disclosure risk* and *information loss*. Disclosure risk measures the capacity of a third part to obtain some information of the original dataset from the protected one, while information loss measures the reduction of the statistical utility respect to the original.

Unfortunately, both measures are complementaries; when one increases, the other decreases. Because of that, it is necessary to achieve a good trade off between each other in order to ensure enough protection level while dataset continue being statistically useful, optimizing such trade off. This can be measured by the score measure [7] as follows.

On the one hand, Information Loss (IL) is measured averaging the difference between several statistical measures of original and protected data. These are the mean absolute error between protected and original records (IL_1), the mean variation between the mean (IL_2) and the variance vectors (IL_3) of attributes and the mean difference of covariance (IL_4) and correlation matrices (IL_5). Hence, the final IL measure is computed as

$$IL = 100(0.2IL_1 + 0.2IL_2 + 0.2IL_3 + 0.2IL_4 + 0.2IL_5)$$

On the other hand, Disclosure Risk (DR) measures two different scenarios: i) the objective of the third part is to link protected data to some other one, identifying protected records, and ii) to get approximated values of the original data.

This project is focused in the first scenario, where the disclosure risk is computed using Distance-based Linkage Disclosure (DLD) and Probabilistic Linkage Disclosure (PLD). In both, record linkage is applied using DBRL and PRL, respectively, making the assumption that a third part or intruder knows one to half of the total attributes of the data for each computation, obtaining an averaged percentage of links as a final result. However, in this project we propose a worse scenario, making the assumption that such third part knows all the attributes of the data.

In the second scenario, *Interval Disclosure* (ID) risk is defined, which is the average percentage of the original values falling into a defined interval around the masked value.

Therefore, Dislcosure Risk measure is computed as

$$DR = (0.5\frac{DLD+PLD}{2} + 0.5ID)$$

Finally, we obtain a score, as

$$score = 0.5IL + 0.5DR$$

4 Hungarian Method for Disclosure Risk Assessment

As we introduced in Chapter 1, current record linkage methods underestimate the disclosure risk of released data; because of that, we have implemented Hungarian Method as described in Section 2.3.1 and we have used as *disclosure risk linkage measure*.

4.1 Hungarian Method as a New Privacy Measure

The current privacy definition is based on the assumption that the intruder knows some quasi-identifiers of some records extracted from and external dataset. Then, the goal of such intruder is to relate any record $x \in X$ with its corresponding record $x' \in X'$ in such a way x and x' belong to the same individual in order to infer from such link the corresponding sensitive information of that individual.

In this particular setting an adversary can rule out many links, since he is able to compute several distances among X and X' records. Specifically, since the attacker knows that the true linkage between the records in X and X' is a bijection, he may rule out links that are not part of any such bijection. However, this simple idea is not considered in the DBRL measure.

Definition 7 *Let $\langle X' \rangle$ be a release of the dataset X. The corresponding full-connected bipartite graph $G = G_{X,\langle X' \rangle}$ is a graph on the set of nodes $V := X \cup X'$, where the set of edges E consists of all pairs $(B-V, B-V')$ where $B-V \in X$, $B-V' \in C'$.*

A perfect matching in G is a set of N edges that cover all $2N$ nodes of G. Namely, it is a set of edges of the form $\{(B-V_n, B-V'_{\pi(n)}) : 1 \leq n \leq N\}$, where π is a permutation on $\{1, \ldots, N\}$. Clearly, G has at least one perfect matching, which is the one that corresponds to the identity permutation and describes the true linkage of all records — $\{(B-V_n, B-V'_n) : 1 \leq n \leq N\}$. Indeed, since G is a full-connected bipartite graph, all of those N pairs of nodes are edges in G (as implied by Definition 7), and that set of edges is indeed a perfect matching in G.

Definition 8 *An edge in G is called perfectly-matchable if there exists a perfect matching in G that includes it. The subgraph of G that consists of all perfectly-matchable edges in G is denoted G^{pm}.*

All true links, $(B{-}V_n, B{-}V'_n)$, $1 \leq n \leq N$, are clearly perfectly-matchable, since the collection of all those N edges is a perfect matching. The edges in the subgraph G^{pm} represent all possible links between records in X and X'. We note that this set of edges cannot be further reduced, since every edge in G^{pm} belongs to some perfect matching in G, and each such perfect matching describes a "possible world", namely, a possible linkage between the records of X and those of X'.

Given a bipartite graph $G = (V, E)$ that has at least one perfect matching Ω, then with the knowledge of Ω it is possible to find all perfectly-matchable edges in G in time that is linear in $|V|+|E|$ (see [26]). If no such perfect matching is known upfront, it is needed first to find one (a procedure that has runtime of $O(|V|^{1/2}|E|)$, see [13]) and then proceed to find all perfectly-matchable edges in additional linear time.

There are several attack strategies that the strong adversary may attempt to launch, given the reduced bipartite graph G^{pm}. Two of them are the following:

1. Link any record $B{-}V \in X$ with any of its neighboring records from X' in G^{pm} with equal probabilities.

2. Find in G^{pm} a minimal-cost perfect matching (where the weight of an edge $(B{-}V, B{-}V')$ is $d(B{-}V, B{-}V')$) and link any record $B{-}V \in X$ with its neighboring record in that perfect matching.

In the first strategy, the probability of correctly linking a record $B{-}V \in X$ is $1/\deg(B{-}V)$, where $\deg(B{-}V)$ is the degree of the node $B{-}V$ in G^{pm}. Indeed, each of the neighboring nodes to $B{-}V$ in G^{pm} represents a record $B{-}V' \in X'$ that could be its true image (and the true image is one of those nodes). Assuming that the adversary selects one of them uniformly at random, the probability of correctly linking a record $B{-}V \in X$ is $1/\deg(B{-}V)$. Hence, the average probability of a successful linkage is

$$P_G(X, \langle X' \rangle) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{\deg(B{-}V_n)} \,,$$

where $\deg(B{-}V_n)$ is the degree of $B{-}V_n$ in the bipartite graph G^{pm}.

We would like to note that a more sophisticated adversary would select a neighbor $B{-}V'$ of $B{-}V$ in probability that is proportional to the number of perfect matchings in G^{pm} that include the edge $(B{-}V, B{-}V')$. However, that attack is infeasible since counting the number of perfect matchings in a bipartite graph is equivalent to computing the

permanent of a $\{0,1\}$-matrix, a problem which is in #P-complete [28], and is hard not only in the worst case, but also in the average case [20].

In the second strategy, the adversary finds a perfect matching in G^{pm} of a minimal cost.

Definition 9 *The cost of a perfect matching* $\Omega = \{(B-V_n, B-V'_{\pi(n)}) : 1 \leq n \leq N\}$ *in* G^{pm} *is* $C(\Omega) := \sum_{n=1}^{N} d(B-V_n, B-V'_{\pi(n)})$.

Let Ω^o be a min-cost perfect matching in G^{pm}. Then if the adversary uses it to link records from X and X', its average success probability is the percentage of true links in Ω^o:

$$P_G^o(T, \langle T', \Delta \rangle) = \frac{|\Omega^o \cap \{(B-V_n, B-V'_n) : 1 \leq n \leq N\}|}{N}.$$

4.2 Implementation Details

In this section we describe the main implementation details of the hungarian algorithm we have used to evaluate the two different attacks described in the previous section.

The Hungarian hethod has been coded as a single class header, which implements the algorithm, to make easy its use in other experiments. Hungarian Method class is constructed with the matrix distance as a parameter; then, it is executed with the *execute()* call and the results are gathered with *get_results()* function. However, because distance is sent as a matrix, it requires $O(n^2)$ bytes to store it, depending on type used (char, integer, float, etc.) to represent distances; given our experimental setting (see Section 4.3.3), we cannot afford an execution over 110,000 records if we use chars, or 55,000 records if we use integers, that typically use 4 bytes, if we want to use just RAM. This issue will be deeply discussed in the following chapter.

Algorithm 1 Execution of Hungarian Method

```
M ← readMatrix()
H ← HungarianAlgorithm(M)
H.execute()
R ← H.get_results()
print results R
```

Algorithm 2 execution() method of HungarianAlgorithm

Require: HungarianAlgorithm initialized with M as squared data matrix
  if M has to be minimized **then**
    step0()
  end if
  step1()
  while not perfect matching **do**
    step2()
    while not augmenting path **do**
      if $N == T$ **then**
        step3()
      end if
      step4()
    end while
  end while
Ensure: R contains a perfect matching

4.3 Experiments

Now, we provide a complete description of the experimental environment and datasets we have used, as well as the experimental results about both disclosure risk measures described in Section 4.1.

4.3.1 Datasets

Used datasets were obtained from U.S. Census Bureau [25]; these datasets have 13 attributes and 1,080, 13,500 and 149,642 records. A summary of the datasets is shown in Table 4.1. Along this project, we will use the ID instead of the name of the dataset.

Name	ID	Attributes	Records
Census	C	13	1,080
Large Census	LC	13	13,500
Very Large Census	VLC	13	149,642

Table 4.1: Description of the datasets used.

4.3.2 Experimental Parameters

We have applied the *Noise Addition* protection method described in Section 3.1.1, with different σ values, concretely $\sigma = \{5,\ 15,\ 25\}$.

All distances in the following experiments are computed using the Euclidean distance. Attributes are normalized to $N(\mu, \sigma)$, with $\mu = 0$ and σ equal to the the maximum of both original and protected records for each attribute; this is, given $\forall x \in X$ as an original record and $\forall x' \in X'$ as protected one, every value v of each attribute i is normalized as

$$N(v_i) = \frac{v_i}{max_i(x_i, x'_i)}$$

4.3.3 Computation Environment

We were granted access to the Computer Architecture Computing Laboratory (LCAC)[1] at the UPC, where we have executed all the experiments. LCAC Computers have a CPU Intel Xeon 5148 at 2.33 GHz, with 12 GB of RAM. The operating system installed is Ubuntu 11.10, with kernel Linux 3.0.0 x86_64. Finally, the Hadoop framework installed is 0.20.203.0 version.

4.4 Results

In this section, we show the empirical disclosure risk values we have obtained, either using DBRL and Hungarian Method for three datasets.

4.4.1 Distance Based Record Linkage

In order to compare the results obtained with Hungarian Method with DBRL, first we have executed both on to the same datasets.

Despite explained at Section 4.3.2, in this case, and only in this case, distance is normalized according to the maximum protected record (NPR).

We would like to highlight in Table 4.2 that, as it was expected, when σ increases the number of correct link decreases. For instance, in the LC datasets with σ equal to 5

[1] `http://www.ac.upc.edu/serveis-tic/altas-prestaciones`

Dataset	Records	σ	NPR	Time (s)
C	1,080	5	1,080	2.81
		15	972	6.82
		25	680	10.20
LC	13,500	5	11,789	122.21
		15	4,455	124.97
		25	1,896	120.20
VLC	149,642	5	51,773	15,278.48
		15	18,305	15,593.45
		25	8,096	15,428.30

Table 4.2: Results of applying DBRL to datasets

DBRL obtains almost 11,800 correct links, while it only obtains near 1,900 with σ equal to 25.

4.4.2 Hungarian Method

Hungarian Method maximizes or minimizes a given score; in our case as we have introduced it in Section 4.1, we minimize the distance between records of X and X' in order to obtain a min-cost perfect matching in G^{pm}.

Dataset	Records	σ	Correct links	Time (s)
C	1,080	5	1,080	0.42
		15	1,061	0.42
		25	902	0.45
LC	13,500	5	12,435	73.70
		15	5,492	279.73
		25	2,667	241.43

Table 4.3: Results of applying Hungarian Method to datasets

As it is depicted in Table 4.3, we can observe an important execution time difference within the same dataset protected with different σ parameters; this effect it is mainly produced by the fact that it is more difficult to do a match when the noise added increases. In the case of doing a very good initial greedy assignment, as in the case of small σ cases, it avoids updating labels and constructing lots of alternating trees, reducing, in this way, the number of operations the algorithm must perform when σ increases.

We have not executed experiments with VLC because it has about 150,000 records, significantly over the number of records we can allocate in RAM, as already explained

in this chapter; moreover, being the proportion between VLC and LC about 11, time scales 11^3, hence if we take mean time of LC executions, execute VLC would take almost 3 days for each different σ.

4.4.3 Method Comparison

Dataset	Records	σ	HM-DBRL links	HM-DBRL links (%)	HM-DBRL time (s)	HM-DBRL time (%)
C	1,080	5	0	0.00	-2.39	-84.95
		15	89	+9.19	-6.40	-93.82
		25	222	+32.63	-9.74	-95.56
LC	13,500	5	647	+5.49	-48.51	-39.70
		15	1,037	+23.27	154.76	+123.83
		25	771	+40.66	119.23	+97.57

Table 4.4: Comparison between Hungarian Method (HM) and Distance Based Record Linkage (DBRL)

Comparing tables 4.2 and 4.3, the first we perceive is the difference in the execution time. In Census dataset, the difference is not very significant, taking account DBRL code is written in Java and it requires allocate first JVM, while Hungarian Method is written in C++. Nevertheless, in LC the difference becomes more significant, being clear that Hungarian Method generally requires more time to be executed.

It is noticeable that all Hungarian Method executions have better results compared to DBRL, using the same σ. A clear example is the case of LC with $\sigma = 25$, which increases the number of links in 771, a 40.66% better than DBRL.

Table 4.4 summarizes these two observations. It depicts the differences in the execution time and obtained links between DBRL and HM in an absolute and relative manner. If we observe in detail those results, we can say that Hungarian Method outperforms DBRL at the cost of having a larger execution time. For example, LC with $\sigma = 15$, which has an improvement of 23.57% in the number of links, implies increasing the execution time in a 123.83%. However, as the attacker can spent as much as time he has then, it is not a real problem for him.

5 Hungarian Method for Large Data

As we have already seen, Hungarian algorithm is highly time consuming ($O(n^3)$) compared to DBRL ($O(n^2)$). Therefore, a feasible strategy to face this problem and reduce its execution time is to split the dataset into smaller parts, i.e. to split the bipartite graph into sub-components not greater than an arbitrary size. As a result, splitting the data in components of size t, we get n/t sub-components, which takes $O(t^3)$ each execution; finally, in order to solve the complete matching problem in all partitions takes $O(nt^2)$ time. This approach will be deeper discussed in Sections 5.1 and 5.2.

Another way to address this problem is to parallelize the data access and processing, which represents the n^2 of $O(n^3)$. This is, indeed, a very similar approach to actually splitting the data. This strategy is discussed in Section 5.3.

5.1 Clustering Method for Data Partitioning

Clustering, as explained in Section 2.1, is a method that groups records within similar characteristics; this permits us to split and distribute the data in several groups. Our approach is to apply the Hungarian Method to all these groups which are similar separately to link pair of records belonging to the same individual.

Roughly speaking, the algorithm proceeds as follows: first, we apply a top-down clustering algorithm by means of creating a kd-tree [2] data structure with the original records, which produce partitions of close records not greater than a given size. This is possible splitting the space at a midpoint of tree structure. We call T to this set of partitions. After that, we fix the centroids of T as the initial clusters centroids for a k-means algorithm, then, we create a new set of partitions T' with the protected records obtaining the bipartite sub-components to apply the Hungarian algorithm. However, proportion between t and $t' \in T'$ may not be 1, i.e. not all the partitions t' will have the same size that its corresponding partition t. For this reason, we balance them moving records from $l' \in T'$ to $s' \in T'$, for l' and s' being the partitions of closer records to $l, s \in T$, $|l'| > |l|, |s'| < |s|$ and being s the closer partition after l to the record which is moved. Later, we apply Hungarian Method to balanced partitions t and t' of original and protected records, respectively. Finally, we merge results, setting as link between t and t'

the most probable, according to those results.

We have used the ANN library [22] for the kd-tree clustering. However, this library is designed to search nearest neighbours, not to provide partitions; because of that, we modified it to get the partitions generated internally.

In order to distribute the execution, we got every partition and executed it in a single node of the LCAC (see Section 4.3.3) at the same time.

Dataset	**Records**	**Block Size**	σ	**Links**
LC	13,500	4,500	5	14
			15	17
			25	17
VLC	149,642	15,000	5	24
			15	5
			25	3

Table 5.1: Results of applying Hungarian Method to clustered data

Despite this clustering strategy, results obtained were discouraging, with negligible number of correct links, as we can see in Table 5.1 with LC and $\sigma = 25$. Because of these poor results, we have been forced to change our approach, proposing a simpler but more effective method described in Section 5.2.

One possible cause of these bad results is the *curse of dimensionality* [4]. This issue arises in the presence of a metric space with a high dimensionality, which provokes that elements in such space seem sparse and different, augmenting the variance and making much more challenging the clustering task.

Another reason behind these results is that, in the first place, we compute a distance between the partition centroid and the records in order to connect them, adding some kind of extra noise; and second, extra records are moved to another partition which its centroid is farther than the previous one, computing a new distance with a new centroid. Because of this, we are distorting even more the possible correct links between records, worsening the obtained results.

5.2 Blocking Method for Data Partitioning

Blocking [12] is another method which can help us to split the data into smaller partitions. In blocking, data is sorted and then it is split in blocks. Our approach is to sort

the dataset by one of its attributes, minimizing the *curse of dimensionality* problem presented on the clustering method.

Constructing blocks taking care of just one data attribute introduces a great bias; in order to counteract this effect, we propose to construct different blocks for each of the attributes in the data. Then, final matching is done aggregating the obtained linkage results for each blocking configuration. However, to do that implies to multiply the execution time by the number of attributes; despite of this, as executions are independent, it is very easy to parallelize.

Votes emitted by different blocking configurations may produce ties. To solve this drawback we have tested three different voting systems:

- Basic Vote (BV). Ties are broken giving the preference vote to the minor index
- Best Score (BSV). Ties are broken considering the best score, i.e the minimum distance
- Weighted Vote (WV). Ties are broken as in Best Score, but vote is wheighted by the score

However, observing the results depicted in Table 5.2, we cannot observe a great difference among different voting criteria.

Dataset	**Records**	**Block Size**	σ	**Basic Vote**	**Best Score Vote**	**Weighted Vote**
LC	13,500	4,500	5	11,853	11,831	11,777
			15	5,687	5,683	5,674
			25	2,742	2,739	2,739
VLC	149,642	15,000	5	42,020	42,341	41,705
			15	21,692	21,800	21,731
			25	10,463	10,622	9,688

Table 5.2: Results of applying Hungarian Method to blocking data

We also observe that in case of $\sigma = 5$, results are close to those produced by the DBRL approach; moreover, VLC are worse. Nevertheless, we get better results with greater σ, as perceived in Table 5.3, getting about $18.50\% - 31.69\%$ more correct links with $\sigma \geq 15$ in VLC.

We should also note that, according to time spent as described in Table 5.4, this method has a large time variance. Some configurations are executed very fast whilst some of them need much more time to finish.

Dataset	σ	BV	BV (%)	BSV	BSV (%)	WV	WV (%)
LC	5	64	+0.54	42	+0.36	-12	-0.10
	15	1,232	+27.65	1,228	+27.56	1,219	+27.36
	25	846	+44.62	843	+44.46	843	+44.46
VLC	5	-9,753	-18.84	-9,432	-18.22	-10,068	-19.45
	15	3,387	+18.50	3,495	+19.09	3,426	+18.72
	25	2,367	+29.24	2,566	+31.69	1,592	+19.66

Table 5.3: Comparison between Blocking Method and DBRL

Dataset	Records	Block Size	Blocks	σ	Min. time	Max. time	Mean time
LC	13,500	4,500	3	5	6.14	194.17	37.50
				15	17.46	220.99	61.05
				25	25.89	247.08	75.23
VLC	149,642	15,000	10	5	135.72	59,631.26	9,333.19
				15	218.74	46,713.97	7,552.97
				25	242.30	53,379.71	8,121.76

Table 5.4: Time, in seconds, spent applying Hungarian Method to one block of data. Note that the number of blocks shown is the amount of blocks produced by each attribute.

Hence, according to this results, it is possible to conclude that DBRL works pretty well compared to this method with low σ, while blocking combined with Hungarian works better when σ increases. Although the execution time penalty, the combination of the Hungarian Method with blocking arises as a very interesting option for intruders with access to any type of cloud computing environment. A possible option to reduce blocking execution time is to select those attributes that provide more information using some feature selection method, e.g. *Principal Component Analysis*.

5.3 Hadoop Hungarian Method

In previous sections, we discussed about possible methods to split data into smaller parts in order to reduce the time and space complexity. Nonetheless, this approach has the lack that data splitting sometimes introduce bias in the data, reducing in this way the Hungarian Algorithm performace.

In this section we follow a different approach. Instead of splitting the data into smaller partitions to process them in a moderated time inside a distributed environment, we split data access and processing operations, executing them in a distributed computing environment, and solving the global linkage problem on the entire data. To do that, we have

implemented the Hungarian Method as a Hadoop Java algorithm.

Hadoop works executing at least one map task and an optional reduce task each time; this makes not possible to chain several map tasks, which would perfectly fit into the Hungarian Method because each data matrix modification must consider all the data to compute the new updated matrix cost value. Because of that, we have implemented a map-reduce task for all operations related to data processing in the Hungarian algorithm. The operations have been coded as mappers to access the matrix. They are the following:

- **getMax mapper:** This mapper maps every value to the key *max*, which then is reduced in order to get the max value of the whole matrix. Such maximum value is passed to the *mapMaxToMin*.

- **getSlack mapper:** It maps nodes that have to be tracked in the slack vector (see Section 2.3.1). It also maps, with a different key, the nodes which are in the equality graph to keep track of potential neighbours in order to avoid to launch a job each time it is required to generate the new neighbours.

- **initializeLabeling mapper:** This function maps every value of the matrix to a key based on its row and column position; then, these keys are reduced by the maximum, obtaining the initial labeling.

- **mapCandidates:** Maps every value (node) of the matrix belonging to the equality graph, i.e. all candidates to be a record assignment. This mapper is used by the initial matching operation.

- **mapMaxToMin:** This is a chain mapper which is used to transform the maximizing task of HM into a minimizing one.

Therefore, the actual Hadoop Hungarian algorithm is executed on a node, which sends every operation as a different job that is enqueued in Hadoop; meanwhile the main node waits for the Hadoop results and continues the algorithm.

Dataset	**Records**	σ	**Links**	**Time (s)**
C	1,080	5	1,079	41,100.84
		15	1,045	52,155.69
		25	820	380,915.49

Table 5.5: Results of applying the Hadoop algorithm to datasets

Unfortunately, the Hadoop approach is slower than the blocking distributed version of the Hungarian Method as it is illustrated in Table 5.5. Hadoop implementation, executed in the C dataset, has a time cost comparable to processing the most time

costly blocks of 15,000 records of VLC using the blocking technique in the case of $\sigma = \{5, 15\}$; and it is even worse with $\sigma = 25$. Due to this, we decided do not execute more experiments.

If we pay attention to Hadoop executions, the main reason behind those bad results is the following one: if we observe every executed job, i.e. every data access operation, it has an execution overhead around 30 seconds queueing the job, allocating a JVM inside the node, transferring the job and data to it and starting the job. All this process is detailed in Section 2.2.1. Arguably, those 30 seconds are worthy if the number of operations performed in the node is larger enough, but this is not our case, because each operation must be executed isolated from the other having access to the complete cost matrix, therefore, we can conclude that Hadoop is not the most suitable framework for this kind of algorithms.

6 Project Analysis

In this chapter we detail the planning and cost analysis of the project. Section 6.1 is dedicated to the planning while Section 6.2 analyzes the cost of the project and how much it would cost if it were done in a private company.

6.1 Planning

As any engineering project, it is required to have a planning. However, our initial planning it has not been fulfilled; because of that, we provide both the initial (IP) and the mended planning (MP) in Table 6.1, expressed in days. The obtaining of the experimental results is not considered because it is distributed along all the project, but mainly in the report elaboration phase.

Task	**IP**	**MP**
Documentation and implementation of Hungarian Method	30	39
Documentation and implementation of Hungarian Method in Hadoop	30	16
Clustering implementation	14	9
Report elaboration	14	29
Total	88	93

Table 6.1: Planning of the project

As observed, we have underestimated the documentation and implementation of the Hungarian Method in C++ and the report elaboration, while Hadoop and clustering implementation were overestimated. The reasons because we have needed more time in the tasks that were underestimated are the following:

In the case of the Hungarian Method, we found several versions of the algorithm; this brought us to try to implement a version that seemed complete and time efficient, but it was not. After we realized that it was a dead end, we opted for the version which we have finally used.

On the other hand, report elaboration has required more time due to the unknowing of LaTeX, the waiting for some experimental results and the continued improvement of such report.

6.2 Cost Analysis

In order to analyze the cost, first, we should get the time spent on it, which is shown in Table 6.1; we should note that we have dedicated about 6 hours per day. However, we must add the hours spent by our director as project manager, too. Then, according to FIB recommendations[1], our cost is 7.5 €/h, while cost of our director is 30 €/h. Analysing all these costs, Table 6.2 shows the cost of this project, which signifies 4,935 €.

Worker	**Cost/h**	**Hours**	**Cost**
Student	7.5	558	4,185
Director	30	25	750
Total	8.46	583	4,935

Table 6.2: Cost of the project

If the project was done in a company, we would required an analyst, a programmer and a project manager, whose cost would be 35, 20 and 45 €/h, respectively. Hence, according to our estimation, as shown in Table 6.3, project would cost 16,095 €.

Worker	**Cost/h**	**Hours**	**Cost**
Programmer	20	354	7,080
Analyst	35	129	4,515
Project Manager	45	100	4,500
Total	27.61	583	16,095

Table 6.3: Cost of the project in a company

In any case, we have not considered the cost of the software, basically because we have used open source software, without cost. However, we are neither considering the cost of the hardware used, which includes the workstation used by the student and the data center provided by LCAC.

[1] `http://www.fib.upc.edu/en/empresa/practiques/empresa.html`

7 Conclusions and Future Work

Along this final degree project, we have presented our work applying Hungarian Method to the Record Linkage scenario. This includes the implementation of such method in C++ and the development of different types of data splitting and distribution; in particular, we have implemented a clustering and a blocking method and a Hadoop version of the Hungarian Method. We have also showed our experimental results applying them. Now, we expose our conclusions and we point out some future work.

7.1 Conclusions

The main conclusion we extracted from this project is that Hungarian Method is a potential improvement for the disclosure risk assessment. Table 4.4 and 5.3 show that Hungarian Method is always better than Distance Based Record Linkage with $\sigma \geq 15$, with an improvement in number of correct links up to 44%, and so is it with $\sigma = 5$, with the only exception of VLC.

Despite the cubic complexity of Hungarian algorithm, we saw that it is possible to process large quantities of data applying some clustering methods, as exposed on Section 5.2. However, although we already know that splitting data introduces a bias, we saw that if we do not take account the high dimensionality of the records and we introduce more noise, this has a crucial impact on the results, as seen in Section 5.1.

On the other hand, still it is a largely used and promisingly technology, parallelizing the Hungarian Method with Hadoop was proved as unsatisfactory, being a model that does not fit in this kind of algorithms. This is mainly caused by the restriction which only allows one reduction on the job process and the overhead that signifies to send a job to Hadoop, which needs to set up a JVM on nodes involved on the task execution.

7.2 Personal Conclusions

Making this project I have learned many new technologies and algorithms, and I practiced in real scenarios many techniques and methods learned in this degree.

The skills I have learned include:

- How the Hungarian Algorithm works, as well as Statistical Disclosure Control does and the protection methods it proposes.
- How it works and how to use the Hadoop framework, which, nowadays, is having increasingly focus because of the distributed computing and the *big data* model.
- How to start from zero implementing an algorithm with several specifications, many of them different and sometimes contradictories, and how to deal with them.
- How to work within a data center.
- How to write this report with LaTeX.

I have put in practice some knowledge received from subjects coursed in the Informatics Engineering degree:

- Programming in C++ and Java, learnt at subjects like P1, PRAP, PRED, PROP, etc.
- Working only with a command line and using queues, at LCAC, like I have learnt at SO and ASO.
- Taking into account complexity, as I learned at ADA, and realize how important it is.
- Applying clustering methods, as learned at MD and A.
- Use a distributed system, like Hadoop, and take advantage from it, as I learned at SODX.

I also proved myself that I can do something like this project.

7.3 Future Work

Despite the results presented on this project, we still have some open problems to look into them in order to advance in this research.

We have seen that on equal conditions, Hungarian Method is generally better than Distance Based Record Linkage on linkage results. However, we focused on a dataset, protected with just one method and counting on with all attributes in the protected data. This suggests these experiments need to be repeated with other datasets and intruder knowledge.

At the clustering scope, other methods can be tried, reducing the noise and the bias introduced by the process of splitting and grouping records, avoiding compute distance based on centroids. We can also avoid the *curse of dimensionality* and the waste of time with techniques like Blocking if we reduce the number of attributes used to compute such distance using feature selection methods.

Finally, parallelization process can be reformulated, applying some tricks to Hadoop in order to able us to execute all data process in a single job or programming an ad-hoc parallelization scheme, eluding as much as possible the overhead time for every data processed.

Bibliography

[1] Agrawal, Rakesh; Srikant, Ramakrishnan. Privacy-preserving data mining. *ACM SIGMOD Record*, 2000, Vol. 29, No. 2, p. 439-450.

[2] Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of The ACM*, 1975 Vol. 18, No. 9, p. 509-517. DOI 10.1145/361002.361007.

[3] Blakely, Tony; Salmond, Clare. Probabilistic record linkage and a method to calculate the positive predictive value. *International Journal of Epidemiology* [online], 2002, Vol. 31, No. 6, p. 1246-1252. [Consulted: April 19th 2013] Available at: <`http://ije.oxfordjournals.org/content/31/6/1246.full`>. DOI 10.1093/ije/31.6.1246.

[4] Chávez, Edgar; Navarro, Gonzalo; Baeza-Yates, Ricardo; Marroquín, José Luis. Searching in metric spaces. *ACM Computing Surveys*, 2001, Vol. 33, No. 3, p. 273-321.

[5] Dawes, Mike. *The Optimal Assignment Problem* [online PDF]. Ontario, Canada: University of Western Ontario, Department of Mathemathics, Discrete Optimization Course, January 2005. [Consulted: June 6th 2012]. Available at: <`http://www.math.uwo.ca/~mdawes/courses/344/kuhn-munkres.pdf`>.

[6] Defays, D.; Nanopoulos, P. Panels of enterprises and confidentiality: The small aggregates method. *Proceedings of 92th Symposium on Design and Analysis of Longitudinal Surveys*. Ottawa, CA: Statistics Canada, 1993, p. 195-204.

[7] Domingo-Ferrer, Josep; Torra, Vicenç. Disclosure control methods and information loss for microdata *Confidentiality, Disclosure, and Data Access: Theory and Practical Applications for Statistical Agencies*, 2001, p. 91-110.

[8] Domingo-Ferrer, Josep; Torra, Vicenç. Validating Distance-Based Record Linkage with Probabilistic Record Linkage. At: Catalonian Conference on AI. *Proceedings of the 5th Catalonian Conference on AI: Topics in Artificial Intelligence, Castellón, Spain, October 24-25, 2002.* London, UK: Springer-Verlag, 2002. p. 207-215.

[9] Edmonds, Jack; Karp, Richard M. Theoretical improvements in algorithmic efficiency

for network flow problems. *Journal of the Association Computer Machinery*, 1972, Vol. 19, No. 2, p. 248-264.

[10] Ghemawat, Sanjay; Gobioff, Howard; Leung, Shun-Tak. The Google file system. At: ACM Symposium on Operating Systems Principles. *Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA, October 19-22, 2003.* New York, NY, USA: ACM, 2003. p. 29-43.

[11] Golin, Mordecaj J. *Bipartite Matching and the Hungarian Method* [online PDF]. Hong Kong, China: University of Science & Technology, Department of Computer Science & Engineering, Introduction to Combinatorial Optimization Course, December 2004. [Consulted: June 5th 2012]. Available at: <`http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf`>.

[12] Hernández, Mauricio A.; Stolfo, Salvatore J. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 1998, Vol. 2, No. 1, p. 9-37.

[13] Hopcroft, J.E.; Karp, R.M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 1973, Vol. 2, p. 225-231.

[14] Hundepool, Anco; Domingo-Ferrer, Josep; Franconi, Luisa; Giessing, Sarah; Schulte Nordholt, Eric; Spicer, Keith; De Wolf, Peter-Paul. *Statistical Disclosure Control.* 1st Ed. Wiley, 2012. ISBN: 978-1-1183-4821-5.

[15] Jardine, N.; Sibson, R. The Construction of Hierarchic and Non-Hierarchic Classifications. *The Computer Journal*, 1968, Vol. 11, No. 2, p. 177-184.

[16] Kim, J.J. A method for limiting disclosure in microdata based on random noise and transformation. *Proceedings of the ASA Section on Survey Research Methodology, 1986.* Alexandria, VA, USA: American Statistical Association, 1986, p. 303-308.

[17] Kuhn, Harold W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955, Vol. 2, No. 1-2, p. 83-97.

[18] Lane, Julia; Heus, Pascal; Mulcahy, Tim. Data Access in a Cyber World: Making Use of Cyberinfrastructure. *Transactions on Data Privacy*, 2008, Vol. 1, No. 1, p. 2-16.

[19] Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, Vol. 10, p. 707-710.

[20] Lipton, R. New directions in testing. *Distributed Computing and Cryptography, Vol.*

2 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1991, p. 191-202.

[21] Lloyd, S. Least squares quantization in PCM. *IEEE Trans. on Information Theory*, 1982, Vol. 28, p. 129-137.

[22] Mount, David M.; Arya, Sunil. ann_1.1.2.tar.gz. At: *ANN - Approximate Nearest Neighbor Library* [online compressed file]. College Park, MD, USA: University of Maryland, January 27th 2010. [Consulted: February 25th 2013]. Available at: <http://www.cs.umd.edu/~mount/ANN/>.

[23] Moore, Richard A. *Controlled Data Swapping Techniques for Masking Public Use Microdata Sets*. U. S. Bureau of the Census, 1996.

[24] Munkres, James. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 1957, Vol. 5, No. 1, p. 32-38.

[25] Nin Guerrero, Jordi. Contributions to Record Linkage for Disclosure Risk Assessment. Ph.D. thesis, UAB, Departament de Ciències de la Computació. 2008. [Biblioteca de Comunicació i Hemeroteca General, UAB, Bellaterra].

[26] Tassa, Tamir. Finding all maximally-matchable edges in a bipartite graph. *Theoretical Computer Science*, 2012, Vol. 423, p. 50-58.

[27] U.S. Census Bureau. FAQ. At: *Statistical Disclosure Control* [online PDF] Suitland, MD, USA: U.S. Census Bureau, June 21th 2001. [Consulted: April 16th 2013] Available at: <http://www.census.gov/srd/sdc/>

[28] Valiant, L.G.. The complexity of computing the permanent. *Theoretical Computer Science*, 1979, Vol. 8, p. 189-201.

[29] White, Tom. *Hadoop: The Definitive Guide*. 2nd Ed. Sebastopol, CA, USA: O'Reilly, 2011. ISBN: 978-1-449-38973-4.