# Contents

# List of Figures

# List of Abbreviations

API: Application Programming Interface.

IEEE: Institute of Electrical and Electronics Engineers.

JS: Jensen and Shannon.

RDF: Resource Description Framework.

OWL: Web Ontology Language.

TFIDF: Term Frequency / Inverse Document Frequency.

SFS: Simplified Fellegi and Sunter.

URI: Uniform Resource Identifier.

# 1 Introduction

When communication between two parties is established, it has to be granted that both parties understand the information which is being exchanged in the same way. That is one of the principals of communication, and without that guarantee, a proper understanding can not be provided.

The same understanding has to be granted when two distributed ontologies are communicating. Both parties have to make sure that the concepts they are sharing are understood in the proper way.

Problems in communication can especially appear when both ontologies deal with overlapping domains of concepts. In those cases they have to relate the concepts in one of the domains to the concepts in the other one. For that purpose we use ontology matching.

This thesis aims to present and describe a new approach for matching ontologies, as well as compare it with some existing methods.

The new method proposed represents the common aspects the compared ontologies by means of polygons in order to obtain the final parameter determining the relationship between them.

## 1.1 Background

Ontology matching has become a key concept in the field of ontology research. Researchers all around the world have concluded that without a proper integration between ontologies, the knowledge will never be interpreted correctly, and therefore, the information will not be accurate.

Moreover, many solutions have been proposed to find correspondence between ontologies and many different applications are using those ideas, such as the AI community, the Semantic Web, Warehouses, Ontology integration, etc.

## 1.2 Purpose/Objectives

Despite the extensive research conducted in this field, there are still problems unsolved when dealing with ontology matching. With the aim of solving those problems and improving efficiency, a method using polygons to determine similarity between ontologies is proposed.

The algorithm for this polygon method has been developed by the Ph.D. Student in Information technology at the University of Jönköping Feiyu Lin (Feiyu Lin and Kurt Sandkuhl, 2007). The programming implementation for that algorithm and the research performed for the methods belonging to the class SecondString are the contributions of the author of this thesis.

In this thesis, the method based in polygons is described. The polygons are used to represent the main characteristics of the ontologies being compared, and by comparing the polygons, the ontologies will be compared as a result.

The purpose of the idea is giving an alternative way of matching ontologies, and solving the possible problems that the current methods have nowadays.

A new system has to be implemented in order to provide the functionality to run the new application. For this purpose a programming language has to be used as well as different programs and plug-ins.

Java is chosen as the programming language for this project due to the reason that it is the most common programming language in this field, providing a simple integration of the code into other applications.

The inputs we are going to deal with are the ontologies. Those ontologies will be implemented in OWL language. The main features of OWL will be described later in this document.

The polygons have to be represented in some mathematical way. Therefore, Matlab has been chosen as the standard program used for the mathematical requirements, given its integrating qualities.

## 1.3 Limitations

When we face the implementation of a system, some delimitation has to be made to clarify in which domain the project is going to be valid. Some delimitation for this project is provided in this section.

The topic of this thesis will be restricted to ontologies; no other data structure will be taken into account. The results will neither be extensible to other data integration techniques out of ontology matching.

If the input ontologies are written in other languages different from OWL, a translation should be made in order to fit the specifications of the system.

Regarding the perspective, some restrictions have to be taken into account, especially when dealing with such a large subject, where many approaches can be chosen.

The core part of the report aims at describing the implementation of the system. The sections correspond to the implementation steps followed while developing the application. That makes that the purpose of the thesis is to clarify the research tasks and justify the decisions made throughout the design and programming steps. Much technical information is also included. We can conclude that the scientific approach is the main one followed and the objective is to present the scientific community the new research done in this area.

On the other hand, the proposed system is going to be accessible for users willing to find the correspondence between two ontologies. That situation also has to be regarded in this thesis, implying the appearance of some guidance to the users as the implementation of the system is being described. The steps followed and the methodology will be constantly aimed at facilitating the role of the user when using the application.

Moreover, another important objective of this thesis is to simplify any possible further work carried out in this field and the tasks of other programmers. For that purpose, all the implementation features are explained in detail.

## 1.4 Thesis outline

In the introduction; the background, purpose and objectives, limitations and perspective of the project are explained. This information gives the reader a general idea about what one is going to read in the following pages, why this information is relevant, and how it is going to be presented.

In the theoretical background section, the theoretic information is provided.

The core part arrives at section 3, where the implementation of the system is described. The sub-sections correspond to the tools, languages and other implementation details used, which gives an idea to the reader about the steps that were followed during the implementation of the system.

Section 4 describes the results of the project, while section 5 introduces other ontology matching methodologies and compares the results obtained with the new method and the ones corresponding to other methods in the same area of study. It also expresses some ideas for further work.

The references and appendixes can be found at the end of the thesis.

# 2 Theoretical Background

## 2.1 Ontologies

This thesis describes a new methodology for automatic ontology matching. Before arriving to that concept, it can be useful for the reader to have a general idea about what an ontology is, its main features, and the purpose of ontology designing. This section of the thesis covers all those theoretical aspects.

When defining an ontology, we have to make a reference to Gruber, (Gruber, 1993) who describes an ontology as "*Specification of a conceptualization*".

As Gruber also defined (Gruber, 1992) ontology is a "*Explicit formal specifications of the terms in the domain and relations among them*".

According to Gruber (Gruber, 1992) an ontology can be described as well as "*Description of the concepts and relationships that can exist for an agent or a community of agents*".

According to the definition given by Natalya F. Noy and Deborah L. McGuinness (Noy and McGuinness, 2001) an ontology is a "*Formal explicit description of concepts in a domain of discourse*".

The definitions are many, and sometimes they contradict each other. To have a closer idea about what an ontology is, we are going to describe its functionality, its design and its components.

The sharing and reuse of knowledge among software entities is the main target of the ontology development. Ontologies give the support to implement this knowledge reuse and sharing.

Before the appearance of ontologies, other systems were in charge of the communication between knowledge producers and consumers: knowledge-based systems.

Knowledge-based systems are computer systems programmed to deal with databases in charge of knowledge management. By means of methods and techniques of artificial intelligence, the collection, organization, and retrieval of knowledge are performed.

When dealing with knowledge-based systems, problems are found due to the heterogeneity of the platforms, languages and protocols.

Ontologies are considered to be the solution for the heterogeneity problems as the same time as they provide with means to have knowledge libraries available from the networks.

In order to support knowledge sharing, the design of ontologies has to follow some standards and steps of implementation.

Every time we try to represent some knowledge from the real world into any kind of model, concepts have to be abstracted and simplified in order to represent them. That process is called conceptualization and is followed during the design of Ontologies.

By means of conceptualization, we get an ontological commitment. We say that an ontology deals with a specific domain which contains concepts. This domain is also called the ontological commitment, and in the other way round we can say that one concept commits to an ontology when it can be identified to a part of the existing ontology.

Consequently, we have taken concepts from reality and organized them into ontologies which can communicate between each other throughout the networks sharing and reusing all the knowledge they have stored.

Some of the components described by an ontology are: classes (representing the concepts in the domain), properties of each concept (describing various features and attributes of the concept), and individuals (belonging to one of the classes).

Classes are usually considered to be the most important part of the ontologies as they describe the concepts. Subclasses and super classes can be included, to explain more or less specific concepts and form a hierarchy.

Properties deal with the characteristics of the classes and the relationships between them, while individuals represent each of the individual examples belonging to each of the classes.

According to these components of the ontology, developing an ontology has to cover each of these parts: Definition of the classes in the ontology, setting the class hierarchy, definition of the properties and relationships between classes and finding instances for the classes.

## 2.2 Matching Ontologies

Once ontologies have been defined, the concept of ontology matching will be handled.

More and more each day, ontologies are proposed as the ideal way to deal with information and knowledge. That causes that many communities and organizations decide to edit and create their own ontologies to save and store their information. But as we have already said, ontologies are not structures for storage. They aim at sharing and reusing knowledge.

That fact leads us to the next conclusion: ontologies created by distributed parties have to be put in common in order to satisfy their main objective: the sharing and reuse of information.

If two distributed parties are dealing with different domains they will not find any problem, but when the domains being treated overlap in some aspects, the interacting ontologies can face difficulties.

In order to avoid those difficulties, the overlapping domains have to agree on the way concepts are faced. One of the possible solutions for that problem is ontology matching, which can also be called ontology mapping or ontology alignment. This technique identifies the entities in both parties which address to the same concept, and match them for the ontologies to know that they are talking about the same concept even if the terms used are different.

Other techniques, such as ontology merging prefer to combine the two ontologies creating a new one in which both concepts are combined.

Once the intuitive idea of ontology matching is presented, we can present some formal definitions.

According to Rahm and Bernstein (Rahm and Bernstein, 2001) an "*ontology mapping process is the set of activities required to transform instances of a source ontology into instances of a target ontology*".

According to IEEE (EEE Computer Society) *"Ontology Mapping is the process whereby two ontologies are semantically related at conceptual level and the source ontology instances are transformed into target ontology entities according to those semantic relations"*.

Another point of view can express the ontology matching as the process of finding correlation between ontologies developed by distributed parties.

The process of ontology matching can be done following various methodologies. The aim of this thesis is to present one possible approach for ontology matching based on String comparisons and representation of the similarity measures obtained by the use of polygons. That technique will be explained step by step in the following section.

In section number five, other approaches for ontology matching will be studied and compared to the polygon method.

# 3 Implementation

## 3.1 OWL

OWL Web Ontology Language is, according to the requirements, the language which is going to be used to describe the input ontologies. This language does not only deal with the representation of information for humans, but also with processing the content of the information. It is a Semantic Web Standard for sharing and reuse of data on the Web.

The bases for this language are taken from RDF, and after, additional vocabulary for the formal semantics is added. We are going to have a short overview about RDF first.

According to the RDF Schema (RDF Schema), "*The Resource Description Framework (RDF) is a general-purpose language for representing information in the Web*".

RDF is a standard to describe information about resources, which are typically URIs. This information is classified into resources, statements (or properties) and individuals.

Each of the statements is represented by an arc, and each of the arcs has three parts: subject (resource from where the arc leaves), predicate (property that labels the arc), and object (resource pointed to by the arc).

A set of statements create a RDF model.

Based on this implementation of a RDF model, it is easy to understand OWL models. OWL has three sublanguages: OWL Lite, OWL DL, and OWL Full, in increasing order of expressiveness.

OWL Full can be viewed as an extension of RDF and every OWL (Lite, DL, and Full) document is an RDF document. In general we can say that with OWL, everything that can be expressed with RDF can be expressed and also more complex concepts about the classes of the ontologies. OWL is considered to be the most expressive language for ontology description.

## 3.2 Editing the ontologies: Protégé OWL

In order to implement the ontologies that are going to be compared by the polygon method, we use the Protégé OWL editor and knowledge-base framework. This free open source editor provides us with the accurate tools to create ontologies and modify their main characteristics. A graphical interface is provided as well as the means to visualize the OWL code.

Through the entire thesis, different examples and scenarios of comparisons between ontologies will be used to illustrate the reader with the steps followed to arrive to the final result. The main example used deals with some hypothetical aspects of an accommodation. The different ontologies dealing with accommodation have some concepts in common and some different ones. At this moment we are going to introduce the ontologies of the example as they were edited with Protégé OWL.

It should be pointed out, that the first ontology introduced to our polygon method is going to be considered as the standard ontology, and the second one is going to be compared to it. This means that the similarity is going to be described as "how similar the second ontology is to the standard one". The standard ontology receives the name of Accommodation1.

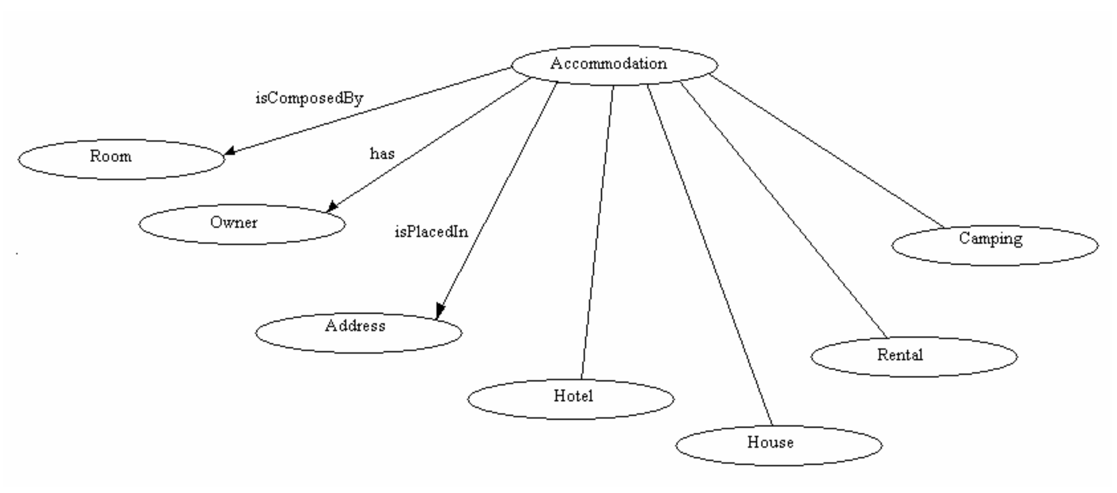


Figure 1: Standard Ontology.

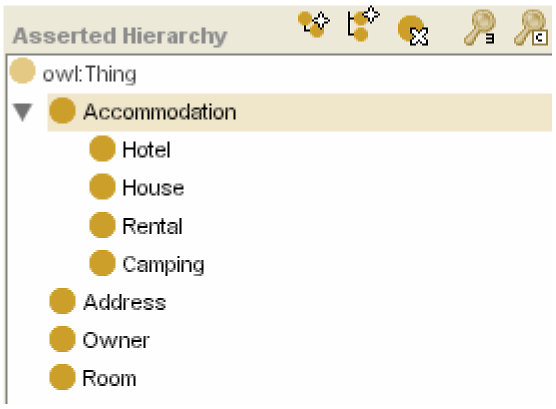


Figure 2: Classes in the Standard Ontology.

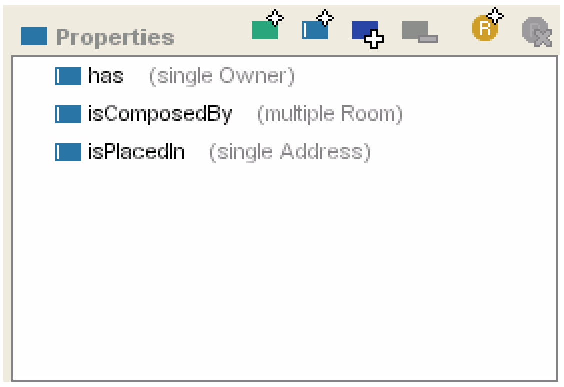


Figure 3: Properties in the Standard Ontology.

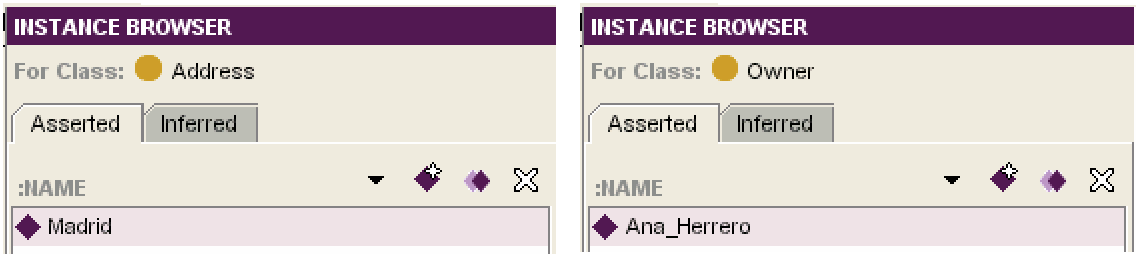


Figure 4: Individuals in the Standard Ontology.

The three main features of an ontology (classes, properties and individuals), are identified for this ontology, which is going to work as the standard one.

We now implement with Protégé the ontology that is going to be compared to the previous one. When referring to this ontology we are going to use Accommodation2.

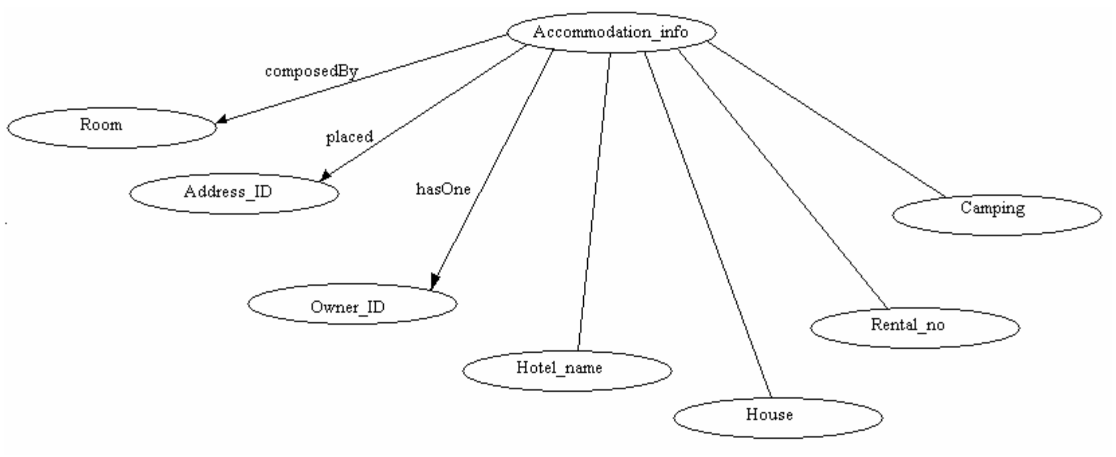
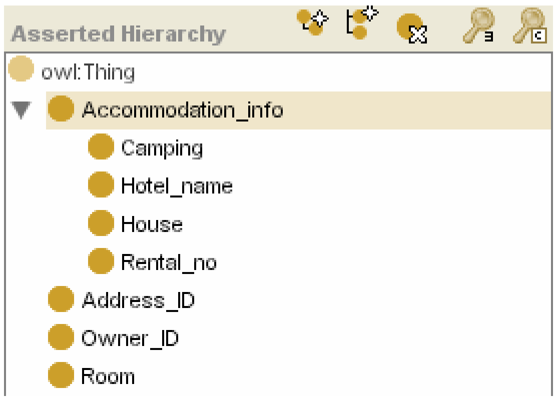
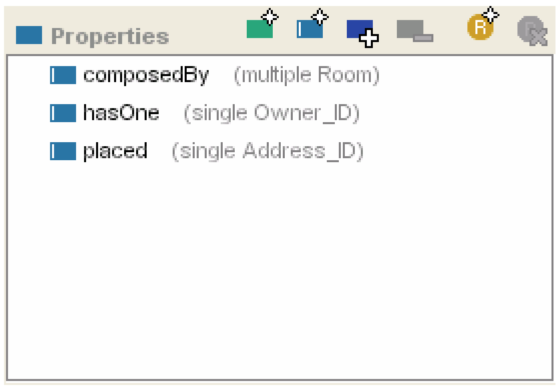


Figure 5: Second Ontology.


Asserted Hierarchy

- owl:Thing
  - Accommodation_info
    - Camping
    - Hotel_name
    - House
    - Rental_no
  - Address_ID
  - Owner_ID
  - Room

Figure 6: Classes in the Second Ontology.


Properties

- composedBy (multiple Room)
- hasOne (single Owner_ID)
- placed (single Address_ID)

Figure 7: Properties in the Second Ontology.

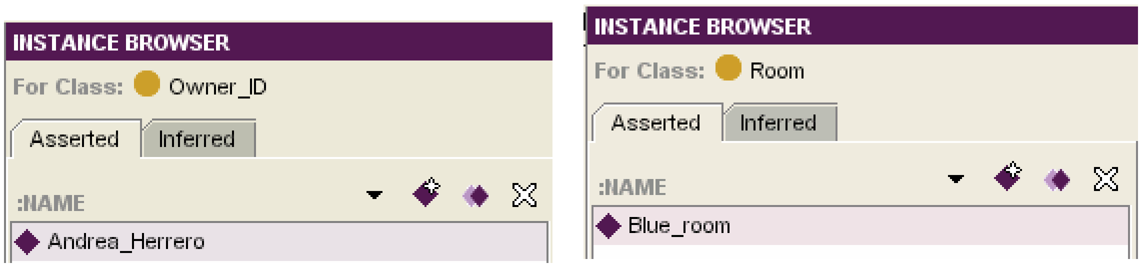


Figure 8: Individuals in the Second Ontology.

Once the implementation is done, the OWL code for each of the ontologies is also generated. In the next step we find the way to access that code with java methods.

## 3.3 Access the ontologies from Java: Jena

As the general requirements specify, the code of the method has to be implemented in java. The requirements also specify that the input ontologies have to be written in OWL. Consequently, OWL code has to be accessible from the java editor.

That problem is aimed by Jena, a Semantic Web Framework for Java. This open source framework used for building Semantic Web applications provides a programmatic environment for RDF, RDFS and OWL. Jena was at first developed to be a Java API for RDF, but later implementations include other functionalities such as the Jena2 ontology API.

This API provides an interface for the Semantic Web application developers. That makes it an ideal programming toolkit when we want to process the ontologies created in Protégé-OWL.

When we deal with Jena, the class `Model` is the one used to access the statements in a collection of RDF data. If, in stead of accessing RDF data, OWL data has to be processed, the class `OntModel` is the one used. This class is an extension of the previous `Model` class and makes accessible the main features of an ontology: classes, properties and individuals. The methods included in this class provide the needed functionalities to access these features in different ways.

In order to use this class the first step is the creation of an ontology model using the `Jena` `ModelFactory`. The polygon method is going to deal with two different ontologies in order to compare them. Therefore, there is a need to create two ontology models, one for each of them.

```
OntModel m1 = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

OntModel m2 = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
```

Accommodation1 is the source file used as input for the first model `m1`, while Accommodation2 is the file issued by `m2`.

From now on, the ontologies will be available by using Java methods through their respective models.

At this point of the implementation we are going to take an overview of the general method we are developing.

The polygon method is based on the comparison of the features from both ontologies, the standard one and the one which is going to be compared. As it has been already pointed out, these features can be grouped into classes, properties and individuals.

Hence, classes in ontology1 have to be compared to classes in ontology2 and the same procedure has to be followed for the properties and the individuals.

There is no need to compare classes in ontology1 with properties in ontology2 and vice versa. This comparison does not add any relevant data to our study, because that would not mean that the ontologies are more similar. Therefore, the comparisons between the features are going to be restricted to the same kind of feature in both ontologies.

Moreover, subclasses will also be considered and grouped regarding their super class. Two subclasses will only be compared to each other if some similarity is found between super classes.

Once the models are created the information from each of the ontologies can be easily retrieved. We use the following methods to retrieve the existing classes, properties and individuals.

```
Iterator i = m1.listClasses(); i.hasNext();

OntClass c = (OntClass) i.next();

Iterator s = c.listSubClasses(true); s.hasNext();

Iterator f = m1.listObjectProperties(); f.hasNext();

Iterator s = c.listInstances(); s.hasNext();
```

By means of these methods the names of the features are stored, and after compared with the ones belonging to the other ontology taking into account the fact that we are only interested in comparing the corresponding groups of features.

The comparisons are made in the next section of this thesis.

## 3.4 Comparing Strings in Ontology matching

After the information retrieval, the names of the elements corresponding to each of the ontologies are stored and converted to Java Strings.

The next step deals with the comparison of these Strings. According to those results of the comparison, the polygons can be represented, and finally, based on those representations, the final matching result can be obtained.

Comparison of strings is a much issued topic in the field of ontology research. The reason is that many different ways of comparing strings can be used, and efficiency and accurateness of the result depends on the situation and the parties involved.

Nevertheless, in this study, we are going to deal with the comparison of strings oriented to Ontology matching. The examples proposed, and the cases considered will all be dealing with the names of the classes, subclasses, properties and individuals in the involved ontologies.

Moreover, there is also a great variety of methods that can be used for the comparison of ontologies and its elements. Consequently, and before taking a decision about which one to use, some studies have to be conducted in order to determine which of the existing methods has better results in the field of ontology matching.

### 3.4.1 Previous research

Previous studies have already conducted research about the effectiveness of the string comparing methods. According to Cohen (Cohen et al., 2003) we are going to have a classification of the methods belonging to the class SecondString.

SecondString is an open-source package of string matching methods based on the java language. These methods follow a big range of approaches and have been designed according to different criteria and perspectives, such as statistics, artificial intelligence, information retrieval, and databases.

Previous classifications divide these methods into three groups according to the methodology they use to establish the correspondence between strings: Edit-distance metrics, Token-based distance metrics and hybrid methods.

1) Edit-distance metrics

This method considers the two strings that are going to be compared. One of them is taken as the input and the other one as the output. Transformations are done between both Strings for them to be the same. The distance between both strings can be seen as the shortest sequence of edit commands that transform the input into the output. These transforming commands are copy, delete, substitute and insert.

Depending on how the cost of the editing operations is considered, two edit distance methods are regarded by the SecondString class.

The first of these methods consider that all the editing operations have unit cost and therefore, the distance increases the same independent from the edit methods used to transform the input into the output. The methods included in this modality are the Levenstein distance and Level2Levenstein distance.

In the second modality, each of the transforming commands has a particular cost. That makes that depending on the parameter used; the distance between the two strings is going to increase in a different way. The methods belonging to this modality are the Monge-Elkan and the Level2Monge-Elkan.

Some similar metrics not based on the edit-distance models are Jaro, and its variants: Level2Jaro, JaroWinkler, and Level2JaroWinkler.

For these methods, the distance between two Strings is determined according to the number and order of the common characters between them.

Given strings:

$s = a_1, \ldots. a_K,$ and $t = b_1, \ldots.. b_L,$

A character $a_i$ in $s$ is said to be *common with t* if there is a

$b_j = a_i$ in $t$ such that $i - H <= j <= i + H$,

where $H = \min(|s|, |t|) / 2$

Let $s' = a_1', \ldots. a_K',$ be the characters in $s$ which are common with $t$ (in the same order they appear in $s$) and

let $t' = b_1', \ldots.. b_L',$ be the characters in $t$ which are common with $s$ (in the same order they appear in $t$)

The definition of a *transposition for s', t'* to be a position $i$ such that $a_i' = b_i'$. Let $T_{s',t'}$ be half the number of transpositions for $s'$ and $t'$. The Jaro similarity metric for $s$ and $t$ is

$$Jaro(s,t) = \frac{1}{3} \cdot \left( \frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|} \right)$$

For the JaroWinkler variant, we can define:

$$Jaro\text{-}Winkler(s,t) = Jaro(s,t) + \frac{P'}{10} \cdot (1 - Jaro(s,t))$$

Where $P$ is the longest common prefix of $s$ and $t$ and $P' = \max(P, 4)$.

2) Token-based distance metrics

In this metric, the strings which are going to be compared are considered as multi-sets of tokens (words).

Jaccard similarity is one of the methods included in this section. It computes the similarity between the sets of words S and T as

$$\frac{|S \cap T|}{|S \cup T|}.$$

TFIDF (or cosine similarity method) is based on the frequency of words inside the tokens as well as the size of the core part. In order to define it, we have to define:

$$V'(w,S) = \log(\mathrm{TF}_{w,S}+1)\cdot\log(\mathrm{IDF}_w)$$

Where $\mathrm{TF}_{w,S}$ is the frequency of word w in S, N is the size of the "corpus" and $\mathrm{IDF}_w$ is the inverse of the fraction of names in the corpus that contain w.

With those concepts we can define:

$$V(w,S) = V'(w,S)/\sqrt{\sum_{w'} V'(w,S)^2}.$$

And finally:

$$\mathit{TFIDF}(S,T) = \sum_{w \in S \cap T} V(w,S)\cdot V(w,T)$$

Jensen and Shannon distance regards a token set as samples from an unknown distribution $P_s$ of tokens. The distance between S and T is computed based on the distributions $P_s$ and $P_T$. KL(P||Q) is the Kullback-Lieber divergence where

$$Q(w) = \tfrac{1}{2}(P_S(w) + P_T(w))$$

The distance can be defined as follows:

$$\mathit{Jensen\text{-}Shannon}(S,T) = \frac{1}{2}\left(KL(P_S||Q) + KL(P_T||Q)\right)$$

$P_s$ distribution can be calculated using the function of Dirichlet or Jelenik-Mercer. That combination results into two methods: Dirichlet JS and Jelinek-Mercer JS.

Finally, Fellegi and Sunter and SFS (Simplified Fellegi and Sunter) methods can also be pointed out.

3) Hybrid methods

An important aspect of these methods to compare Strings is that they can be combined between themselves to obtain better results. When two methods belonging to different branches are combined, we obtain a hybrid method, with some characteristics from each of the modalities.

There are three hybrid methods that should be studied: SlimTFIDF, SoftTFIDF and JaroWinklerTFIDF.

### 3.4.2 Cases and scenarios

Once that all the methods under study have been introduced, and before a comparison between all them is made, the scenarios in which we are going to work are presented in this section.

We are going to evaluate separately the three different types of methods: edit-distance metrics, token-based distance metrics and hybrid methods.

In each of the three evaluations, pairs of strings to be compared are proposed. The pairs of strings are divided into three groups:

a) Very similar pairs of strings.

b) Pairs of strings with some similarities.

c) Pairs of strings with nothing in common.

Inside each of these groups, some features are going to be taken into account for the comparisons. When editing ontologies different styles can be followed for the names of the classes, properties and individuals. The sub-groups of pairs cover those differences. Some of the aspects taken into account are the differences between capital letters and lower-case, the differences between long and short words, the influence of the underscore or the influence of the spaces.

For each of these features, the different string comparing methods will be checked.

The strings used for the evaluation will try to make clear the feature tested. For that purpose, some examples and its variations will be conducted through all the tests.

Depending on the results obtained, it will be decided if a normalization process is needed before the Strings are compared to each other or not.

According to the Unicode Normalization definition the process of normalization produces one binary representation for any of the binary representations of a character. The process removes some differences but preserves the case and reduces alternate string representations that have the same linguistic meaning.

Finally, we have to point out that the next section the best method will be chosen for each of the possible situations and finally one only method will be chosen for the implementation of the application.

### 3.4.3 Comparison of methods

We compare the methods running examples in Java. After that, we present the results in the tables at the appendixes.

It has to be pointed out that to consider two strings to have a relevant similarity the value of the similarity between them has to be higher than 60%.

The decision of that value is based on empirical practices. After reviewing examples and considering different scenarios, it is demonstrated that strings with a similarity above that value can be considered to have something in common, while strings with a similarity coefficient below that percentage are not really related to each other and the similarity can be just considered as a coincidence.

Due to that reason, pairs of strings with a similarity coefficient lower than 0.6 will not be taken into account when comparing the classes, properties and individuals in the ontologies.

In this section we analyze the results:

1) Edit-distance metrics:

The metrics included in this section are Levenstein, Monge-Elkan, Jaro and Jaro-Winkler. All of them in the simple form and in the Level2 form. That makes a total number of 8 metrics to compare. The way results are presented in each of the metrics has to be taken into account as well. Monge-Elkan, Jaro and Jaro-Winkler show the similarity with a number between 0.0, when there is any similarity, and 1.0, when the strings coincide 100%. However, Levenstein shows the similarity with a number between 0.0, meaning maximum similarity and -• meaning no similarity at all.

a) Very similar pairs of strings:

The comparison tables are in the appendixes 1 and 2.

In the first one we can find the comparisons of the pairs of strings in one sense while in the other table the comparisons are made in the other direction. The position in which the strings are introduced to the comparing method affects the result for the Level2 edit distance metrics while the Level1 metrics are not influenced by the order of the strings. We will analyze that influence for different cases.

When we compare very similar pairs of strings we can make several remarks: Any of the metrics considers the influence of capital letters, regardless the length of the word or the order of the strings.

Adding a particle before or after the main word (*isComposed*, *composedBy*) Levenstein, Jaro and Jaro-Winkler consider a difference with the original word (*composed*). In the case of Levenstein the difference is quite big, while in Jaro and Jaro-Winkler the percentage of similarities goes down to 80.833%. Monge-Elkan considers that the strings continue with 100% similarity. For all these first examples, the order in which we place the strings for the comparison does not influence the final result.

When the particle is introduced with the underscore (*is_composed*, *composed_by*) Levenstein considers that the similarity between them and the original word is even smaller. Monge-Elkan is the only metric which continues considering 100% similarity between those variations and the original word. Jaro and Jaro-Winkler estimate the similarity in 78.40%, which is still a considerable similarity.

As soon as the length of the original word begins to decrease, Jaro and Jaro-Winkler do not consider the similarities to be so obvious, and when the original word has 5 or less characters (*is_compo*) the similarity arrives to 0%.

In general, the Level2 metrics detect more differences between the strings, and show lower values for the similarities when more changes are introduced. We can say that these metrics are more sensible to changes and stricter when considering differences between strings. A good example of this can be the difference between Jaro-Winkler and Level2 Jaro-Winkler when evaluating the similarity between *composed_by* and *composed*. Jaro-Winkler estimates 94,54% while Level2 Jaro-Winkler estimates 50% similarity.

The length of the words when the particle is added with underscore does not vary the result with the Level2 metrics which continue considering 50% of similarity in all the cases (*is_composed, is_compose, is_compos, is_compo, is_comp*).

On the other hand, when we change the order of the introduced strings, the difference between the results is remarkable. While in the first sense, the results are as we have already described, in the other sense the Level1 metrics continue with the same behavior but the Level2 metrics consider that all the pairs are 100% similar no matter the length of the word or the prefix we add with underscore.

That means that if we introduce first the original word and then its variations, the Level2 metrics consider 100% similarity between the strings, while in the other sense the percentage is always 50% no matter the length of the word. We will take into account this characteristic when taking a decision to choose one of the metrics and implementing the method.

When we consider the influence of the spaces we find the same results as the ones for the underscore. We add short particles before and after the main string (*composed by*, is *composed*) and only Monge-Elkan considers that the strings continue the same. The rest of the methods make the difference between the strings with a similarity of (78.40%, 90.90%) and in the case of Level2, the percentage goes to 50% in all the cases.

The same results are obtained when changing the order of the strings for the comparison and when reducing the length of the words.

Finally we study the difference between the underscore and the space. In this case the Level2 metrics consider that there is no difference between them and the normal metrics consider a small similarity between them (around 94%).

Jaro and Jaro-Winkler have very similar results in all the cases, but we can notice that Jaro tends to be more exigent and finds more differences between the cases proposed.

As the pairs of strings compared in these two tables are considered to be very similar, the results obtained are supposed to be higher than 60% of similarity (that value will be considered as relevant). In these tables, the values under 60% are pointed out for the reader to identify easily, where the metrics do not obtain the expected result.

b) Pairs of strings with some similarities:

The comparison tables are in the appendixes 3 and 4.

In these tables we can see that the conducting example is still the same, but in this case we are going to compare the string *composed* with three others with some similarities (*component, compound, comprehend*). As in the previous table, the influence of the capital letters, the underscore, the spaces and other features are going to be studied.

When we compare just the original strings and the strings with some particle before and after it we realize that the method showing a higher similarity is Jaro-Winkler, after that Jaro, then Monge-Elkan and finally Levenstein. In all of the cases, the results do not vary from the normal metric to the Level2 metric. Changing the order of the strings introduced does not vary the results neither.

When we introduce the underscore variations, Jaro-Winkler, Jaro and Monge-Elkan still detect similarity while the Level2 metrics and Levenstein do not detect important similarities between the strings.

This results change drastically if we change the order of the strings. When the original string is introduced first and after the variations, the similarities found are higher at it happened in the previous tables.

In this case, the results from the Level1 metrics remain the same, but the results obtained from the Level2 metrics change from 40% to 70% (or even 90% in the case of Jaro-Winkler), which means that they become relevant to our study.

The same behavior is found for the spaces. Jaro-Winkler, Jaro and Monge-Elkan give always values higher than 60% with small variations between them, while the rest of the methods do not consider relevant similarities between the proposed strings.

As it happened in the previous table, the Level2 metrics do not consider similar any of the pairs of strings when one of the strings belonging to the pair includes underscore or space, but when we go to the table in 4 and we change the order of the strings, Level2 metrics consider even more similarities between the strings.

c) Pairs of strings with nothing in common:

The comparison tables are in the appendixes 5 and 6.

In these examples, we will compare the string *composed* to three strings with no similarity to that one: *formed*, *integrated* and *organized*.

No similarity should be found between the proposed pairs provided that there are not enough things in common to think that the pairs should be matched to each other in a real example. The only cases which have a similarity higher than 60% are marked in yellow to indicate that the values obtained are not the ones expected.

For the original strings and the strings with the particles before and after, Levenstein, Level2 Levenstein, Monge-Elkan and Level2 Monge-Elkan find no relevant similarity in any of the cases.

Jaro-Winkler, Jaro-Winkler Level2, Jaro and Jaro Level2 find similarities between two pairs of strings, when the string *formed* is compared.

When introducing the underscores and the spaces, all the methodologies consider that the similarity found is not relevant enough, and only Jaro and Jaro Winkler obtain a value higher than 60% in one occasion.

Level2 Jaro-Winkler and Level2 Jaro are more exigent and the similarities they find can not be considered to be relevant in any of the examples.

When we go to table 6 and consider the comparisons in the other order, we realize that more similarities are found. Without underscores or spaces, the results are the same, but when we introduce those elements Jaro and Jaro-Winkler detect similarity in 4 cases while Level2 Jaro and Level2 Jaro-Winkler detect similarity in 10 cases.

2) Token-based distance metrics:

In this section we will analyze six different metrics based on the study of tokens. Three of them are variations from the Jensen-Shannon method. When this method is implemented with the Dirichlet function we obtain the Dirichlet JS, when it is implemented with the Jeliken-Mercer function, the obtained method is Jeliken-Mercer JS, and finally we have the Unsmoothed JS. Apart from that, we will analyze the results for the Jaccard metric, TFIDF metric and Fellegi and Sunter metric.

When regarding the way the metrics express the results, Jaccard, TFIDF and Unsmoothed JS express the similarity with values from 0,0 (no similarity) to1,0 (100% similarity). Dirichlet JS and Jeliken-Mercer JS do not result in any numeric value expressing the similarity, the only characteristic they express is the lack of similarity by means of 0,0 numeric value. The rest of the possibilities are expressed by NaN (Not a Number), which will be not useful for the aim of our study. Finally, Fellegi and Sunter metric expresses the value 0,0 when no similarity is found, and the maximum value varies depending on the length of the strings.

We can point out that these metrics will not be influenced by the order of the strings. Due to that, we will have just one table for each of the three groups of pairs:

a) Very similar pairs of strings:

The comparison table is in the appendix 7.

In this case we compare the same strings as in the table for the edit distance metrics. The use of capital letters is not relevant for any of the methods, but they all consider that adding a short particle at the beginning or at the end of the string causes the total loss of similarity with the original string.

When introducing the underscore, the length of the strings is not relevant, but only TFIDF and Unsmoothed JS consider a similarity in those cases, as it happens with the use of the spaces.

The difference between the use of the underscore or the space is not appreciated by any of the methods.

b) Pairs of strings with some similarities:

The comparison table is in the appendix 8.

When the pairs of strings begin to have some different aspects, the results are the same for all of the methods: any of them consider that the strings have any similarity, so all the similarity coefficients obtained are 0, for all the metrics and for all the possibilities.

c) Pairs of strings with nothing in common:

The comparison table is in the appendix 9.

The same thing happens when the pairs of strings get even more different. None of the methods is able to find any similarity between them, so all the displayed results are 0.

We can conclude saying that the Token-based distance metrics are appropriate for pairs of Strings that have many aspects in common, but as soon as the examples get more and more different these methods are not able to notice the minimum aspects they have in common.

3) Hybrid methods:

It is time to analyze how hybrid methods are used in order to compare ontologies. Hybrid methods are formed by a combination of an edit-distance metric and a token-based metric. We are going to analyze 3 methods following these characteristics: Slim TFIDF, Jaro-Winkler TFIDF and Soft TFIDF. For each of them we will propose the same tables of strings already used, and the strings will be introduced in both senses in order to analyze how that affects the results.

a) Very similar pairs of strings:

The comparison tables are in the appendixes 10 and 11.

When comparing similar pairs of strings with hybrid methods, characteristics from both edit-distance metric and a token-based metric are observed.

The first change introduced is the presence of the capital letter. In this situation the three methods consider that the strings are exactly the same, no matter the order in which they are introduced to the method.

When particles are added before and after the main string (*isComposed, composedBy*), the same results are obtained in both tables: Slim TFIDF and Jaro-Winkler TFIDF consider a very high similarity, while Soft TFIDF sees no similarity at all. When the underscore is introduced between the particle and the string (*is_composed, is_com, composed_by*), the three methods consider a relevant similarity, regardless the length of the string.

Only when both particles are introduced, before and after the string (*is_composed_by*) Soft TFIDF considers that the similarity is 0% if the original string (*isComposedBy*) is introduced in the second place, and the three methods consider a non-relevant similarity if the original string is introduced in the first place.

Finally, when comparing the original strings with the ones which have a particle introduced and separated by a space (*is composed*), all of them show relevant similarities between the pairs.

It can be pointed out that while Slim TFIDF and Jaro-Winkler TFIDF are influenced by the order of the strings, Soft TFIDF returns same results.

b) Pairs of strings with some similarities:

The comparison tables are in the appendixes 12 and 13.

The table with pairs of strings with some similarities is very revealing. In Slim TFIDF and Jaro-Winkler TFIDF the results are around 60% of similarity. It could be expected because that is the value marking the relevance of the string similarity and in this cases the similarity if the strings is relevant only in some of the cases. The unexpected is that Soft TFIDF finds 0% similarity in some of the cases, as it is considered that all of them have at list some similarities.

That behavior is found regardless the length of the word, the features introduced and the order of the strings.

c) Pairs of strings with nothing in common:

The comparison tables are in the appendixes 14 and 15.

The behavior of Slim TFIDF and Jaro-Winkler TFIDF is coherent with the one we have been observing in the past tables. Both detect that the pairs of strings have few things in common so the similarity found between them is not considered to be relevant in most of the cases, except from some occasions.

As it could be expected Soft TFIDF continues with the strict view and considers 0% similarity for all the pairs of strings in both senses.

### 3.4.4 Results

This sub section has presented the results obtained after comparing the pairs of strings. Once those results have been explained and analyzed, it is time to take a decision according to the data and decide which methods are more convenient for the comparison of ontologies at which this thesis is aimed.

1) Edit-distance metrics:

Edit-distance metrics were the ones we first compared. Among the eight methods compared, Levenstein is the one we can consider to be less appropriate for our purposes. The results are not given in percentages and after the studies made we can say that the results are not very accurate. Level2 Levenstein has the same problems found for Levenstein.

Monge-Elkan and Level2 Monge-Elkan give results included in the expected ranges. Even though, the metrics do not make differences between the different examples, and different cases have the same result. Due to this reason we can conclude that these methods do not discriminate very well between the strings and though, the results given are quite approximated.

We will focus on the four methods left: Jaro, Level2 Jaro, Jaro-Winkler and Level2 Jaro-Winkler.

The results obtained for Jaro and Jaro-Winkler are very similar, the same as the ones obtained for Level2 Jaro and Level2 Jaro-Winkler. In this situation, we will first compare the two groups of metrics: Level1 and Level2.

The first characteristic we can point out is the fact that Jaro and Jaro-Winkler consider no similarity for the strings with particles for short words (*comp, is_comp*). That problem is solved by Level2 metrics, which consider 100% similarity in all those cases in one of the senses.

Even if Level2 metrics have to make the evaluation in both senses in order to deal with all the possibilities, once that has been taken into account, the results obtained are more accurate than the ones obtained with Level1 metrics.

When we move to more different pairs of strings, Level2 metrics detect more similarities. At this point we have to take into account that ontology matching is used in the cases where the domains of the ontologies are overlapping. That means that the domains compared will have many features in common. In that situation we can conclude that Level2 metrics will give a better result due to the fact that they work better with similar strings than Level1 metrics do.

When taking a decision between Level2 Jaro and Level2 Jaro-Winkler, we realize most of the times both methods obtain similar results.

Level2 Jaro-Winkler obtains better results in some occasions so we can conclude this subsection with the conclusion that Level2 Jaro-Winkler provides the best functionality among the edit distance metrics for ontology matching.

2) Token-based distance metrics:

Once the comparison of these methods is finished we can reach the conclusion that any of them is valid for our objective. The results obtained with all of them do not consider many of the similarities needed when comparing ontologies, so we conclude saying that token-based distance metrics are not appropriate for the implementation of applications dealing with ontology matching.

3) Hybrid methods:

Hybrid methods introduce a new option with the concept of combining the two previous groups of methods.

Among the three methods presented, Soft TFIDF is the first one to be ruled out. The results obtained for the pairs of strings with some similarities show no similarity in some cases where there should be some found. Our objective is to put in common ontologies developed in distributed environments, and even if the domains issued will be the same, it is necessary to have a margin in order to regard possible pairs only with some common features.

We move then to the evaluation of Slim TFIDF and Jaro-Winkler TFIDF. The results obtained by these metrics are similar in all the situations, but when regarding very different pairs of strings we can easily observe that the results obtained by Slim TFIDF are more accurate while Jaro-Winkler TFIDF finds more similarities between some pairs of strings.

According to that fact we can say that Slim TFIDF is the most appropriate metric for our purposes, among the hybrid metrics.

Finally, we have to make the comparison between the two metrics which have been chosen as the most appropriate ones: Level2 Jaro-Winkler and Slim TFIDF.

Considering the similar pairs of strings both methods have a very similar behavior, especially if we consider that when implementing the application we will take into account the maximum value found doing the comparisons in both senses.

The most important differences are found when we move to the pairs of strings with nothing in common. In this case Slim TFIDF finds fewer similarities and considers that the strings have fewer things in common. That leads to more accurate results.

We can conclude the study of the comparing methods included in the Class Second String saying that when dealing with Matching Ontologies, the most appropriate method, and therefore the one we are going to use for our application is Slim TFIDF.

## 3.5 From the comparison results to the polygons: JMatLink

At the end of section 3.4, a method was chosen for the implementation of the application. Once we have made a decision on the best comparison method for editing ontologies, we can apply that method to obtain the results of the comparisons between the strings. Those results correspond to the best matching found between the two ontologies. That means that for each of the classes, properties and individuals in the standard ontology, the most similar string in the second ontology is found and matched to it.

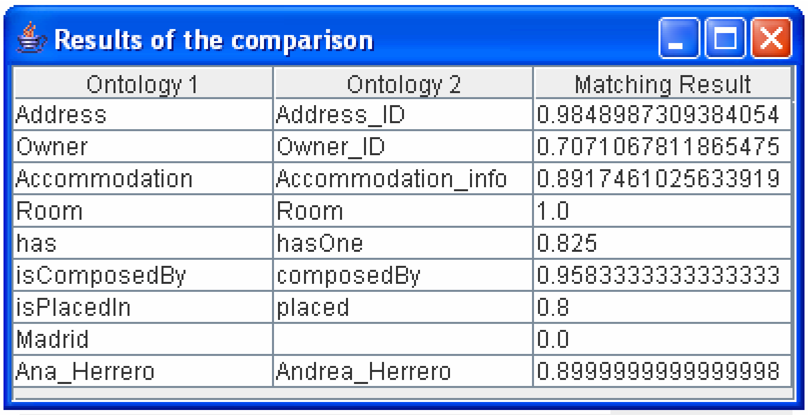
That information is showed in a table containing the pairs found and the similarity coefficient between them.

Concerning subclasses, if two classes are found to have a relevant similarity and consequently matched to each other, it is checked if they have subclasses. If subclasses are found in both ontologies, another comparison is made between them and another table shows the similarity values between those Strings.

Therefore, there will be a table for the classes in first level, and another similarity table for each of the groups of subclasses belonging to the same super class similar to another one.

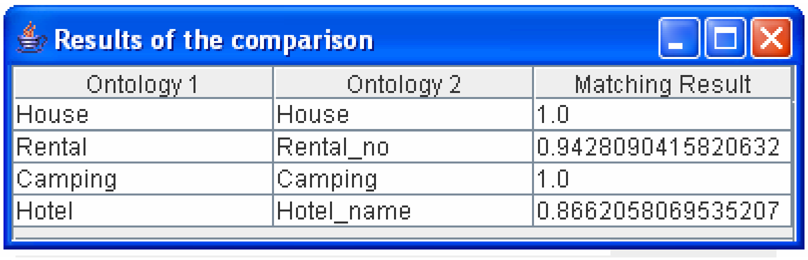
Regarding the example already proposed, when we run the application we obtain two different tables. The first table includes the similarity measures for the classes, properties and individuals, while the second one shows the similarities found for the subclasses belonging to the classes *Accommodation* and *Accommodation_info*.

We can see that the pairs are matched according to the maximum similarity found and the matching result is also expressed in the third column of each table.


Results of the comparison

| Ontology 1 | Ontology 2 | Matching Result |
|---|---|---|
| Address | Address_ID | 0.9848987309384054 |
| Owner | Owner_ID | 0.7071067811865475 |
| Accommodation | Accommodation_info | 0.8917461025633919 |
| Room | Room | 1.0 |
| has | hasOne | 0.825 |
| isComposedBy | composedBy | 0.9583333333333333 |
| isPlacedIn | placed | 0.8 |
| Madrid | | 0.0 |
| Ana_Herrero | Andrea_Herrero | 0.8999999999999998 |

Figure 9: Similarity table for the comparison.



| Ontology 1 | Ontology 2 | Matching Result |
|---|---|---|
| House | House | 1.0 |
| Rental | Rental_no | 0.9428090415820632 |
| Camping | Camping | 1.0 |
| Hotel | Hotel_name | 0.8662058069535207 |

Figure 10: Similarity table for the comparison of subclasses.

The next step of the implementation consists on finding the general similarity coefficient of the ontologies. That is going to be done by representing polygons based on the results obtained and measure their area.

For the representation of the polygons, a software package with mathematical functions has to be used. Matlab was found to be the ideal tool for this purpose and thereby was chosen for the implementation of the software.

Matlab is a very widely extended tool with its own programming language for technical computing. Its wide use has made it connectable with many other applications. When trying to access Matlab with Java, we find that we need a library called JMatLink. Those libraries allow us to use the functions available in Matlab using the Java programming language.

In the next section we will see how the polygons are represented in Matlab according to the similarity coefficients found between the pairs of strings.

## 3.6 Representation of results: Matlab

Based on the information in the tables, the polygons are represented in Matlab.

Many approaches can be followed in order to arrive to the final polygons. In this section the approach chosen for the application is explained.

When the standard polygon has to be represented, the similarity values are not taken into account. All the points have a distance 1 to the centre of the axes. The first pair is represented by a point placed in the x axis (0º). The point for the second pair found is placed 180º from that point. The third one is placed 90º far from the first pair and the fourth one is placed 270º far from the first pair. Once the four first pairs are situated in the standard polygon we obtain four points corresponding to the four axes.

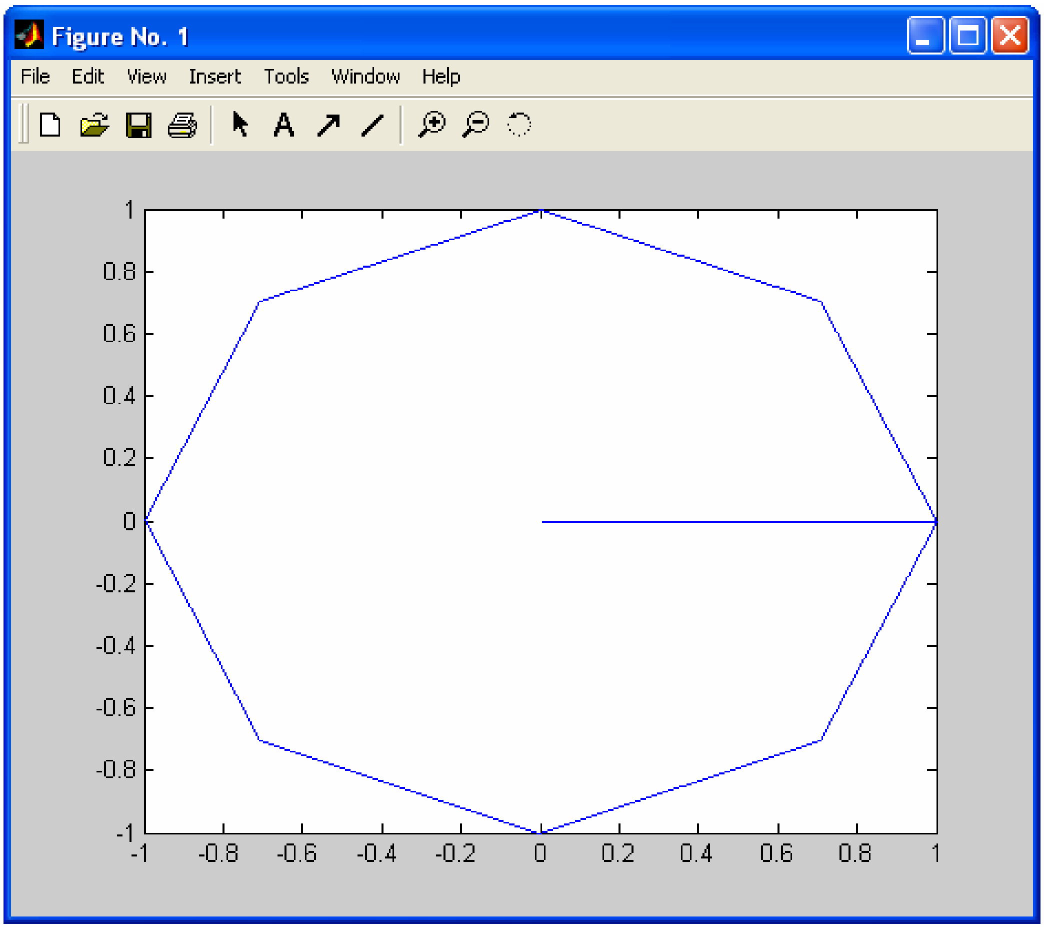
When more pairs are found, new points are added to the standard polygon. Another axis is introduced cutting in two halves the first quadrant and the perpendicular one is also added. When placing the four next points they will have a distance to the x axis of 45º, 225º, 135º and 315º, respectively.

After introducing those points, the same tasks are performed again in order to introduce new points, the 45º angle is divided in two halves and new axes are represented.

The number of points represented is equal to the number of pairs which have a relevant similarity.

According to that procedure, when we represent the standard polygon for the table in Figure 7, we obtain the representation of eight points, providing that one of the elements in the standard ontology does not match any of the elements in the second ontology with a relevant similarity value.

The area of the standard polygon is calculated and stored in order to be used to calculate the final result.



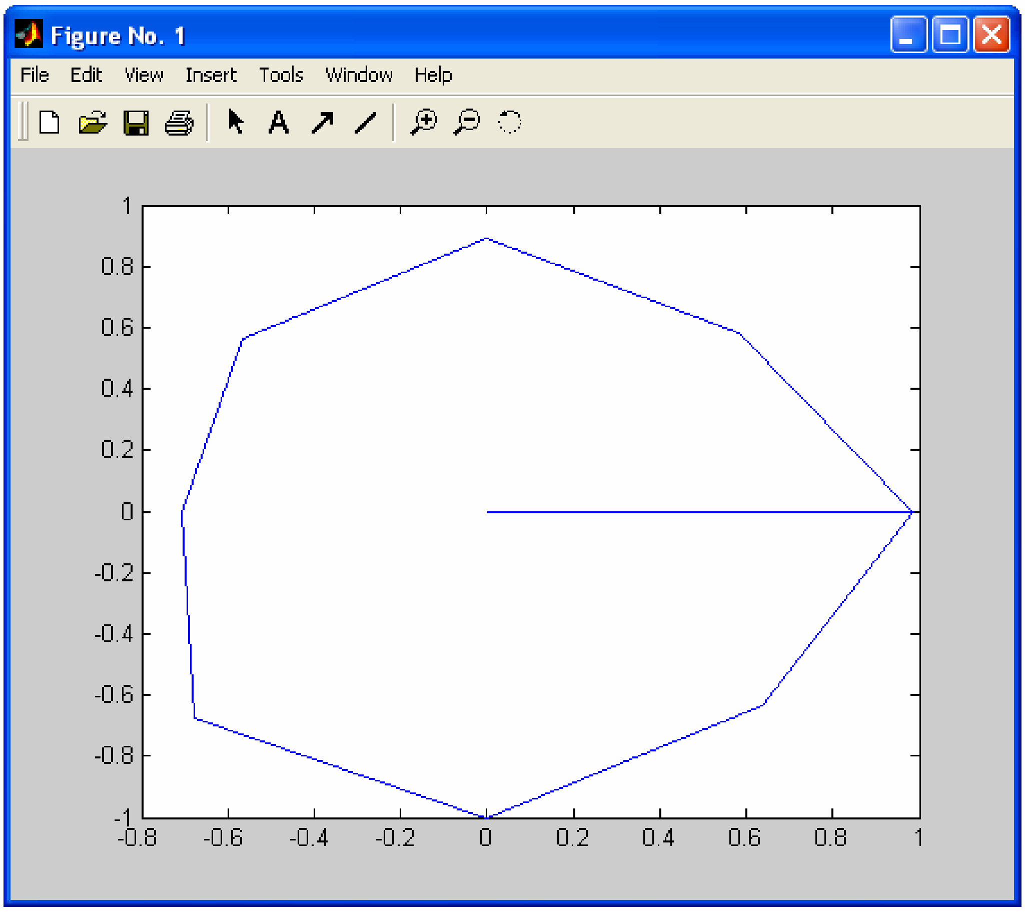
```
Area1= 2.8284271247461903
```

Figure 11: Standard polygon.

The second polygon is represented using the similarity coefficients from the table in Figure 7. Each of the points in the standard ontology, corresponding to a pair matched is multiplied by the similarity coefficient corresponding to that pair.

Therefore, we obtain a polygon with the same number of vertices but a different area, varying according to the similarity values shown in the table.

The polygon for the comparison of the classes, properties and individuals and the value of its area are shown in Figure 10.



```
Area2= 2.2096027964711165
```

Figure 12: Second polygon

The second table has to be taken into account in this point of the implementation. The subclasses of *Accommodation* and *Accommodation_info* are represented in the same way.

For the standard polygon of the subclasses we introduce two multiplying factors.

The first multiplying factor is related to the similarity measure of the super class. In the example, *House*, *Hotel*, *Rental* and *Camping* are subclasses of the super class *Accommodation*. *Accommodation* is related to the class *Accommodation_info* with a similarity of 0,891746. That makes that all the subclasses are going to be represented in their polygon multiplied by that factor.

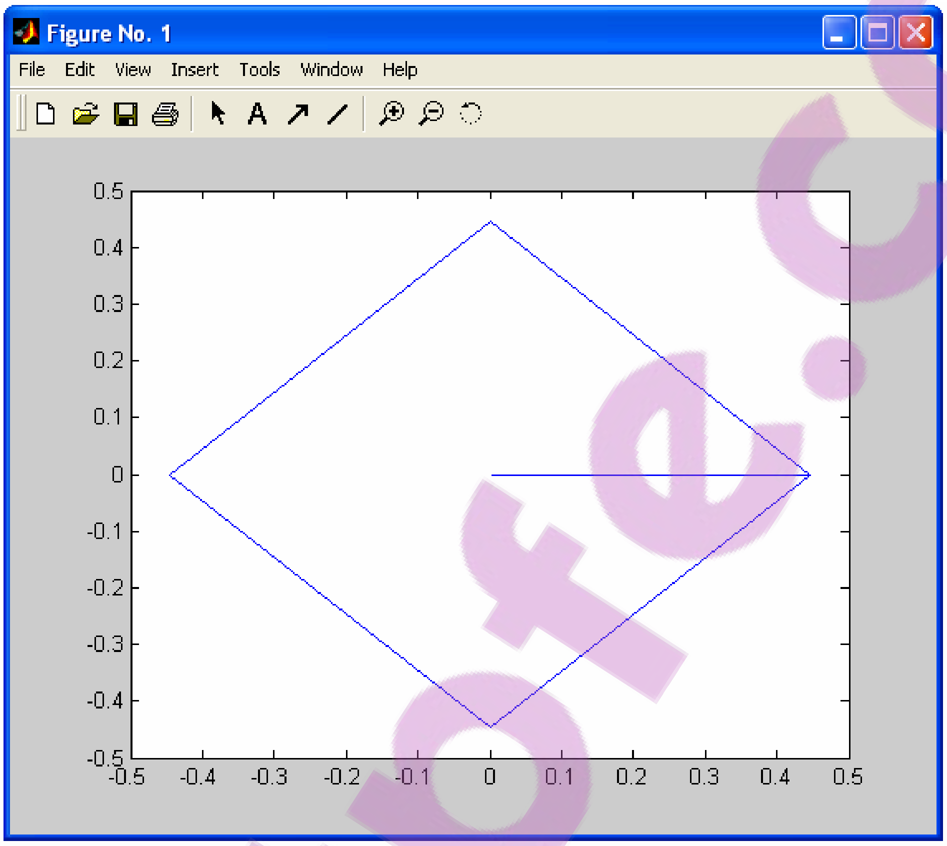
The second multiplying factor depends on the level where the subclass is found. Subclasses placed closer to the original classes are considered to be more relevant for the comparison of two ontologies, while subclasses in lower levels are less relevant for the study.

Consequently, the multiplying factor for the subclasses in first level is 0,5, while the factor for the rest of the levels is the factor in the first level divided by the number of the level.

According to that procedure, the resulting factors for this polygon are:

0,5*0,891746 = 0,445873.

Therefore, the standard polygon for the group of subclasses found has four vertices at the points of (0, 445873,0), (0, 0,445873), (- 0,445873, 0), and (0, -0,445873).



```
Area1 subclasses= 0.39760555571849976
```

Figure 13: Standard polygon for the group of subclasses.

When the second polygon has to be represented for the group of subclasses, the value of each point in the standard polygon is multiplied by the corresponding similarity value.

By that simple operation, the area of the second polygon varies once more and the second polygon is obtained.

In this example, the similarities found are:

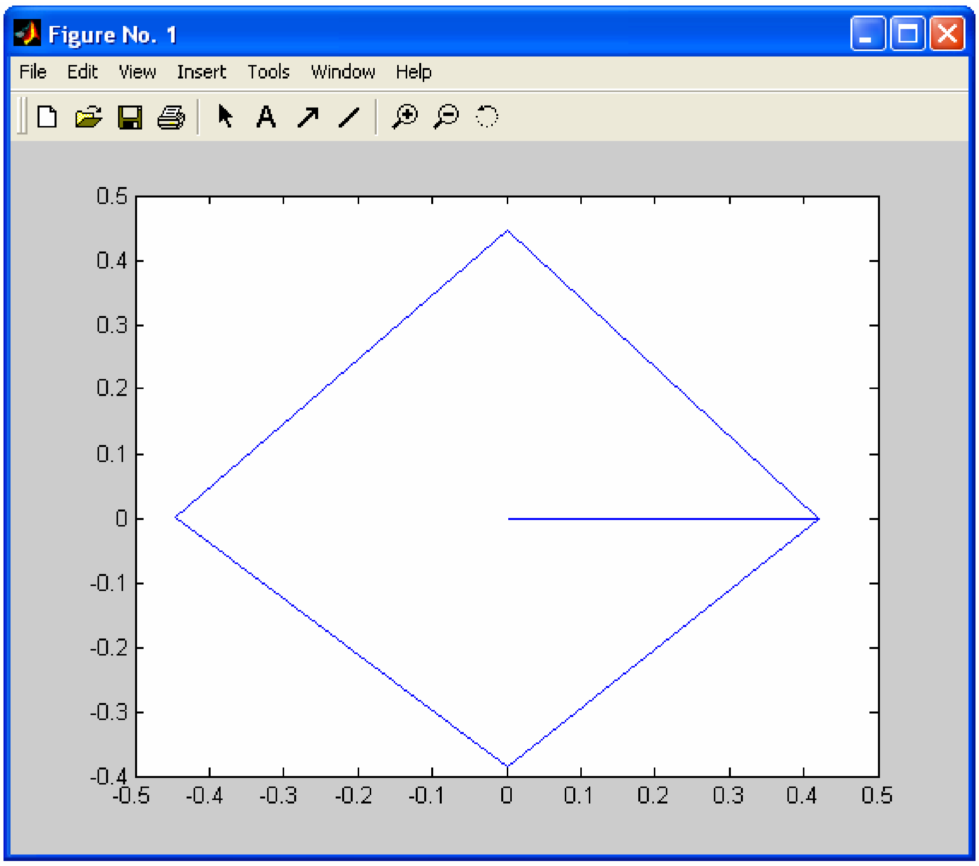
| | |
|---|---|
| House – House | 1,0. |
| Hotel – Hotel_name | 0,8662. |

| | |
|---|---|
| Camping – Camping | 1,0. |
| Rental – Rental_no | 0,9428. |

When multiplying those values by the factors that we already had, the resulting values are

| | |
|---|---|
| House – House: | 1,0*0,5*0,891746=0,445873. |
| Hotel – Hotel_name: | 0,8662*0,5*0,891746=0,3862. |
| Camping – Camping: | 1,0*0,5*0,891746=0,445873. |
| Rental – Rental_no: | 0,9428*0,5*0,891746=0,42069. |

Representing those distances in the correct axes we obtain the polygon and its area.



```
Area2 subclasses= 0.36039777842757076
```

Figure 14: Second polygon for the group of subclasses.

When all the areas of the polygons have been stored, the final value can be calculated.

Areas corresponding to the standard ontologies are added together, as well as areas corresponding to the non-standard ontologies.

Area1 Total = 2.8284271247461903 + 0.39760555571849976.

Area2 Total= 2.2096027964711165 + 0.36039777842757076.

The similarity value is: $\sqrt{\frac{\text{Area2 Total}}{\text{Area1 Total}}}$

We can conclude that the similarity between ontology 1 and ontology 2 is: **<u>0.8925493312983556.</u>**

# 4 Conclusion and Discussion

In this section of the thesis, some other methods for ontology matching will be studied and compared to the polygon method. According to the results achieved, some future work will be proposed.

As we have already said, the ontology community has developed various strategies for ontology matching, according to different approaches. Many classifications have also been made for these strategies, tools and methods depending on the aspect of the ontology compared, the input received, the result given, etc.

One of the most famous classifications groups the strategies for matching ontologies into de following groups:

- Hierarchical clustering techniques.
- Formal concept analysis.
- Analysis of terminological features of concepts and relations.
- Analysis of structure.

All of these methodologies are motivated by the same fact: the distributed ontology development. Distributed organizations and communities develop ontologies covering different domains. The problem comes when two or more ontologies cover overlapping domains and we need to put them in common. In that moment, some technique has to be used: ontology matching, ontology alignment, ontology merging, etc.

The election of the approach and the method varies from one community to another. Therefore, it is interesting to consider solutions given to the problem from different points of view.

Six different methods are presented in order to give a general view about the possibilities to solve the problem presented above.

## 4.1 Method 1: GLUE.

According to Doan, Madhavan, Domingos and Halevy (Doan, Madhavan, Domingos & Halevy, 2003) GLUE solves the problem of finding semantic mappings, given two ontologies to ensure interoperability between them.

This method uses learning techniques to semi-automatically create semantic mapping between ontologies. The key step consists on looking for semantic correspondence. This technique is still conducted by hand, which means an important bottleneck in building large scale information management systems.

For GLUE, the ontologies are organized in taxonomy trees. Each of the nodes in the trees represents one concept and each of these concepts has associated a set of instances and a set of attributes.

The matching procedure consists on mapping between taxonomies. For each of the concepts in one taxonomy the most similar concept node in the other taxonomy is found.

GLUE uses the joint probability distribution to compute the similarity within the multi-strategy learning approach. This approach combines a set of learners and their predictions with some domain constraints and heuristics in order to obtain matching accuracy.

When taking a closer look to the implementation we distinguish the following steps: In the first step, the distribution estimator takes two ontologies as an input, including their structure and data instances. For every pair of concepts in the ontologies the learning techniques compute the joint probability distribution. After that, those numbers go to the similarity estimator, which applies similarity functions and obtains the similarity matrix. In the final step, the relation labeler takes the matrix, domain specific constraints and heuristics and obtains the final mapping configuration as an output.

This method is evaluated in three real world domains in order to evaluate the matching accuracy. A high accuracy was obtained (from 66% to 97%) but some problems were still found: there was not enough training data, the learners used were not appropriate and some nodes were considered to be unambiguous.

This method has been placed in the first position considering the similarities with the polygon method. Both of them follow the same schema of comparisons.

In GLUE, all the possible pairs of concepts are made, and then the similarity between them is evaluated. A matrix is formed with those values and finally the highest value shows which concepts are more likely to be matched together.

In the polygon method that procedure is also followed. The classes in one of the ontologies are compared to the classes in the other one, obtaining similarity measures. The same happens with the properties and the individuals, and once all of them are computed, they are placed into a matrix where the highest value is obtained showing the matching pairs as a result.

Both of them have also a semantic approach, but the techniques to compare the concepts are not the same ones. GLUE uses learning techniques while the polygon method uses the comparison between strings.

One advantage which can be found in the polygon method is the fact that it is an automatic method. Therefore, there is no need for any hand conducted part in the implementation. That fact makes it faster to compute the similarity and improves the accuracy of the result. Moreover, the polygon method is a complete system while GLUE is considered to be just one piece of a more complete ontology matching solution.

## 4.2 Method 2: Anchor-PROMPT.

According to Noy and Musen (Noy & Musen, 2001) they have developed an algorithm which finds semantically similar terms automatically. The input for this method is a set of anchors, which are pairs of related terms defined by the user or automatically identified by lexical matching.

The algorithm analyses paths in the sub-graph limited by the anchors and determines which classes frequently appear in similar positions on similar paths. These classes are likely to present semantically similar concepts. It also uses a set of heuristics to analyze non logical context. It tries to determine additional possible points of similarity between ontologies.

From the anchors from the source ontologies, and by means of going through the paths between anchors in the ontologies and comparing the terms along these paths to find similar terms, a set of new pairs of semantically class terms is obtained.

If we arrive to two classes in parallel at the same time while following two paths we can deduce they have something in common. We know for sure some terms in the ontologies are similar. There are paths connecting these terms, so we can say that the terms in those paths are similar too.

Some other systems follow a similar procedure, but they do not take into account the internal structure of concept representation and the structure of the ontology itself. Anchor-PROMPT does, and that represents a big advantage.

It can be considered to be a method augmenting existing methods such as CHIMAERA and PROMPT, but not providing a global solution.

CHIMAERA is an interactive merging tool based on ontolingua. The only relations that are considered are the subclasses, super classes and the slot attachments.

PROMPT is a method for semi automatic guided ontology merging. It identifies candidates for merging as pairs of matching terms (from different sources representing similar concepts).

It has a semantic and syntactic base and takes into account the content and the structure of the source ontologies and the user actions.

One disadvantage has to be pointed out: the input has to be already ordered.

When the Anchor-PROMPT is evaluated, the accuracy achieved for ontologies developed separately is around 75%. The range is extended between 61% and 100% for general ontologies.

This method relies very much in the relationships between concepts and in the fact that concepts between two similar concepts are going to be similar. Sometimes those assumptions can be risky and bad results can be obtained as a result.

As it happened with the previous method, human interaction is needed for the implementation. In this case, the anchors have to be ordered manually before the beginning of the method. That ordering will determine the whole implementation, which makes the method reliable up to a certain point and always dependable on a human point of view.

## 4.3 Method 3: S-Match.

Studying the paper about S-Match (Giunchiglia, Shvaiko and Yatskevich, 2004) we find another interesting approach for ontology matching.

S-Match is a semantic matching approach which aims at combining ontologies developed by distributed communities. The method takes two ontologies as input and produces a mapping between the nodes in those two structures which correspond semantically to each other. Two different components of the ontologies are taken into account: nodes (concepts) and labels (relationships).

S-Match takes two trees. For each pair of nodes from the two trees, it compares the strongest semantic relation. The possible semantic relations are: “equivalence”, “more general”, “less general”, “mismatch” and “overlapping”. These relations have to be found among the nodes of the ontologies.

The algorithm follows four steps in order to identify the components.

- Identify labels in both nodes.
- Identify concepts in both nodes

- Identify relations among all pairs of labels.
- Identify relations among all pairs of nodes.

The labels are written in an external language, so they have to be translated into an internal language with defined syntax and semantics and used to express the concepts. The language is a logical propositional language where atomic formulas are atomic concepts, written as single words. Complex formulas are obtained combining simple ones.

A translation process takes place and converts the labels into concepts. After this process all the labels have been translated into sentences of the international concept language.

The result is a matrix containing relations existing between any two concepts in labels in the two trees.

In the next step the concepts in nodes are computed as the intersection of the concepts at labels of all the nodes from the root to the node itself.

The result is a matrix containing relations existing between any two concepts in nodes in the two trees.

Once these matrixes are computed, the algorithm implements the semantic matching within the platform S-Match.

According to the developers, very good results are obtained with this method compared to others. It is supposed to be a very promising method in real domains and to have very good running times.

The suggested future work is related to the handling of attributes, the testing of the system and the implementation of some iterative semantic matching.

As we can see after this review, the same approach is followed in this thesis. Matrixes are created with all the possible combinations of labels and nodes (classes and properties) between the two ontologies. The similarity between all the possible pairs is computed, and the best results are chosen.

The difference is that in S-Match, the labels have to be translated into the internal language for the comparison process, while in the polygon method, the external language is the one chosen for the comparisons: the strings are compared directly without any translation.

## 4.4 Method 4: Axiom-Based Ontology Matching.

The objective of this method is to define correspondence between ontologies covering overlapping domains. According to Fürst and Trichet (Fürst & Trichet, 2005) this method is based on the use of axioms and the analysis of natural language expressions, instances and/or taxonomical structures of ontologies.

This new technique is defined in the context of Conceptual Graph model (CG), where axioms are explicitly represented in terms of conceptual graphs. This helps to match the concepts and the relations of two ontologies.

Normally, other ontology matching techniques only consider lightweight ontologies, which are the ones composed by a taxonomy of concepts and a taxonomy of relationships. These kinds of matching do not include axioms, which are the main building blocks for fixing semantic interpolation of concepts and relations.

This new approach for heavyweight ontologies includes all axioms needed to represent the semantics of the domain. It requires explicit representation of the axioms of two ontologies at the conceptual level (and not at the operational level as happens with other languages such as OWL). For this purpose, the ontology conceptual graph language is used. Its graphical syntax constitutes the ideal tool to represent terminological knowledge through concepts, relations and properties.

Every time an ontology is represented, two aspects have to be taken into account: to specify conceptual vocabulary through concepts, relations and instances and to specify semantics with axioms.

The objective of the ontology matching is to discover and evaluate identity links between conceptual primitives (concept relations) of 2 ontologies supposed to be built on connected domains. Then, the similarity coefficient has to be calculated, to indicate how closely two concepts or relations are related.

Finally, we obtain the final result as a table with the similarity concepts placed in columns as well as the relations coefficients.

Some of the conclusions reached while the testing of this tool reveals the effectiveness of the tool for heavyweight ontologies but also the lack of applicability for lightweight ontologies.

This methodology takes into account other aspects that our methodology does not consider: axioms. That makes the method useful in a wider area, which includes heavyweight ontologies. Even if that area is out of our studies, the steps followed are interesting and can be considered for further work.

Apart from the fact that axioms are considered, the handling of the information works in the same way as in the previous methods, and the results are presented in a table in the same way as the polygon method.

## 4.5 Method 5: FCA-Merge.

FCA-Merge is the next method we are going to consider in our revision. According to Stumme and Maedche (Stumme & Maedche, 2001) the method proposes merging ontologies as the solution for knowledge overlaps.

FCA-Merge takes as input a set of documents from which concepts and ontologies are structured. The documents cover all the concepts from the ontologies that are going to be matched during the process. In order to have good results, the ontologies need to have classified instances, but many times that will not happen so the instances have to be obtained from the text documents.

That can be considered to be the first step of the method: the population of the ontologies.

In the second step, a context for each of the ontologies has to be generated. That is done with a lexical analysis to retrieve domain specific information. With these two formal contexts that have been created, we create a new pruned one by merging them.

In the next step, TITANIC algorithm computes the pruned context, and finally in the last phase of the process, the new merged ontology is constructed with a non automatic construction including human iteration based on the pruned context.

It can be considered to be a semi automatic method as it requires background knowledge about the domain.

Comparing it to our method we can easily see that the non automatic computation represents a big disadvantage compared to our automatic method for matching ontologies. Human interaction is required and that makes it less accurate, slower and out of standards.

We can also point out that merging is the issued task, and not matching. That makes the method a bit out of the field we are dealing with, but as it includes new concepts such as the introduction of documents together with the ontologies, it was considered to be helpful to include it in the review.

## 4.6 Method 6: MAFRA.

Maedche, Motik, Silva and Volz (Maedche, Motik, Silva & Volz, 2002) present MAFRA as an interactive, incremental and dynamic framework for mapping distributed ontologies.

This approach introduces the concept of bridge to define mappings between two schemas. Semantic Bridges establish the correspondence between entities from source and target ontology.

A meta-ontology (ontology of ontologies) of bridges is defined during the process.

The matching process can be divided into 5 fundamental phases

1) Lift and normalization: The data which has to be mapped is placed into the same level of representation in order to make both ontologies to have a uniform representation.

2) Similarity: Similarities between the source and the target ontologies are established: lexical similarity, property similarity, bottom-up similarity and top-down similarity.

3) Semantic bridging: Based on the computed similarities, correspondences between the source and the target ontology are found. For each of the entities in one ontology, a similar concept is found in the other ontology, and a bridge is established between both entities. This phase can be subdivided into the next steps:

> 3.1) Pairs of entities to be bridged are found.
>
> 3.2) Property bridging: The matching properties for each concept bridge are specified.
>
> 3.3) Deducing step: Endowing the mapping with bridges for concepts that do not have a specific counterpart target concept.
>
> 3.4) Refinement step: Improvement of the quality of bridges between a source concept and sub concepts of target concepts.
>
> 3.5) Transformation specification step: Associates a transformation procedure to the translation, in a way that source instance may be translated into target instances.

4) Execution. This module actually transforms instances from the source ontology into target ontology by evaluating the semantic bridges already defined.

5) Post-processing. The post-processing component takes the results of the execution module to check and improve the quality of the transformation results.

While these steps are being carried out, four components run along the entire mapping process, interacting with the previous steps.

1) Evolution: This aspect focuses on keeping semantic bridges obtained by the “Semantic Bridge” module, which must be kept in synchrony with the changes in the source and target ontologies.

2) Cooperative Consensus Building: Establishes a consensus on semantic bridges between two communities participating in the mapping process.

3) Domain Constraints and Background Knowledge: Introducing background knowledge and domain constraints can improve the quality of similarity computation and semantic bridging very much.

4) Graphical User Interface.

The introduction of bridges is the main contribution of the MAFRA methodology. This concept of bridge relating two concepts was represented in different ways by other techniques. Most of them introduced the similarity values into tables or matrixes where the correlations between the concepts could be seen. Our methodology in particular relates one concept to others in the matrixes we use to store the data retrieved from the ontologies.

With this method we finish the review of examples included in this thesis as a comparison to the polygon method. Other approaches give us the possibility to realize the weaknesses and the strengths of our implementation and take into account some of the aspects developed by other researchers for future work.

## 4.7 Future work

Once the automatic ontology matching method based in polygons has been implemented weaknesses have to taken into account to propose future changes. These future changes will result into new versions of the system improving the results of the previous versions.

The first aspect which should be taken into account is the relevance of the elements which once matched have similarity 0.0 with other element belonging to the other ontology. In the current implementation when a pair of strings does not have a similarity value higher than 0.6, the value is converted to 0.0, and consequently is not taken into account. These pairs with nothing in common make the ontologies to be less similar to each other and therefore, it has to change the final result.

Currently that data is not taken into account but for future versions, the non-similar pairs have to be relevant for the final result.

Another situation which is not regarded in the system is when a group of classes or subclasses has less than three elements. With less than three vertices a polygon can not be represented so those pairs of matched data, regardless if they have features in common or not, are not represented in any case. For future versions a solution has to be found to take into account the area even is there are only two elements.

Reviewing the comparison made with other methods, we can see that different characteristics are taken into account by other groups of research, such as learning techniques, relationships between concepts, axioms, the introduction of documents together with the ontologies, or bridges.

Each of the methods is based on different principles, while the polygon method is based on the string comparison and the representation of polygons. For future implementations it could be useful to take into account aspects such as the synonyms or the way the relationships are established between concepts.

Synonyms are not included in the current implementation, and they will not be matched together as they are not necessary similar strings. When two ontologies are edited by different communities, synonyms can appear and make ontologies more similar.

The relationships between the classes are another characteristic which should be taken into account. In the described implementation, a property is matched to another one taking into account only its name as a string, but not its domain (class from which it departs) and its range (class to which it arrives). The similarity is influence by this fact and therefore, it should be taken into account when calculating the matching between two ontologies.

Another fact which has been identified is that the result of the comparison is different depending on which ontology is chosen as the standard one. A study about the effect of the ontology chosen as the standard one has to be made in order to clear out how that fact can affect to the result of the matching.

# 5 Results

The purpose and objective of the thesis, described in section 1.2, expresses the aim of the author to describe the implementation of a new method for automatic ontology matching according to the ontology matching algorithm developed by Feiyu Lin

The previous sections have explained in detail the steps of the implementation and the tools needed in each of the phases. Upon completing the implementation, a numerical result is reached corresponding to the similarity of the two ontologies proposed for the example. This similarity value is the purpose of the implementation and all the decisions taken during the process focus on the accuracy of that result.

Once the system is described and the result is reached, the first objective is fulfilled.

The second objective deals with the comparison between the method proposed and other methods implemented for ontology matching. As a result of the comparison, conclusions have been reached which will lay the foundations for future work.

We can conclude saying that the results of the thesis are the implementation of a new system for automatic ontology matching and the theoretical study comparing that method to other existing methods.

# 6 References

Carroll, Jeremy J.; Reynolds, Dave; Dickinson, Ian; Seaborne, Andy; Dollin, Chris; Wilkinson, Kevin (2004) *Jena: Implementing the Semantic Web Recommendations.*
HP Labs, Bristol, UK, HP Labs, Palo Alto, CA. USA.

Cohen, WilliamW.; Ravikumar, Pradeep; Fienberg, Stephen E. (2003) *A Comparison of String Distance Metrics for Name-Matching Tasks.*
Carnegie Mellon University.

Doan, AnHai; Halevy, Alon Y. (2005) *Semantic Integration Research in the Database Community: A Brief Survey.*
University of Illinois. University of Washington. USA.

Doan, AnHai; Madhavan, Jayant; Domingos, Pedro; Halevy, Alon (2003) *Ontology Matching: A Machine Learning Approach.*
University of Illinois, Urbana-Champaign, IL, U.S.A, University of Washington, Seattle, WA, U.S.A.

Ferrara, Alfio (2004) *Methods and Techniques for Ontology Matching and Evolution in Open Distributed Systems.*
Università degli Studi di Milano, Italy.

Fürst, Frédéric; Trichet, Francky (2005) *Axiom-based ontology matching: a method and an experiment.*
Laboratoire d'informatique de Nantes- Atlantique, Université de Nantes. France.

Giunchiglia, Fausto; Shvaiko, Pavel; Yatskevich, Mikalai (2004) *S-Match: An algorithm and an implementation of semantic matching.*
Department of information and communication technology, University of Trento, Trento, Italy.

Gruber, T. R. (1992) *A translation approach to portable ontology specifications.*
Stanford Knowledge Systems Laboratory, Stanford University, Palo Alto, CA. USA.

Gruber, T. R. (1993) *Toward Principles for the Design of Ontologies used for Knowledge Sharing.*
Stanford Knowledge Systems Laboratory, Stanford University, Palo Alto, CA. USA.
Kalfoglou, Yannis; Schorlemmer, Marco (2003) *Ontology mapping: the state of the art.*
Dep. of Electronics and Computer Science, University of Southampton, UK. School of Informatics, University of Edinburgh, UK. Escola Superior de Tecnologies d'Informació i Comunicació, Universitat Internacional de Catalunya, Spain.

Lin, Feiyu; Sandkuhl, Kurt (2007) POLYGON-BASED SIMILARITY AGGREGATION Towards a Contribution to Ontology Matching, Submitted to 3rd International Conference on Web Information Systems and Technologies (WEBIST07), Barcelona, Spain, March, 2007.

Maedche, Alexander; Motik, Boris; Silva, Nuno; Volz, Raphael (2002) *MAFRA— A MApping FRAmework for Distributed Ontologies.*
Forschungszentrum Informatik at the Univ. Karlsruhe, Karlsruhe, Germany. ISEP Instituto Superior de Engenharia, Instituto Politecnico do Porto, Portugal.

Noy, Natalya F.; McGuinness, Deborah L. (2001) *Ontology Development 101: A Guide to Creating Your First Ontology.*
Stanford University, Stanford, CA, USA.

Noy, Natalya F.; Musen, Mark A. (2002) *Anchor-PROMPT: Using non-local context for semantic matching.*
Stanford University, Stanford, CA, USA.

Rahm, E.; Bernstein, P. (2001) A survey of approaches to automatic schema matching. VLDB Journal, 10(4):334–350, 2001.

Rajagopal, Hari (2005) *Jena: A Java API for ontology management.*
Colorado software submit, IBM Corporation.

Shvaiko, Pavel; Euzenat, Jérôme (2004) *A Survey of Schema-based Matching Approaches.*
University of Trento, Povo, Trento, Italy. INRIA, Rhône-Alpes, France.

Stumme, Gerd; Maedche, Alexander (2001) *FCA-MERGE: Bottom-Up Merging of Ontologies.*
Institute AIFB, University of Karlsruhe, Karlsruhe, Germany.

HP Labs Semantic Web Research http://www.hpl.hp.com/semweb/ (Acc. 12/15/2006)

IEEE Computer Society. http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/wi/2003/1932/00/1932toc.xml&DOI=10.1109/WI.2003.1241177 (Acc. 12/15/2006)

Index Java Classes http://jena.sourceforge.net/ontology/index.html (Acc. 12/15/2006)

Jena 2 Ontology API http://jena.sourceforge.net/ontology/index.html (Acc. 12/15/2006)

Jena 2.0 Library Plugin. http://owl-eclipse.projects.semwebcentral.org/Jena/ (Acc. 12/15/2006)

Jena SourceForge. http://jena.sourceforge.net/src-examples/jena/examples/rdf/Tutorial01.java (Acc. 12/15/2006)

Jena Tutorial http://jena.sourceforge.net/tutorial/index.html (Acc. 12/15/2006)

Jena, a semantic web framework for java. http://www.try.idv.tw/static-resources/jena/doc/ (Acc. 12/15/2006)

JmatLink SourceForge. http://jmatlink.sourceforge.net/ (Acc. 12/15/2006)

Matlab Tutorial. http://www.cyclismo.org/tutorial/matlab/ (Acc. 12/15/2006)

Ontology Matching. http://www.ontologymatching.org/publications.html (Acc. 12/15/2006)

OWL Web ontology Language Overview http://www.w3.org/TR/owl-features/ (Acc. 12/15/2006)

Protégé Home page. http://protege.stanford.edu/overview/protege-owl.html (Acc. 12/15/2006)

RDF Schema http://www.w3.org/TR/2004/REC-owl-features-20040210/#ref-rdf-schema (Acc. 12/15/2006)

Second String Project Page. http://www.cyclismo.org/tutorial/matlab/ (Acc. 12/15/2006)

SecondString SourceForge. http://sourceforge.net/project/showfiles.php?group_id=75872 (Acc. 12/15/2006)

Unicode Normalization.
http://msdn2.microsoft.com/en-us/library/ms776393.aspx (Acc. 12/15/2006)

# 7 Appendix

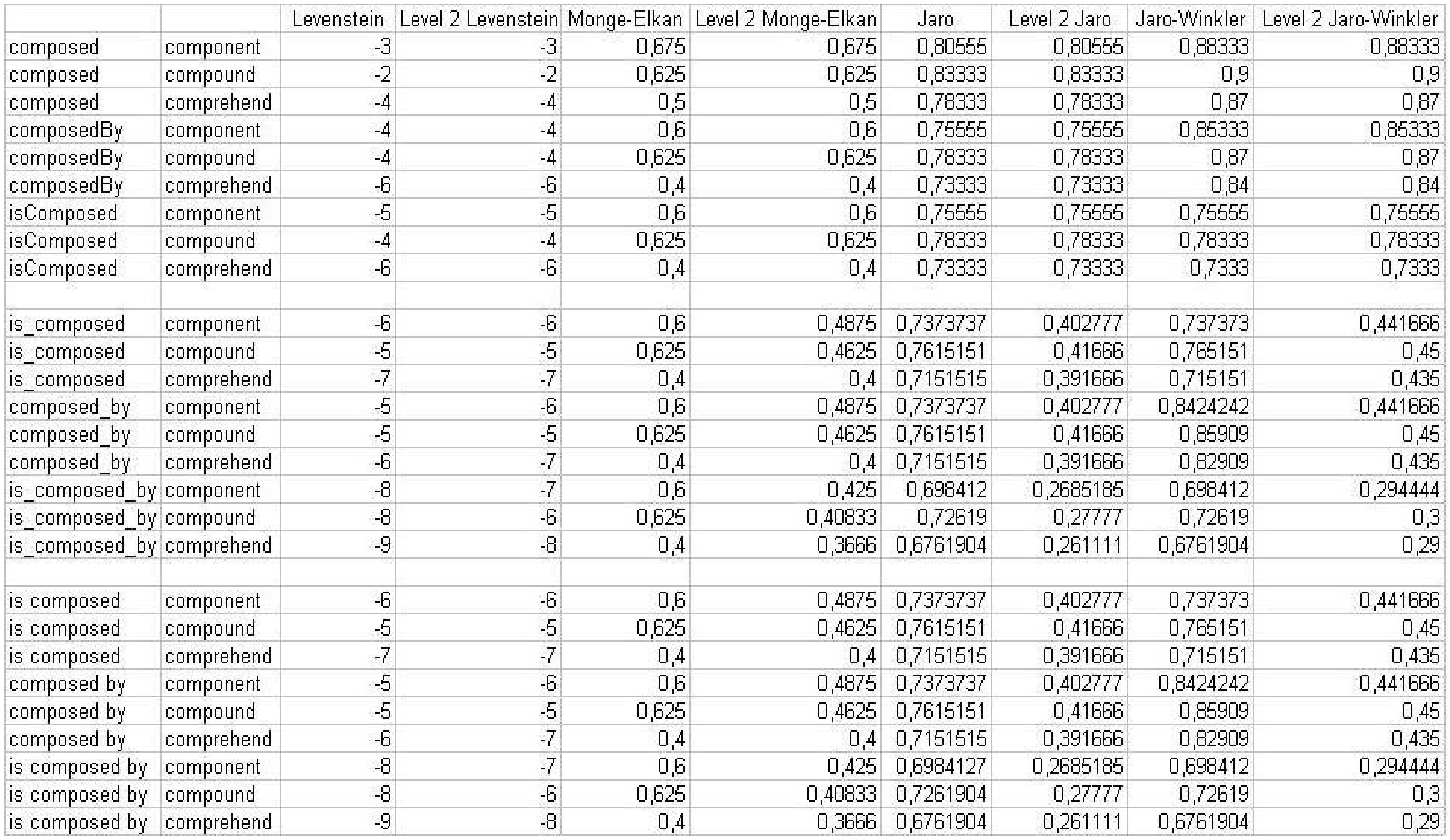
Appendix 1: Table: Comparison of Edit-distance metrics for very similar pairs of strings.

| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| composed | Composed | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| id | ID | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| has | Has | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| isComposed | Composed | -2 | -2 | 1 | 1 | 0,808333 | 0,808333 | 0,808333 | 0,808333 |
| ComposedBy | Composed | -2 | -2 | 1 | 1 | 0,93333 | 0,93333 | 0,96 | 0,96 |
| iscomposed | Composed | -2 | -2 | 1 | 1 | 0,808333 | 0,808333 | 0,808333 | 0,808333 |
| composedby | Composed | -2 | -2 | 1 | 1 | 0,93333 | 0,93333 | 0,96 | 0,96 |
| | | | | | | | | | |
| is_composed | composed | -3 | -3,5 | 1 | 0,9 | 0,7840909 | 0,5 | 0,784090909 | 0,5 |
| is_compose | compose | -3 | -3 | 1 | 0,9 | 0,9 | 0,5 | 0,9 | 0,5 |
| is_compos | compos | -3 | -2,5 | 1 | 0,9 | 0,888888 | 0,5 | 0,88888 | 0,5 |
| is_compo | compo | -3 | -2,5 | 1 | 0,65 | 0 | 0,5 | 0 | 0,5 |
| is_comp | comp | -3 | -2 | 1 | 0,65 | 0 | 0,5 | 0 | 0,5 |
| is_com | com | -3 | -1,5 | 1 | 0,65 | 0 | 0,5 | 0 | 0,5 |
| | | | | | | | | | |
| composed_by | composed | -3 | -4 | 1 | 0,65 | 0,9090909 | 0,5 | 0,94545454 | 0,5 |
| is_composed_by | isComposedBy | -2 | -8 | 0,83333 | 1 | 0,952381 | 0,49537037 | 0,9619047 | 0,513888889 |
| id_no | idNo | -1 | -2 | 0,75 | 1 | 0,93333 | 0,4166666 | 0,9466666 | 0,4333333 |
| lives_with | livesWith | -1 | -4,5 | 0,88888 | 1 | 0,966666 | 0,652777 | 0,98 | 0,6824074 |
| | | | | | | | | | |
| is composed | composed | -3 | -3,5 | 1 | 0,9 | 0,7840909 | 0,5 | 0,784090909 | 0,5 |
| composed by | composed | -3 | -4 | 1 | 0,65 | 0,9090909 | 0,5 | 0,9454545 | 0,5 |
| is composed | Composed | -3 | -3,5 | 1 | 0,9 | 0,7840909 | 0,5 | 0,784090909 | 0,5 |
| composed by | Composed | -3 | -4 | 1 | 0,65 | 0,9090909 | 0,5 | 0,9454545 | 0,5 |
| | | | | | | | | | |
| is composed | is_composed | -1 | 0 | 0,854545 | 1 | 0,9393939 | 1 | 0,95151515 | 1 |
| composed by | composed_by | -1 | 0 | 0,854545 | 1 | 0,9393939 | 1 | 0,963636363 | 1 |
| is composed by | is_composed_by | -2 | 0 | 0,7714285 | 1 | 0,9047619 | 1 | 0,923809523 | 1 |

Appendix 2: Table: Comparison of Edit-distance metrics for very similar pairs of strings, changing the order of the strings.

| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| Composed | composed | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| ID | id | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Has | has | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| Composed | isComposed | -2 | -2 | 1 | 1 | 0,808333 | 0,808333 | 0,808333 | 0,808333 |
| Composed | ComposedBy | -2 | -2 | 1 | 1 | 0,93333 | 0,93333 | 0,96 | 0,96 |
| Composed | iscomposed | -2 | -2 | 1 | 1 | 0,808333 | 0,808333 | 0,808333 | 0,808333 |
| Composed | composedby | -2 | -2 | 1 | 1 | 0,93333 | 0,93333 | 0,96 | 0,96 |
| | | | | | | | | | |
| composed | is_composed | -3 | 0 | 1 | 1 | 0,7840909 | 1 | 0,7840909 | 1 |
| compose | is_compose | -3 | 0 | 1 | 1 | 0,9 | 1 | 0,9 | 1 |
| compos | is_compos | -3 | 0 | 1 | 1 | 0,888888 | 1 | 0,888888 | 1 |
| compo | is_compo | -3 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| comp | is_comp | -3 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| com | is_com | -3 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | | | |
| composed | composed_by | -3 | 0 | 1 | 1 | 0,9090909 | 1 | 0,945454 | 1 |
| isComposedBy | is_composed_by | -2 | -4 | 0,666666 | 1 | 0,952381 | 0,7638888 | 0,961904 | 0,7777777 |
| idNo | id_no | -1 | -2 | 0,5 | 1 | 0,93333 | 0,8333333 | 0,946666 | 0,866666 |
| livesWith | lives_with | -1 | -4 | 0,55555 | 1 | 0,966666 | 0,8518518 | 0,98 | 0,9111111 |
| | | | | | | | | | |
| composed | is composed | -3 | 0 | 1 | 1 | 0,7840909 | 1 | 0,7840909 | 1 |
| composed | composed by | -3 | 0 | 1 | 1 | 0,9090909 | 1 | 0,9454545 | 1 |
| Composed | is composed | -3 | 0 | 1 | 1 | 0,7840909 | 1 | 0,7840909 | 1 |
| Composed | composed by | -3 | 0 | 1 | 1 | 0,9090909 | 1 | 0,9454545 | 1 |
| | | | | | | | | | |
| is_composed | is composed | -1 | 0 | 0,8545454 | 1 | 0,9393939 | 1 | 0,9515151 | 1 |
| composed_by | composed by | -1 | 0 | 0,8545454 | 1 | 0,9393939 | 1 | 0,9636363 | 1 |
| is_composed_by | is composed by | -2 | 0 | 0,771428 | 1 | 0,9047619 | 1 | 0,923809 | 1 |

Appendix 3: Table: Comparison of Edit-distance metrics for pairs of strings with some similarities.

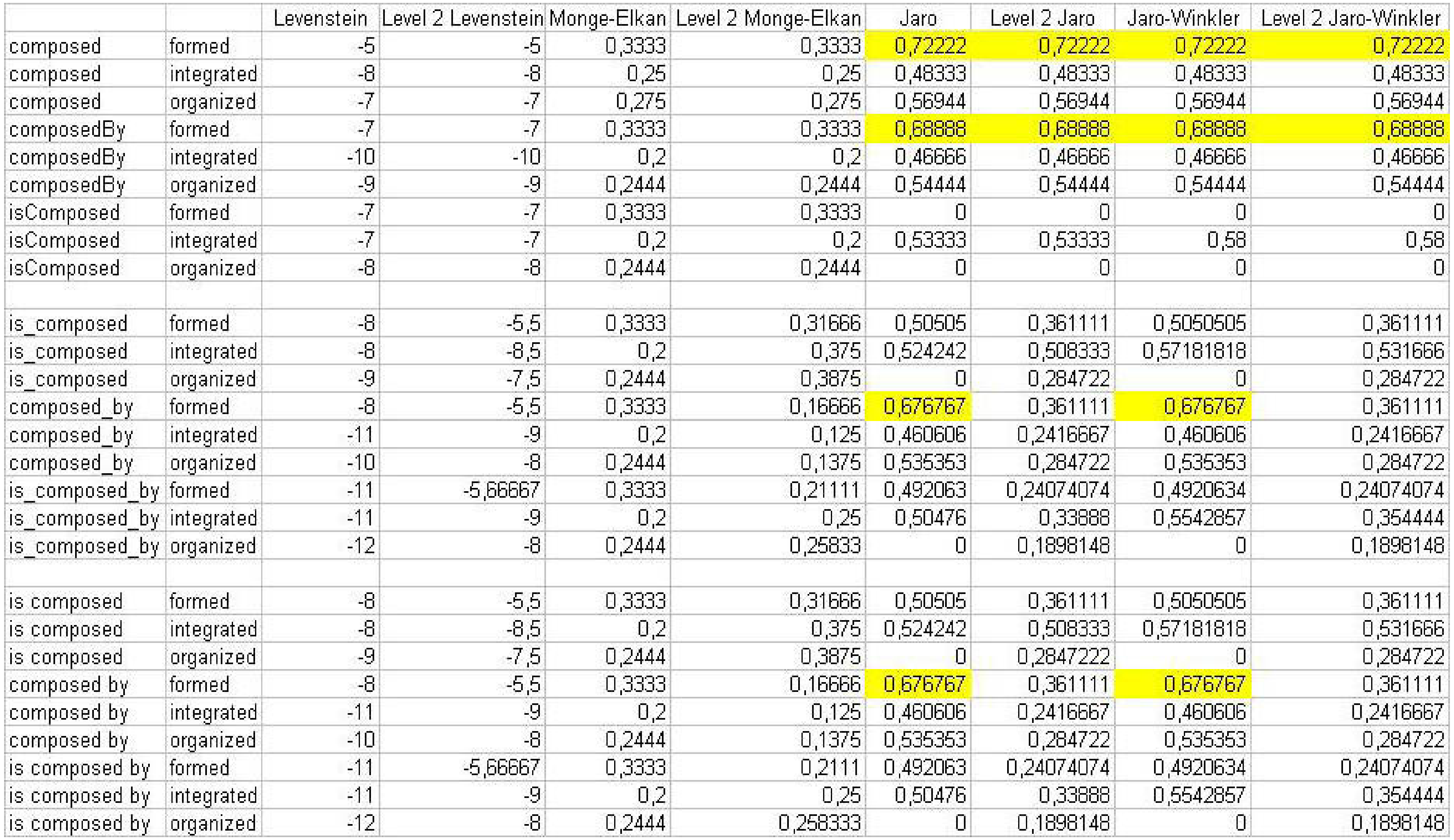


| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| composed | component | -3 | -3 | 0,675 | 0,675 | 0,80555 | 0,80555 | 0,88333 | 0,88333 |
| composed | compound | -2 | -2 | 0,625 | 0,625 | 0,83333 | 0,83333 | 0,9 | 0,9 |
| composed | comprehend | -4 | -4 | 0,5 | 0,5 | 0,78333 | 0,78333 | 0,87 | 0,87 |
| composedBy | component | -4 | -4 | 0,6 | 0,6 | 0,75555 | 0,75555 | 0,85333 | 0,85333 |
| composedBy | compound | -4 | -4 | 0,625 | 0,625 | 0,78333 | 0,78333 | 0,87 | 0,87 |
| composedBy | comprehend | -6 | -6 | 0,4 | 0,4 | 0,73333 | 0,73333 | 0,84 | 0,84 |
| isComposed | component | -5 | -5 | 0,6 | 0,6 | 0,75555 | 0,75555 | 0,75555 | 0,75555 |
| isComposed | compound | -4 | -4 | 0,625 | 0,625 | 0,78333 | 0,78333 | 0,78333 | 0,78333 |
| isComposed | comprehend | -6 | -6 | 0,4 | 0,4 | 0,73333 | 0,73333 | 0,7333 | 0,7333 |
| | | | | | | | | | |
| is_composed | component | -6 | -6 | 0,6 | 0,4875 | 0,7373737 | 0,402777 | 0,737373 | 0,441666 |
| is_composed | compound | -5 | -5 | 0,625 | 0,4625 | 0,7615151 | 0,41666 | 0,765151 | 0,45 |
| is_composed | comprehend | -7 | -7 | 0,4 | 0,4 | 0,7151515 | 0,391666 | 0,715151 | 0,435 |
| composed_by | component | -5 | -6 | 0,6 | 0,4875 | 0,7373737 | 0,402777 | 0,8424242 | 0,441666 |
| composed_by | compound | -5 | -5 | 0,625 | 0,4625 | 0,7615151 | 0,41666 | 0,85909 | 0,45 |
| composed_by | comprehend | -6 | -7 | 0,4 | 0,4 | 0,7151515 | 0,391666 | 0,82909 | 0,435 |
| is_composed_by | component | -8 | -7 | 0,6 | 0,425 | 0,698412 | 0,2685185 | 0,698412 | 0,294444 |
| is_composed_by | compound | -8 | -6 | 0,625 | 0,40833 | 0,72619 | 0,27777 | 0,72619 | 0,3 |
| is_composed_by | comprehend | -9 | -8 | 0,4 | 0,3666 | 0,6761904 | 0,261111 | 0,6761904 | 0,29 |
| | | | | | | | | | |
| is composed | component | -6 | -6 | 0,6 | 0,4875 | 0,7373737 | 0,402777 | 0,737373 | 0,441666 |
| is composed | compound | -5 | -5 | 0,625 | 0,4625 | 0,7615151 | 0,41666 | 0,765151 | 0,45 |
| is composed | comprehend | -7 | -7 | 0,4 | 0,4 | 0,7151515 | 0,391666 | 0,715151 | 0,435 |
| composed by | component | -5 | -6 | 0,6 | 0,4875 | 0,7373737 | 0,402777 | 0,8424242 | 0,441666 |
| composed by | compound | -5 | -5 | 0,625 | 0,4625 | 0,7615151 | 0,41666 | 0,85909 | 0,45 |
| composed by | comprehend | -6 | -7 | 0,4 | 0,4 | 0,7151515 | 0,391666 | 0,82909 | 0,435 |
| is composed by | component | -8 | -7 | 0,6 | 0,425 | 0,6984127 | 0,2685185 | 0,698412 | 0,294444 |
| is composed by | compound | -8 | -6 | 0,625 | 0,40833 | 0,7261904 | 0,27777 | 0,72619 | 0,3 |
| is composed by | comprehend | -9 | -8 | 0,4 | 0,3666 | 0,6761904 | 0,261111 | 0,6761904 | 0,29 |

Appendix 4: Table: Comparison of Edit-distance metrics for pairs of strings with some similarities, changing the order of the strings.

| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| component | composed | -3 | -3 | 0,675 | 0,675 | 0,80555 | 0,80555 | 0,88333 | 0,88333 |
| compound | composed | -2 | -2 | 0,625 | 0,625 | 0,83333 | 0,83333 | 0,9 | 0,9 |
| comprehend | composed | -4 | -4 | 0,5 | 0,5 | 0,78333 | 0,78333 | 0,87 | 0,87 |
| component | composedBy | -4 | -4 | 0,6 | 0,6 | 0,75555 | 0,75555 | 0,85333 | 0,85333 |
| compound | composedBy | -4 | -4 | 0,625 | 0,625 | 0,78333 | 0,78333 | 0,87 | 0,87 |
| comprehend | composedBy | -6 | -6 | 0,4 | 0,4 | 0,73333 | 0,73333 | 0,84 | 0,84 |
| component | isComposed | -5 | -5 | 0,6 | 0,6 | 0,75555 | 0,75555 | 0,75555 | 0,75555 |
| compound | isComposed | -4 | -4 | 0,625 | 0,625 | 0,78333 | 0,78333 | 0,78333 | 0,78333 |
| comprehend | isComposed | -6 | -6 | 0,4 | 0,4 | 0,73333 | 0,73333 | 0,7333 | 0,7333 |
| | | | | | | | | | |
| component | is_composed | -6 | -3 | 0,6 | 0,675 | 0,73737 | 0,7373737 | 0,737373 | 0,88333 |
| compound | is_composed | -5 | -2 | 0,625 | 0,625 | 0,76152 | 0,76515151 | 0,765151 | 0,9 |
| comprehend | is_composed | -7 | -4 | 0,4 | 0,5 | 0,71515 | 0,7151515 | 0,715151 | 0,87 |
| component | composed_by | -5 | -3 | 0,6 | 0,675 | 0,73737 | 0,7373737 | 0,8424242 | 0,88333 |
| compound | composed_by | -5 | -2 | 0,625 | 0,625 | 0,76152 | 0,76515151 | 0,85909 | 0,9 |
| comprehend | composed_by | -6 | -4 | 0,4 | 0,5 | 0,71515 | 0,7151515 | 0,82909 | 0,87 |
| component | is_composed_by | -8 | -3 | 0,6 | 0,675 | 0,69841 | 0,6984126 | 0,698412 | 0,88333 |
| compound | is_composed_by | -8 | -2 | 0,625 | 0,625 | 0,72619 | 0,7261904 | 0,72619 | 0,9 |
| comprehend | is_composed_by | -9 | -4 | 0,4 | 0,5 | 0,67619 | 0,6761904 | 0,6761904 | 0,87 |
| | | | | | | | | | |
| component | is composed | -6 | -3 | 0,6 | 0,675 | 0,73737 | 0,73737373 | 0,737373 | 0,88333 |
| compound | is composed | -5 | -2 | 0,625 | 0,625 | 0,76152 | 0,76515151 | 0,765151 | 0,9 |
| comprehend | is composed | -7 | -4 | 0,4 | 0,5 | 0,71515 | 0,7151515 | 0,715151 | 0,87 |
| component | composed by | -5 | -3 | 0,6 | 0,675 | 0,73737 | 0,7373737 | 0,8424242 | 0,88333 |
| compound | composed by | -5 | -2 | 0,625 | 0,625 | 0,76152 | 0,76515151 | 0,85909 | 0,9 |
| comprehend | composed by | -6 | -4 | 0,4 | 0,5 | 0,71515 | 0,7151515 | 0,82909 | 0,87 |
| component | is composed by | -8 | -3 | 0,6 | 0,675 | 0,69841 | 0,6984126 | 0,698412 | 0,88333 |
| compound | is composed by | -8 | -2 | 0,625 | 0,625 | 0,72619 | 0,7261904 | 0,72619 | 0,9 |
| comprehend | is composed by | -9 | -4 | 0,4 | 0,5 | 0,67619 | 0,6761904 | 0,6761904 | 0,87 |

Appendix 5: Table: Comparison of Edit-distance metrics for pairs of strings with nothing in common.



| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| composed | formed | -5 | -5 | 0,3333 | 0,3333 | 0,72222 | 0,72222 | 0,72222 | 0,72222 |
| composed | integrated | -8 | -8 | 0,25 | 0,25 | 0,48333 | 0,48333 | 0,48333 | 0,48333 |
| composed | organized | -7 | -7 | 0,275 | 0,275 | 0,56944 | 0,56944 | 0,56944 | 0,56944 |
| composedBy | formed | -7 | -7 | 0,3333 | 0,3333 | 0,68888 | 0,68888 | 0,68888 | 0,68888 |
| composedBy | integrated | -10 | -10 | 0,2 | 0,2 | 0,46666 | 0,46666 | 0,46666 | 0,46666 |
| composedBy | organized | -9 | -9 | 0,2444 | 0,2444 | 0,54444 | 0,54444 | 0,54444 | 0,54444 |
| isComposed | formed | -7 | -7 | 0,3333 | 0,3333 | 0 | 0 | 0 | 0 |
| isComposed | integrated | -7 | -7 | 0,2 | 0,2 | 0,53333 | 0,53333 | 0,58 | 0,58 |
| isComposed | organized | -8 | -8 | 0,2444 | 0,2444 | 0 | 0 | 0 | 0 |
| | | | | | | | | | |
| is_composed | formed | -8 | -5,5 | 0,3333 | 0,31666 | 0,50505 | 0,361111 | 0,5050505 | 0,361111 |
| is_composed | integrated | -8 | -8,5 | 0,2 | 0,375 | 0,524242 | 0,508333 | 0,57181818 | 0,531666 |
| is_composed | organized | -9 | -7,5 | 0,2444 | 0,3875 | 0 | 0,284722 | 0 | 0,284722 |
| composed_by | formed | -8 | -5,5 | 0,3333 | 0,16666 | 0,676767 | 0,361111 | 0,676767 | 0,361111 |
| composed_by | integrated | -11 | -9 | 0,2 | 0,125 | 0,460606 | 0,2416667 | 0,460606 | 0,2416667 |
| composed_by | organized | -10 | -8 | 0,2444 | 0,1375 | 0,535353 | 0,284722 | 0,535353 | 0,284722 |
| is_composed_by | formed | -11 | -5,66667 | 0,3333 | 0,21111 | 0,492063 | 0,24074074 | 0,4920634 | 0,24074074 |
| is_composed_by | integrated | -11 | -9 | 0,2 | 0,25 | 0,50476 | 0,33888 | 0,5542857 | 0,354444 |
| is_composed_by | organized | -12 | -8 | 0,2444 | 0,25833 | 0 | 0,1898148 | 0 | 0,1898148 |
| | | | | | | | | | |
| is composed | formed | -8 | -5,5 | 0,3333 | 0,31666 | 0,50505 | 0,361111 | 0,5050505 | 0,361111 |
| is composed | integrated | -8 | -8,5 | 0,2 | 0,375 | 0,524242 | 0,508333 | 0,57181818 | 0,531666 |
| is composed | organized | -9 | -7,5 | 0,2444 | 0,3875 | 0 | 0,2847222 | 0 | 0,284722 |
| composed by | formed | -8 | -5,5 | 0,3333 | 0,16666 | 0,676767 | 0,361111 | 0,676767 | 0,361111 |
| composed by | integrated | -11 | -9 | 0,2 | 0,125 | 0,460606 | 0,2416667 | 0,460606 | 0,2416667 |
| composed by | organized | -10 | -8 | 0,2444 | 0,1375 | 0,535353 | 0,284722 | 0,535353 | 0,284722 |
| is composed by | formed | -11 | -5,66667 | 0,3333 | 0,2111 | 0,492063 | 0,24074074 | 0,4920634 | 0,24074074 |
| is composed by | integrated | -11 | -9 | 0,2 | 0,25 | 0,50476 | 0,33888 | 0,5542857 | 0,354444 |
| is composed by | organized | -12 | -8 | 0,2444 | 0,258333 | 0 | 0,1898148 | 0 | 0,1898148 |

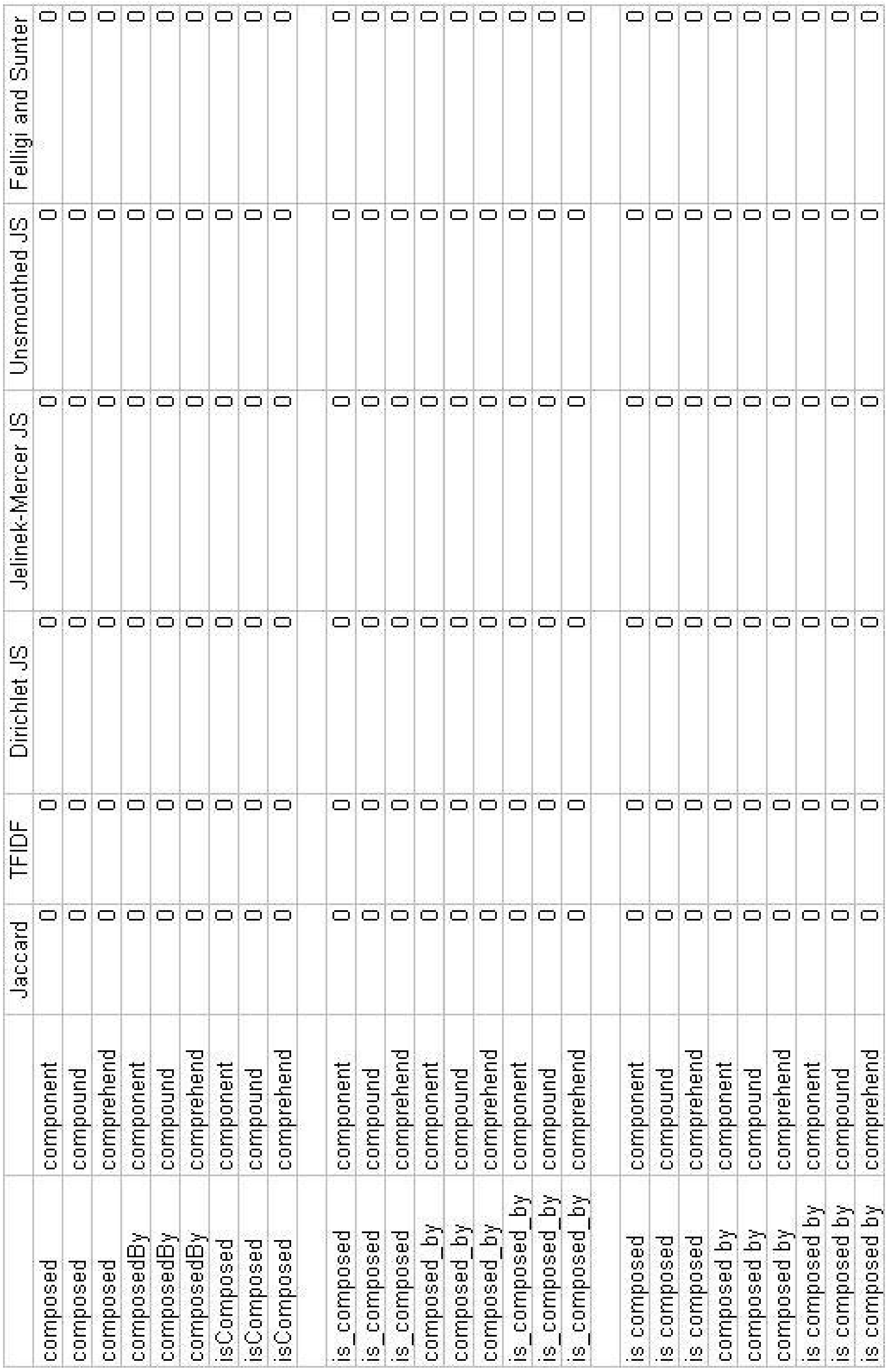
Appendix 6: Table: Comparison of Edit-distance metrics for pairs of strings with nothing in common, changing the order of the strings.

| | | Levenstein | Level 2 Levenstein | Monge-Elkan | Level 2 Monge-Elkan | Jaro | Level 2 Jaro | Jaro-Winkler | Level 2 Jaro-Winkler |
|---|---|---|---|---|---|---|---|---|---|
| formed | composed | -5 | -5 | 0,3333 | 0,3333 | 0,72222 | 0,72222 | 0,72222 | 0,72222 |
| integrated | composed | -8 | -8 | 0,25 | 0,25 | 0,48333 | 0,48333 | 0,48333 | 0,48333 |
| organized | composed | -7 | -7 | 0,25 | 0,25 | 0,56944 | 0,56944 | 0,56944 | 0,56944 |
| formed | composedBy | -7 | -7 | 0,3333 | 0,3333 | 0,68888 | 0,68888 | 0,68888 | 0,68888 |
| integrated | composedBy | -10 | -10 | 0,2 | 0,2 | 0,46666 | 0,46666 | 0,46666 | 0,46666 |
| organized | composedBy | -9 | -9 | 0,2222 | 0,2222 | 0,54444 | 0,54444 | 0,54444 | 0,54444 |
| formed | isComposed | -7 | -7 | 0,3333 | 0,3333 | 0 | 0 | 0 | 0 |
| integrated | isComposed | -7 | -7 | 0,2 | 0,2 | 0,53333 | 0,53333 | 0,58 | 0,58 |
| organized | isComposed | -8 | -8 | 0,2222 | 0,22222 | 0 | 0 | 0 | 0 |
| | | | | | | | | | |
| formed | is_composed | -8 | -5 | 0,3333 | 0,33333 | 0,50505 | 0,722222 | 0,50505 | 0,72222 |
| integrated | is_composed | -8 | -8 | 0,2 | 0,5 | 0,52424 | 0,533333 | 0,57181818 | 0,58 |
| organized | is_composed | -9 | -7 | 0,2222 | 0,5 | 0 | 0,5694444 | 0 | 0,56944 |
| formed | composed_by | -8 | -5 | 0,3333 | 0,33333 | 0,67677 | 0,722222 | 0,676767 | 0,72222 |
| integrated | composed_by | -11 | -8 | 0,2 | 0,25 | 0,46061 | 0,483333 | 0,460606 | 0,48333 |
| organized | composed_by | -10 | -7 | 0,2222 | 0,25 | 0,53535 | 0,569444 | 0,535353 | 0,569444 |
| formed | is_composed_by | -11 | -5 | 0,3333 | 0,33333 | 0,49206 | 0,722222 | 0,49206349 | 0,72222 |
| integrated | is_composed_by | -11 | -8 | 0,2 | 0,5 | 0,50476 | 0,533333 | 0,5542857 | 0,58 |
| organized | is_composed_by | -12 | -7 | 0,2222 | 0,5 | 0 | 0,569444 | 0 | 0,56944 |
| | | | | | | | | | |
| formed | is composed | -8 | -5 | 0,3333 | 0,33333 | 0,50505 | 0,722222 | 0,50505 | 0,72222 |
| integrated | is composed | -8 | -8 | 0,2 | 0,5 | 0,52424 | 0,533333 | 0,57181818 | 0,58 |
| organized | is composed | -9 | -7 | 0,2222 | 0,3 | 0 | 0,5694444 | 0 | 0,56944 |
| formed | composed by | -8 | -5 | 0,3333 | 0,33333 | 0,67677 | 0,722222 | 0,676767 | 0,72222 |
| integrated | composed by | -11 | -8 | 0,2 | 0,5 | 0,46061 | 0,483333 | 0,460606 | 0,48333 |
| organized | composed by | -10 | -7 | 0,2222 | 0,5 | 0,53535 | 0,569444 | 0,535353 | 0,569444 |
| formed | is composed by | -11 | -5 | 0,3333 | 0,333333 | 0,49206 | 0,722222 | 0,49206349 | 0,72222 |
| integrated | is composed by | -11 | -8 | 0,2 | 0,5 | 0,50476 | 0,533333 | 0,5542857 | 0,58 |
| organized | is composed by | -12 | -7 | 0,2222 | 0,5 | 0 | 0,569444 | 0 | 0,56944 |

Appendix 7: Table: Comparison of Token-based distance metrics for very similar pairs of strings.

| | | Jaccard | TFIDF | Dirichlet JS | Jelinek-Mercer JS | Unsmoothed JS | Fellegi and Sunter |
|---|---|---|---|---|---|---|---|
| Composed | composed | 1 | 1 | Nan | Nan | 1 | 4,605170180 |
| ID | id | 1 | 1 | Nan | Nan | 1 | 4,605170180 |
| Has | has | 1 | 1 | Nan | Nan | 1 | 4,605170180 |
| Composed | isComposed | 0 | 0 | 0 | 0 | 0 | 0 |
| Composed | ComposedBy | 0 | 0 | 0 | 0 | 0 | 0 |
| Composed | iscomposed | 0 | 0 | 0 | 0 | 0 | 0 |
| Composed | composedby | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| composed | is_composed | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| compose | is_compose | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| compos | is_compos | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| compo | is_compo | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| comp | is_comp | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| com | is_com | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| | | | | | | | |
| composed | composed_by | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| isComposedBy | is_composed_by | 0 | 0 | 0 | 0 | 0 | 0 |
| idNo | id_no | 0 | 0 | 0 | 0 | 0 | 0 |
| livesWith | lives_with | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| composed | is composed | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| composed | composed by | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| Composed | is composed | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| Composed | composed by | 0,5 | 0,707107 | Nan | Nan | 0,688721876 | 3,917035547 |
| | | | | | | | |
| is_composed | is composed | 1 | 1 | Nan | Nan | 1 | 6,467776390 |
| composed_by | composed by | 1 | 1 | Nan | Nan | 1 | 6,467776390 |
| is_composed_by | is composed by | 1 | 1 | Nan | Nan | 1 | 7,343430833 |

Appendix 8: Table: Comparison of Token-based distance metrics for pairs of strings with some similarities.

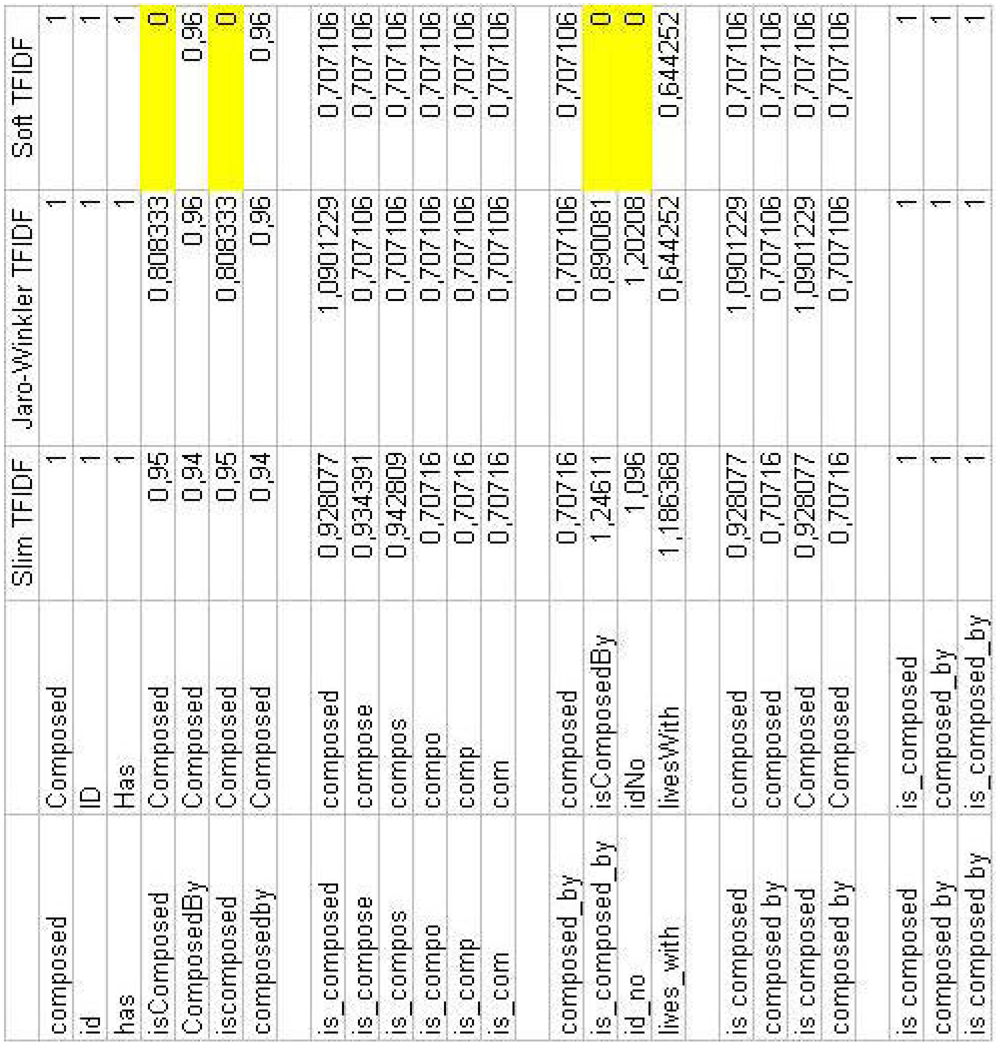


| | | Jaccard | TFIDF | Dirichlet JS | Jelinek-Mercer JS | Unsmoothed JS | Felligi and Sunter |
|---|---|---|---|---|---|---|---|
| composed | component | 0 | 0 | 0 | 0 | 0 | 0 |
| composed | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| composed | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | component | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | component | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| is_composed | component | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | component | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | component | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| is composed | component | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | component | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | component | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | compound | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | comprehend | 0 | 0 | 0 | 0 | 0 | 0 |

Appendix 9: Table: Comparison of Token-based distance metrics for pairs of strings with nothing in common.

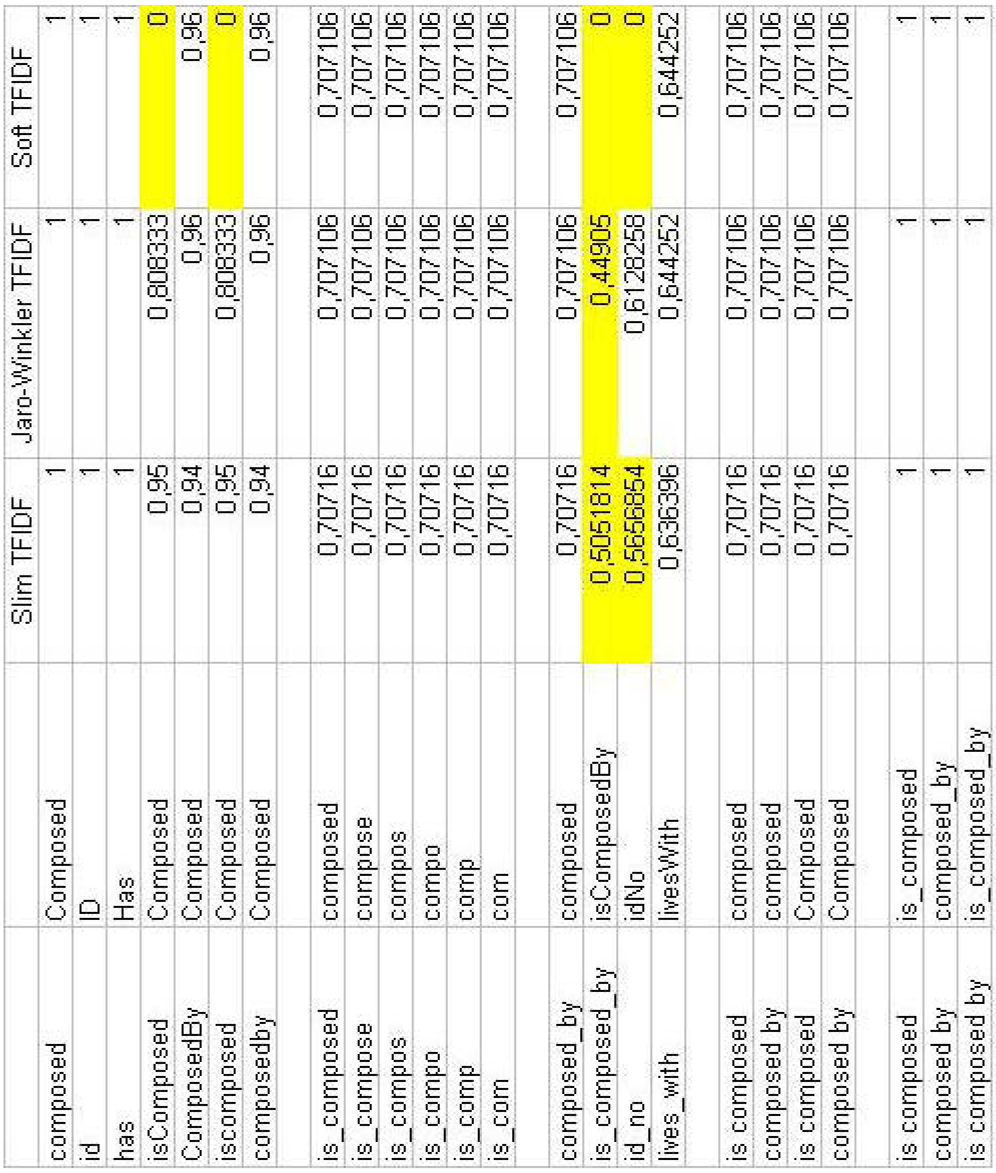
| | | Jaccard | TFIDF | Dirichlet JS | Jelinek-Mercer JS | Unsmoothed JS | Felligi and Sunter |
|---|---|---|---|---|---|---|---|
| composed | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| composed | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| composed | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| composedBy | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| isComposed | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| is_composed | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| composed_by | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| is_composed_by | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | |
| is composed | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| composed by | organized | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | formed | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | integrated | 0 | 0 | 0 | 0 | 0 | 0 |
| is composed by | organized | 0 | 0 | 0 | 0 | 0 | 0 |

Appendix 10: Table: Comparison of Hybrid methods for very similar pairs of strings.



| | | Slim TFIDF | Jaro-Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | Composed | 1 | 1 | 1 |
| id | ID | 1 | 1 | 1 |
| has | Has | 1 | 1 | 1 |
| isComposed | Composed | 0,95 | 0,808333 | 0 |
| ComposedBy | Composed | 0,94 | 0,96 | 0,96 |
| iscomposed | Composed | 0,95 | 0,808333 | 0 |
| composedby | Composed | 0,94 | 0,96 | 0,96 |
| | | | | |
| is_composed | composed | 0,928077 | 1,0901229 | 0,707106 |
| is_compose | compose | 0,934391 | 0,707106 | 0,707106 |
| is_compos | compos | 0,942809 | 0,707106 | 0,707106 |
| is_compo | compo | 0,70716 | 0,707106 | 0,707106 |
| is_comp | comp | 0,70716 | 0,707106 | 0,707106 |
| is_com | com | 0,70716 | 0,707106 | 0,707106 |
| | | | | |
| composed_by | composed | 0,70716 | 0,707106 | 0,707106 |
| is_composed_by | isComposedBy | 1,24611 | 0,890081 | 0 |
| id_no | idNo | 1,096 | 1,20208 | 0 |
| lives_with | livesWith | 1,186368 | 0,644252 | 0,644252 |
| | | | | |
| is composed | composed | 0,928077 | 1,0901229 | 0,707106 |
| composed by | composed | 0,70716 | 0,707106 | 0,707106 |
| is composed | Composed | 0,928077 | 1,0901229 | 0,707106 |
| composed by | Composed | 0,70716 | 0,707106 | 0,707106 |
| | | | | |
| is composed | is_composed | 1 | 1 | 1 |
| composed by | composed_by | 1 | 1 | 1 |
| is composed by | is_composed_by | 1 | 1 | 1 |

Appendix 11: Table: Comparison of Hybrid methods for very similar pairs of strings, changing the order of the strings.

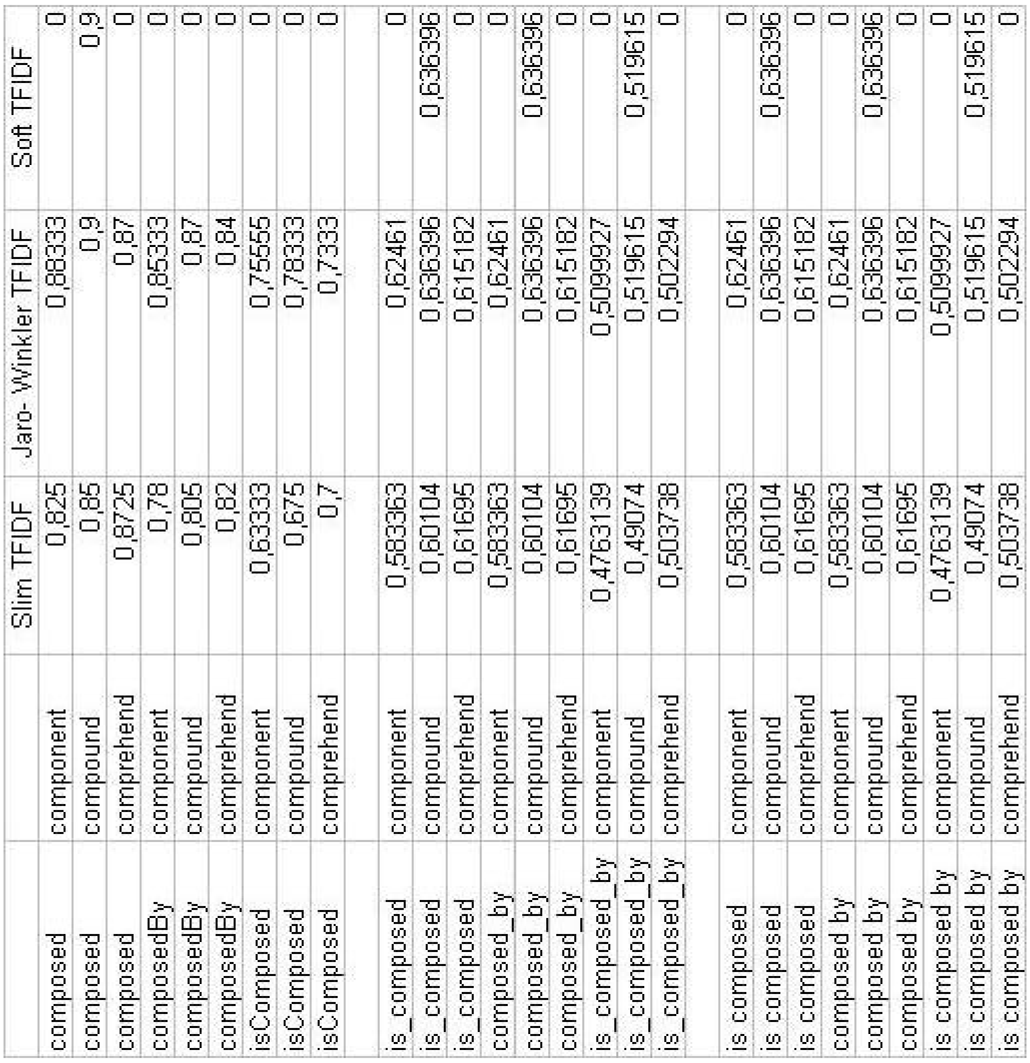


| | | Slim TFIDF | Jaro-Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | Composed | 1 | 1 | 1 |
| id | ID | 1 | 1 | 1 |
| has | Has | 1 | 1 | 1 |
| isComposed | Composed | 0,95 | 0,808333 | 0 |
| ComposedBy | Composed | 0,94 | 0,96 | 0,96 |
| iscomposed | Composed | 0,95 | 0,808333 | 0 |
| composedby | Composed | 0,94 | 0,96 | 0,96 |
| | | | | |
| is_composed | composed | 0,70716 | 0,707106 | 0,707106 |
| is_compose | compose | 0,70716 | 0,707106 | 0,707106 |
| is_compos | compos | 0,70716 | 0,707106 | 0,707106 |
| is_compo | compo | 0,70716 | 0,707106 | 0,707106 |
| is_comp | comp | 0,70716 | 0,707106 | 0,707106 |
| is_com | com | 0,70716 | 0,707106 | 0,707106 |
| | | | | |
| composed_by | composed | 0,70716 | 0,707106 | 0,707106 |
| is_composed_by | isComposedBy | 0,5051814 | 0,44905 | 0 |
| id_no | idNo | 0,5656854 | 0,6128258 | 0 |
| lives_with | livesWith | 0,636396 | 0,644252 | 0,644252 |
| | | | | |
| is composed | composed | 0,70716 | 0,707106 | 0,707106 |
| composed by | composed | 0,70716 | 0,707106 | 0,707106 |
| is composed | Composed | 0,70716 | 0,707106 | 0,707106 |
| composed by | Composed | 0,70716 | 0,707106 | 0,707106 |
| | | | | |
| is composed | is_composed | 1 | 1 | 1 |
| composed by | composed_by | 1 | 1 | 1 |
| is composed by | is_composed_by | 1 | 1 | 1 |

Appendix 12: Table: Comparison of Hybrid methods for pairs of strings with some similarities.

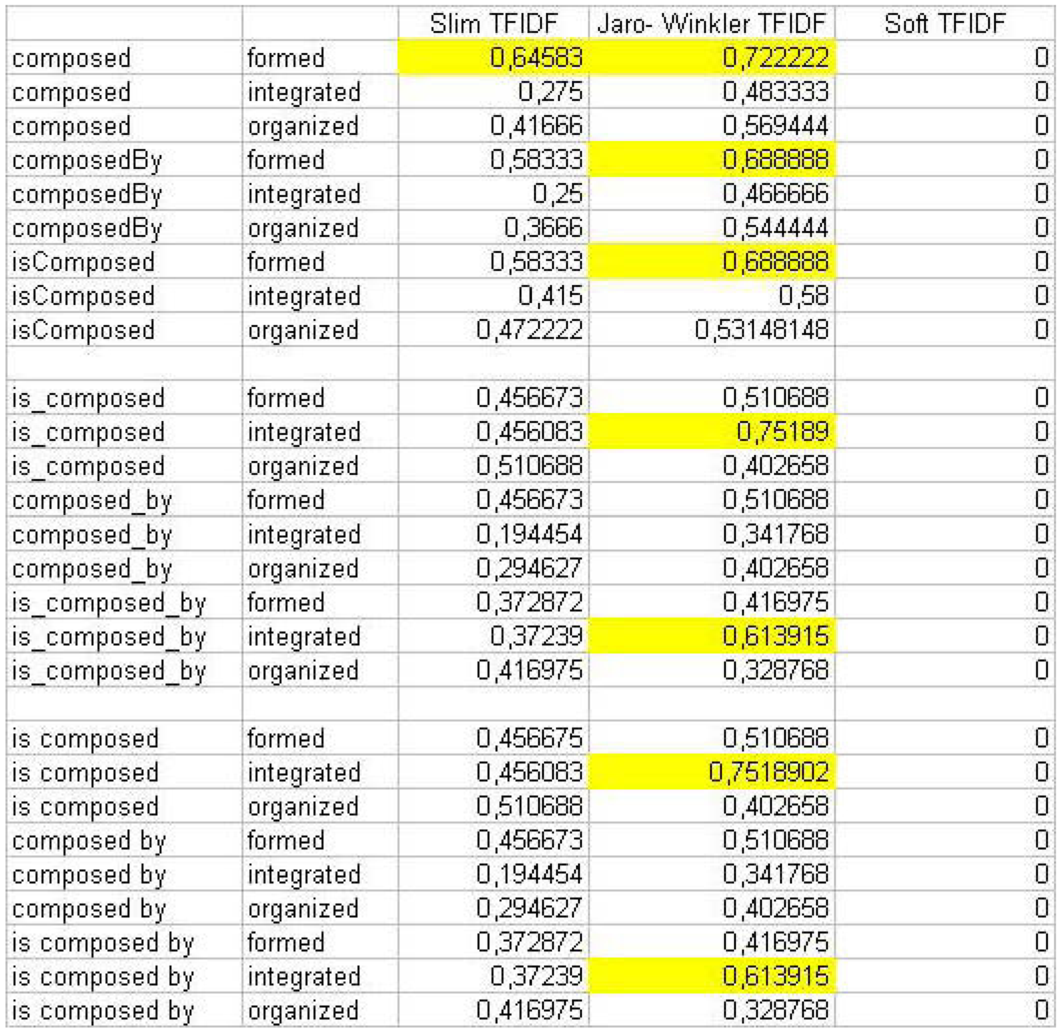
| | | Slim TFIDF | Jaro- Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | component | 0,825 | 0,88333 | 0 |
| composed | compound | 0,85 | 0,9 | 0,9 |
| composed | comprehend | 0,8725 | 0,87 | 0 |
| composedBy | component | 0,78 | 0,85333 | 0 |
| composedBy | compound | 0,805 | 0,87 | 0 |
| composedBy | comprehend | 0,82 | 0,84 | 0 |
| isComposed | component | 0,63333 | 0,75555 | 0 |
| isComposed | compound | 0,675 | 0,78333 | 0 |
| isComposed | comprehend | 0,7 | 0,7333 | 0 |
| | | | | |
| is_composed | component | 0,583363 | 0,62461 | 0 |
| is_composed | compound | 0,60104 | 0,636396 | 0,636396 |
| is_composed | comprehend | 0,61695 | 0,615182 | 0 |
| composed_by | component | 0,583363 | 0,62461 | 0 |
| composed_by | compound | 0,60104 | 0,636396 | 0,636396 |
| composed_by | comprehend | 0,61695 | 0,615182 | 0 |
| is_composed_by | component | 0,4763139 | 0,5099927 | 0 |
| is_composed_by | compound | 0,49074 | 0,519615 | 0,519615 |
| is_composed_by | comprehend | 0,503738 | 0,502294 | 0 |
| | | | | |
| is composed | component | 0,583363 | 0,62461 | 0 |
| is composed | compound | 0,60104 | 0,636396 | 0,636396 |
| is composed | comprehend | 0,61695 | 0,615182 | 0 |
| composed by | component | 0,583363 | 0,62461 | 0 |
| composed by | compound | 0,60104 | 0,636396 | 0,636396 |
| composed by | comprehend | 0,61695 | 0,615182 | 0 |
| is composed by | component | 0,4763139 | 0,5099927 | 0 |
| is composed by | compound | 0,49074 | 0,519615 | 0,519615 |
| is composed by | comprehend | 0,503738 | 0,502294 | 0 |

Appendix 13: Table: Comparison of Hybrid methods for pairs of strings with some similarities, changing the order of the strings.



| | | Slim TFIDF | Jaro- Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | component | 0,825 | 0,88333 | 0 |
| composed | compound | 0,85 | 0,9 | 0,9 |
| composed | comprehend | 0,8725 | 0,87 | 0 |
| composedBy | component | 0,78 | 0,85333 | 0 |
| composedBy | compound | 0,805 | 0,87 | 0 |
| composedBy | comprehend | 0,82 | 0,84 | 0 |
| isComposed | component | 0,63333 | 0,75555 | 0 |
| isComposed | compound | 0,675 | 0,78333 | 0 |
| isComposed | comprehend | 0,7 | 0,7333 | 0 |
| | | | | |
| is_composed | component | 0,583363 | 0,62461 | 0 |
| is_composed | compound | 0,60104 | 0,636396 | 0,636396 |
| is_composed | comprehend | 0,61695 | 0,615182 | 0 |
| composed_by | component | 0,583363 | 0,62461 | 0 |
| composed_by | compound | 0,60104 | 0,636396 | 0,636396 |
| composed_by | comprehend | 0,61695 | 0,615182 | 0 |
| is_composed_by | component | 0,4763139 | 0,5099927 | 0 |
| is_composed_by | compound | 0,49074 | 0,519615 | 0,519615 |
| is_composed_by | comprehend | 0,503738 | 0,502294 | 0 |
| | | | | |
| is composed | component | 0,583363 | 0,62461 | 0 |
| is composed | compound | 0,60104 | 0,636396 | 0,636396 |
| is composed | comprehend | 0,61695 | 0,615182 | 0 |
| composed by | component | 0,583363 | 0,62461 | 0 |
| composed by | compound | 0,60104 | 0,636396 | 0,636396 |
| composed by | comprehend | 0,61695 | 0,615182 | 0 |
| is composed by | component | 0,4763139 | 0,5099927 | 0 |
| is composed by | compound | 0,49074 | 0,519615 | 0,519615 |
| is composed by | comprehend | 0,503738 | 0,502294 | 0 |

Appendix 14: Table: Comparison of Hybrid methods for pairs of strings with nothing in common.



| | | Slim TFIDF | Jaro- Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | formed | 0,64583 | 0,722222 | 0 |
| composed | integrated | 0,275 | 0,483333 | 0 |
| composed | organized | 0,41666 | 0,569444 | 0 |
| composedBy | formed | 0,58333 | 0,688888 | 0 |
| composedBy | integrated | 0,25 | 0,466666 | 0 |
| composedBy | organized | 0,3666 | 0,544444 | 0 |
| isComposed | formed | 0,58333 | 0,688888 | 0 |
| isComposed | integrated | 0,415 | 0,58 | 0 |
| isComposed | organized | 0,472222 | 0,53148148 | 0 |
| | | | | |
| is_composed | formed | 0,456673 | 0,510688 | 0 |
| is_composed | integrated | 0,456083 | 0,75189 | 0 |
| is_composed | organized | 0,510688 | 0,402658 | 0 |
| composed_by | formed | 0,456673 | 0,510688 | 0 |
| composed_by | integrated | 0,194454 | 0,341768 | 0 |
| composed_by | organized | 0,294627 | 0,402658 | 0 |
| is_composed_by | formed | 0,372872 | 0,416975 | 0 |
| is_composed_by | integrated | 0,37239 | 0,613915 | 0 |
| is_composed_by | organized | 0,416975 | 0,328768 | 0 |
| | | | | |
| is composed | formed | 0,456675 | 0,510688 | 0 |
| is composed | integrated | 0,456083 | 0,7518902 | 0 |
| is composed | organized | 0,510688 | 0,402658 | 0 |
| composed by | formed | 0,456673 | 0,510688 | 0 |
| composed by | integrated | 0,194454 | 0,341768 | 0 |
| composed by | organized | 0,294627 | 0,402658 | 0 |
| is composed by | formed | 0,372872 | 0,416975 | 0 |
| is composed by | integrated | 0,37239 | 0,613915 | 0 |
| is composed by | organized | 0,416975 | 0,328768 | 0 |

Appendix 15: Table: Comparison of Hybrid methods for pairs of strings with nothing in common, changing the order of the strings.

| | | Slim TFIDF | Jaro- Winkler TFIDF | Soft TFIDF |
|---|---|---|---|---|
| composed | formed | 0,64583 | 0,722222 | 0 |
| composed | integrated | 0,275 | 0,483333 | 0 |
| composed | organized | 0,41666 | 0,569444 | 0 |
| composedBy | formed | 0,58333 | 0,688888 | 0 |
| composedBy | integrated | 0,25 | 0,466666 | 0 |
| composedBy | organized | 0,3666 | 0,544444 | 0 |
| isComposed | formed | 0,58333 | 0,688888 | 0 |
| isComposed | integrated | 0,415 | 0,58 | 0 |
| isComposed | organized | 0,472222 | 0,53148148 | 0 |
| | | | | |
| is_composed | formed | 0,456673 | 0,510688 | 0 |
| is_composed | integrated | 0,26162 | 0,41012 | 0 |
| is_composed | organized | 0,29462 | 0,402658 | 0 |
| composed_by | formed | 0,4566731 | 0,510688 | 0 |
| composed_by | integrated | 0,194454 | 0,341768 | 0 |
| composed_by | organized | 0,294627 | 0,402658 | 0 |
| is_composed_by | formed | 0,372872 | 0,416975 | 0 |
| is_composed_by | integrated | 0,2136195 | 0,334863 | 0 |
| is_composed_by | organized | 0,240562 | 0,328768 | 0 |
| | | | | |
| is composed | formed | 0,456673 | 0,510688 | 0 |
| is composed | integrated | 0,26162 | 0,4101219 | 0 |
| is composed | organized | 0,29462 | 0,402658 | 0 |
| composed by | formed | 0,4566731 | 0,510688 | 0 |
| composed by | integrated | 0,194454 | 0,341768 | 0 |
| composed by | organized | 0,294627 | 0,402658 | 0 |
| is composed by | formed | 0,372872 | 0,416975 | 0 |
| is composed by | integrated | 0,2136195 | 0,334863 | 0 |
| is composed by | organized | 0,240562 | 0,328768 | 0 |