UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Zhengjie FAN

Thèse dirigée par **Jérôme Euzenat**
et codirigée par **François Scharffe**

préparée au sein **INRIA & LIG**
et de **Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Concise Pattern Learning for RDF Data Sets Interlinking

Thèse soutenue publiquement le **4 Avril 2014**,
devant le jury composé de :

Prof. Éric Gaussier
Université de Grenoble, France, Président

Prof. Chantal Reynaud
LRI Orsay, Paris XI, France, Rapporteur

Prof. Christel Vrain
Université d'Orléans, France, Rapporteur

Prof. Zohra Bellahsene
Université Montpellier 2, France, Examinateur

Dr. Jérôme Euzenat
INRIA, France, Directeur de thèse

Dr. François Scharffe
LIRMM, Université Montpellier 2, France, Co-Directeur de thèse

Acknowledgments

First, I would like to express my sincere gratitude to my Supervisor Dr. Jérôme Euzenat. His deep insight of Semantic Web's development and his patient discussions on my research progress keep me on the right track of the topic. Therefore, at the time of graduation, I am able to show my work to other researchers in my field with pride. I also would like to thank my co-advisor Dr. François Scharffe. He is an enthusiastic researcher on doing challenging research topics. During my Ph.D period, he acted like a mental stimulator who encouraged me to step forward whenever I doubted of my ideas. The good results of my work should be attributed to their kind suggestions and great help.

Second, the colleagues of EXMO team gave me a lot of support when I had research difficulties. For instance, Dr. Cássia Trojahn dos Santos provided a great help on how to use GIT; Dr. Jérôme David answered me many technical questions; Manuel Atencia, Jose Luis Aguirre and Luz Maria Priego Roche encouraged me on writing paper and thesis; Melisachew Wudage Chekol provided me valuable information on research methodologies; Tatiana Lesnikova and Armen Inants shared various research experiences and cultural topics with me; Ngoc Nguyen Thinh Dong cooperated with me on translating EDOAL correspondences into graph patterns. Our secretary Ahlem helped me a lot during the initial period that I just arrived in France.

Third, I should thank the people who participate in the Datalift project. After communicating and cooperating with them, I have gained a lot of knowledge on not only how to code in a big platform and write reports, but also how to cooperate with people with different cultural backgrounds. In particular, Laurent Bihanic taught me so much on designing web services. I owe him a lot of thanks.

Fourth, thanks to my friends, my life become colorful and exciting in France. They also give me support on my research and life planning. I would like to give special thank to Prof. Enrico Pagello, who has encouraged me to study in Europe and insist on doing research.

Fifth, I think my thank to my parents is wordless, because of their invaluable support on my life. What I can do best for them in return is trying my best to not let them down.

Sixth, I would thank all the members of my thesis defense committee for their insightful comments, which are really helpful to improve the quality of my thesis.

Finally, I am really more and more interested in doing research, especially in the last half year of my Ph.D period. I will surely continue to do the research, and further improve the work proposed in the thesis in the near future.

Abstract:

There are many data sets being published on the web with Semantic Web technology. The data sets contain analogous data which represent the same resources in the world. If these data sets are linked together by correctly building links, users can conveniently query data through a uniform interface, as if they are querying one data set. However, finding correct links is very challenging because there are many instances to compare. Many existing solutions have been proposed for this problem. (1) One straight-forward idea is to compare the attribute values of instances for identifying links, yet it is impossible to compare all possible pairs of attribute values. (2) Another common strategy is to compare instances according to attribute correspondences found by instance-based ontology matching, which can generate attribute correspondences based on instances. However, it is hard to identify the same instances across data sets because there are the same instances whose attribute values of some attribute correspondences are not equal. (3) Many existing solutions leverage Genetic Programming to construct interlinking patterns for comparing instances, while they suffer from long running time.

In this thesis, an interlinking method is proposed to interlink the same instances across different data sets, based on both statistical learning and symbolic learning. The input is two data sets, class correspondences across the two data sets and a set of sample links that are assessed by users as either "positive" or "negative". The method builds a classifier that distinguishes correct links and incorrect links across two RDF data sets with the set of assessed sample links. The classifier is composed of attribute correspondences across corresponding classes of two data sets, which help compare instances and build links. The classifier is called an interlinking pattern in this thesis. On the one hand, our method discovers potential attribute correspondences of each class correspondence via a statistical learning method, the K-medoids clustering algorithm, with instance value statistics. On the other hand, our solution builds the interlinking pattern by a symbolic learning method, Version Space, with all discovered potential attribute correspondences and the set of assessed sample links. Our method can fulfill the interlinking task that does not have a conjunctive interlinking pattern that covers all assessed correct links with a concise format.

Experiments confirm that our interlinking method with only 1% of sample links already reaches a high F-measure (around 0.94-0.99). The F-measure quickly converges, being improved by nearly 10% than other approaches.

Keywords: Interlinking, Ontology Matching, Machine Learning

Résumé :

De nombreux jeux de données de données sont publiés sur le web à l'aide des technologies du web sémantique. Ces jeux de données contiennent des données qui représentent des liens vers des ressources similaires. Si ces jeux de données sont liés entre eux par des liens construits correctement, les utilisateurs peuvent facilement interroger les données à travers une interface uniforme, comme s'ils interrogeaient un jeu de données unique. Mais, trouver des liens corrects est très difficile car de nombreuses comparaisons doivent être effectuées. Plusieurs solutions ont été proposées pour résoudre ce problème : (1) l'approche la plus directe est de comparer les valeurs d'attributs d'instances pour identifier les liens, mais il est impossible de comparer toutes les paires possibles de valeurs d'attributs. (2) Une autre stratégie courante consiste à comparer les instances selon les attribut correspondants trouvés par l'alignement d'ontologies à base d'instances, qui permet de générer des correspondances d'attributs basés sur des instances. Cependant, il est difficile d'identifier des instances similaires à travers les ensembles de données car, dans certains cas, les valeurs des attributs en correspondence ne sont pas les mêmes. (3) Plusieurs méthodes utilisent la programmation génétique pour construire des modèles d'interconnexion afin de comparer différentes instances, mais elles souffrent de longues durées d'exécution.

Dans cette thèse, une méthode d'interconnexion est proposée pour relier les instances similaires dans différents ensembles de données, basée à la fois sur l'apprentissage statistique et sur l'apprentissage symbolique. L'entrée est constituée de deux ensembles de données, des correspondances de classes sur les deux ensembles de données et un échantillion de liens "positif" ou "négatif" résultant d'une évaluation de l'utilisateur. La méthode construit un classifieur qui distingue les bons liens des liens incorrects dans deux ensembles de données RDF en utilisant l'ensemble des liens d'échantillons évalués. Le classifieur est composé de correspondances d'attributs entre les classes correspondantes et de deux ensembles de données, qui aident à comparer les instances et à établir les liens. Dans cette thèse, le classifieur est appelé motif d'interconnexion. D'une part, notre méthode découvre des correspondances potentielles entre d'attributs pour chaque correspondance de classe via une méthode d'apprentissage statistique : l'algorithme de regroupement K-medoids, en utilisant des statistiques sur les valeurs des instances. D'autre part, notre solution s'appuie sur un modèle d'interconnexion par une méthode d'apprentissage symbolique : l'espace des versions, basée sur les correspondances d'attributs potentielles découvertes et l'ensemble des liens de l'échantillon évalué. Notre méthode peut résoudre la tâche d'interconnexion quand il n'existe pas de motif d'interconnexion combiné qui couvre tous les liens corrects évalués avec un format concis.

L'expérimentation montre que notre méthode d'interconnexion, avec seulement 1% des liens totaux dans l'échantillon, atteint une F-mesure élevée (de 0,94 à 0,99). La F-mesure converge rapidement, ameliorant les autres approches de près de 10%.

Mots-clés : Interconnexion, L'alignement D'ontologies, Apprentissage

Contents

Chapter 1
Introduction

Nowadays, many organizations and individuals publish their data sets on the web for sharing information with other users. There are countless RDF[1] data sets that are published[2], and they tend to be more and more heterogeneous. Integrating heterogeneous data sets can help search web data efficiently. For example, without interlinking across library databases, a librarian has to manually browse different library databases to find a requested book, easily leading to response delay and mistakes. Hence, it is important to provide an easy-to-use and effective method to interlink the same instances across different data sets.

In the interlinking method defined in this thesis, the input is two data sets, class correspondences [Euzenat 2007] across the two data sets and a set of sample links that are assessed by users as either “positive” or “negative”. Class correspondences help find out the same instances from two classes. The set of assessed sample links helps build a classifier that distinguishes correct links and incorrect links across two data sets. The classifier is called an interlinking pattern in this thesis, which can help compare instances of two corresponding classes and generate links. The output of the interlinking method of this thesis is an interlinking pattern.

In this thesis, we assume that class correspondences of two data sets are available. The way to find out the same instances is computing a similarity value by comparing attribute values of instances from two corresponding classes [Suchanek 2012], where attributes refer both to value-oriented attribute (i.e., *property*) and to object-oriented attribute (i.e., *relation*). This requires to find out which attribute to compare and how to generate a similarity value. Nevertheless, it is not necessary to compare each attribute of an instance with each attribute of another instance. Only some attributes should be compared. Assume that we are going to compare instances of two data sets that describe people. It is useful to compare instances by comparing people’s names. But it is useless to compare instances by comparing people’s age with people’s address. Thus, it is better to compare attributes when the attributes have similar semantics and overlapping ranges. The attributes with similar semantics and overlapping ranges are corresponding attributes, which can form attribute correspondences.

Attribute correspondences may be obtained from an alignment between attributes. Instance-based ontology matching works [Dhamankar 2004, Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005, Tran 2011, Qin 2007] can help generate

[1] http://www.w3.org/RDF/

[2] http://lod-cloud.net/

attribute correspondences. However, quality alignments are difficult to obtain. Some existing interlinking approaches [Nikolov 2010, Ngonga Ngomo 2011c] explore attribute correspondences based on sample links, while they are subject to the assumption that some sample correct links should be known beforehand.

In order to solve the problems illustrated above, the interlinking method in this thesis will build attribute correspondences by analyzing value features of each attribute. Since attributes that have similar value features are more likely to be attribute correspondences, this thesis utilizes a clustering method to classify attributes of each class into several groups by value features. The thesis proposes an ontology matching approach of constructing attribute correspondences by classifying the attributes of each class with a K-medoids clustering algorithm [Kaufman 1987]. Then, attributes of different classes are mapped together as potential attribute correspondence, if they have similar value features. The ontology matching method in the thesis can largely reduce the number of attribute pairs that are not corresponding with each other and the number of undiscovered attribute correspondences.

Nevertheless, even if attribute correspondences across two corresponding classes are discovered, correct links cannot be generated by comparing all instances according to all attribute correspondences. Due to the irregularity of instance structure in RDF data sets, correct links may contain different attribute correspondences. If two attribute values of two instances that formed one correct link are equal, we assume that the attribute correspondence that is built with the two attributes is contained in such a correct link. Therefore, two instances cannot be simply judged as a correct link even though all attribute correspondences are compared. Besides discovering attribute correspondences, an interlinking pattern should be constructed. The interlinking pattern helps compute the similarity value of two attribute values with respect to each attribute correspondence of two instances, and combines the similarity values of all attribute correspondences into one value, which represents the similarity of two compared instances. Two instances are generated as a link if their similarity is above a pre-defined threshold. But usually the interlinking pattern cannot be obtained easily without referring to correct links. A learning model is required to improve the interlinking pattern with assessed links from users. The interlinking method in this thesis employs the Version Space model [Mitchell 1982], a symbolic learning method that searches the interlinking pattern precisely, to construct such an interlinking pattern that is composed of all attribute correspondences, so as to generate a link set across two RDF data sets.

To summarize, this thesis proposes an interlinking method which helps interlink RDF data sets with high F-measure, according to the in-depth analysis of instances' attribute value features. The interlinking method in this thesis adopts two machine learning steps mixing statistical and symbolic learning [Hastie 2001, Mitchell 1997]: (1) construct attribute correspondences by classifying the attributes with K-medoids clustering; (2) generate interlinking patterns with Version Space.

The key contributions of this thesis are thus:

- *Attribute Clustering Algorithm*: A K-medoids clustering is implemented to find out all potential attribute correspondences across corresponding classes. The K-medoids method is able to classify large data sets like attribute set of each class with statistics. Specifically, K-medoids classifies attributes of each class in a class correspondence into several groups, based on a set of statistical features of instance values. Statistical feature is a kind of value feature that reflects attribute values' characteristics with statistics. Then, each group of one corresponding class is mapped to a group of the other corresponding class that contains the most similar features. Potential attribute correspondences are obtained by mapping attributes across the matched groups. In other words, in each potential attribute correspondence, one attribute comes from one group, the other attribute comes from a group which is mapped to the former group. Hence, attribute pairs that are not corresponding with each other are largely avoided. The final output of the clustering algorithm is a set of potential attribute correspondences.

- *Version Space Algorithm*: After the clustering step, an interlinking pattern is constructed based on the Version Space model. The Version Space method is able to construct an interlinking pattern when the input is a set of symbols (i.e., potential attribute correspondences). Nevertheless, the Version Space model suffers from a limitation for the interlinking problem. It can only build the interlinking pattern that is represented by a conjunctive pattern, which is a conjunction of attribute correspondences that are contained in all correct links. If there is no such a conjunctive pattern, the Version Space cannot produce any interlinking pattern. It outputs a null result. Disjunctive Version Space is one model that is designed to overcome such a limitation. It is a method that is able to express cases that cannot be represented by a conjunctive pattern into a disjunction of conjunctive patterns. When there are several conjunctive patterns each of which satisfies only some but not all correct links, the interlinking pattern can be expressed into a disjunction of conjunctive patterns by Disjunctive Version Space. However, the interlinking pattern it learns is a long expression, which will largely increase the interlinking running time. In this thesis, an Extended Version Space algorithm is proposed to build the interlinking pattern that includes (excludes) correct links (incorrect links), with a more concise expression than Disjunctive Version Space. Such a concise expression in turn speeds up the interlinking process, in that the more concise an interlinking pattern is, the shorter running time the interlinking process requires to produce a link set. The final output of the Extended Version Space algorithm is an interlinking pattern than help generate a link set of two RDF data sets.

The interlinking method of this thesis is evaluated against a set of large-scale data sets, by comparing to related works. Experiments confirm that the interlinking solution with only 1% sample links can already return a fairly precise link set. The F-measures quickly converge to a higher level by nearly 10% than other state-of-the-art approaches.

This thesis is supported by the Datalift[3] project. It is a project that weaves data sets of various domains and formats into linked data [Scharffe 2012]. There are several modules within the Datalift platform. The content of this thesis contributes to the interlinking module, which interlinks RDF data sets.

The remainder of the thesis is organized as follows. The thesis formulates the interlinking problem in Chapter 2. Thereafter, related works are introduced in Chapter 3. Chapter 4 presents the proposed solution and a system overview. The K-medoids clustering method is presented in Chapter 5, in order to explore attribute correspondences across corresponding classes. The transforming rules that transforms ontology alignment into executable interlinking scripts are shown in Chapter 6, which are not only used to generate a set of sample links that are assessed by users for constructing and improving an interlinking pattern, but also used to generate a set of links across two corresponding classes. Chapter 7 describes the Extended Version Space method, which is used to generate an interlinking pattern with a concise disjunctive expression. Chapter 8 presents the experimental results based on real-world data sets. Finally, conclusions as well as future work are summarized in Chapter 9.

[3] http://datalift.org/

Chapter 2

The Data Interlinking Problem

Contents

In this chapter, a general scenario for interlinking is presented first, in Section 2.1. Then, the interlinking problem is defined in Section 2.2. Finally, three main research tasks required to solve the interlinking problem are illustrated in Section 2.2.1.

2.1 Scenario

Many data sets on the web are expressed in RDF. The Resource Description Framework (RDF) is a W3C standard language that organizes data into a set of triples. Each triple is made up of *subject*, *predicate*, and *object*. Subject can be a Uniform Resource Identifier (i.e., *URI*[1]) or a blank node. Object can be a URI, a blank node or a data value such as a string or an integer. Predicate is a relationship between the subject and the object. A predicate can be an object-oriented property, which is usually called *relation*. Its range is a set of entities. It also can be a data-oriented property, which is usually called *property*. Its range is a set of values. The domain of a predicate is always a set of entities. Each RDF data set can be built an ontology[2]. It is a schema that specifies classes, relations, and properties in the data set. A class usually has several relations and properties. Each relation has one or several classes as its domain. It also has one or several classes as its range. Each property has one or several classes as its domain. It also has one data type as its range. In the data set, all entities which are created conforming to the structure of a class are called *instances* of the class [Antoniou 2008].

Interlinking data sets is highly required in web applications, such as data searching and querying. Users would like to weave all data sets into a web, so as to query them easily. For example, a person would like to buy a computer on the web. In order to buy the computer with a relatively cheaper price, he needs to browse all

[1] http://www.w3.org/TR/uri-clarification/

[2] http://semanticweb.org/wiki/Ontology

appliance shopping web sites. But, it is probably a heavy task to search one desired computer several times on different web sites, due to the fact that the information of the same computer is described in different data sets. If we interlink the data set of each shopping site together, users can obtain the prices of the computer in all shopping sites only from one shopping site's querying portal on the web.

Interlinking is a task that attaches the same instances from different data sets together by comparing instances. Given two different data sets, we can compare instances' URIs or property values to find the same instances. It is not easy to compare instances' URIs. Usually, URIs are not expressed with the same prefix. For example, all instances of geographical RDF data set *INSEE* are named with the prefix *http://rdf.insee.fr/geo/2010/*. Instance *http://rdf.insee.fr/geo/2010/DEP_38* is the department "Isère" of France in *INSEE*. All instances of geographical RDF data set *EUROSTAT* are named with the prefix *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/*. Instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714* is the department "Isère" in France in *EUROSTAT*. Thus, different prefixes make the comparison of URIs quite difficult [Nikolov 2010]. Nevertheless, it is easier to compare property values of instances. Each instance has one or more property values. The values are either strings or numbers. They do not contain prefixes. They can help compare instances. Instances' relations also can help compare instances. Here, relations' subjects are the compared instances. The comparison is not executed upon the URIs of relations' objects, but the property values of relations' objects. The comparison also can be executed upon property values of relations' subjects, when relations' objects are the compared instances. In one sentence, it is easier to compare instances' property values rather than their URIs.

An interlinking example is given here to illustrate the interlinking process of this thesis. We are going to interlink two geographical data sets. One is *INSEE*, and the other is *EUROSTAT*. There are instances that describe departments in France in both data sets. The interlinking task is to compare the property values of instances in order to find out links, such as *http://rdf.insee.fr/geo/2010/DEP_38 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*. *owl:sameAs*[3] is a property that defines a link across two instances that refer to the same resource in the world.

Figure 2.1 shows the corresponding classes of "department" as well as their corresponding attributes in both data sets. A class correspondence is composed of two classes that have similar semantics and instances. A attribute correspondence is composed of two attributes that have similar semantics and overlapping ranges. In this thesis, "attribute" refers to both "relation" and "property" in RDF data sets. There are three correspondences in this example. They are class correspondence *insee:Departement*↔*eurostat:NUTSRegion*, property correspondence *insee:nom*↔*eurostat:name*, and relation correspondence *insee:subdivision*$^{-1}$↔*eurostat:hasParentRegion*. Here, $\cdot^{-1}$ means the reverse rela-

[3] http://www.w3.org/TR/owl-ref/#sameAs-def

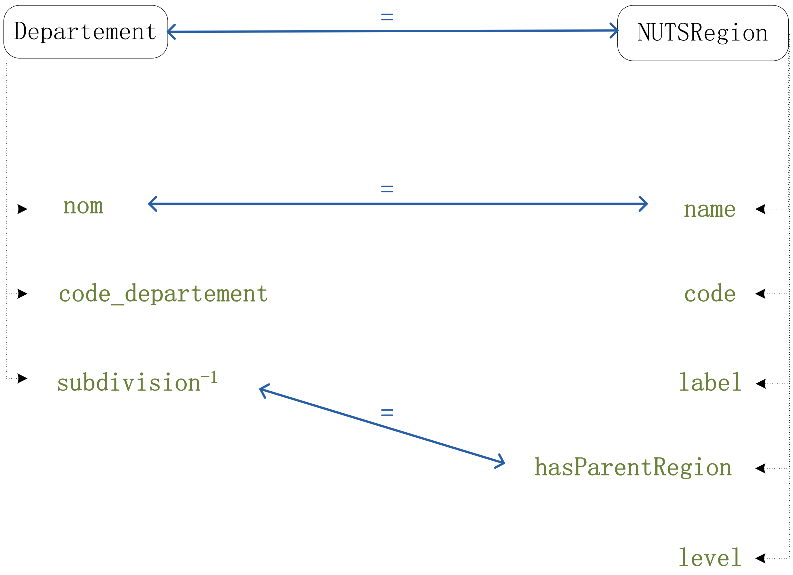


Figure 2.1: Correspondences of Departments in *INSEE* and *EUROSTAT*

tion. Thus, relation *eurostat:hasParentRegion* is corresponding to the reverse relation of relation *insee:subdivision*.

In order to link the same instances of class "department" in both data sets, we need to compare instances' property values rather than instances' URIs. Class *insee:Departement* has two attributes. They are property *insee:nom* and relation *insee:code_ departement*. It is also the object of the relation *insee:subdivision*. The following is an instance of the class "Department" in the data set *INSEE*, which is *http://rdf.insee.fr/geo/2010/DEP_ 38*. It represents the department "Isère" in France.

```
<insee:Departement rdf:about="DEP_38">
    <insee:code_departement>38</insee:code_departement>
    <insee:nom xml:lang="fr">Isère</insee:nom>
</insee:Departement>
```

In contrast, the class *eurostat:NUTSRegion* has five attributes. They are property *eurostat:level*, property *eurostat:name*, property *eurostat:code*, property *rdfs:label*, and relation *eurostat:hasParentRegion*. The following is an instance of the class "Department" in the data set *EUROSTAT*, which is *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*. It also represents the department "Isère" in France.

```
<NUTSRegion rdf:about="FR714">
    <level rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</level>
    <hasParentRegion rdf:resource="FR71"/>
    <code>FR714</code>
    <name>Isère</name>
    <rdfs:label>FR714 - Isère</rdfs:label>
</NUTSRegion>
```

In order to build a link between the same instances, we should compare instances' property values according to the attribute correspondences of the two corresponding classes to which the instances belong. Intuitively, all property values of each instance in the class *insee:Departement* should be compared with all property values of each instance in the class *eurostat:NUTSRegion*. However, most of the comparisons are not useful for assessing whether two instances are identical or not. For example, it is meaningless to compare the value of the property *insee:code_departement* with the value of the property *eurostat:name*. Because in each correct link like *http://rdf.insee.fr/geo/2010/DEP_38 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*, the similarity of the two values is 0. Such a similarity value cannot help us to evaluate whether the instance *http://rdf.insee.fr/geo/2010/DEP_38* and the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714* are the same or not. The reason is that the property *insee:code_departement* and the property *eurostat:name* do not have similar semantics and overlapping ranges. The property *insee:code_departement* denotes the codes of the departments in France in the data set *INSEE*. While the property *eurostat:name* denotes the names of the departments in France in the data set *EUROSTAT*. There is not an attribute correspondence between the two properties. Thus, a comparison of two attributes is only useful when there is an attribute correspondence that can be built with the two attributes.

There are only two comparisons that should be done for evaluating the two instances that refer to the department "Isère", because there are two attribute correspondences of the two corresponding classes *insee:Departement* and *eurostat:NUTSRegion*. One comparison is executed between the value of the property *insee:nom* in the class *insee:Departement* with the value of the property *eurostat:name* in the class *eurostat:NUTSRegion*. That is to say, *insee:nom*'s value, *Isère*, of the instance *http://rdf.insee.fr/geo/2010/DEP_38* should be compared with *eurostat:name*'s value, *Isère*, of the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*. Another comparison is executed between property values of the relation *insee:subdivision*'s subjects in the class *insee:Departement* with property values of the relation *eurostat:hasParentRegion*'s objects in the class *eurostat:NUTSRegion*. As for the two instances referring to *Isère*, instance *http://rdf.insee.fr/geo/2010/REG_82* is the subject of the relation *insee:subdivision* when the object is the instance *http://rdf.insee.fr/geo/2010/DEP_38*. Instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR71* is the object of the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*' relation *eurostat:hasParentRegion*. The two instances are shown below.

```
<insee:Region rdf:about="REG_82">
    <insee:code_region>82</insee:code_region>
    <insee:nom xml:lang="fr">Rhône-Alpes</insee:nom>
    <insee:subdivision>
        <insee:Departement rdf:about="DEP_38">
            <insee:code_departement>38</insee:code_departement>
            <insee:nom xml:lang="fr">Isère</insee:nom>
        </insee:Departement>
    </insee:subdivision>
</insee:Region>

<NUTSRegion rdf:about="FR71">
    <level rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2</level>
    <hasParentRegion rdf:resource="FR7"/>
    <code>FR71</code>
    <name>Rhône-Alpes</name>
    <rdfs:label>FR71 - Rhône-Alpes</rdfs:label>
</NUTSRegion>
```

Since it is difficult to compare URIs of the above two instances, property values of the instance *http://rdf.insee.fr/geo/2010/REG_82* should be compared with property values of the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR71*. Therefore, the property *insee:nom*'s value, *Rhône-Alpes*, of the instance *http://rdf.insee.fr/geo/2010/REG_82* should be compared with the property *eurostat:name*'s value, *Rhône-Alpes*, of the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR71*. Based upon the two similarities of property values, the instance *http://rdf.insee.fr/geo/2010/DEP_38* and the instance *http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714* are assessed as "the same". The link *http://rdf.insee.fr/geo/2010/DEP_38 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714* is built with the two instances.

From the example above, the scenario of the interlinking process can be formulated in this way. A RDF graph g is a set of triples made over a set of URIrefs $\mathcal{U}$, blanks, and literals $\mathcal{V}$, which is described by ontology o. Similarly, g' is another set of triples made over a set of URIrefs $\mathcal{U}'$, blanks, and literals $\mathcal{V}'$, which is described by ontology o'. There is an alignment between classes of two ontologies o and o', i.e., a set of class correspondences $< e, =, e' >$ over entities e and e' such that $e \in \mathcal{U}$, $e' \in \mathcal{U}'$. If two instances $i \in \mathcal{U}$ and $i' \in \mathcal{U}'$ actually reflect the same object in the world, we call there is a *link* between them, denoted as i owl:sameAs i'.

In order to find out correct links across each pair of corresponding classes in two data sets, it is required to construct a classifier that distinguishes correct links and incorrect links. Such a classifier is called an interlinking pattern in this thesis. An interlinking pattern combines all attribute correspondences of the two corresponding classes into an expression that satisfies correct links and dissatisfies incorrect links. With the interlinking pattern, we can compare instances and generate links. Users need a method that can generate such an interlinking pattern that helps interlink the data sets. Therefore, the interlinking problem to be solved in this thesis is formulated below.

2.2 The Interlinking Problem

Given two data sets being RDF graphs g and g' described by ontology o and o' and an alignment between classes of two ontologies o and o', the objective is to find an interlinking pattern that covers a set of links $L \in \mathcal{U} \times \mathcal{U}'$ such that instance pair $< i, i' > \in L$ if and only if i *owl:sameAs* i'.

Since there are many existing effective ontology matchers that can generate class correspondences, we assume that there are a set of class correspondences already given for two interlinking data sets. However, many ontology matchers cannot provide qualified attribute correspondences of two interlinking data sets. So the attribute correspondences between two ontologies are unknown in the problem formulation of this thesis.

2.2.1 Three Main Research Tasks

According to the interlinking scenario and problem definition described in the last two sections, an interlinking pattern that is composed of attribute correspondences across two corresponding classes is required to generate links between two RDF data sets.

Since the interlinking pattern satisfies correct links and dissatisfies incorrect links, it is a classifier that is composed of the set of attribute correspondences that exist in each correct link. Moreover, it is also a classifier that does not contain the set of attribute correspondences that exist in any incorrect link. If two attribute values of two instances that form a link is similar, we assume that the attribute correspondence built with the two attributes is contained in the link. For instance, in the example of Section 2.1, the set of attribute correspondences of the correct link *http://rdf.insee.fr/geo/2010/DEP_38 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714* is {*insee:nom*↔*eurostat:name*, *insee:subdivision*$^{-1}$↔*eurostat:hasParentRegion*}. So the interlinking pattern of the example should contain such a set of attribute correspondence in order to generate the correct link *http://rdf.insee.fr/geo/2010/DEP_38 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR714*.

Without knowing some correct and incorrect links, it is hard to construct an interlinking pattern that is able to distinguish links. Different links usually have different sets of attribute correspondences. Assume that there are n attribute correspondences in a pair of corresponding classes. There are totally $C_n^1 + C_n^2 + \ldots + C_n^n = 2^n - 1$ sets of attribute correspondences. Since we cannot estimate which sets exist in correct links, we do not know which sets of attribute correspondences are needed to build the interlinking pattern. We also have no idea how many sets of attribute correspondences are needed to build the interlinking pattern. Similarly, we cannot estimate which sets exist in incorrect links, we do not know which sets of attribute correspondences should not be put into the interlinking pattern.

Here we give an example to illustrate the difficulty of constructing an interlinking pattern across two corresponding classes in two RDF data sets. Assume that there are three links of two RDF data sets. One is a correct link, which has a set of the attribute correspondences $\{AC_i, \ldots, AC_j\}$. The second is also a correct link, which has a set of attribute correspondences $\{AC_k, \ldots, AC_l\}$. The third is an incorrect link, which has a set of attribute correspondences $\{AC_p, \ldots, AC_q\}$. Here, $AC_i, \ldots, AC_j$, $AC_k, \ldots, AC_l$ and $AC_p, \ldots, AC_q$ are attribute correspondences of the two corresponding classes. If we do not know the first link, we cannot estimate the set $\{AC_i, \ldots, AC_j\}$. If we do not know the second link, we cannot estimate the set $\{AC_k, \ldots, AC_l\}$. Therefore, we may do not put those two sets of attribute correspondences in the interlinking pattern, so as to let the pattern disable to cover the two correct links. Similarly, if we do not know the third link, we cannot estimate the set $\{AC_p, \ldots, AC_q\}$. Therefore, we may put this set into the interlinking pattern, so as to let the pattern disable to filter the third incorrect link. Therefore, in order to produce as many correct links as possible in the link set and reduce the number of incorrect links in the link set, some assessed links are needed. The assessed links are links marked as either "correct" or "incorrect" by users. Since interlinking pattern should be constructed by extracting the sets of attribute correspondences from assessed links, the process of constructing the interlinking pattern can be a Machine Learning process that improves the interlinking pattern by referring to assessed links. The definition of Machine Learning is below.

Machine Learning builds a classifier for a task with an amount of training data so as to classifier other data. Assume that we are going to train a robot to distribute mails in France into different boxes that represent different departments. Some training mails are collected to teach the robot to which box each mail should be distributed. The robot then learns the relationships between cities in the addresses of the training mails and departments that are represented by the boxes. Based on the relationships, the robot can distribute other mails into the corresponding boxes of departments. In this example, the training data are the training mails. The classifier is the relationships between cities and departments that the robot learns. Generally speaking, there are four types of learning methods, *Supervised Learning*, *Unsupervised Learning*, *Semi-supervised Learning*, and *Reinforcement Learning*.

1 Supervised Learning is a method that builds a classifier with labeled examples [Kotsiantis 2007].

2 Unsupervised Learning is a method that builds a classifier with unlabeled examples [Ghahramani 2004, Hinton 1999]. Clustering is a widely-used Unsupervised Learning method. It divides unlabeled examples into several groups according to some characteristics of the examples [Xu 2005].

3 Semi-supervised Learning is a method that builds a classifier with both labeled and unlabeled examples [Zhu 2005, Chapelle 2006]. Self-training is a commonly-used semi-supervised Learning method. It first builds a classifier with the available labeled examples. Then it classifies the rest unlabeled examples with the classifier. After classification, the most reliable examples are

selected to improve the classifier with their predicting labels. The classifier is prone to stay in the local optimum.

4 Reinforcement Learning is a method that builds a classifier by collecting feedback from the environment where the task takes place [Gosavi 2009, Kaelbling 1996].

According to the definition of Machine Learning, the process of constructing the interlinking pattern is a supervised learning process. The learned classifier is the interlinking pattern across two corresponding classes in two RDF data sets. The labeled examples are assessed links.

There are several learning methods that fall into the classification of Supervised Learning.

i Support Vector Machine is a learning method that builds a linear classifier of classifying the examples into two categories according to the information extracted from the labeled examples [Cristianini 2000, Burges 1998]. The linear classifier combines all conditions that each positive example satisfies into a linear function.

ii Neural Network is a learning method that builds a classifier composed of a group of artificial neurons for parsing the input information. Usually it is an adaptive system, which can change its structure according to the information it processes [Haykin 1998, Stergiou 1996, Chow 2007]. Such a structure shows the relationship between the labeled examples and the target effect.

iii Naive Bayesian classifier [Witten 2005] is a probabilistic method that classifies the examples according to their probabilities of several relevant variables. The probabilities are computed according to the Bayes' Theorem.

iv Decision Tree is a tree-structured classifier that classifies the examples according to their feature values [Murthy 1997, Safavian 1991].

v The k-Nearest Neighbor method classifies data by considering the label of their nearest labeled example [Jiang 2007, Bhatia 2010].

vi Rule learning extracts a set of rules from the labeled examples that reflect the logic relationships of several relevant variables [Piatetsky-Shapiro 1991].

vii Version Space is a learning process that maintains two boundaries for separating labeled positive examples and labeled negative examples [Mitchell 1982, Mitchell 1997]. After all labeled examples are learned, the two boundaries are merged into one. It is the classifier that distinguishes positive and negative examples.

To summarize, there are three tasks to be done for the interlinking problem of this thesis.

1 Discovering Attribute Correspondences
This task is to find out all attribute correspondences of each pair of corresponding classes across two interlinking data sets.

2 Generating Links
This task is to generate a set of assessed links that can be used to construct and improve the interlinking pattern with a supervised learning method.

3 Constructing and Improving the Interlinking Pattern with Assessed Links
This task is to construct and improve the interlinking pattern with the assessed links that are collected in the second task.

2.2.1.1 Discovering Attribute Correspondences

All attribute correspondences of each pair of corresponding classes are demanded for interlinking. In order to judge whether two instances are the same or not, instances' attribute values should be compared. But, it is a heavy task to compare each attribute value of one instance with each attribute value of another instance. It is better to reduce the number of comparisons as many as possible. In fact, there is no need to compare attributes that do not correspond with each other. For example, it is meaningless to compare values of the property *insee:nom* in the class *insee:Departement* with values of the property *eurostat:code* in the class *eurostat:NUTSRegion*. The former property refers to departments' names in France. The latter property refers to departments' codes in France. The two properties do not have similar semantics and overlapping ranges, so it is a waste of time to compare their values. The first task is defined as follows:

Given two classes C and C', find out at least one attribute correspondence (AC) $A \leftrightarrow A'$ on attribute A of class C and attribute A' of class C' such that $range(A) \cap range(A') \neq \varnothing$, where $range(x)$ represents the range of attribute x.

2.2.1.2 Generating Links

After finding out all attribute correspondences, a set of assessed links should be generated. They are used to extract the sets of attribute correspondences for constructing and improving the interlinking pattern with a supervised learning method. Without knowing some assessed links, the interlinking pattern is hard to be constructed, because it is difficult to estimate which and how many sets of attribute correspondences are required for the interlinking pattern. So the task is defined as:

Given n attribute correspondences $AC_1, \ldots, AC_n$ of two corresponding classes, a set of assessed links $\{l_1, \ldots, l_n\}$ are generated to help construct and improve the interlinking pattern that can distinguish links precisely.

2.2.1.3 Constructing and Improving the Interlinking Pattern with Assessed Links

Based on assessed links, a learning model can be applied here to construct an interlinking pattern P and improve it into a new one P'. As illustrated before, the interlinking pattern cannot be built precisely without extracting the set of attribute correspondences from assessed links produced from Task 2.2.1.2. After learning the sets of attribute correspondences of the assessed links, the number of correct links in the link set produced by the new interlinking pattern P' will be bigger than the one of the link set produced by the old interlinking pattern P, or the number of incorrect links in the link set produced by the new interlinking pattern P' will be smaller than the one of the link set produced by the old interlinking pattern P.

There are three measures that can be used to evaluate the link set. They are *Precision*, *Recall*, and *F-measure*. Precision is a ratio of the number of generated correct links on the number of all generated links in a link set. Recall is a ratio of the number of generated correct links on the number of all correct links in an interlinking task. Nevertheless, neither Precision nor Recall gives a comprehensive view of the link set. The reason is that Precision only shows the interlinking performance of filtering incorrect links, while Recall only shows the interlinking performance of generating correct links. In other words, an interlinking pattern with high precision may be unable to produce some correct links. An interlinking pattern with high recall may also produce some incorrect links. F-measure is a balancing measure of Precision and Recall, which is defined based on the two measures. Its definition is

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{2.1}$$

Usually F_1 measure (i.e., $\beta = 1$) is applied to evaluate the generated link set.

So the last task is defined as:

Given a set of assessed links $\{l_1,\ldots,l_n\}$ and a learning model F such that $F(l_1, l_2, \ldots, l_n, P) = P'$, where $A(P') > A(P)$. $A(x)$ is a measure of evaluating the pattern x. $A(x)$ can be Precision, Recall, or F-measure.

2.3 Conclusion

This chapter defines the interlinking problem of this thesis, which is to find out the interlinking pattern for generating a link set of two RDF data sets.

The research problem can be divided into three tasks.

- Discovering Attribute Correspondences
- Generating Links
- Constructing and Improving the Interlinking Pattern with Assessed Links

In the next chapter, we will introduce related works of the interlinking process. The works will be introduced according to techniques that are used for interlinking. Afterwards, the works are analyzed according to the tasks that they make contributions to.

CHAPTER 3

Related Work

Contents

This chapter analyzes related works of the interlinking process. First, a brief overview of the related works is given in Section 3.1. The related works are introduced according to the techniques that are used for interlinking. Second, related works are presented to show the state of the art on the three tasks for the interlinking problem of this thesis. They are, related works on discovering attribute correspondences (Section 3.2), related works on generating links (Section 3.3), and related works on constructing and improving the interlinking pattern with assessed links (Section 3.4).

3.1 Interlinking

Interlinking technologies for heterogeneous data sets have been studied for years. Although domain specific [Raimond 2008, Sleeman 2010] and dataset specific [Auer 2009, Hassanzadeh 2009] interlinking methods have already been proposed, it is not enough to interlink new-coming RDF data sets from various domains, such as Geographical Information System or bioinformatics. In the following text, we are going to introduce the interlinking works that are able to interlink data sets from various domains. The works are presented with respect to the techniques that are used for interlinking.

3.1.1 Record Linkage

In the Database field, *Record Linkage* [Cao 2011] is similar to the interlinking problem of RDF data sets. It is to find out the same records across two tables of the database [Winkler 2006]. It is also called object identification [Tejada 2001, Tejada 2002], approximate matching/joins [Gravano 2003, Guha 2004], and entity resolution [Whang 2009, Benjelloun 2009]. Record Linkage is needed in the tasks Data Cleaning [Guha 2004] and Merging [Ananthakrishna 2002].

Some definitions of database can be applied for interlinking RDF data sets. One is inverse functional property [Niu 2012, Hogan 2010, Hu 2011], the other is key [Atencia 2012, Song 2011, Song 2013].

3.1.1.1 Inverse Functional Property

Inverse Functional Property is used to identify instances within a RDF data set. An inverse functional property is a property that whenever an object is the property's value of a subject, there is no other subject whose property's value is the same object. It also means that the subject is the one and the only subject that has the object as the property value. In RDF data set, inverse functional property's subjects are URIs. Inverse functional property's objects are data values. If one value is given, an instance is identified uniquely [Niu 2012, Hogan 2010, Hu 2011]. So inverse functional property is used to identify instances of one data set. If inverse functional properties across two data sets can be matched, the same instances are uniquely identified in both data sets. Then, a link can be built.

Niu et al. [Niu 2012], Hogan et al. [Hogan 2010] and Hu et al. [Hu 2011] all compute inverse functional properties so as to find links across two RDF data sets. Attribute correspondences are built between inverse functional properties of two RDF data sets to help compare instances.

Specifically, Niu et al. [Niu 2012] apply the Expectation-Maximization algorithm to discover frequent-used attribute correspondences that are built with inverse functional properties. The paper assumes that there are several available assessed correct links, which are utilized to mine potential attribute correspondences across data sets. The paper constructs a graph M showing the relationships between available assessed links and each potential attribute correspondence. Each node represents an instance, and each edge refers to an available assessed correct link between two neighboring instances. The weight of each edge shows the confidence of the link. The likelihood function is an approximate precision of the graph when each correspondence exists, which is a ratio of the number of connected instances in the graph on the total number of edges in the graph. Besides, there is one parameter, *support*, denoting the probability of each correspondence in all available correct links. In the paper, *support* is set to 10 empirically. The paper states that such a value needs to be tuned according to different mining tasks with a small amount of training data.

Hogan et al. [Hogan 2010] build attribute correspondences for interlinking, based on not only inverse functional properties but also functional properties and cardinality restrictions of data sets. Functional property is a property that each subject has a unique object as the property's value. It is possible that several subjects have the same object. Cardinality is used to restrain the number of objects of a subject's property.

Hu et al. [Hu 2011] introduce a method to learn discriminative property-value pairs that identify an instance in a RDF data set with some available assessed correct links. Such a concept of property-value pairs is similar to the definition of inverse functional property that identifies instances uniquely.

To conclude, inverse functional property can be used to build attribute correspondences, which in turn help identify the same instances across two RDF data sets. Intuitively, inverse functional property identifies instances in a RDF data set. There should be two mapped inverse functional properties from both RDF data sets, so that instances of both data sets can be identified and linked. If there is no attribute correspondence that is formed by one inverse functional property of one data set and one inverse functional property of the other data set, the same instances cannot be identified by inverse functional properties.

3.1.1.2 Key

Key is also used to identify instances for interlinking. In relational databases, each table has at least one key. The key identifies each record in a table of database. If keys across two data sets can be matched, the same instances are uniquely identified in both data sets. Then, a link can be built. Related works [Atencia 2012, Scharffe 2012, Pernelle 2013, Song 2011, Song 2013] on key for interlinking are presented as follows.

Pernelle et al. [Pernelle 2013] introduce KD2R, a method that interlinks data sets by two steps. The first step finds out all minimal keys in each class of each data set. The second step discovers mapped keys across two mapped classes of two data sets by referring to a set of available attribute correspondences. This work assume that class correspondences and attribute correspondences across two data sets are available. In order to find out the minimal keys, all combinations of attributes within a class should be considered. Nevertheless, there are $2^n - 1$ attribute combinations to be evaluated, if there are n attributes in a class. In order to reduce the number of evaluated attribute combinations, the method computes two set of attribute combinations before finding out minimal keys. One set is composed of maximal undetermined keys. The other set is composed of maximal non keys. In each undetermined key, there are some attributes that have the same value in at least two instances. Furthermore, remaining attributes within each undetermined key cannot be used to identify instances, because they do not have values in all instances. The paper assumes that if an attribute combination is part of any maximal non key or any maximal undetermined key, it is not a key. Thus, the method only evaluates the attribute combinations that are not part of any maximal non key and any maximal undetermined key. If there are not two instances that have the same attribute values on an attribute combination, such an attribute combination is recognized as a key. The method finally choose the keys that contain the least attributes as the minimal keys of a class.

Atencia et al. [Atencia 2012, Scharffe 2012] broaden the definition of key. Since there are many erroneous, redundant and irregular data in large RDF data sets, not all RDF data sets have keys. The authors define a key to be a property or a group of properties that can identify most of the instances in a RDF data set. This contrasts with the definition of key in relational database, which is a property or a group of properties that can identify each instance in a RDF data set. They call such an extended key *pseudo-key*. The pseudo-key of a RDF data set is discovered in several steps. First, a partition on subjects is computed according to the objects of each property. For each property, there is a subject partitioning. Subjects that have the same object of one property are classified into the same partitioned group. Second, the paper treats any property or a set of properties to be candidate keys. A property is a key only if the number of partitioned groups, which contain only one subject, is larger than a threshold. It means that the property is a key only when it is able to identify a pre-defined proportion of instances. In order to judge whether a property set is a key, an extra operation is needed before counting the number of instances that are distinguished by the property set. The operation is that computing the intersection set of all properties' partitioned groups in the property set. The intersection set consists of the partitioned groups that are distinguished by all properties in the property set respectively. A property set is considered as a key only when the number of partitioned groups, which contain only one subject, in the intersection set is larger than a threshold. It means that only when a property set is able to distinguish a pre-defined proportion of instances in the RDF data set, it is regarded as a key. The definition of *pseudo-key* makes the interlinking work more easily executed on various data sets, especially for the interlinking data sets that do not have any key. Based on the pseudo-keys of each RDF data set, instances are identified with regard to the properties in the key of each data set. If each property of a key in one class can be mapped to a property or several properties in a key in the corresponding class, the same instances are identified by only comparing properties within two pseudo-keys.

Song and Heflin [Song 2011] make use of unsupervised learning to find out keys. There are three measures being used for discovering keys. *Discriminability* is defined as a ratio of a property's range size on the number of triples that contains the property. *Coverage* is a ratio of the number of instances that have a property on the number of instances in the RDF data set. F_L is the harmonic mean of *Discriminability* and *Coverage*.

$$F_L = \frac{2 * Discriminability * Coverage}{Discriminability + Coverage}$$

If a key's F_L is lower than a pre-defined threshold, it will not be regarded as a key. The work searches the key by evaluating each property. If there is no property that can be the key of the data set, each set of any two properties is evaluated. The key-discovering process will continue by evaluating each set of any three properties, if there is no key that consists of two properties. This key-discovering process will stop until the F_L of one property or one property set is higher than the pre-defined threshold. When interlinking data sets, the paper compares instances according to the properties of two keys across two corresponding classes. If each property of a key in one class can be mapped to a property or several properties in a key in the corresponding class, the same instances are identified by only comparing properties within two keys. Yet, the work does not compare all instances. It applies inverted index [Baeza-Yates 1999] to reduce the number of instance comparisons. A set of instances are selected if each of them has at least one property value, which contains a token that appears in the range of one property in the key. Only these instances are compared and evaluated to be the same or not.

[Song 2013] is an improved work of [Song 2011]. The paper assumes that corresponding classes and a set of attributes are pre-known. The method in the paper first computes distinguishable attributes of the instances, which form the keys of each corresponding class. Then, a group of comparable instances are picked out to compare according to the distinguishable attributes. These two steps are similar with the work of [Song 2011]. The improvement is that instances are not only compared on property values, but also compared on relations' URIs or properties' values at the end of each attribute path [Scharffe 2009]. An attribute path is a sequence of attributes. The first attribute of the path must be a relation. The last attribute of the path is a relation or a property. The object of each attribute in the path is the subject of the following attribute. In the implementation of the paper, the depth of each attribute path is 2, which means that only the attribute paths that have one or two relations are used to compare instances. This work does not build attribute correspondences for comparing instances. Each attribute path of an instance in one corresponding class is compared with all attribute paths of another instance in the other corresponding class. The similarity values between paths are computed. The pair of attribute paths that has the highest similarity value of all comparisons is selected. Then, the similarity value is multiplied with a weight. It is the average weight of two attribute paths. The weight of each path is determined by two parameters P and F. The definition of path weight is below.

$$W_{path} = \prod_{i=1}^{depth(path)} P_i * F_i$$

P_i represents the discriminability of a triple i in the path. Its definition is

$$P_i = \frac{\mid set\ of\ distinct\ objects\ of\ property\ i\mid}{\mid set\ of\ triples\ that\ use\ property\ i\mid *P_{max}}$$

P_i is a normalized ratio. F_i represents the importance of a triple to an object node. If the object node has three attributes, F_i equals to $\frac{1}{3}$. If the object node has five attributes, F_i equals to $\frac{1}{5}$. When all similarity values are generated on each selected pair of attribute paths according to the above equations, an average value of all similarity values is computed to represent the similarity of two compared instances. The average value is used to judge whether two instances are a link or not.

To conclude, when key is used for interlinking, both classes of a class correspondence should have at least one key. Furthermore, the two keys must be mapped. In other words, each property of one key should have one or more mapping properties in the other key. It also means that there is a surjective function between the two mapped keys. Assume that there are two corresponding classes about persons in two data sets. One corresponding class has a key containing three properties "identity number", "name" and "birth date". The other corresponding class has a key that also consists of three properties "identity number", "name" and "address". But, the two keys cannot be mapped completely. Specifically, the first two properties of both classes can be mapped as two attribute correspondences, but the property "birth date" cannot be mapped to the property "address". There is no surjective function between the two keys, because there is a property in one key that cannot be mapped to any property in the other key. Therefore, the same instances cannot be identified according to the two keys. Links cannot be built.

3.1.1.3 Conclusion

The techniques of Record Linkage have been used for interlinking RDF data set in many related works. Once the inverse functional properties or keys of two corresponding classes are found, they can be used to identify instances within each data set. If attribute correspondences can be built between inverse functional properties or keys of two corresponding classes, the same instances can be identified by only comparing instances' property values according to the attribute correspondences. But, there is an assumption when applying the two definitions. As for inverse functional property, there should be at least one property correspondence that is composed by two inverse functional properties of two corresponding classes respectively. If there is no attribute correspondence, instances cannot be identified uniquely. As for the key, a similar assumption is held, too. That is, all properties in a key of one corresponding class can be mapped to all properties in a key of the other corresponding class. It means that the alignment of two keys is a surjective function. Otherwise, the two mapped keys cannot help identify the same instances and further build links of two data sets.

3.1.2 Trust Propagation

Interlinking RDF data sets can also be executed by inferring correct links and incorrect links in the RDF graph of the data set with some available assessed links. Such a method is called trust propagation [Cudré-Mauroux 2009, Demartini 2012].

The method in [Cudré-Mauroux 2009] interlinks RDF data sets in a decentralized setting. In such a setting, there are many users who provide assessed correct links, the qualities of which are diverse. The author designs a framework that d-educes the correct links and incorrect links with trust metrics on each user who provides assessed links. Yet, it is not easy to interlink data sets at runtime with this interlinking method, because getting feedback from various users requires a lot of time.

[Demartini 2012] is an improved work based on [Cudré-Mauroux 2009]. It solves the problem of how to combine the links that are generated by an automatic interlinking method with user-provided links. A large collection of users is appealed to help assess the links or provide manual links across RDF data sets. There are two sub-problems that should be taken into account. One is how to merge these "crowdsourcing" [Brabham 2008] links with the links that are generated by machines. The other is how to infer other correct links from these known correct links. Probabilistic reasoning is applied to find out the best way of utilizing links from both sides. The authors state that although their framework does not perform interlinking in real-time, it can be utilized by the automatic interlinking tools to improve the Precision, Recall and F-measure of the generated link set.

To conclude, the works of applying trust propagation on interlinking are likely to provide a blueprint of future interlinking platform. For the sake of interlinking the large data sets on the web, the related works focus on improving the Precision, Recall and F-measure of generated link set by inferring correct links from assessed links. Yet, those frameworks are not easy to be realized at runtime, because a lot of time is required to obtain assessed links from various users when interlinking.

3.1.3 Statistical technique

Statistical technique is also used to interlink RDF data sets [Shen 2005, Suchanek 2012]. If there are many correct links that are formed by instances that have similar values of two attributes, these two attributes can be recognized as an attribute correspondence. It is easy to infer that if two attributes' values share similar features or constraints, the two attributes probably forms an attribute correspondence. Suchanek et al. [Suchanek 2012] design a framework that involves interlinking and ontology matching. The two processes provide feedback for improving the results of each other. Attribute correspondences are computed by referring to the probabilities of equivalent instances and equivalent property values. The work mentioned in [Shen 2005] takes advantage of the Expectation Maximization algorithm [Dempster 1977] and the relaxation labeling algorithm [Marshall 1992] to find out semantic constraints of attribute values, so as to build attribute correspondences. Besides, Hu et al. [Hu 2011] and Hogan et al. [Hogan 2010] use statistical technique to find out frequently-appeared attribute correspondences in assessed correct links. Statistical technique also acts as an auxiliary when other techniques are used to link RDF data sets [Demartini 2012, Niu 2012, Cudré-Mauroux 2009].

To conclude, statistical technique is used to summarize value features or constraints of attributes or discover similar instances and property values in order to build attribute correspondences or extract attribute correspondences from available assessed correct links. It is also used with other techniques to interlink RDF data sets. The weak point of the statistical technique is that usually the infrequent information cannot be discovered by the statistical technique, because the statistical technique usually extracts frequently-appeared information. For example, assume that there are 10 assessed correct links. 9 links contain an attribute correspondence AC_1. 1 link contains an attribute correspondence AC_2. Both of the two correspondences can be extracted from assessed correct links. But, the correspondence AC_1 is more likely to be recognized as an attribute correspondence than the correspondence AC_2 by the statistical technique, because most of the assessed correct links contain the correspondence AC_1. The mining process of [Hu 2011] may ignore less frequent attribute correspondences that are contained in some correct links. It will in turn cause the interlinking framework not to generate some correct links that contain such less frequent attribute correspondences.

3.1.4 Ontology Matching

Ontology Matching is used to discover attribute correspondences for comparing instances [Fan 2012]. It is more likely to build link with instances from corresponding classes. It is also more easy to compare two instances by comparing their corresponding attributes' values.

Ontology matching is the process of defining corresponding classes, properties, and relations between two ontologies [Euzenat 2007]. Basically, methods of ontology matching can be classified into two categories. The first one is element-level matching and the other one is structural-level matching. Element-level methods align the ontologies by analyzing each entity and their instances independently, without considering the relations with other entities and instances [Euzenat 2007]. In contrast, structural-level methods align the ontologies by analyzing the structures of ontologies as well as the relations among entities [Euzenat 2007]. The existing literature [Euzenat 2007] shows that structural-level techniques are often more efficient. This is due to the fact that the structural-level methods match ontologies by comparing the relations among entities and recursively propagating the similarity within neighboring entities. Accordingly, structural-level methods have been extensively investigated for years.

In the literature, there are many existing structural-level techniques, such as graph-based technique, taxonomy-based technique, repository-of-structure based technique, model-based technique, data-based technique, and statistics-based technique. Graph-based technique treats each of the two ontologies as a labeled graph (or weighted graph) and tries to match their entities based on the positions in the graphs. Taxonomy-based techniques not only treat an ontology as a labeled graph but also take into account the taxonomical relationships between concepts. Repository-of-structure-based techniques matches the ontologies by estimating the similarity of two ontologies' structures. Model-based technique matches the ontologies based on their similar interpretations. The techniques based on data-based and statistics both match ontologies by leveraging the population or distribution of instances.

Among all techniques of structural-level ontology matching, data-based techniques are more useful for interlinking, because the corresponding attributes that are used for interlinking should be comparable. Compared attributes of two interlinked instances that should have overlap ranges. The interlinked instances must have some similar attribute values of those attributes, which can form attribute correspondences. In one sentence, attribute correspondences that are used for interlinking should be built according to the values of attributes. Therefore, ontology matching that applies data-based techniques are introduced primarily in this section.

Attribute correspondences can be discovered by analyzing instances or available correct links. Since attributes of each attribute correspondence have overlapping ranges, instance-based ontology matching is used to find out attribute correspondences [Dhamankar 2004, Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005, Tran 2011, Qin 2007]. Some interlinking methods extract attribute correspondences from available correct links [Nikolov 2010, Ngonga Ngomo 2011c, Isele 2013, Niu 2012].

3.1.4.1 Instance-based Ontology Matching

In this subsection, we first discuss the related works of instance-based ontology matching systems, and then present related works of instance-based ontology alignment algorithms.

There are many instance-based ontology matching systems. T-tree [Euzenat 1994], CAIMAN [Lacher 2001], FCA-merge [Stumme 2001], GLUE [Doan 2004], SBI [Ichise 2003, Ichise 2004] and oPLMap [Nottelmann 2006] only build class correspondences [Euzenat 2007]. iMAP [Dhamankar 2004], Automatch [Berlin 2002], Kang and Naughton [Kang 2003], Dumas [Bilke 2005] and sPLMap [Nottelmann 2005] are works that discover attribute correspondences.

iMAP [Dhamankar 2004] introduces a semi-automatic matcher that finds out attribute correspondences. It employs a set of *searchers* to find out potential correspondences according to attributes' types. There are totally five searchers: Numeric Searcher, Category Searcher, Schema Mismatch Searcher, Unit Conversion Searcher, and Date Searcher. Numeric Searcher finds out correspondences that are composed of numeric properties. Category Searcher exploits correspondences that are composed of category properties that either have boolean values or have a range with a few numerated values. The Schema Mismatch Searcher searches correspondences that are composed of properties and relations. The Unit Conversion Searcher discovers correspondences that require conversions on property values. The Date Searcher finds out correspondences that are composed of date properties. Each searcher only maps properties and relations that satisfy the types of the searcher. Once the 1-*to*-1 correspondences are constructed, K correspondences with the highest similarities are selected to construct complex correspondences. Such a method of constructing complex correspondences is called *beam search* [Russell 1996]. The matching process will stop when the difference between the highest similarity of the current round and the one of the last round is below a pre-specified threshold δ. Furthermore, the ontology matcher of this paper also exploits a group of context information, such as past complex matches, domain integrity constraints and overlap data, to help evaluate candidate attribute correspondences. The past complex matches are the correspondences of similar or relevant schemas. The domain integrity constraints are the restrictions of the attributes in the mapped schemas. The overlap data refer to the correct links that are generated as a by-product by the five searchers introduced above. Based on the context information above, the method proposes a user-interaction process to evaluate candidate attribute correspondences. There are two features of this system that prevents it from being applied for discovering attribute correspondences of interlinking at runtime. First, the ontology matching method in this system relies on some available information, such as past complex matches and domain integrity constraints, which takes time to be obtained. Second, the ontology matching system also should spend some time to tune the pre-specified threshold δ.

Automatch [Berlin 2002] is a matcher that discovers property correspondences. The paper assumes that an *attribute dictionary* is provided by domain experts, in which each attribute is represented by a set of selected values from their ranges and a estimated probability. The value that are selected should contain the features of the properties. The authors utilize three feature selection [Guyon 2003] algorithms. They are Mutual Information, Information Gain, and Likelihood Ratio. When matching two ontologies, they first match the properties of each ontology to the attributes in the dictionary. Each property correspondence has a score, which is a probability of the set of selected values that exist in both properties of the correspondence. Based on the score, attributes from one class can be mapped to the ones from another class which are also mapped to the same attribute in the dictionary. They are potential property correspondences. Afterwards, the authors introduce a directed graph to describe the alignments between two mapped ontologies and the dictionary, so as to apply Minimum Cost Maximum Flow network [Ahuja 1993] for picking out the final property correspondences. The reason of describing the potential property correspondences into a directed graph is below. If the property correspondence is picked out when it has the highest score among all property correspondences mapped to the same property in the dictionary, it will easily lead to ambiguous property correspondence. The weak point of this ontology matcher is that it relies on one external information, attribute dictionary, to fulfill the matching task. However, such an attribute dictionary takes time to be obtained from domain experts. Thus, this ontology matching system cannot discover attribute correspondences for interlinking at runtime.

Kang and Naughton [Kang 2003] introduce a matching method based on instances, which maps opaque properties that are hard to be interpreted. This matching method builds correspondences by exploiting the dependencies between properties. They make use of two concepts of information theory [Cover 1991], *Entropy* and *Mutual Information*, to discover the dependencies between properties. Entropy is a measure that presents uncertainty of a property's values. Mutual information is a measure that presents the reduction of a property's uncertainty when the information of another property is given. It shows the amount of information, which is contained in one attribute, of another attribute. Afterwards, the matching method construct a graph with all properties within a class. Each node in the graph presents a property. Each edge represents a dependency relationship between two properties. All nodes and edges have weights. A node's weight is the entropy. An edge's weight is the mutual information of two connected properties. Property Correspondences are constructed according to the graph structure of both ontologies by a graph matching algorithm. The algorithm is implemented to optimize the distance metric that is computed on entropy and mutual information of properties. To conclude, such an ontology matching system builds correspondences by comparing dependencies between properties across data sets' ontologies. Nevertheless, the dependencies do not always exist in each ontology. Thus, this ontology matching system cannot provide correspondences for ontologies that do not have dependencies between properties. It also means that this matching system cannot provide correspondences for all interlinking tasks.

Dumas [Bilke 2005] presents an ontology matching algorithm that extracts property correspondences from some duplicate instances. The algorithm first generates K duplicate instances from two mapped ontologies. The method treats the property values of one instance as a string. Each token of the string is analyzed with a computed weight based on TFIDF scheme [Euzenat 2007]. There are two advantages of using TFIDF. First, it is order-independent. Second, it will give high weights to the infrequent tokens that have higher identifying power. Then, the WHIRL algorithm [Cohen 1998] is used to find out several fuzzy duplicate instances. The algorithm selects duplicate instances that have tokens of high TFIDF weights. Then, the property correspondences are extracted by comparing each property value from one duplicate instance with each property value from the other duplicate instance. The weak point of this ontology matching system is that it only discovers property correspondences. It cannot discover relation correspondences. Thus, this system is not applicable for the interlinking tasks that require relation correspondences for comparing instances.

sPLMap [Nottelmann 2005] builds property correspondences across different ontologies based on Horn Predicate Logic and Probability Theory. The system uses several classifiers to predict the probability of each instance that has one property when the instance has another property. Then a set of potential property correspondences are generated based on the probability of each instance that contains the properties. The most likely property correspondences are selected if their probabilities are the highest ones among all correspondences. This system is subject to two assumptions. First, all data types of properties are known in advance. Second, data transformation operations are also known between any two data types. Therefore, such a matching system cannot provide attribute correspondences for the interlinking task, if the data types of properties and data transformation operations of data types are not known before interlinking.

In addition to instance-based ontology matching systems, there are also many instance-based ontology alignment algorithms. Not all algorithms detect attribute correspondences across data sets, such as [Duan 2012] and [Schopman 2012]. Our analysis focuses on the works of detecting attribute correspondences.

In [Tran 2011], the authors design a clustering method for ontology matching.

They define five measures to compute the similarities between classes and properties. They are string edit distance of classes names' lexical similarity, semantic similarity of classes names' in WordNet, profile similarity of classes' id, label and comments, structure of classes, and instances of classes. Similarities of classes and properties are computed according to each measure. After that, this ontology matching method uses K-means [Jain 1999], which is a widely-used clustering method in many domains [Oyelade 2010, Fausett 2013, Ray 1999, Sree 2014, Wang 2005, Lu 2004, Broder 2014, McCallum 2000, Di 2013, Xing 2003, Lu 2012], to classify the compared class/property pairs of each measure into two groups: matching or non-matching. Then, the matching pairs will be extracted to be final correspondences according to the occurrence of each matching pair in the matching group of each measure. However, this ontology matching method cannot discover relation correspondences. Therefore, such an ontology matching method cannot be applied in the interlinking tasks that require relation correspondences for comparing instances.

Qin et al. [Qin 2007] design an iterative process to find out both class correspondences and attribute correspondences. There are two steps in the matching process: finding potential correspondences and verifying correspondences by counting the number of links that contain the correspondences. The first step finds potential correspondences by applying three methods. Class correspondences are discovered by comparing names of the classes. Property correspondences are discovered by comparing both names and data types of the properties across two corresponding classes. Relation correspondences are extracted by referring to the correct links that are produced by the method introduced in [Dong 2005]. Once a set of potential correspondences are built, correspondences are organized into groups. Each group is composed of a class correspondence and all related attribute correspondences. The paper utilizes and optimizes a query strategy named FARMER [Nijssen 2003] to evaluate the correspondences. A threshold K is given. If there are more than K links that contain the correspondences in each query, all correspondences in the query are the final correspondences. This matching method relies on some available correct links for building relation correspondences. However, available correct links are difficult to obtain, since users need to compare all instances of one corresponding class with all instances of the other corresponding class. It means that users should do many comparison on instances, in order to find out correct links of two corresponding classes. Therefore, such an ontology matching method cannot be applied for interlinking at runtime.

There is also an ontology matching work that do not generate ontology alignment from scratch [Rivero 2011]. [Rivero 2011] assumes there are a set of available correspondences. The work makes use of chase algorithm [Deutsch 2008] to produce a closure of the available correspondences. For example, if attribute A is corresponding to attribute A', then A's subject class C should correspond to A''s subject class C'. However, the assumption of the method is that some correspondences are already given before the matching process. There should be at least one available attribute correspondence. Otherwise, there will be no class correspondence that can be generated by this work. Thus, such an ontology matching method is not applicable for the interlinking task, if there is no available attribute correspondences across ontologies of the two interlinking data sets.

To summarize, instance-based ontology matching is a commonly used method for discovering attribute correspondences of two data sets for interlinking. There are many instance-based ontology matching works, but only some of them produce attribute correspondences [Dhamankar 2004, Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005, Qin 2007]. Among which, most of the works only generate property correspondences, while do not generate relation correspondences [Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005, Tran 2011]. The works that are able to generate relation correspondences all require external information to fulfill the matching process [Dhamankar 2004, Nottelmann 2005, Qin 2007, Rivero 2011], which is hard to help interlinking data sets at runtime.

3.1.4.2 Extracting Correspondences from Available Correct Links

There are interlinking works that extract attribute correspondences from available correct links [Nikolov 2010, Ngonga Ngomo 2011c, Isele 2013, Niu 2012, Hu 2011].

[Nikolov 2010] is a work that extracts class correspondences and property correspondences from available correct links. It first computes a transitive closure on all available correct links. Second, this work classifies instances of each available correct link according to their belonging classes and the classes of the instances that are linked to. For example, assume that there is a correct link i *owl:sameAs* i'. i is an instance of class C. i' is an instance of class C'. Thus, the instances i and i' are classified into the group of the class C. They are also classified into the group of the class C'. The similarity of two classes are computed according to the equation below.

$$\mathrm{sim}(A, B) = \frac{\mid c(A) \cap c(B) \mid}{\min(\mid c(A) \mid, \mid c(B) \mid)}$$

where $c(A)$ and $c(B)$ are classified groups of class A and class B respectively. $\mid c(x) \mid$ is the number of instances in the group $c(x)$. The similarity of two classes is a ratio of the number of instances that exist in both classified groups $c(A)$ and $c(B)$ on the minimum number of instances between the classified groups $c(A)$ and $c(B)$. The similarity of two properties are computed according to the equation below.

$$\mathrm{sim}(r_1, r_2) = \frac{\mid c(X) \mid}{\mid c(Y) \mid}$$

In the set $c(X)$, there are two instances that have equivalent values of property r_1 and property r_2; $c(Y)$ is a set of instances which have values of property r_1 or property r_2. The similarity of two properties is a ratio of the size of $c(X)$ on the size of $c(Y)$. If the similarity value of two classes/properties is above a pre-defined threshold, the two classes/properties can form a correspondence. This ontology matching method cannot produce relation correspondences.

Ngonga Ngomo et al. [Ngonga Ngomo 2011c] extract class correspondences from available correct links. This work maps two classes if there are two instances from both classes that can form an available correct link. It maps two properties if the two properties have overlapping ranges. The work does not build relation correspondences.

Isele and Bizer [Isele 2013] extract property correspondences from available correct links. The work maps two properties that exist in two instances of any available correct link, if the two properties have overlapping ranges. The method first tokenizes and lowercases each property value. Then, similarity measure *Levenshtein* is used to compare one property's range with another property's range. If there are two tokens that appear in both ranges, the two properties form a property correspondence. The work does not build relation correspondences from available correct links.

Niu et al. [Niu 2012] also map classes and properties with available correct links. Their work is introduced in Section 3.1.1.1.

Hu et al. [Hu 2011] apply association rule to mine frequently-used attribute correspondences from available correct links. The attribute correspondences whose attribute ranges are not overlapping will be deleted from the set of candidate attribute correspondences. The rules are built based on the frequency of the attribute correspondences in the available correct links with respect to a pre-defined threshold. In the paper, the threshold is set to 0.98 (98% of correct links). The threshold should be tuned according to different ontology matching tasks.

To conclude, all these related works extract correspondences from some available correct links. They assume that users are able to provide them some correct links. Nevertheless, it is not easy for users to provide correct links. Assume that there is a class correspondence across two RDF data sets. There are m instances of one corresponding class. There are n instances of the other corresponding class. Thus, users should compare $m*n$ pairs of instances to find out the same instances in order to build links. It is a heavy task for users.

3.1.4.3 Conclusion

To summarize, instance-based ontology matching methods and the methods that extract correspondences from available correct links are all used to discover attribute correspondences across two correspondences classes in two data sets. However, both two kinds of methods have their weak points. As for instance-based ontology matching methods, only a few works produce attribute correspondences [Dhamankar 2004, Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005, Qin 2007]. Among which, works that generate property correspondences cannot generate relation correspondences [Berlin 2002, Kang 2003, Bilke 2005, Nottelmann 2005]. The works that generate both property correspondences and relation correspondences all require external information [Dhamankar 2004, Nottelmann 2005, Qin 2007]. Those external information usually take some time to be obtained. Therefore, there is no instance-based ontology matching work that can be applied for discovering attribute correspondences at runtime for the interlinking process. Besides the works of instance-based ontology matching, some interlinking methods [Nikolov 2010, Ngonga Ngomo 2011c, Isele 2013, Niu 2012] build attribute correspondences based on some available correct links. Yet it is non-trivial to collect correct links from users, because users should compare a lot of pairs of instances so as to find out correct links across two RDF data sets. Therefore, extracting attribute correspondences from available correct links also cannot help interlinking data sets at runtime.

3.1.5 Machine Learning

Some Machine Learning methods are also used to find out links across data sets. The interlinking process is not completed successfully with a first interlinking pattern, because usually the first interlinking pattern does not cover enough correct links. It means that the interlinking pattern should be improved iteratively. A general method is selecting a small set of assessed links. By learning the assessed links, the interlinking pattern are improved so as to generate more correct links. There are a group of related works that make use of machine learning [Nikolov 2008, Shen 2005, Hu 2011, Niu 2012, Song 2011, Isele 2011a, Ngonga Ngomo 2011c, Ngonga Ngomo 2011a, Rong 2012, Nikolov 2012, Ngonga Ngomo 2013, Isele 2013] for improving the interlinking pattern.

Genetic Programming is a supervised learning method that is used for learning interlinking patterns in some interlinking methods [Ngonga Ngomo 2012b, Isele 2013, Ngonga Ngomo 2013, Isele 2011a]. It leverages two biological concepts, *crossover* and *mutate*, in the learning process, so as to improve the interlinking patterns by making changes on fragments of the interlinking patterns [Banzhaf 1998]. This learning method first generates k interlinking patterns, where k is a number that is defined before the learning process. Second, these interlinking patterns are evaluated with the F-measures of their generated link sets. Third, $k - l$ interlinking patterns are selected to be improved in the next round if their evaluations are the best ones of all interlinking patterns. The selected patterns are changed by crossing over some randomly selected fragments with each other, or by mutating some randomly selected fragments. Fourth, the changed interlinking patterns are evaluated again, and only $k - 2 * l$ patterns are selected to be improved in the next round. The algorithm will stop until there are no more than l patterns left.

A genetic programming algorithm is proposed in the interlinking method of [Ngonga Ngomo 2012b]. The method changes the selected interlinking pattern with all possible aggregation methods like "average" and "max" for constructing a better interlinking pattern. Nevertheless, Genetic Programming is not a method suitable for interlinking at runtime. In each learning round, all candidate interlinking patterns should be evaluated. The method of evaluating the interlinking patterns is to interlink the data sets with the interlinking patterns and compute the Precision, Recall and F-measure of the generated link sets. When interlinking data sets with each interlinking pattern, the machine should query the two interlinking data sets at least once in order to get instances and instances' attribute values. The queries require I/O operations of the machine, which take a lot of time. It means that a lot of time will be spent on link generation.

One of the most recent works on interlinking is proposed by Ngonga Ngomo et al. [Ngonga Ngomo 2013]. The method uses not only Genetic Programming but also Active Learning to construct and improve the interlinking pattern of two RDF data sets. Active Learning is a learning strategy that actively chooses unlabeled examples which bring the biggest information and change to the classifier [Alajlan 2014, Fu 2013, Druck 2011, Ebert 2012, Rashidi 2011, Guyon 2012, Settles 2009, Saar-tsechansky 2004, John 1996, Lewis 1994, Brain 2003, Dagan 1995]. Usually Active Learning is embedded in the learning process of other learning methods. Active Learning helps speed up the learning process when there are a large amount of assessed examples to be learned. The interlinking approach in this paper outperforms many other solutions like [Isele 2011a] and [Ngonga Ngomo 2012b]. The F-measure of generated link set grows higher with the same amount of assessed links than [Isele 2011a] and [Ngonga Ngomo 2012b].

Another recent work on interlinking by learning the interlinking pattern is proposed by Isele et al. [Isele 2013]. It utilizes Genetic Programming to initialize a set of candidate interlinking patterns. A variety of changes are permitted to be applied on the selected patterns when crossing over and mutating fragments of the patterns. Moreover, it exploits two active learning strategies, query-on-committee [Seung 1992] and uncertainty learning [Fu 2013], to select the most informative sample links and send to users to assess. Its convergence speed on many data sets is very fast.

[Ngonga Ngomo 2011c] is another interlinking work that uses Active Learning for interlinking. The author selects some sample links for users to assess so as to extract a set of class correspondences and property correspondences for comparing instances. Then, the authors design a linear classifier and a boolean classifier to distinguish correct links and incorrect links that are assessed by users. A linear classifier is composed of an attribute correspondence. The boolean classifier is a conjunction of several linear classifiers. It means that the interlinking classifiers in this work only can generate correct links by a set of attribute correspondences. As for each link, there are two instances. If a pair of attribute values across the two instances are the same, we assume that the two attributes form an attribute correspondence in the link. So if a link have the same set of attribute correspondences that is contained in the boolean classifier, such a link can be generated as a correct link. In other words, the interlinking classifiers of this work cannot produce the correct links that do not have the same set of attribute correspondences that is contained in the boolean classifier. However, there are many interlinking tasks whose correct links do not share the same set of attribute correspondences. Therefore, this method cannot interlink the data sets that cannot be classified by such a boolean classifier.

In addition to the works on generating and improving the interlinking pattern for interlinking, there are works on improving the parameters of the interlinking pattern, such as, transformation between property values [Ngonga Ngomo 2012a]. However, only these works are not able to generate an interlinking pattern for interlinking.

To conclude, most works of interlinking RDF data sets with Machine Learning focus on constructing and improving the interlinking pattern. These works all have their weak points. The interlinking method of [Ngonga Ngomo 2011c] cannot be applied for all kinds of interlinking tasks, because its classifiers only can generate correct links that have the same set of attribute correspondences. They cannot generate links for the interlinking tasks that have several sets of attribute correspondences. The works that make use of Genetic Programming take higher running time for improving the interlinking pattern [Ngonga Ngomo 2012b, Isele 2013, Ngonga Ngomo 2013, Isele 2011a]. Therefore, a learning method that costs shorter time and is able to generate interlinking patterns for all interlinking tasks is required.

3.1.6 Conclusion

Interlinking RDF data sets is executed by many techniques. They are Record Linkage, Trust Propagation, Statistical Technique, Ontology Matching and Machine Learning. As interlinking is similar to the record linkage in Database, researchers usually leverage key and inverse functional property for identifying instances when comparing instances. Meanwhile, from the perspective of web, interlinking is treated as a decentralized task that finds links across data sets by reasoning on available correct links either generated by machines or provided by users with regard to the trust of link sources. Such a method is called Trust Propagation. Statistical techniques mostly assist other techniques to extract correspondences and enhance the Precision, Recall and F-measure of the generated link set. Furthermore, Ontology Matching is used to generate correspondences for comparing instances of the interlinking process. Machine Learning is applied for interlinking by constructing and improving the interlinking pattern.

According to the analysis, no technique can solve the interlinking problem completely. All these different techniques have pros and cons, so they are better to be used together. In the following section, they are presented in their corresponding involved tasks respectively.

3.2 Discovering Attribute Correspondence

Discovering attribute correspondence is highly required for comparing instances when interlinking two RDF data sets. During the interlinking process, any two instances across two different data sets should be compared on their attribute values, so that they can be judged to be the same or not. Intuitively, any two instances that come from classes across which there is a class correspondence are likely to be identical to each other. Besides, any two instances that hold more similar attribute values are more likely to be links than the ones with less similar attribute values. Most of the time, similar attribute values exist in the two instances across which there is an attribute correspondence. Therefore, an ontology alignment is better to be explored first before comparing instances. It is necessary to narrow the instances to be compared to a small range, so as to avoid comparing attributes which are not corresponding with each other.

Related works that are introduced in Section 3.1.1, Section 3.1.3 and Section 3.1.4 all make contributions to discover attribute correspondences. Nevertheless, each of them has weak points.

- Key and Inverse Functional Property: In order to build attribute correspondences for interlinking, there should be a surjective function between one key/inverse functional property of a class with one key/inverse functional property of the corresponding class. Otherwise, the same instances cannot be identified and further form links.

- Statistical Technique: Attribute correspondences that do not appear frequently may not be discovered by Statistical Technique.

- Instance-based Ontology Matching and Correspondences Extraction from Available Correct Links: These works require external information or available correct links for discovering attribute correspondences, which takes time to obtain or demand users to compare many pairs of instances.

By introducing related works of discovering attribute correspondences, we find that the interlinking process requires a correspondence discovering method that can produce attribute correspondences at runtime according to instances' attribute values without requiring external information.

3.3 Generating Links

Since we are going to learn the interlinking pattern for two data sets iteratively with assessed links, a set of assessed links should be obtained in advance. Most of research assumes that users are able to provide a portion of correct links as the assessed links, such as [Nikolov 2010, Ngonga Ngomo 2011c, Isele 2013, Niu 2012]. In general, letting users themselves to find correct links is a huge burden for users. They should compare all instances of one corresponding class with all instances of the other corresponding class. There are many comparison that should be done by users. Therefore, it is better to provide a set of sample links for users to assess.

Isele and Bizer [Isele 2013] propose a method to produce a group of sample links for users to assess. This sample link production algorithm queries all values for each property in both data set. Each property value is separated into several tokens according to the MultiBlock blocking method of [Isele 2011b]. An index is created for each token that appears in any property value of either data set. If two instances both fall into at least 5 indexes, the two instances are chosen to construct a sample link. It is a method that can create a set of sample links with a fast speed. However, only a few property values of each instance are useful for comparing instances. There are only a small portion of property values in each instance that are useful to judge whether two instances are the same or not. They are property values of attribute correspondences. Thus, the link production method of this paper may do a lot of computation on building indexes for tokens that only appear in the ranges of properties that do not have corresponding properties. These tokens will in turn help create a lot of sample links that are not correct across two corresponding classes. These sample links will be useless for constructing and improving the interlinking pattern, because they do not contain any attribute correspondence.

Ngonga Ngomo et al. [Ngonga Ngomo 2012b, Ngonga Ngomo 2013] generate sample links by comparing instance's property value of "rdfs:label" or some predefined interlinking pattern. Such a sample link process cannot be applied in all kinds of interlinking tasks, because not all data sets have the property "rdfs:label" or a predefined interlinking pattern.

Therefore, we need a sample link generating process that produce less irrelevant sample links that do not contain attribute correspondences and can be applied for different interlinking tasks without external information.

3.4 Constructing and Improving the Interlinking Pattern with Assessed Links

After attribute correspondences are obtained, they need to be organized into an interlinking pattern to generate links by learning assessed links with a learning method. Thus, this section is to illustrate how to build an interlinking pattern.

It is not a straight-forward process to construct an interlinking pattern that is able to produce correct links. [Song 2013, Ngonga Ngomo 2011c] build an interlinking pattern by combining all attribute correspondences into a conjunction of all attribute correspondences. Yet, it is not a universal interlinking pattern that can generate correct links for all interlinking tasks. Here is an example. Assume that we interlink two data sets D and D'. There are two correct links l_1 and l_2. Link l_1 has three attribute correspondences AC_1, AC_2 and AC_3. It means that attribute values of each attribute correspondence across two linked instances of the link l_1 are the same. Moreover, link l_2 has an attribute correspondence AC_3. If the interlinking pattern is a conjunction of attribute correspondences AC_1, AC_2, and AC_3, the link l_2 cannot be generated. The reason is that the link l_2 does not have all attribute correspondences in the interlinking pattern. Since it is hard to estimate which and how many sets of attribute correspondences are required for constructing an interlinking pattern, we need some assessed links to learn the sets of attribute correspondences.

The reasonable way of constructing an interlinking pattern is to improve the pattern iteratively with assessed links. It is a supervised learning process. Thus, there are two elements for constructing an interlinking pattern. One is a set of assessed links, the other is a supervised learning method. The related work of producing the assessed links is illustrated in Section 3.3. The related works of constructing and improving the interlinking pattern are introduced in Section 3.1.5. According to the analysis in Section 3.1.5, related works of constructing and improving the interlinking pattern by learning have several weak points. Some interlinking methods cannot be applied for all kinds of interlinking tasks. Some interlinking methods take longer running time. Therefore, a learning method that costs shorter running time and is able to generate interlinking patterns for all interlinking tasks is required to construct and improve the interlinking pattern.

3.5 Conclusion

From the analysis above, Record Linkage, Trust Propagation, Statistical Techniques, Ontology Matching and Machine Learning are all used to interlink RDF data sets. Ontology Matching and Machine Learning are two primary techniques for interlinking. The reasons are below.

- First, Ontology Matching is needed to discover attribute correspondences across two corresponding classes. Interlinking is the process which aims to find out links by comparing instances. URIs of instances are difficult to compare due to the naming differences, so links are generated by comparing instances' attribute values. Nevertheless, there are a lot of comparisons of attribute values across two compared instances. Most of the comparisons are useless for evaluating the similarity of two instances, because these comparisons are executed on attributes that are not corresponding with each other. Accordingly, discovering attribute correspondences is a necessary step.

- Second, Machine Learning is used to improve the linking pattern. It is not enough to compare instances with attribute correspondences to find out correct links, in that different correct links usually have different attribute correspondences. The interlinking pattern needs to be constructed and improved with assessed links by a supervised machine learning method, so as to cover more correct links.

The related works on interlinking by applying Ontology Matching and Machine Learning still have some points to improve.

- As for the works on Ontology Matching, instance-based ontology matching requires external information to find out attribute correspondences. Other related works on interlinking rely on some available assessed links provided by users to extract attribute correspondences. Both kinds of related works take long running time to produce attribute correspondences, because external information and available assessed links take time to be obtained. Thus, it is required to design an ontology matching method that discovers attribute correspondences at runtime without external information.

- As for the machine learning techniques that are used for interlinking, most related works utilize Genetic Programming. Genetic Programming costs long running time on evaluating candidate interlinking patterns when improving the interlinking pattern. Hence, the interlinking process requires a supervised learning method that can improve the interlinking pattern with short running time and is able to generate interlinking patterns for all interlinking tasks. Furthermore, in order to improve the interlinking pattern, a set of sample links should be generated for the user to assess. The related works on generating sample links either produce many irrelevant sample links that are useless for constructing and improving the interlinking pattern, or cannot be applied for all interlinking tasks. Therefore, the required link production method should produce fewer irrelevant sample links and can be applied for all interlinking tasks.

Next chapter will introduce the solutions of the three tasks for the interlinking problem of this thesis. They are:

- Discovering attribute correspondences by classifying attributes of each class according to attributes' value features with the K-medoids clustering method and matching attributes with regard to the clustered groups that share similar value features.

- Generating links by constructing a sample interlinking pattern with a disjunction of discovered potential attribute correspondences and sending sample links to users for assessing.

- Constructing and improving the interlinking pattern of two RDF data sets with assessed links and the Version Space learning method.

Chapter 4

Proposed Solution

Contents

For the sake of overcoming the weak points of the related works introduced in Chapter 3, an interlinking method is proposed for the interlinking problem of this thesis, especially for all three tasks, to be discussed briefly in Section 4.2, Section 4.3 and Section 4.4 respectively.

4.1 The Interlinking Process

The interlinking process can be fulfilled by running an interlinking script that is transferred from an interlinking pattern with a semi-automatic interlinking tool. As it is introduced in Chapter 2, an interlinking pattern can be constructed to help compare instances' attribute values and generate links of two RDF data sets. The interlinking pattern is composed of the set of attribute correspondences of each correct link. Furthermore, it aggregates all these sets together. With the interlinking pattern, we can design a JAVA program to compare instances' attribute values according to the attribute correspondences in the interlinking pattern and aggregate all similarities of compared attribute values into one final value. The final value represents the similarity of two compared instances. Then, we can evaluate whether two compared instances are the same or not according to the final value. Nevertheless, there are already some tools that have the same function of such a program, such as semi-automatic interlinking tool Silk[1] and LIMES[2]. The tools require users to specify an interlinking script that expresses the interlinking pattern in a specific syntax as well as a group of comparison methods for different property data types. In this thesis, we choose Silk to generate links since it is a open-source

[1] http://www4.wiwiss.fu-berlin.de/bizer/silk/

[2] http://aksw.org/Projects/LIMES.html

tool. Furthermore, Silk provides a rich set of comparison methods and aggregation methods.

To conclude, in order to find out links across two RDF data sets, we should first find out all attribute correspondences across two corresponding classes. We should second aggregate all attribute correspondences into an interlinking pattern that distinguish correct links and incorrect links. Finally, we should transfer the interlinking pattern into an executable interlinking script, so that Silk can generate a link set of the two RDF data sets.

4.2 Discovering Attribute Correspondences

The interlinking process requires an ontology matching method that discovers attribute correspondences across two corresponding classes at runtime without external information. According to the analysis of discovering attribute correspondences by instance-based ontology matching works and other interlinking works in Section 3.1.4 of Chapter 3, either extracting attribute correspondences from available correct links or matching attributes with regard to external information is not applicable for the interlinking process. Thus, an intuitive method is to match all attributes of a corresponding class with all attributes of the other corresponding class.

It is difficult to build attribute correspondences across two corresponding classes by matching all attributes of one corresponding class with all attributes of another corresponding class. From one hand, it is hard to select a threshold of similarity to evaluate whether two attributes are similar enough to form a correspondence. If the threshold of similarity is set to a high value, there will be a lot of attribute correspondences being undiscovered. If the threshold of similarity is set to a low value, there will be a lot of attribute pairs that are not corresponding being produced as attribute correspondences. From the other hand, there will be a big number of comparisons on attributes for detecting attribute correspondences. The attribute correspondences are not only composed of properties of the corresponding classes, but also composed of attribute paths of the corresponding classes. An attribute path is a sequence of attributes. It begins with one relation and ends with either one relation or one property. The object of each attribute in the path is the subject of the following attribute. Since URIs are difficult to be compared for finding links, we do not consider attribute paths that end with a relation. We only consider attribute paths that end with a property. For example, in data set *INSEE*, there is an attribute path for the class *insee:Departement*, which is *insee:subdivision*$^{-1}$*/insee:nom*. It denotes the name of an instance whose subdivision is an instance of the class *insee:Departement*. Assume that there are m properties and attribute paths of one corresponding class. There are n properties and attribute paths of the other corresponding class. Consequently, the number of attribute comparisons is $m \times n$. It is not applicable to detect correspondences by comparing each property/attribute path of one corresponding class with each property/attribute path of the other corresponding class.

Classification is one strategy to avoid many comparisons on attributes. If attributes are classified by their features, the attribute correspondences will only be discovered among attributes that share similar features. There are many kinds of features, such as structure feature and value feature. Since we are going to find out corresponding attributes that have overlapping ranges, it is intuitive to use attributes' value features for classification. For example, the value feature of properties that denote email addresses is that each property value contains a symbol @. The value feature of numerical properties is that there is no letter in each property value except e and E. Thus, if we classify attributes according to their value features, we only need to compare attributes with similar value features. Consequently, the properties that denote email addresses will not be mapped to numerical properties like people's age. Therefore, attribute pairs that are not corresponding will be largely reduced.

Clustering is a strategy for classification. It is a classification method that requires no external information. For the interlinking method of this thesis, K-medoids clustering is utilized to classify properties and attributes paths into several groups according to value features. K-medoids clustering algorithm is a clustering algorithm that is easy to implement. It is applicable for clustering large data sets. What is important is that it is tolerable when there is irregular data in the data set. Irregular data is a widely-existing problem of RDF data set. Irregular data may contain spelling or punctuation errors. It also may be incomplete or outdated data. As for the clustering task on attributes, there is no special requirement on the shape of clustered groups. Thus, there is no need to apply other clustering methods.

K-medoids is an improving clustering method based on K-means. K-means is a clustering algorithm that classify a set of sample points in the Euclidean space into several groups. It selects the center of each clustering group by computing the average coordinate of all sample points in the group. The K-means clustering centers are easy to be impacted by the irregular data. Since irregular data usually have different features with other regular sample data, its coordinate in the Euclidean space is usually far away with the ones of other regular points. If there is an irregular sample point in the set of sample points, the center may probably be computed as a point that is far away from regular sample points. To contrast, K-medoids selects one sample point among all sample points within the same clustered group as the center. Such a sample point has the smallest sum of distance between its coordinate and each other sample point. Since irregular data is scarce in each clustered group, it is less likely to be selected as the group center. Thus, the center of each clustered group will not be far away from the regular sample points. That is the reason that K-medoids can cluster sample points without being influenced too much by the irregular sample points.

The discovered attribute correspondences are expressed in Expressive and Declarative Ontology Alignment Language (*EDOAL*)[3] [Scharffe 2009, David 2011], which is used to express various kinds of correspondences.

The details of the clustering process can be found in Chapter 5.

[3] http://alignapi.gforge.inria.fr/edoal.html

4.3 Generating Links

It is appropriate to generate a sample link set for users to assess for constructing the interlinking pattern. There are two methods of obtaining assessed links. First, let users compare instances and provide links. It is time-consuming to find links manually. Since each instance of one class should be compared with each instance of the corresponding class, there are a lot of instance pairs that should be assessed. Alternatively, the other method explores some sample links and send them to users to compare. The second method is better than the first one, in that it sets users free from a big number of comparisons on instances. We will use the second method in the interlinking method of this thesis to generate assessed links.

In order to produce a sample link set with Silk, we need to design transformation rules that transfer class correspondences and attribute correspondences into queries in a Silk script. With the discovered attribute correspondences of Section 4.2, a sample link set can be generated by constructing a sample interlinking pattern, transferring the sample interlinking pattern into an executable script, and running the script with Silk. Since Silk is a semi-automatic tool, it can only produce links based on an executable script. The script is a file, which is composed of all sets of attribute correspondences in the interlinking pattern and some comparison methods for comparing corresponding attributes' values across two instances. The script is expressed with the syntax of Silk script. Silk compares instances of two RDF data sets by comparing attributes' values with regard to each attribute correspondence in the interlinking pattern. Attributes' values are obtained by querying RDF data sets. Hence, in a Silk script, each attribute correspondence is expressed as a pair of queries. To conclude, in order to obtain a sample link set for assessing and a link set of two RDF data sets, a set of transformation rules is demanded for transferring the correspondences that are expressed in EDOAL into the queries of Silk script.

The sample links should be produced by a sample interlinking pattern that is expressed as a disjunction of all discovered potential attribute correspondences, so that all correct links that contain any potential attribute correspondence are generated as sample links to be assessed. From the solution of Section 4.2, we have a set of discovered potential attribute correspondences. However, we have no idea which potential attribute correspondence is an attribute correspondence and which potential attribute correspondence is not. We also do not know which set of attribute correspondences is contained in correct links. Thus, we design the sample interlinking pattern as a disjunction of all potential attribute correspondences. It means that any instance pair that have at least one discovered potential attribute correspondence should form a sample link. Here, if two compared instances have similar attribute values of one discovered potential attribute correspondence, we say that the two compared instances have such a discovered potential attribute correspondence. We also say that the sample link that is formed by the compared instances has such a discovered potential attribute correspondence. Therefore, the sample link set contains all correct links, if in each link there is at least one attribute correspondence. Then, the interlinking pattern can cover more correct links and filter more incorrect links after learning the set of attribute correspondences of each assessed sample link. Comparing to the related work that is introduced in Section 3.3, this sample link generation process creates much less irrelevant sample links than the method in [Isele 2013]. The reason is that the number of tokens will be definitely much larger than the number of discovered potential attribute correspondences. Comparing to [Ngonga Ngomo 2012b], such an algorithm does not require any external information for generating sample links. Thus, it can be applied in different interlinking tasks.

The details of the solution will be presented in Chapter 6.

4.4 Constructing and Improving the Interlinking Pattern with Assessed Links

The interlinking pattern of two RDF data sets should be improved with a supervised learning method. As illustrated in Chapter 2, the interlinking pattern usually cannot be built correctly the first time. The interlinking pattern should be improved iteratively with some assessed links. The iterative process is executed according to a supervised learning method. The method ought to enhance the interlinking pattern to cover more correct links and filter more incorrect links.

Expressing the interlinking pattern as a disjunction of sets of attribute correspondences is applicable for most interlinking tasks. There are two kinds of interlinking pattern. The first one is a conjunction of all attribute correspondences that appear in all correct links. In this case, all correct links of two data sets have the same set of attribute correspondences, so that a conjunctive expression satisfies all correct links. It means that if any two instances have these set of attribute correspondences, a link can be built between them. The second one is a disjunction of all conjunctive expressions. Each conjunctive expression only satisfies some correct links. In this case, the interlinking pattern is a disjunctive expression that combines all conjunctive expressions. It means that if any two instances have any set of attribute correspondences in the disjunctive expression, a link can be built between them. Since most RDF data sets are not regular, not all correct links share the same set of attribute correspondences. Thus, disjunctive expression is more applicable than conjunctive expression for most interlinking tasks.

Since the interlinking pattern distinguishes correct links and incorrect links, assessed links play a very important role during learning. Each link can be expressed as a conjunction of several attribute correspondences. Formally, such an expression is in the form of $AC_i \bigcap \ldots \bigcap AC_j$, where $AC_i, \ldots, AC_j$ are attribute correspondences that exist in the link. Therefore, the interlinking pattern is an expression that combines all assessed correct links' expressions disjunctively. Formally, the expression of the interlinking pattern is in the form of $(AC_i^1 \bigcap \ldots \bigcap AC_j^1) \bigcup \ldots \bigcup (AC_k^n \bigcap \ldots \bigcap AC_l^n)$. In the expression, $AC_i^1, \ldots, AC_j^1$ are attribute correspondences that exist in the assessed correct link No.1. $AC_k^n, \ldots, AC_l^n$ are attribute correspondences that exist in the assessed correct link No.n. That is why the expression of the interlinking pattern can be constructed based on the assessed links.

In the thesis, Version Space is chosen as the supervised learning method to build and improve the interlinking pattern into a disjunctive expression with assessed links. Compared to other learning methods, the main advantages of Version Space are:

- It is a learning method that is able to build classifiers for various kinds of learning tasks. It can not only build the classifier expressing conjunctive expressions, but also the classifier expressing a disjunction of conjunctive expressions. It means that Version Space can build a classifier which is the composition of all classifiers, each of which only covers some correct links. Yet the composition covers all correct links. In contrast, other learning methods such as Support Vector Machine and Neural Network cannot build a classifier expressing a disjunction of conjunctive expressions.

- Comparing to the previous interlinking pattern, the interlinking pattern is always improved whenever a new assessed link is learned. It will in turn enhance the F-measure of the generated link set. Any revision on the interlinking pattern in the Version Space is based on the set of attribute correspondences in new-coming assessed links. The interlinking pattern of Version Space never turns bad when more assessed links are learned, if the links are not assessed wrongly. Other learning methods such as Genetic Programming are non-deterministic. The interlinking pattern may be different, if the learning process is executed several times with the same set of assessed links. The interlinking pattern may turn bad when more assessed links are learned, which cover less correct links or filter less incorrect links. The reason is that the random changes of chromosome by crossing over or mutating in Genetic Programming may degrade the interlinking pattern. It will in turn decrease the Precision, Recall and F-measure of the generated link set.

Besides a supervised learning method, a sample link selection method is required. There are lots of links in the sample link set to be assessed. According to the Active Learning method, links that possess the most information should be assessed first, because they can help the Version Space model find out the final interlinking pattern with fewer learning rounds. In our interlinking problem, the information is the set of attribute correspondences that exist in a sample link. Thus, all links in the sample link set can be classified by their set of attribute correspondences. Links that have the same set of attribute correspondences are classified into the same group. When one link of a classified group is assessed and used to improve the interlinking pattern, the improved interlinking pattern can distinguish all links that have the same set of attribute correspondences as the selected sample link. Therefore, it is only necessary to select one sample link per classified group to assess and improve the interlinking pattern.

The details of the Version Space learning method are described in Chapter 7.

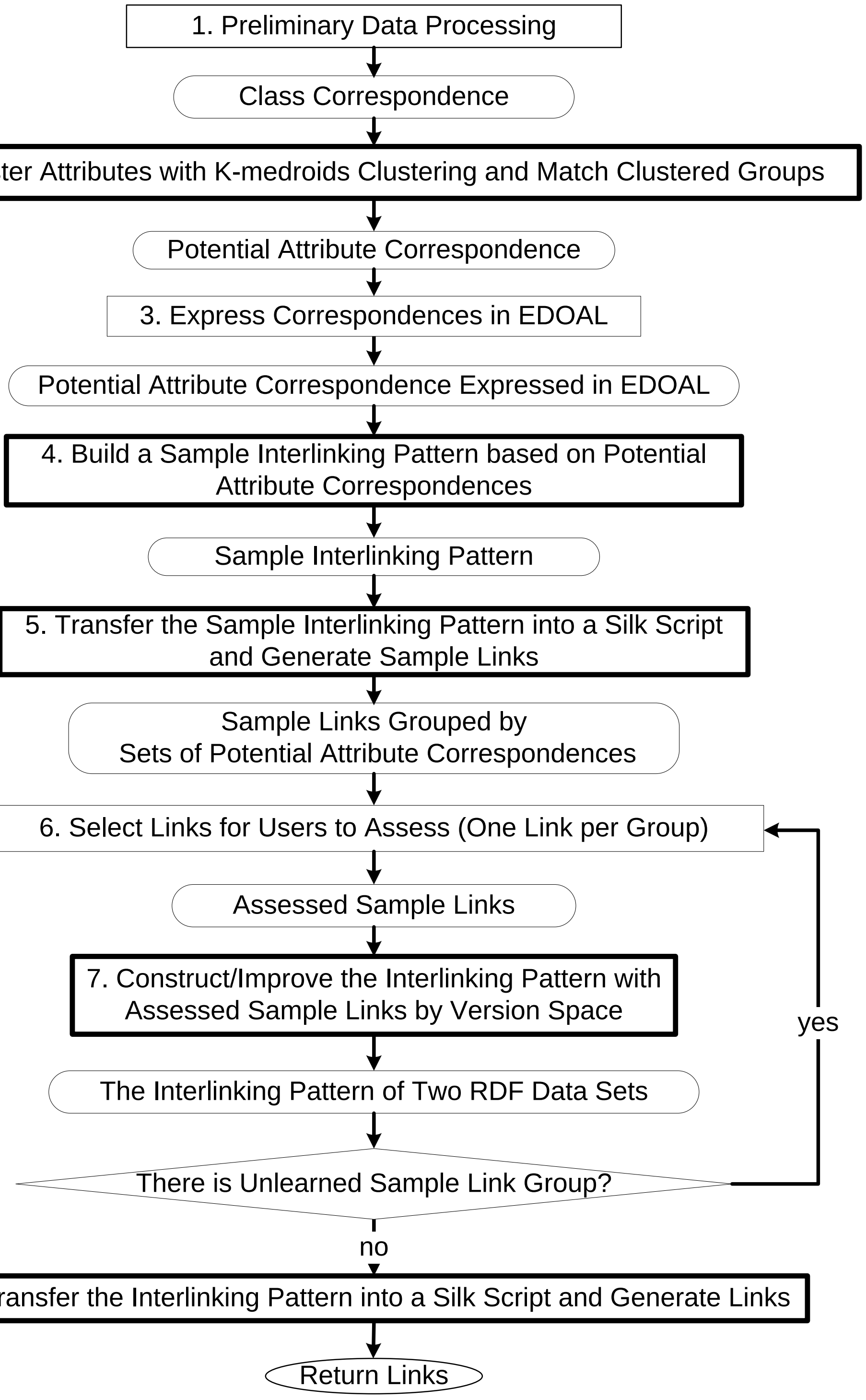

1. Preliminary Data Processing
Class Correspondence
2. Cluster Attributes with K-medroids Clustering and Match Clustered Groups
Potential Attribute Correspondence
3. Express Correspondences in EDOAL
Potential Attribute Correspondence Expressed in EDOAL
4. Build a Sample Interlinking Pattern based on Potential Attribute Correspondences
Sample Interlinking Pattern
5. Transfer the Sample Interlinking Pattern into a Silk Script and Generate Sample Links
Sample Links Grouped by Sets of Potential Attribute Correspondences
6. Select Links for Users to Assess (One Link per Group)
Assessed Sample Links
7. Construct/Improve the Interlinking Pattern with Assessed Sample Links by Version Space
The Interlinking Pattern of Two RDF Data Sets
There is Unlearned Sample Link Group?
yes
no
8. Transfer the Interlinking Pattern into a Silk Script and Generate Links
Return Links

Figure 4.1: Eight Steps of Interlinking Data Sets

4.5 System Overview

I propose eight steps for interlinking instances across two data sets with the interlinking method of this thesis, as shown in Figure 4.1.

1 Step 1 generates a set of class correspondences based on the ontologies of two data sets. This step can be implemented by leveraging some ontology matching methods like [Ngo 2012], such that some class correspondences are determined.

2 Step 2 uses the K-medoids clustering method to cluster all attributes in each class of a class correspondence according to attributes' value features. Then, each clustered group of one class is matched with a clustered group of the corresponding class, if their value features are similar. Potential attribute correspondences are attribute pairs across two matched clustered groups.

3 Step 3 expresses class correspondences and potential attribute correspondences discovered by preceding steps into correspondences in EDOAL.

4 Step 4 builds a sample interlinking pattern into a disjunction of all potential attribute correspondences that are discovered by the K-medoids clustering method.

5 Step 5 expresses the sample interlinking pattern in Silk syntax and generates a sample link set. These sample links are classified into groups according to their sets of attribute correspondences.

6 Step 6 selects links out of classified groups (one link per group) and assess them as correct or incorrect links one by one based on users' dynamic feedback over time.

7 Step 7 constructs (1^{st} learning round) or improves (2^{nd}, 3^{rd}, ..., learning round) the interlinking pattern by learning the sets of attribute correspondences of assessed sample links. If there is unlearned sample link group, the learning process continues. Otherwise, it stops.

8 With the interlinking pattern improved in the last step, a script is built and run by Silk in Step 8.

Step 2, 4, 5, 7 and 8 are the solutions to the three tasks of the interlinking problem in this thesis. Specifically, Step 2 fulfills the task *Discovering Attribute Correspondence*. Step 4, 5 and 8 complete the task *Generating Links*. Step 7 accomplishes the final task *Constructing and Improving the Interlinking Pattern with Assessed Links*. The three steps play the key roles in determining the F-measures of the generated link set, which will be the focus in the thesis.

4.6 Conclusion

This chapter introduces the interlinking method of this thesis. In detail, three solutions are given for each task of the interlinking process. First, K-medoids clustering is used to discover potential attribute correspondences across corresponding classes. Second, a set of sample links should be generated to help construct the interlinking pattern. The links are produced with a Silk script that is transformed from a sample interlinking pattern, which is expressed as a disjunction of all discovered potential attribute correspondences. Third, Version Space is applied to construct and improve an interlinking pattern that is able to produce correct links across two RDF data sets by learning user-assessed sample links. Finally, a system overview is presented to show the data flow of the interlinking method.

The following chapters will introduce the solutions of the three tasks in detail. Specifically, next chapter will introduce the solution of discovering attribute correspondences across two corresponding classes in two RDF data sets. K-medoids is used to classify attributes of each class into several groups according to attributes' value features. Then, clustered groups are mapped if their value features are similar. Attribute pairs across mapped clustered groups are considered to be potential attribute correspondences.

Chapter 5

Discovering Attribute Correspondences by K-medoids Clustering

Contents

The work for discovering the attribute correspondences by K-medoids clustering is introduced in this chapter. First, the chapter presents how to classify attributes of each class with K-medoids clustering in Section 5.1. Second, clustered groups are mapped by their statistical value features in Section 5.2. This section also shows how to map attributes of a clustered group to attributes of the mapped clustered group, generating a set of potential attribute correspondences. Third, an example is given in Section 5.3.

5.1 Clustering Attributes of Each Class

Before constructing an interlinking pattern for each class correspondence, it is necessary to specify a group of attribute correspondences across two corresponding classes by clustering. Clustering reduces the size of potential attribute correspondences considered in the interlinking task, significantly reducing the computation workload. Otherwise, all attributes of a class should be compared with all attributes of the corresponding class. Since attributes of each attribute correspondence share similar value features, we can justify the attribute correspondences by matching attributes with similar value features. Value feature is a characteristic or pattern of values. For example, value feature of dates is "XXXX-XX-XX". It means that there are 8 numbers and the two symbols "-" in each date. Value feature of email addresses is that there is a symbol "@" in each email address. Thereafter, clustered groups from two corresponding classes can be mapped according to their value features. The potential attribute correspondences are attribute pairs across mapped clustered groups.

Since there may be irregular data in the RDF data sets on the web, we choose K-medoids clustering to classify attributes of each class. It is a clustering method that is tolerable when there is irregular data in the data set. In other words, the clustering results will not influenced much by the irregular data if K-medoids is applied. Irregular data is a kind of data that does not have the same value feature with the other regular data. Regular data is a kind of data that is created according to the schema of the data sets. In Euclidean space, the coordinates of irregular data are usually far away from the ones of regular data.

K-medoids clustering is an unsupervised learning method [Kaufman 1987], which is able to partition a large-scale data set into Voronoi cells [Okabe 2000] based on the coordinates of the data in the Euclidean space. The time complexity of K-medoids clustering is $O(n^2)$. n is the data size. Two information should be given before executing K-medoids. One is the number of clustering groups k, and the other one is the initial centers of these k clustering groups. In practice, K-medoids clustering effect differs a lot, with different k and different initial centers of clustering groups.

In the interlinking setting of this thesis, each attribute of a class is treated as a sample point to be clustered based on value features. Each sample point has an coordinate in the Euclidean space. There are d dimensions constructed by d *value features*, thus each coordinate is a d-tuple. The ith item in a d-tuple for an attribute is the value of the ith feature based on the instances of the attribute.

It is intuitive to cluster attributes according to type features. Type feature is a kind of value feature that describes the values of the same type. There are several type features that can be used for constructing attribute coordinates. For example, the type feature of numeric attributes is that there is no letter (except the letter e and E indicating exponent) but only numbers. The type feature of people's name and book's title is that the first letter is always capitalized. If we count the type features of all attributes in a RDF data set, there will be a lot. There are many types of attributes that describe the resources in the world. Moreover, type features can only distinguish attributes of different types. They cannot be used to distinguish attributes of the same type. Thus, we assume that there are two types of attributes of each class in general: numeric and non-numeric. For attributes of the same type, we define four statistical features, which is a kind of value feature that reflects attribute values' characteristics with statistics, to distinguish attributes of the same type.

If an attribute is a number, the statistical features are

1 The maximum value of the attribute's range

2 The minimum value of the attribute's range

3 The average value of the attribute's range

4 The deviation of the attribute's range

These four statistical features are used to distinguish attributes according to attribute values' distribution. Assume that there are two properties. One is about people's age, the other is about people's height. Assume that the values of the above four statistical features of people's age (in years) are 1, 100, 40, 20. The values of the above four statistical features of people's height (in centimeters) are 30, 210, 170, 20. In the Euclidean space, these two points (1,100,40,20) and (30,210,170,20) are relatively far. Therefore, these two properties will be clustered into different attribute groups, so that they can be distinguished.

If an attribute is a string (a sequence of characters), the statistical features are

1 The ratio of the values that capitalize the first letter

2 The ratio of the values that contains special symbols, such as “@” or “$”

3 The average length of the values

4 The average number of words in the values

These four statistical features are used to distinguish attributes according to the characteristics of different non-numeric attributes. For example, properties about address usually do not have a first capitalized letter like properties about people's name and book's title. Properties about email and currency can be distinguished by the second statistical feature from other properties (using the special symbols like “@” or “$”). Properties about people's name can be identified by 3^{rd} and 4^{th} statistical features, because usually people's names are shorter than properties like book's title. Besides, the words of people's names are fewer than the words of book's titles.

Thus, before clustering, attributes of each class are divided into two sets. One is composed of numerical attributes, and the other is composed of non-numerical attributes. Then, values of each attribute are collected to compute the coordinate of the attribute according to the four statistical features defined before. The numerical attributes and non-numerical attributes are both clustered based on their coordinates, but they are clustered in two different Euclidean spaces.

Figure 5.1 shows how to cluster some non-numerical attributes in the example of Chapter 2. In the figure, there are two corresponding classes that refer to the departments in France. We are going to cluster the attributes of each class. For each class, there are three non-numerical attributes, {*insee:nom*, *insee:code_ departement*, *insee:subdivision*$^{-1}$*/nom*} for the class *insee:Departement*; {*eurostat:name*, *eurostat:code*, *eurostat:hasParentRegion/name*} for the class *eurostat:NUTSRegion*. All attribute values are collected to compute the coordinate for each attribute (i.e., Step ① and ②). In the figure, three statistical features are evaluated. They are the ratio of the values that capitalize the first letter, the average number of words in the values, and the average length of the values, which denote X, Y, and Z respectively. With the K-medoids clustering, all non-numerical attributes of each class are classified into several groups based on the coordinates (Step ③ in Figure 5.1). Afterwards, each clustered group of non-numerical attributes in one class is mapped to one clustered group of non-numerical attributes in the corresponding class whose center is the closest one. In Figure 5.1, the clustered group

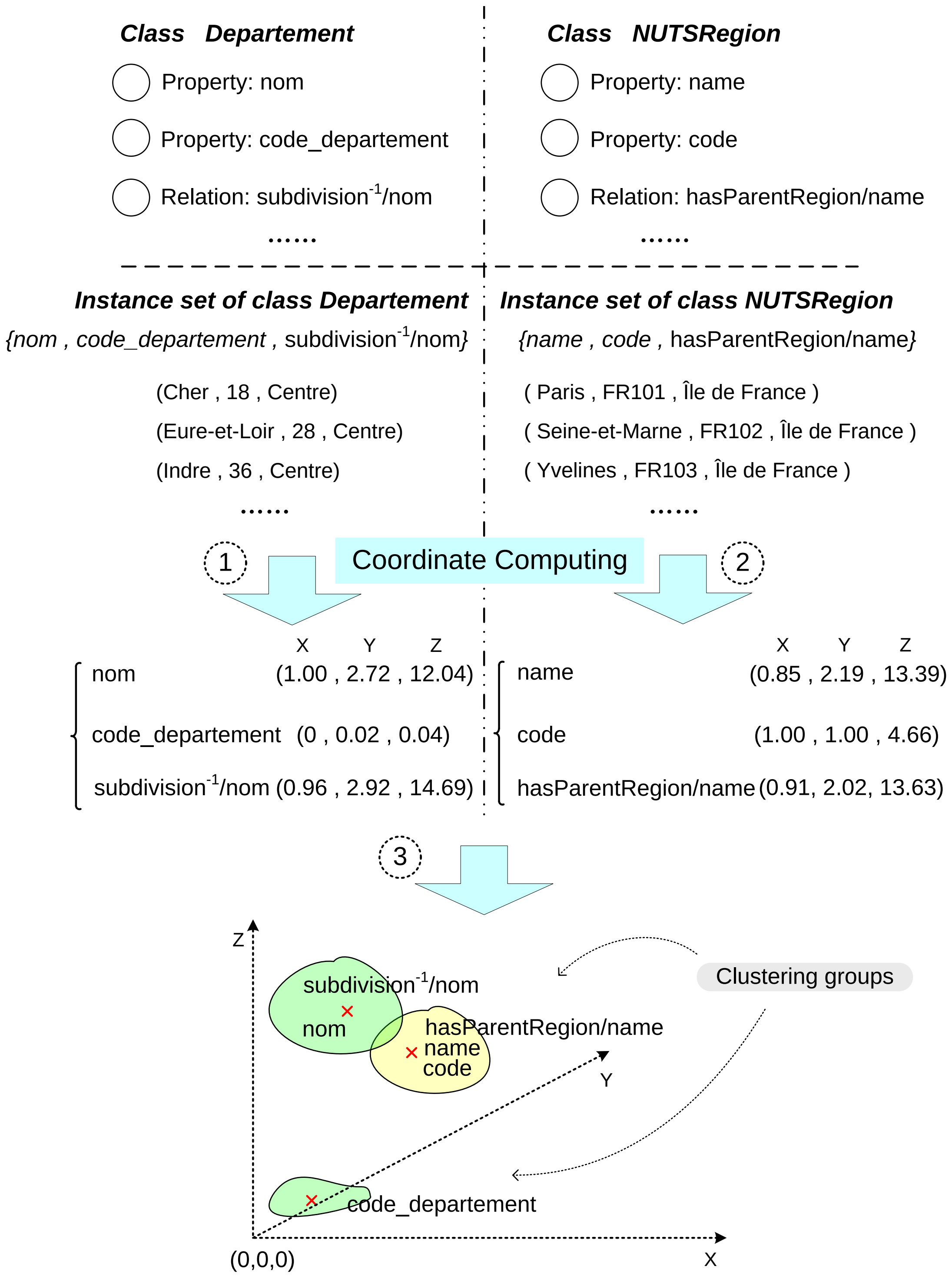

Class Departement
Property: nom
Property: code_departement
Relation: subdivision^{-1}/nom
……
Class NUTSRegion
Property: name
Property: code
Relation: hasParentRegion/name
……
Instance set of class Departement
{nom , code_departement , subdivision^{-1}/nom}
(Cher , 18 , Centre)
(Eure-et-Loir , 28 , Centre)
(Indre , 36 , Centre)
……
Instance set of class NUTSRegion
{name , code , hasParentRegion/name}
(Paris , FR101 , Île de France)
(Seine-et-Marne , FR102 , Île de France)
(Yvelines , FR103 , Île de France)
……
1
Coordinate Computing
2
X Y Z
nom (1.00 , 2.72 , 12.04)
code_departement (0 , 0.02 , 0.04)
subdivision^{-1}/nom (0.96 , 2.92 , 14.69)
X Y Z
name (0.85 , 2.19 , 13.39)
code (1.00 , 1.00 , 4.66)
hasParentRegion/name (0.91, 2.02, 13.63)
3
Z
subdivision^{-1}/nom
nom
hasParentRegion/name
name
code
Y
Clustering groups
code_departement
(0,0,0)
X

Figure 5.1: Constructing Euclidean Space for Clustered Non-numerical Attributes

$\{$*insee:subdivision*$^{-1}$*/insee:nom*, *insee:nom*$\}$ is mapped to the clustered group $\{$*eurostat:hasParentRegion/eurostat:name*, *eurostat:name*, *eurostat:code*$\}$. Finally, all attribute pairs across mapped groups are treated as potential corresponding attributes. The potential attribute correspondences in the figure are listed below.

- insee:subdivision^{-1}/insee:nom↔eurostat:name
- insee:subdivision^{-1}/insee:nom↔eurostat:hasParentRegion/eurostat:name
- insee:subdivision^{-1}/insee:nom↔eurostat:code
- insee:nom↔eurostat:name
- insee:nom↔eurostat:hasParentRegion/eurostat:name
- insee:nom↔eurostat:code

Besides a set of sample points, there are two other information that should be given before applying K-medoids. One is the initial centers of the clustering groups. The other is the number of clustering groups. In order to optimize the clustering effect, the two information is given in the following way. First, the initial central point of each clustering group is created several times with the Forgy method [Forgy 1965]. According to the illustration in [Hamerly 2002], the Forgy method is suitable for the clustering scenario than other initialization methods. There are a lot of initial sets of centers created, and the most suitable set of centers is the one that has the best convergence effect. It means that the total distance between each sample point and its center is the smallest one. Second, in order to make the sample points evenly distributed, a merging operation is used to merge nearby clustering groups if the distances of their centers are relatively smaller than the average distance between centers. If a merging operation is applied, all sample points should be re-clustered because the centers are changed. The clustering process stops until either of the following two conditions is satisfied. One condition is that there are only two clustered groups left, which means that there is no need to merge. The other condition is that the distances between any two centers are relatively the same.

In order to compute the distance between any two coordinates, two distance measures are applied. One is Euclidian distance, and the other is Cosine similarity. Euclidean distance represents the absolute distance between two sample points in the Euclidean space. Cosine similarity shows the orientation similarity of two sample points. The two measures are combined here to show the distance between two coordinates with an all-round perspective. Thus, the distance of two points is computed as the average value of the two measures. First, values of both measures are normalized. So the values are both in the range $[0, 1]$. When the Euclidean distance is 1, it means that the two coordinates are far away. In contrast, when the Cosine similarity is 1, it means that the two coordinates have the same orientation. So second, values of the Cosine similarity should be reversed to show the same mappings from the decimals in $[0, 1]$ to the distances as the mappings of the Euclidean distance. Each value of the Cosine similarity is reversed by $1 - v$, where v represents the value of the Cosine similarity. Third, the average value of the two measures is computed to show the distance of two coordinates.

A formal definition of K-medoids clustering is below.

```
Input: the set $M$ of sample points representing numerical/non-numerical
  attributes in a class, initial number ($N$) of clustered groups, the iteration
  times $T$=30 for finding an optimized set of initial centers;
Output: a set of clustered groups of numerical/non-numerical attributes in
  each class.
  $C$ = the set of $N$ centers computed by the Forgy method;
  Clustering sample points into the nearest centers with $KMedoids(M,C)$;
  Compute $TD$;
  $t = 1$;
  while ($t<T$) do
     $C_{new}$ = the set of $N$ centers computed by the Forgy method;
     Clustering sample points into the nearest centers with
     $KMedoids(M,C_{new})$;
     Compute $TD_{new}$;
     if $TD_{new}<TD$ then
        $TD=TD_{new}$;
        $C = C_{new}$;
     $t$++;
  while ($|C|>2$) do
     Clustering sample points into the nearest centers with $KMedoids(M,C)$;
     Compute new centers $C_{new}$ of each clustering group;
     Compute $ad$;/*Compute the average distance between centers*/
     $C = C_{new}$;
     if ($\exists\, c_i,c_j \in C$, distance$(c_i,c_j)<\frac{1}{2}\cdot ad$) then
        $C=MergeCenters(C,\frac{1}{2}\cdot ad)$;/*Merge nearby centers in $C$*/
        if ($|C|<2$) then
           break;
     else
        break;
  Output $C$.
```

Algorithm 1: K-medoids Clustering

The goal of the K-medoids clustering algorithm is to cluster numerical/non-numerical attributes into several groups based on the coordinates of the attributes, such that the total distance between each sample point and its center (denoted by *TD*) is minimized. In this algorithm, $KMedoids(M,C)$ represents the K-medoids clustering algorithm on the sample point set M and the set of centers C. Experiments indicate that the initial number of centers (denoted by N) is set to 20, if the total number of attributes is equal to 50 or more than 50. If the total number of attributes is lower than 50, N is set to the total number of attributes. T denotes the iteration times of optimizing the initial set of centers with the Forgy method. The set of initial centers that is used for clustering is the one that has the smallest TD. distance(c_i,c_j) denotes the distance between two group centers c_i and c_j. ad represents the average distance of any two centers in C. If the distance of any two centers is smaller than $\frac{1}{2} \cdot ad$, they should be merged as one center. The centers' groups are merged into one group accordingly. The new center is a sample point in the new group, such that the sum of distances between this sample point and other sample points in the new group is the smallest one.

The whole process of the K-medoids clustering (Algorithm 1) is summarized as follows. Firstly, the Forgy method is run T times to find out the most distributed set of centers M (line 1–12). Second, all sample points are clustered to the centers which are the closest centers from them (line 14). Third, two relatively close centers (according to $\frac{1}{2} \cdot ad$) will be merged (line 18–19). The new group's center is a sample point of the new group that has the smallest sum of distance between the point and each other point in the group (line 19). Fourth, a new K-medoids clustering will be performed based on the new set of centers (line 14). The merging operation will repeat until the distance between any two centers is larger than $\frac{1}{2} \cdot ad$ or there are only two groups left.

5.2 Group Matching of Attributes Across Two Classes

After the attributes of each class are classified into several groups via the clustering step, potential attribute correspondences are built easily between mapped clustered groups of corresponding classes C and C'. Each clustered group of the class C can be mapped to one clustered group of the class C', based on the coordinates of the group centers. Note that numerical attributes and non-numerical attributes of each class are clustered separately. Thus, numerical attribute groups of each class are only mapped to numerical attribute groups of the corresponding class. Non-numerical attribute groups of each class are only mapped to non-numerical attribute groups of the corresponding class.

Here is the illustration of how to build mappings for different groups of attributes across the two corresponding classes and how to build potential attribute correspondences across mapped groups. Assume there are p groups of non-numerical attributes in class C, denoted by $G_1, G_2, ..., G_p$. There are four statistical features for building coordinates of the attributes. The coordinate of a clustered group's center is denoted by (x_i, y_i, z_i, w_i), where i=1,2,$\cdots$,p. Assume that there are q groups of non-numerical attributes in class C', denoted by $G'_1, G'_2, \ldots, G'_q$. The coordinate of a clustered group's center is (x'_j, y'_j, z'_j, w'_j), where j=1,2,$\cdots$,q. Then, the distance between G_i and G'_j is reflected by the distance of their coordinates, denoted by $D(G_i, G'_j)$. The mappings of groups can be determined by the formula, $\{\arg\min_{\{(i,j)\}} D(G_i, G'_j)\}$, where arg min indicates the index value (i,j) with minimal

$D(G_i, G'_j)$. It means that for each group G_i of the class C, the mapping group is the group G_j of the corresponding class C' that has the closest center with the group G_i. According to the mapped groups, all attributes from one group of class C are matched with the attributes from the mapping group of class C'. Therefore, a set of potential attribute correspondences can then be produced.

To conclude, the K-medoids clustering algorithm can discover several types of attribute correspondences as follows. They are expressed in Expressive and Declarative Ontology Alignment Language ($EDOAL$)[1] [Scharffe 2009, David 2011], which is defined to express complex attribute correspondences.

- a property P is corresponding to a property P'

```
<Cell>
  <entity1>
    <edoal:Property rdf:about="P"/>
  </entity1>
  <entity2>
    <edoal:Property rdf:about="P'"/>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a property P is corresponding to a path that is composed of one or several relations $R'_1, \ldots, R'_n$ and a property P', where $n \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property rdf:about="P"/>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation rdf:about="R_1'" />
        ...
        <edoal:Relation rdf:about="Rn'" />
        <edoal:Property rdf:about="P'" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a property P is corresponding to a path that is composed of a reverse relation R'^{-1} and a property P'

```
<Cell>
  <entity1>
    <edoal:Property rdf:about="P"/>
```

[1] http://alignapi.gforge.inria.fr/edoal.html

```
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R’" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a property P is corresponding to a path that is composed of a reverse relation R'^{-1}, one or several relations $R'_1, \ldots, R'_m$ and a property P', where $m \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property rdf:about="P"/>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R’" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1’" />
        ...
        <edoal:Relation rdf:about="Rm’" />
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of one or several relations $R_1, \ldots, R_n$ and a property P is corresponding to a path that is composed of one or several relations $R'_1, \ldots, R'_m$ and a property P', where $n \in [1, +\propto)$, $m \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation rdf:about="R1" />
        ...
        <edoal:Relation rdf:about="Rn" />
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation rdf:about="R1'" />
        ...
        <edoal:Relation rdf:about="Rm'" />
        <edoal:Property rdf:about="P'" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of one or several relations $R_1, \ldots, R_n$ and a property P is corresponding to a path that is composed of a reverse relation R'^{-1} and a property P', where $n \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation rdf:about="R1" />
        ...
        <edoal:Relation rdf:about="Rn" />
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R'" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P'" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of one or several relations $R_1, \ldots, R_n$ and a property P is corresponding to a path that is composed of a reverse relation R'^{-1}, one or several relations $R'_1, \ldots, R'_m$ and a property P', where $n \in [1, +\propto)$, $m \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation rdf:about="R1" />
        ...
        <edoal:Relation rdf:about="Rn" />
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R'" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1'" />
        ...
        <edoal:Relation rdf:about="Rm'" />
        <edoal:Property rdf:about="P'" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of a reverse relation R^{-1} and a property P is corresponding to a path that is composed of a reverse relation R'^{-1} and a property P'

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R'" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of a reverse relation R^{-1}, one or several relations $R_1, \ldots, R_n$ and a property P is corresponding to a path that is composed of a reverse relation R'^{-1} and a property P', where $n \in [1, + \propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1" />
        ...
        <edoal:Relation rdf:about="Rn" />
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R’" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of a reverse relation R^{-1} and a property P is corresponding to a path that is composed of a reverse relation R'^{-1}, one or several relations $R'_1, \ldots, R'_m$ and a property P', where $m \in [1, + \propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
```

```
          <edoal:inverse>
            <edoal:Relation rdf:about="R" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R’" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1’" />
        ...
        <edoal:Relation rdf:about="Rm’" />
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

- a path that is composed of a reverse relation R^{-1}, one or several relations $R_1, \ldots, R_n$ and a property P is corresponding to a path that is composed of a reverse relation R'^{-1}, one or several relations $R'_1, \ldots, R'_m$ and a property P', where $n \in [1, +\propto)$,$m \in [1, +\propto)$

```
<Cell>
  <entity1>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1" />
        ...
        <edoal:Relation rdf:about="Rn" />
        <edoal:Property rdf:about="P" />
      </edoal:compose>
    </edoal:Property>
  </entity1>
  <entity2>
    <edoal:Property>
      <edoal:compose rdf:parseType="Collection">
        <edoal:Relation>
          <edoal:inverse>
            <edoal:Relation rdf:about="R’" />
          </edoal:inverse>
        </edoal:Relation>
        <edoal:Relation rdf:about="R1’" />
        ...
        <edoal:Relation rdf:about="Rm’" />
        <edoal:Property rdf:about="P’" />
      </edoal:compose>
    </edoal:Property>
  </entity2>
  <relation>equivalence</relation>
  <measure>1.0</measure>
</Cell>
```

5.3 Example

As for the interlinking scenario 2.1 in Chapter 2, potential attribute correspondences are generated below. There are three types of attributes that are clustered. They are *property*, *relation/property* and *relation*$^{-1}$*/property*.

In the class *insee:Departement*, the attributes being clustered are:

string	insee:nom
string	insee:code_departement
number	insee:subdivision^{-1}/insee:code_region
string	insee:subdivision^{-1}/insee:nom

The clustered group for numerical attributes is

Group 1	{insee:subdivision^{-1}/insee:code_region}; coordinate: (94.00, 1.00, 43.85, 905.75)

The clustered groups for non-numerical attributes are

Group 1	{insee:code_departement}; coordinate: (0.00, 0.00, 0.02, 0.04)
Group 2	{insee:nom, insee:subdivision^{-1}/insee:nom}; coordinate: (0.96, 0.00, 2.92, 14.69)

In the class *eurostat:NUTSRegion*, the attributes being clustered are:

string	eurostat:label
string	eurostat:name
string	eurostat:code
number	eurostat:level
string	eurostat:hasParentRegion/eurostat:label

string	eurostat:hasParentRegion/eurostat:name
string	eurostat:hasParentRegion/eurostat:code
number	eurostat:hasParentRegion/eurostat:level
string	eurostat:hasParentRegion^{-1}/eurostat:label
string	eurostat:hasParentRegion^{-1}/eurostat:name
string	eurostat:hasParentRegion^{-1}/eurostat:code
number	eurostat:hasParentRegion^{-1}/eurostat:level

The clustered groups for numerical attributes are

Group 1 {eurostat:level};
coordinate: (3.00, 0.00, 1.50, 1.25)

Group 2 {eurostat:hasParentRegion/eurostat:level};
coordinate: (2.00, 0.00, 1.00, 0.67)

Group 3 {eurostat:hasParentRegion^{-1}/eurostat:level};
coordinate: (3.00, 1.00, 2.00, 0.67)

The clustered groups for non-numerical attributes are

Group 1 {eurostat:hasParentRegion/eurostat:code};
coordinate: (1.00, 0.00, 1.00, 3.70)

Group 2 {eurostat:name, eurostat:hasParentRegion/eurostat:name,
eurostat:hasParentRegion^{-1}/eurostat:name, eurostat:code,
eurostat:hasParentRegion^{-1}/eurostat:code};
coordinate: (0.92, 0.00, 2.27, 15.01)

Group 3 {rdfs:label, eurostat:hasParentRegion/rdfs:label,
eurostat:hasParentRegion^{-1}/rdfs:label};
coordinate: (1.00, 0.00, 3.02, 22.54)

The mapped groups on numerical attributes are:

- {insee:subdivision^{-1}/insee:code_region}↔{eurostat:level}

The mapped groups on non-numerical attributes are:

- {insee:code_departement}↔{eurostat:hasParentRegion/eurostat:code}
- {insee:nom,insee:subdivision^{-1}/insee:nom}↔{eurostat:name, eurostat:hasParentRegion/eurostat:name, eurostat:hasParentRegion^{-1}/eurostat:name, eurostat:code, eurostat:hasParentRegion^{-1}/eurostat:code}

Thus, potential attribute correspondences that are produced after mapping clustered groups are:

- insee:subdivision^{-1}/insee:code_region↔eurostat:level
- insee:code_departement↔eurostat:hasParentRegion/eurostat:code
- insee:nom↔eurostat:name
- insee:nom↔eurostat:hasParentRegion/eurostat:name
- insee:nom↔eurostat:hasParentRegion^{-1}/eurostat:name
- insee:nom↔eurostat:code
- insee:nom↔eurostat:hasParentRegion^{-1}/eurostat:code
- insee:subdivision^{-1}/insee:nom↔eurostat:name
- insee:subdivision^{-1}/insee:nom↔eurostat:hasParentRegion/eurostat:name
- insee:subdivision^{-1}/insee:nom↔eurostat:hasParentRegion^{-1}/eurostat:name
- insee:subdivision^{-1}/insee:nom↔eurostat:code
- insee:subdivision^{-1}/insee:nom↔eurostat:hasParentRegion^{-1}/eurostat:code

5.4 Conclusion

In this chapter, K-medoids is exploited to discover potential attribute correspondences across two corresponding classes.

First, several statistical features of attributes are defined. If attribute values are numbers (a sequence of digits), the features are *maximum value of the attribute's range*, *minimum value of the attribute's range*, *average value of the attribute's range*, and *deviation of the attribute's range*. If the attribute values are strings (a sequence of characters), the features are *ratio of the attribute values that contain special symbols*, *ratio of the attribute values that capitalize the first letter*, *average number of words*, and *average length of the attribute values*. These statistical features effectively distinguish attributes of a class.

Second, the coordinate of each attribute in each class are computed according to the features defined above. Then, both numerical and non-numerical attributes are classified into groups by K-medoids clustering algorithm respectively.

Third, with respect to a class correspondence, the groups in one class are mapped to the groups in another class. Among which, each group is mapped to the group that has the closest euclidian center with its own. It also means that each group is mapped to a group which has the most similar features. Groups of numerical attributes in one class are mapped to the groups of numerical attributes in the corresponding class. Groups of non-numerical attributes in one class are mapped to groups of non-numerical attributes in the corresponding class. Potential attribute correspondences are attribute pairs across the mapped groups.

Next chapter will introduce two works. One work is the transformation rules that transfer EDOAL correspondences into graph patterns and form a Silk script for generating links across data sets. The other work is the method of generating sample links for users to assess.

Chapter 6

Generating Links

Contents

This chapter introduces how to utilize the potential attribute correspondences that are produced by the K-medoids clustering step to generate a set of sample links that can be assessed by users. First, it introduces expressive correspondence language EDOAL and semi-automatic interlinking tool Silk in Section 6.1. Second, it presents the translation rules that interpret the correspondences expressed in EDOAL into an executable Silk script in Section 6.2. Finally, it illustrates how to produce a set of sample links to be assessed and learned in Section 6.3.

6.1 EDOAL and Silk Link Scripting Language

After attribute correspondences are discovered by the K-medoids clustering algorithm, they should be transformed from EDOAL alignment into a Silk script, which are processed by Silk for generating link sets.

6.1.1 Expressive and Declarative Ontology Alignment Language (EDOAL)

Correspondences that are found by an ontology matcher should be expressed in a language. There is an alignment format[1] that expresses simple correspondence like 1-*to*-1 attribute correspondence. For example, the attribute

[1] http://alignapi.gforge.inria.fr/format.html

correspondence *insee:nom*↔*eurostat:name* is a 1-*to*-1 attribute correspondence. However, the alignment format cannot express complex attribute correspondences like *insee:subdivision*↔*eurostat:hasParentRegion*$^{-1}$. It means that the relation *insee:subdivision* is equivalent to the inverse relation of the relation *eurostat:hasParentRegion*. Therefore, Expressive and Declarative Ontology Alignment Language (*EDOAL*)[2] [Scharffe 2009, David 2011] is defined to express complex attribute correspondences.

EDOAL expresses the attribute correspondence *insee:subdivision*↔*eurostat:hasParentRegion*$^{-1}$ as

```
<Cell>
  <entity1>
    <edoal:Relation rdf:about="&insee;subdivision" />
  </entity1>
  <entity2>
    <edoal:Relation>
      <edoal:inverse>
        <edoal:Relation rdf:about="&eurostat;hasParentRegion"/>
      </edoal:inverse>
    </edoal:Relation>
  </entity2>
  <measure rdf:datatype='&xsd;float'>1.</measure>
  <relation>=</relation>
</Cell>
```

measure 1 indicates that the similarity between these two attribute correspondences is 1. *relation* = indicates that the relationship of the two attributes is *equivalence*, rather than *subsumption* or *disjointness*.

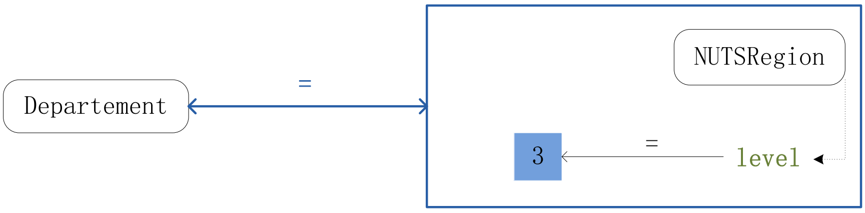


Figure 6.1: Class Correspondence of the concept “Department” in *INSEE* and *EUROSTAT*

EDOAL can also express constructed class correspondences. For instance, the following EDOAL correspondence expresses the correspondence in Figure 6.1:

```
<Cell>
  <entity1>
    <edoal:Class rdf:about="&insee;Departement" />
  </entity1>
```

[2] http://alignapi.gforge.inria.fr/edoal.html

```
  <entity2>
    <edoal:Class>
      <edoal:and rdf:parseType="Collection">
        <edoal:Class rdf:about="&eurostat;NUTSRegion"/>
        <edoal:AttributeValueRestriction>
          <edoal:onAttribute>
            <edoal:Property rdf:about="&eurostat;level"/>
          </edoal:onAttribute>
          <edoal:comparator rdf:resource="&xsd;equals"/>
          <edoal:value>3</edoal:value>
        </edoal:AttributeValueRestriction>
      </edoal:and>
    </edoal:Class>
  </entity2>
  <measure rdf:datatype='&xsd;float'>1.</measure>
  <relation>=</relation>
</Cell>
```

it expresses the equivalence between the class *insee:Departement* in the *INSEE* data set with the class *eurostat:NUTSRegion* whose *eurostat:level* is equal to 3 in the *EUROSTAT* data set.

The EDOAL grammar is summarized below:

For class expressions:

$C ::= u$
$\quad | \neg C \mid C \cap C \mid C \cup C$
$\quad | \; occ(A, cp, n)$
$\quad | \; dom(R, C)$
$\quad | \; type(P, t)$
$\quad | \; val(A, cp, v)$

For property expressions:

$P ::= u$
$\quad | \neg P \mid P \cap P \mid P \cup P \mid R \circ P$
$\quad | \; dom(C)$
$\quad | \; type(t)$
$\quad | \; val(cp, v)$

For relation expressions:

$R ::= u$
$\quad | \neg R \mid R \cap R \mid R \cup R \mid R \circ R$
$\quad | \; reflex(R) \mid sym(R) \mid \overline{R} \mid R^{-1}$
$\quad | \; dom(C)$
$\quad | \; coDom(C)$

For instance & value expressions:

$I ::= u$
$v ::= I \mid literal$

C denotes a class. u denotes a URI. A denotes a property or a relation. P denotes a property. R denotes a relation. I denotes an instance. $\cup, \cap, \neg, \circ, dom$ and $coDom$ denote union, intersection, complement, composition, domain, and range restriction respectively. t denotes a data type. $type(P, t)$ specifies that the data type of a class' property P is t. $type(t)$ specifies that a property's data type is t. v denotes a value expression. cp denotes a comparator. n denotes an integer. R^{-1}, $sym(R)$, $\overline{R}$, and $reflex(R)$ denotes the inverse, the symmetric closure, the transitive closure, and the identity relation respectively. $occ(A, cp, n)$ is a cardinality restriction on a class' attribute A. $val(A, cp, v)$ denotes the value restriction on the attribute A of a class. $val(cp, v)$ denotes the value restriction of a property.

6.1.2 Linkage Specifications

An interlinking specification is a script that describes necessary information on comparing attributes of two given classes for judging whether two instances of the given classes must be linked. It is written with the syntax of interlinking tools. Briefly, the tools require the following information for generating links.

1 Where to get the data sets

2 From which classes to get the instances

3 Which attributes to compare

4 With which comparison method to compare attribute values

5 How to aggregate the similarities

6 How to store the generated links

There are not many differences between these tools. In this thesis, Silk is chosen to execute interlinking and generate a link set, because it is an open source tool. Therefore, this thesis only introduces Silk in order to give a brief introduction to the semi-automatic interlinking tools.

Silk provides a declarative language LSL for specifying which conditions two instances must fulfill in order to be interlinked [Jentzsch 2012]. Assume that we would like to find links on departments of two geographical data sets *INSEE*[3] and *EUROSTAT*[4]. *INSEE* is a data set that describes geographical data in France. *EUROSTAT* is a data set that describes geographical data in Europe. *insee* and *eurostat* are two prefixes that name resources of two data sets respectively. Both data sets describe departments in France. So the interlinking specification written in LSL can be expressed below.

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
    <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
        <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-schema#" />
```

[3] http://rdf.insee.fr/geo/index.html

[4] http://datahub.io/dataset/eurostat-rdf

```
        <Prefix id="insee" namespace="http://rdf.insee.fr/geo/" />
        <Prefix id="eurostat" namespace=
        "http://ec.europa.eu/eurostat/ramon/ontologies/geographic.rdf#" />
    </Prefixes>

    <DataSources>
        <DataSource id="insee" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://localhost:8080/datalift/sparql" />
        </DataSource>

        <DataSource id="eurostat" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://localhost:8080/datalift/sparql" />
        </DataSource>
    </DataSources>

    <Interlinks>
        <Interlink id="departement">
            <LinkType>owl:sameAs</LinkType>

            <SourceDataset dataSource="insee" var="a">
                <RestrictTo>
                    ?a rdf:type insee:Departement .
                </RestrictTo>
            </SourceDataset>

            <TargetDataset dataSource="eurostat" var="b">
                <RestrictTo>
                    ?b rdf:type eurostat:NUTSRegion .
                </RestrictTo>
            </TargetDataset>

            <LinkageRule>
                <Aggregate type="average">
                    <Compare metric="Levenshtein">
                        <TransformInput function="lowerCase">
                            <Input path="?a/insee:nom" />
                        </TransformInput>
                        <TransformInput function="lowerCase">
                            <Input path="?b/eurostat:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="Levenshtein">
                        <TransformInput function="lowerCase">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="lowerCase">
                            <Input path="?e2/id2:hasParentRegion/id2:name" />
                        </TransformInput>
                    </Compare>
                </Aggregate>
            </LinkageRule>

        <Filter />
```

```
            <Outputs>
                <Output type="sparul" >
                    <Param name="uri" value=
                    "http://localhost:8080/openrdf-sesame/repositories/lifted/statements"/>
                    <Param name="parameter" value="update"/>
                </Output>
            </Outputs>
        </Interlink>
    </Interlinks>
</Silk>
```

In the above script, the information for interlinking is specified as follows:

1 Where to get the data sets
Both data sets are queried through Datalift's SPARQL endpoint *http://localhost:8080/datalift/sparql.*

2 From which classes to get the instances
Instances in *INSEE* come from the class *insee:Departement.* Instances in *EUROSTAT* come from the class *eurostat:NUTSRegion.*

3 Which attributes to compare
The property *insee:nom* should be compared with the property *eurostat:name.* The property *insee:nom* of the class that has the relation *insee:subdivision* should be compared with the property *eurostat:name* of the class that is the object of the relation *eurostat:hasParentRegion.*

4 With which comparison method to compare attribute values
Property values will be compared with the method *Levenshtein*, which is a string metric for measuring the difference between two strings. It is used here because all values of the properties *insee:nom* and *eurostat:name* are strings.

5 How to aggregate the similarities
The similarities of both attribute pairs will be aggregated by the method *average* into one similarity value.

6 How to store the generated links
The links will be stored in the public SPARQL endpoint of the Datalift platform. It is *http://localhost:8080/openrdf-sesame/repositories/lifted/statements.*

As an expressive language, LSL provides several comparison methods for attribute values. For strings, the set of comparison methods are *levenshteinDistance*, *levenshtein*, *jaro*, *jaroWinkler*, *equality*, *inequality*, *jaccard*, *dice*, and *softjaccard.* For numbers, there is a comparison method named *num.* For time, the comparison methods are *date* and *dateTime.* For geographical coordinates, the comparison method is *wgs84.*

LSL also provides several aggregation methods to transfer similarities of attribute values into one similarity value. They are, *average*, *max*, *min*, *quadraticMean*, and *geometricMean.*

Operator	Syntax	Description
$/$	$< path > / < prop >$	Leads the query passing from the entity variable as the subject to its object entity
$\backslash$	$< path > \backslash < prop >$	Leads the query passing from the entity variable as the object to its subject entity
[ ]	$< path > [< prop >< comp >< val >]$ $< path > [@lang =< val >]$	Acts as a filter which only passes the values that satisfy the expression defined inside. *comp* can be one of $>, <, >=, <=, =, ! =$

Table 6.1: Silk navigation operators: forward navigation, backward navigation and condition.

Silk offers XPath[5]-inspired navigation operators as presented in Table 6.1. These are used to access the property values of instances.

6.2 From EDOAL to Silk scripts

The goal of this section is to specify how Silk scripts are generated from EDOAL alignments.

First, we present how each component of a Silk script is generated from an alignment:

1. Generating graph patterns from EDOAL expressions allows to fill the Source and Target data set sections (a.k.a., data set parts) of Silk scripts, which selects instances to compare (Section 6.2.1). It does not actually compare resources.
2. Generating linkage rules which actually compare the instances (Section 6.2.2). The linkage rule specifies how to compare the instances selected before.

Then we consider how to assemble a Silk script from these components (Section 6.2.4).

Relating expressive alignments and SPARQL has already been the topic of previous papers [Euzenat 2008, Rivero 2011]. However, these papers are concerned with transferring data with alignments, not interlinking them.

6.2.1 Generation of Source and Target Data Sets

From the correspondence of Figure 6.1, it is required to generate automatically the following graph patterns which respectively extract the two corresponding sets of instances:

[5] http://www.w3.org/TR/xpath20/

```
?r rdf:type insee:Departement .
```

and

```
?n rdf:type nuts:NUTSRegion .
?n nuts:level 3^^xsd:int .
```

These graph patterns will be used in the Silk script as *SourceDataset* and *TargetDataset* selectors. In the data set part of a Silk script, the pair of classes to be compared are defined. The graph patterns of classes are specified here.

In EDOAL, the expressions can be constructed by using the operators or defined through restrictions. Based on the specification of EDOAL expressions with the grammars of Section 6.1.1, all of the concepts are transferred into graph patterns. This transformation depends on the type of entity (Class, Property, Relation), the operations (conjunctions, union, negation, compose, etc.) and the restrictions (domains, values, types, occurrences) they are built from. Each expression (entity) will be translated into a graph pattern.

Silk does not support SPARQL 1.1. Besides, dealing with negation would lead to too many comparisons on instances. Although correspondences with negations ($\neg C$) are considered when designing most of the translation rules, they are not implemented when generating Silk script.

EDOAL translation is designed for the graph pattern T which is described in the following sections. There is one section for each entity types according to the EDOAL grammar.

6.2.1.1 Translation of class expressions

For class expressions, the function $T_s(C)$ takes as argument C the expression to convert and s the entity typed by this class. When invoking T, s is usually a variable.

A class C can be constructed through one of the three operators and, or, not. A class can also be identified by using its URI or defined through a restriction:

- *AttributeDomainRestriction* which restricts the values of relations to be in a particular class;
- *AttributeTypeRestriction* which restricts the values of properties to be in a particular type;
- *AttributeValueRestriction* which restricts the value of properties or relations to a particular value;
- *AttributeOccurrenceRestriction* which constrains the cardinality of a property or relation.

In the following, operators are identified that require SPARQL 1.1. As operation is also used, $!x$ may generate a new variable ($?x$) or a blank (_:x) depending of the context. Here it will always generate a new variable.

$$\begin{aligned}
T_s(u) &= s \texttt{ rdf:type } \langle u \rangle. \\
T_s(\neg C) &= \mathit{MINUS}\ \{T_s(C)\} && (\mathit{SPARQL}\ 1.1) \\
T_s(C \cup C') &= T_s(C)\ \mathit{UNION}\ T_s(C') \\
T_s(C \cap C') &= T_s(C)\ T_s(C') \\
T_s(dom(R, C)) &= T_{s,!o}(R)\ T_{!o}(C) \\
T_s(occ(A, cp, n)) &= T_{s,!o}(A)\ \mathit{FILTER}(\mathit{COUNT}(!o)\ cp\ n) && (\mathit{SPARQL}\ 1.1) \\
T_s(type(P, t)) &= T_{s,!o}(P)\ \mathit{FILTER}(\mathit{datatype}(!o) = t) \\
T_s(val(A, =, v)) &= T_{s,v}(A) \\
T_s(val(A, cp, v)) &= T_{s,!o}(A)\ \mathit{FILTER}(!o\ cp\ v)
\end{aligned}$$

Example: value restriction class

Entity (EDOAL)	**Graph Pattern (SPARQL)** $T_s(u)$ = *?s rdf:type <u> .* $T_s(val(A, cp, v))$ = $T_{s,o}(A)$ *FILTER(?o cp v)* $T_{s,o}(u)$ = *?s <u> ?o .*

`<Class>`
`  <and rdf:parseType="Collection">`
`    <Class rdf:about="#Person" />`
`    <AttributeValueRestriction>`
`      <onAttribute>`
`        <Property rdf:about="#age">`
`      </onAttribute>`
`      <comparator rdf:resource=`
`      "&edoal;#greater-than" />`
`      <value>`
`        <Literal edoal:type="xsd;integer"`
`        edoal:string="17" />`
`      </value>`
`    </AttributeValueRestriction>`
`  </and>`
`</Class>`
	`?x rdf:type <humans:Person> .`
	`?x <humans:age> ?age .`
	`FILTER ( <xsd:integer>(?age) > 17 )`

6.2.1.2 Translation of property expressions

For the property expressions, the translation function $T_{s,o}(P)$ is used such that s is the subject and o is the object in a SPARQL triple pattern, and the predicate P establishes a relationship between the subject s and the object o.

Properties correspond to data properties in OWL. Property expressions can be constructed using one of the operators *and, or, not* and *compose.* They can also be identified by using their URIs or defined through a restriction:

- *PropertyDomainRestriction* which restricts the subjects of properties to be in a particular class;

- *PropertyTypeRestriction* which restricts the values of properties to be in a particular type;
- *PropertyValueRestriction* which restricts the value of properties to a particular value.

$$
\begin{aligned}
T_{s,o}(u) &= s \ \langle u \rangle \ o. \\
T_{s,o}(\neg P) &= \mathit{MINUS}\ \{T_{s,o}(P)\} && (\mathit{SPARQL}\ 1.1) \\
T_{s,o}(P \cup P') &= T_{s,o}(P)\ \ \mathit{UNION}\ T_{s,o}(P') \\
T_{s,o}(P \cap P') &= T_{s,o}(P)\ T_{s,o}(P') \\
T_{s,o}(R \circ P) &= T_{s,!x}(R)\ T_{!x,o}(P) \\
T_{s,o}(\mathit{dom}(C)) &= T_s(C) \\
T_{s,o}(\mathit{type}(t)) &= \mathit{FILTER}(\mathit{datatype}(o) = t) \\
T_{s,o}(\mathit{val}(cp, v)) &= \mathit{FILTER}(o\ cp\ v)
\end{aligned}
$$

Example: type restriction property

Entity (EDOAL)	**Graph Pattern (SPARQL)** $T_{s,o}(type(t)) = FILTER(datatype(?o) = t)$

`<Property>`
`  <PropertyTypeRestriction>`
`    <datatype>`
`      <Datatype rdf:about=`
`      "&xsd;integer"/>`
`    </datatype>`
`  </PropertyTypeRestriction>`
`</Property>`
	FILTER(datatype(?o) = xsd:integer)

6.2.1.3 Translation of relation expressions

For the relation expressions, the translation function $T_{s,o}(R)$ is used such that s is the subject and o is the object in a SPARQL triple pattern, and the predicate R establishes a relationship between the subject s and the object o.

Relations correspond to object properties in OWL. Relation expressions can be constructed using one of the operators *and, or, not* and *compose* as well as closure operators *inverse, symmetric, transitive* and *reflexive.* They can also be identified by using their URIs or defined through a restriction. There are two types of relation restrictions (*RelationDomainRestriction* and *RelationCoDomainRestriction*).

$$\begin{aligned}
T_{s,o}(u) &= s\ \langle u\rangle\ o. \\
T_{s,o}(\neg R) &= \mathit{MINUS}\ \{T_{s,o}(R)\} && (\mathit{SPARQL}\ 1.1) \\
T_{s,o}(R \cup R') &= T_{s,o}(R)\ \ \mathit{UNION}\, T_{s,o}(R') \\
T_{s,o}(R \cap R') &= T_{s,o}(R)\ T_{s,o}(R') \\
T_{s,o}(R \circ R') &= T_{s,!x}(R)\ T_{!x,o}(R') \\
T_{s,o}(R^{-1}) &= T_{o,s}(R) \\
T_{s,o}(sym(R)) &= T_{s,o}(R)\ \ \mathit{UNION}\, T_{o,s}(R) \\
T_{s,o}(\overline{R}) &= s\ R*\ \ o. && (\mathit{SPARQL}\ 1.1) \\
T_{s,o}(reflex(R)) &= T_{s,o}(R)\ \ \mathit{UNION}\ \mathit{FILTER}(s = o) \\
T_{s,o}(dom(C)) &= T_s(C) \\
T_{s,o}(coDom(C)) &= T_o(C)
\end{aligned}$$

Example: transitive relation using the path property (+, *) of SPARQL 1.1

Entity (EDOAL)	**Graph Pattern (SPARQL)** $T_{s,o}(\overline{R})$ = *?s R* ?o*

`<Relation>`
`  <transitive>`
`    <Relation rdf:about="&wine;loc" />`
`  </transitive>`
`</Relation>`
	`?s    <wine:loc>*   ?o .`

6.2.2 Generation of linkage rules

The strategy followed by Silk is made up of four steps, each one corresponding to a different operation:

navigating to the parts of the instances to compare (`Input`);

transforming them to make them comparable (`TransformInput`);

comparing them (`Compare`); and

aggregating the results of their comparison into a single similarity between the two instances (`Aggregate`).

These steps are first presented through an example before presenting the systematic generation of linkage rules.

6.2.2.1 Example

These steps are presented in the following LinkageRule generated for the example:

```
<LinkageRule>
   <Compare metric="levenshteinDistance">
      <TransformInput function="lowerCase">
         <Input path="?a/insee:nom[@lang=’fr’]" />
      </TransformInput>
      <TransformInput function="lowerCase">
         <Input path="?b/eurostat:name" />
      </TransformInput>
   </Compare>
</LinkageRule>
```

The example starts from the two classes to compare. One is *insee:Departement*, the other is *eurostat:NUTSRegions*. For that purpose, the parts of these instances to compare are defined. This is achieved by finding attribute correspondences in the alignment which match these parts. This can be given by the following:

```
<map>
    <Cell>
        <entity1>
            <edoal:Property rdf:about="&insee;nom"/>
        </entity1>
        <entity2>
            <edoal:Property rdf:about="&eurostat;name"/>
        </entity2>
        <relation>equivalence</relation>
        <measure>1.0</measure>
    </Cell>
</map>
```

If for a class correspondence, there is at least one attribute correspondence, then a linkage rule can be generated for the Silk script. In the example, *insee:nom* is a property applying to the class *insee:Departement* and *eurostat:name* is a property applying to the class *eurostat:NUTSRegion*, so the values of these properties can be used to compare the instances. From there, the four steps of the LinkageRule can be generated:

navigation the *nom* property should be inserted into the input path with the same variable that appears in the source data set section. When there is value restriction, it is translated as a constraint, such as *?a/insee:nom[@lang='fr']*. When there is no value restriction, it is simply translated as a path, such as *?b/eurostat:name*.

transformation when two values of the LinkageRule cannot be compared immediately because of coding or format difference, they may be transformed. A transformation function should be set for changing the values into a uniform format. In EDOAL, the transformation may be described in each correspondence. For example, “replace” can change certain letters in a string to other letters, “lowerCase” can change all letters into lower case. In Silk, the transformation can be defined recursively[6].

comparison uses particular metrics to compare the values extracted by the paths. If values are strings, they will use a string comparison method, “levenshteinDistance”; if they are geometric points, the method could compare coordinates (as the “wgs84” method). The choice of metrics is made according to the range of properties. Other optional parameters can be set such as “required”, “weight” or “threshold”. It is possible, for instance, to assign weights depending on the coverage degree of properties.

aggregation is finally performed for issuing one value between two instances from the values obtained by comparing their properties. It can be the “average” which computes the weighted average result of comparing results. It also can be “max” to set the final aggregation value to be the maximum similarity of returned comparing results. Other optional parameters are “required” and “weight”. Weights could be set based on the confidence in the property correspondence.

Silk offers a whole battery of operations:

- navigation operators, inspired from the XPath language are provided in Table 6.1;
- transformation functions (lowerCase, etc.);
- comparison functions (levenshteinDistance, etc.);
- aggregation operators (max, min, average, etc.).

6.2.2.2 Aggregation generation

For each pair of corresponding classes in the alignment $\mathbb{A}$, a linkage rule will be generated aggregating the similarities of all the properties. It is possible to select the relevant properties to compare through a learning algorithm by learning assessed links.

Currently, given two class URIs u and u', we generate the comparisons of all attributes and compute the average value of their comparison results:

$$R_{s,s'}(u,u',\mathbb{A}) = Average_{\langle a,a',=,tr\rangle \in \mathcal{K}(u,u',\mathbb{A})} C_{s,range(a),s',range(a')}(a,a',tr)$$

$\mathcal{K}(u,u',\mathbb{A})$ is the set of property or relation correspondences related to the class u and the class u' in $\mathbb{A}$, i.e., those whose domains are u and u' respectively. s, s' are two variables that will be bound to instances of the classes u and u'. The datatypes associated to the properties are passed to the transformation. The transformation tr if available in the EDOAL correspondence is also passed as argument.

[6] https://www.assembla.com/spaces/silk/wiki/Transformation

Because class description in EDOAL may be non-atomic, it is specified below how the compound expressions are compared:

$$
\begin{aligned}
R_{s,s'}(C \cap C', C'', \mathbb{A}) &= Min(R_{s,s'}(C, C', \mathbb{A}), R_{s,s'}(C', C'', \mathbb{A}))\\
R_{s,s'}(C \cup C', C'', \mathbb{A}) &= Max(R_{s,s'}(C, C', \mathbb{A}), R_{s,s'}(C', C'', \mathbb{A}))\\
R_{s,s'}(C, C' \cap C'', \mathbb{A}) &= Min(R_{s,s'}(C, C', \mathbb{A}), R_{s,s'}(C, C'', \mathbb{A}))\\
R_{s,s'}(C, C' \cup C'', \mathbb{A}) &= Max(R_{s,s'}(C, C', \mathbb{A}), R_{s,s'}(C, C'', \mathbb{A}))\\
R_{s,s'}(\neg C, C', \mathbb{A}) &= \varnothing\\
R_{s,s'}(C, \neg C', \mathbb{A}) &= \varnothing\\
R_{s,s'}(C, occ(A, cp, n), \mathbb{A}) &= \varnothing\\
R_{s,s'}(C, dom(R, C'), \mathbb{A}) &= \varnothing\\
R_{s,s'}(C, type(P, t), \mathbb{A}) &= \varnothing\\
R_{s,s'}(C, val(A, cp, v), \mathbb{A}) &= \varnothing\\
R_{s,s'}(occ(A, cp, n), C', \mathbb{A}) &= \varnothing\\
R_{s,s'}(dom(R, C), C', \mathbb{A}) &= \varnothing\\
R_{s,s'}(type(P, t), C', \mathbb{A}) &= \varnothing\\
R_{s,s'}(val(A, cp, v), C', \mathbb{A}) &= \varnothing
\end{aligned}
$$

here $\varnothing$ values have no influence on operators Min, Max or $Average$. It only means that no comparison is generated in this case. These constraints are usually used in the expression of the data sources and thus are not necessary anymore in the comparison.

6.2.2.3 Comparison generation

The comparison between two attributes is generated by $C_{s,t,s',t'}(a, a', tr)$ which literally means: generate the comparison of instances s and s' attributes a and a' of types t and t', knowing that a transformation tr may be useful for converting a units into a''s.

This comparison is generated in two distinct ways depending on whether the attribute expressions are structural, i.e., built from constructors $\neg$, $\cap$, $\cup$ or sym, or navigational, i.e., built from constructor $\circ$, $^{-1}$, $\bar{\cdot}$, $reflex$ or constraints dom, $coDom$, $type$ or val. In the former case, C will aggregate the comparison of subcomponents, while in the latter it will generate paths for applying the comparison.

The comparisons are applied to attributes based on their correspondences. It is assumed that they are in a form $P \dot{\cap} X$ in which P is a path in inverted normal form, i.e., in which all converses are set at the lower level, and X is the set of constraints on the extremities of attributes. This can be achieved by the normalization function

N:

$$\begin{aligned}
N(u) &= u \\
N(u^{-1}) &= u^{-1} \\
N((A^{-1})^{-1}) &= N(A) \\
N((\neg A)^{-1}) &= \neg N((A^{-1})) \\
N((A \cap A')^{-1}) &= N(A^{-1}) \cap N(A'^{-1}) \\
N((A \cup A')^{-1}) &= N(A^{-1}) \cup N(A'^{-1}) \\
N((R \circ R')^{-1}) &= N(R'^{-1}) \circ N(R^{-1}) \\
N(sym(R)^{-1}) &= sym(N(R)) \\
N(reflex(R)^{-1}) &= reflex(N(R^{-1})) \\
N(\overline{R}^{-1}) &= \overline{N(R^{-1})} \\
N(dom(C)^{-1}) &= coDom(N(C)) \\
N(coDom(C)^{-1}) &= dom(N(C))
\end{aligned}$$

and separating at each nesting levels, the constraints from the paths.

The main navigational part is considered:

$$\begin{aligned}
C_{s,t,s',t'}(\neg a, a', tr) &= \varnothing \\
C_{s,t,s',t'}(a, \neg a', tr) &= \varnothing \\
C_{s,t,s',t'}(a' \cap a'', a, tr) &= Min(C_{s,t,s',t'}(a', a, tr), C_{s,t,s',t'}(a'', a, tr)) \\
C_{s,t,s',t'}(a' \cup a'', a, tr) &= Max(C_{s,t,s',t'}(a', a, tr), C_{s,t,s',t'}(a'', a, tr)) \\
C_{s,t,s',t'}(a, a' \cap a'', tr) &= Min(C_{s,t,s',t'}(a, a', tr), C_{s,t,s',t'}(a, a'', tr)) \\
C_{s,t,s',t'}(a, a' \cup a'', tr) &= Max(C_{s,t,s',t'}(a, a', tr), C_{s,t,s',t'}(a, a'', tr)) \\
C_{s,t,s',t'}(a, sym(a'), tr) &= Max(C_{s,t,s',t'}(a, a', tr), C_{s,t,s',t'}(a, a'^{-1}, tr)) \\
C_{s,t,s',t'}(sym(a), a', tr) &= Max(C_{s,t,s',t'}(a, a', tr), C_{s,t,s',t'}(a^{-1}, a', tr))
\end{aligned}$$

This works with ¬, ∪, ∩ and *sym* at the topmost level. Min is used to compare values when there is a ∩ constructor, because the comparison should satisfy both components. Max is used to compare values when there is a ∪ constructor, because the comparison only has to satisfy one of the components. In case of symmetric closure, Max is used as well because it is sufficient that one of the component be satisfied.

When none of the above equations can be made to work, the expression is expanded in a comparison. The way it is expanded depends on the value at the extremity of the path, i.e., if the correspondence is between properties or relations.

In case of relations, i.e., if t and t' are classes, then the extremities of the paths are, in turn, compared as instances:

$$C_{s,t,s',t'}(a, a', tr) = R_{tr(s/P(a)),s'/P(a')}(t, t', \mathbb{A})$$

In case of properties, if t and t' are datatypes, then the paths are directly compared:

$$C_{s,t,s',t'}(a, a', tr) = M_{t,t'}(tr(s/P(a)), s'/P(a'))$$

These operations require:

- reducing each member of the comparison to a path (P);
- replacing the comparison by a concrete operator ($M_{t,t}$);
- using the transformation (tr) if necessary.

The transformation operation tr, originating from the correspondence, is here only applied to the first element. It is possible, in EDOAL, to define transformations of the second element as well. In such a case, it should be applied to the second element.

Some transformations, such as `concat` are available in both EDOAL and Silk, some other have to be converted, such as `&fn;safe-divide` which is converted into `divide`, some other are external such as the EDOAL web service calls, e.g.,“*http://www.google.com/finance/converter*”.

The comparison operator ($M_{t,t'}$) is chosen depending on the datatype of the values to be compared:

$$\begin{array}{lcl}
M_{t,t'} & = & \text{“}levenshtein\text{”} \\
 & \text{if} & t = t' = string \\
 & = & \text{“}jaccard\text{”} \\
 & \text{if} & t = token\ or\ t' = token \\
 & = & \text{“}num\text{”} \\
 & \text{if} & t = t' = integer \mid float \mid other\ number\ types, \\
 & = & \text{“}date\text{”} \\
 & \text{if} & t = t' = \mathit{YYYY\text{-}MM\text{-}DD} \\
 & = & \text{“}dateTime\text{”} \\
 & \text{if} & t = t' = xsd{:}dateTime \\
 & = & \text{“}wgs84\text{”} \\
 & \text{if} & t = t' = coordinate
\end{array}$$

6.2.3 Navigation generation

Silk offers XPath-inspired navigation operators as presented in Table 6.1. The function P generates a path corresponding to a property or relation definition from a node s.

$$\begin{aligned}
P(u) &= /u \\
P(u^{-1}) &= \backslash u \\
P(R \circ A) &= P(R)\ P(A) \\
P(P \dot{\cap} X) &= P(P)[Q(X)] \\
P(reflex(R)) &= s/(P_s(R) - s/|\backslash) \\
P(\bar{R}) &= P(R)*
\end{aligned}$$

Silk cannot express the restrictions of $dom(C)$, $coDom(C)$, $type(t)$ and $val(cp, v)$ (except numeric operator and language tag operator), so the XPath syntax is used here to express them.

$$\begin{aligned}
Q(dom(C)) &= parent :: C && (NoSilk) \\
Q(type(t)) &= child :: t && (NoSilk) \\
Q(coDom(C)) &= child :: C && (NoSilk) \\
Q(val(cp, v)) &= child :: node()\ cp\ v
\end{aligned}$$

Here, *NoSilk* means there is no corresponding expression in Silk for the expressions in EDOAL.

6.2.4 Generation of Silk scripts

After setting the source and target data sets and the linkage rules, there are a few other information to be provided for obtaining a complete Silk script.

- All used prefixes should be specified, especially the prefixes of the two data sets. This can be directly obtained from the prefixes that are contained in the correspondence expressions in EDOAL.
- The data set source and target source can be a local file or a SPARQL endpoint. They are obtained from the information provided by the user.
- The output of the link set can also be a local file or a SPARQL endpoint.
- A parameter named "limit" may restrain how many links are going to be returned after the comparison. If it is not set, all links will be returned[7].

A Silk script can thus be created formally as in Algorithm 2.

The input of the algorithm is an alignment between the data sets. The output is the Silk script. For all class correspondences in the alignment, if there exists an attribute correspondence belonging to the classes of any class correspondence, a Silk script is created. The script is mainly composed of three parts. One is the source data set *SourceDataset($T_{?s}$(e))*. Another one is the target data set *TargetDataset($T_{?s'}$(e'))*. The last is the linkage rule section

[7] https://www.assembla.com/spaces/silk/wiki/Link_Specification_Language

```
Input: $\mathbb{A}$
Output: Silk script
for all $< e, e', r > \in \mathbb{A}$ do
    if $\exists < a, a', r > \in \mathbb{A}$,
    such that $\{?s\ a\ ?o.\ ?s\ rdf{:}type\ e.\}$ and $\{?s'\ a\ ?o'.\ ?s'\ rdf{:}type\ e'.\}$ then
        generate Interlink section with SourceDataset($T_{?s}(e)$);
        TargetDataset($T_{?s'}(e')$);
        LinkageRule($R_{?s,?s'}(e, e', \mathbb{A})$);
    end
end
return the LinkageRule;
```

Algorithm 2: The Silk Script Building Algorithm

LinkageRule($R_{?s,?s'}(e, e', \mathbb{A})$) of all attribute correspondences belonging to the class correspondence.

So, the final Silk script for the example in Figure 6.1 is:

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
    <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
        <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-schema#" />
        <Prefix id="xsd" namespace="http://www.w3.org/2001/XMLSchema#" />
        <Prefix id="dc" namespace="http://purl.org/dc/elements/1.1/" />
        <Prefix id="cc" namespace="http://creativecommons.org/ns#" />
        <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#"/>
        <Prefix id="dcterms" namespace="http://purl.org/dc/terms/" />
        <Prefix id="xmlns" namespace=
        "http://ec.europa.eu/eurostat/ramon/ontologies/geographic.rdf#" />
        <Prefix id="insee" namespace="http://rdf.insee.fr/geo/" />
        <Prefix id="eurostat" namespace=
        "http://ec.europa.eu/eurostat/ramon/ontologies/geographic.rdf#" />
    </Prefixes>

    <DataSources>
        <DataSource id="insee" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://localhost:8080/datalift/sparql" />
        </DataSource>
        <DataSource id="eurostat" type="sparqlEndpoint">
            <Param name="endpointURI" value="http://localhost:8080/datalift/sparql" />
        </DataSource>
    </DataSources>

    <Interlinks>
        <Interlink id="region">
            <LinkType>owl:sameAs</LinkType>

            <SourceDataset dataSource="insee" var="a">
                <RestrictTo>
                    ?a rdf:type insee:Departement .
                </RestrictTo>
            </SourceDataset>
            <TargetDataset dataSource="eurostat" var="b">
                <RestrictTo>
                    { ?b rdf:type eurostat:NUTSRegion .
                      ?b eurostat:level 3^^xsd:int . }
                </RestrictTo>
            </TargetDataset>

            <LinkageRule>
                <Compare metric="levenshteinDistance">
                    <TransformInput function="lowerCase">
                        <Input path="?a/insee:nom[@lang='fr']" />
                    </TransformInput>
                    <TransformInput function="lowerCase">
                        <Input path="?b/eurostat:name" />
                    </TransformInput>
                </Compare>
            </LinkageRule>

           <Filter />

            <Outputs>
                <Output type="sparul" >
                    <Param name="uri" value=
                    "http://localhost:8080/openrdf-sesame/repositories/lifted/statements"/>
                    <Param name="parameter" value="update"/>
                </Output>
            </Outputs>
        </Interlink>
    </Interlinks>
</Silk>
```

6.2.5 Conclusion

The transformation from an EDOAL alignment file to a Silk script has been theoretically defined and implemented. This allows for generating links across two RDF data sets.

6.3 Sample Link Generation by Silk

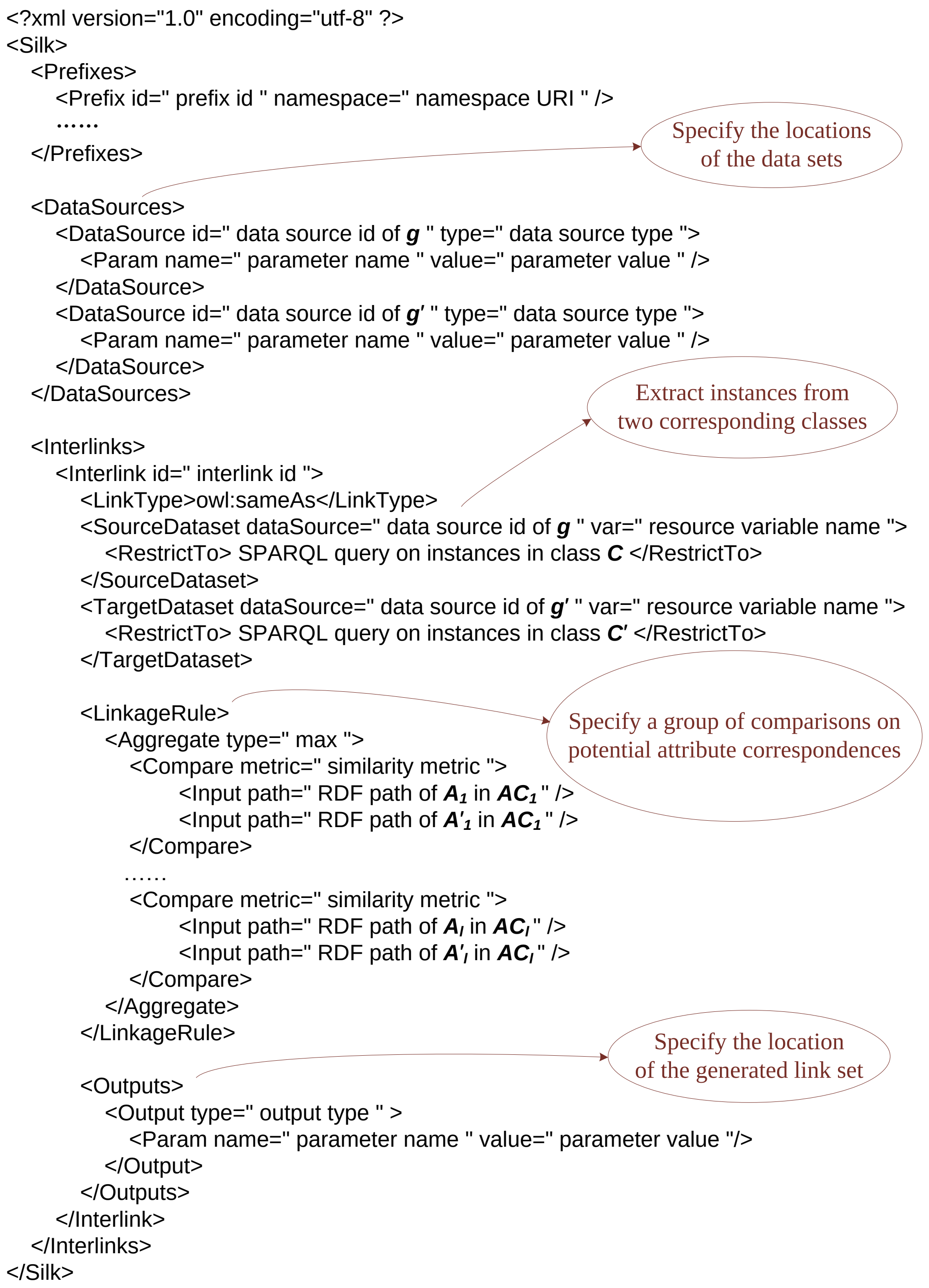
A set of sample links can be produced by executing a Silk script that is transferred from an interlinking pattern that combines all potential attribute correspondences into a disjunction expression. The interlinking pattern is shown in Figure 6.2. In the figure, $AC_1, AC_2, \ldots, AC_l$ stand for l potential attribute correspondences resulting from the K-medoids clustering step. A_i and A'_i are two attributes in the attribute correspondence AC_i, where $i = 1, 2, \ldots, l$. In order to obtain the sample link set, the graph queries of potential attribute correspondences are aggregated with the aggregation method "max". Since the aggregation method "max" helps select all links each of which contains at least one potential attribute correspondence, any correct link that contains at least one potential attribute correspondences will be generated. Here, if there are two similar attribute values across two compared instances, we assume that the two attributes form an attribute correspondence in the link that is formed by the two compared instances. With such a sample link set, users are able to assess the links of the set.

Other aggregation methods are not suitable for generating the sample link set. In fact, Silk also can generate a set of links for the two data sets with other aggregation methods, such as “average”. Unlike the aggregation method “max”, the aggregation method “average” returns the mean value of all similarities on attribute values, which are normalized into the range [-1,1]. As specified by Silk, one link is generated if and only if the similarity between two instances fall in $[0, 1]$. For example, in order to compare two instances between which there are three pairs of corresponding attributes (e.g., derived by K-medoids), Silk can help compute similarities for the three pairs of corresponding attributes. Assume that their similarity values are -0.9, -0.6, and 0.6 respectively, then, the aggregation method “max” and the aggregation method “average” return max{-0.9,-0.6,0.6}=0.6 and $\frac{-0.9+(-0.6)+0.6}{3}=-0.3$ respectively. Hence, according to the aggregation method “max”, this pair of instances can be viewed as a link, while it cannot be treated as a link according to the aggregation method “average”. Obviously, the aggregation method “max” represents a more relaxed similarity restriction than the aggregation method “average” in generating different sets of presumed links.

Figure 6.3 shows the interlinking result by using two different aggregation methods. In the figure, there are two potential attribute correspondences. One is *insee:code_ departement↔eurostat:code*, which is incomparable. The other is *insee:nom↔eurostat:name*, which is comparable. The fragment for comparing these two attribute correspondences are expressed in LSL. Obviously different aggregation methods will result in different interlinking results. If the aggregation method is “max”, the correct link *http://rdf.insee.fr/geo/2010/DEP_ 75 owl:sameAs http://ec.europa.eu/eurostat/ramon/rdfdata/nuts2008/FR101* is produced, because there is an attribute correspondence *insee:nom↔eurostat:name* that exists across the two instances. Property *insee:nom*’s value “Paris” equals to property *eurostat:name*’s value “Paris”. The aggregation method “max” recognizes two instances as a correct link when there is one attribute correspondence existing between the two instances. But such a link will not be produced if the aggregation method “average” is used, because it requires more than one attribute correspondence existing between the two instances, otherwise the two instances will not be recognized as the same.

Thus, the sample link set of the example presented in Section 2.1 of Chapter 2 is generated by the following Silk script, which is transferred from the interlinking pattern that is expressed as a disjunction of potential attribute correspondences.

```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
    <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
        <Prefix id="rdfs" namespace="http://www.w3.org/2000/01/rdf-schema#" />
```



```
<?xml version="1.0" encoding="utf-8" ?>
<Silk>
   <Prefixes>
      <Prefix id=" prefix id " namespace=" namespace URI " />
      ……
   </Prefixes>

   <DataSources>
      <DataSource id=" data source id of g " type=" data source type ">
         <Param name=" parameter name " value=" parameter value " />
      </DataSource>
      <DataSource id=" data source id of g′ " type=" data source type ">
         <Param name=" parameter name " value=" parameter value " />
      </DataSource>
   </DataSources>

   <Interlinks>
      <Interlink id=" interlink id ">
         <LinkType>owl:sameAs</LinkType>
         <SourceDataset dataSource=" data source id of g " var=" resource variable name ">
            <RestrictTo> SPARQL query on instances in class C </RestrictTo>
         </SourceDataset>
         <TargetDataset dataSource=" data source id of g′ " var=" resource variable name ">
            <RestrictTo> SPARQL query on instances in class C′ </RestrictTo>
         </TargetDataset>

         <LinkageRule>
            <Aggregate type=" max ">
               <Compare metric=" similarity metric ">
                     <Input path=" RDF path of A1 in AC1 " />
                     <Input path=" RDF path of A′1 in AC1 " />
               </Compare>
              ……
               <Compare metric=" similarity metric ">
                     <Input path=" RDF path of Al in ACl " />
                     <Input path=" RDF path of A′l in ACl " />
               </Compare>
            </Aggregate>
         </LinkageRule>

         <Outputs>
            <Output type=" output type " >
               <Param name=" parameter name " value=" parameter value "/>
            </Output>
         </Outputs>
      </Interlink>
   </Interlinks>
</Silk>
```

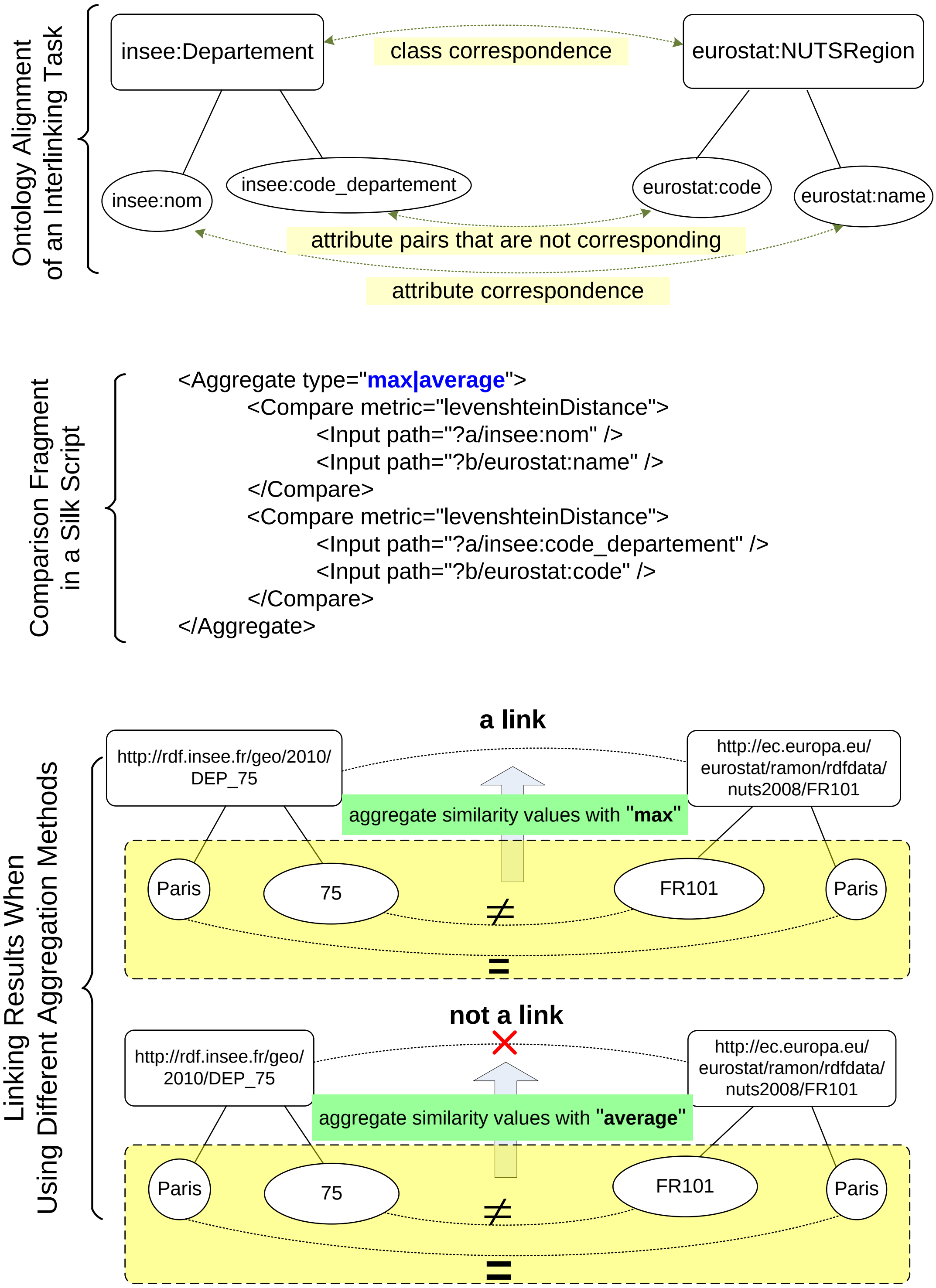
Specify the locations of the data sets

Extract instances from two corresponding classes

Specify a group of comparisons on potential attribute correspondences

Specify the location of the generated link set

Figure 6.2: Silk Script of the Interlinking Pattern that Generates Sample Links



```
<Aggregate type="max|average">
        <Compare metric="levenshteinDistance">
                <Input path="?a/insee:nom" />
                <Input path="?b/eurostat:name" />
        </Compare>
        <Compare metric="levenshteinDistance">
                <Input path="?a/insee:code_departement" />
                <Input path="?b/eurostat:code" />
        </Compare>
</Aggregate>
```

Figure 6.3: Two Consequences of Using Different Aggregation Methods

```
        <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#" />
        <Prefix id="id2" namespace=
        "http://ec.europa.eu/eurostat/ramon/ontologies/geographic.rdf#" />
        <Prefix id="id1" namespace="http://rdf.insee.fr/geo/" />
    </Prefixes>

    <DataSources>
        <DataSource id="id1" type="file">
            <Param name="file" value=
            "C:/Zhengjie/study/eclipse3/SILKscript2/regions-2010.rdf"/>
            <Param name="format" value="RDF/XML" />
        </DataSource>

        <DataSource id="id2" type="file">
            <Param name="file" value=
            "C:/Zhengjie/study/eclipse3/SILKscript2/nuts2008_complete.rdf"/>
            <Param name="format" value="RDF/XML" />
        </DataSource>
    </DataSources>

    <Interlinks>
        <Interlink id="no1">
            <LinkType>owl:sameAs</LinkType>
            <SourceDataset dataSource="id1" var="e1">
                <RestrictTo>
                ?e1 rdf:type id1:Departement .
                </RestrictTo>
            </SourceDataset>
            <TargetDataset dataSource="id2" var="e2">
                <RestrictTo>
                ?e2 rdf:type id2:NUTSRegion . ?e2 id2:level ?ov1 . FILTER (?ov1=3) .
                </RestrictTo>
            </TargetDataset>
            <LinkageRule>
                <Aggregate type="max">
                    <Compare metric="levenshtein">
                        <Input path="?e1\ id1:subdivision/id1:code_region" />
                        <Input path="?e2/id2:level" />
                    </Compare>
                    <Compare metric="levenshtein">
                        <Input path="?e1/id1:code_departement" />
                        <Input path="?e2/id2:hasParentRegion/id2:code" />
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:hasParentRegion/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2\ id2:hasParentRegion/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:name" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:code" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:name" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2\ id2:hasParentRegion/id2:code" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:hasParentRegion/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2\ id2:hasParentRegion/id2:name" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
```

```
                        <TransformInput function="tokenize">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:code" />
                        </TransformInput>
                    </Compare>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1\ id1:subdivision/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2\ id2:hasParentRegion/id2:code" />
                        </TransformInput>
                    </Compare>
                </Aggregate>
            </LinkageRule>

            <Filter />

            <Outputs>
                <Output type="file">
                    <Param name="file" value=
                    "C:/Zhengjie/study/eclipse3/trunk-test/accepted_links.xml"/>
                    <Param name="format" value="alignment"/>
                </Output>
            </Outputs>
        </Interlink>

    </Interlinks>
</Silk>
```

6.4 Conclusion

In this chapter, a set of translation rules are presented to show how to translate E-DOAL correspondences into graph patterns. A Silk script can be built for generating links across two RDF data sets with these graph patterns.

Furthermore, the chapter also proposes the method of generating a set of sample links for users to assess. Each link in the set satisfies at least one potential attribute correspondence that is discovered by Chapter 5. These sample links will be sent to users to assess. The assessed sample links will assist a learning method to construct and improve the interlinking pattern that covers correct links across two data sets.

Next chapter will introduce a learning method to construct and improve the interlinking pattern of two interlinking RDF data sets with the assessed sample links, so as to find out a link set with high F-measure.

Chapter 7

Constructing and Improving the Interlinking Pattern with Assessed Links by Version Space

Contents

This chapter introduces the learning method of constructing and improving the interlinking pattern. First, it introduces the definition of Version Space (VS), as well as its pros and cons in Section 7.1 and Section 7.2. Second, it shows an improvement on Version Space, Extended Version Space (EVS), in Section 7.3. Thirdly, it presents the way to construct and improve the interlinking pattern with Extended Version Space in Section 7.4.

7.1 Preliminary Definition

Version Space [Mitchell 1982, Mitchell 1997] is a supervised learning method that constructs and improves a classifier by learning labeled examples one by one. Figure 7.1 shows the learning process of Version Space. The entire objective is to build a composition of some conditions such that all labeled examples are compatible with, which is called a *hypothesis*. During the learning process, there are two sets of hypotheses always being maintained. One is the set of the most strict compositions of the conditions that cover all learned positive examples, and they are called *Specialized Hypotheses*. The other is the set of the most general compositions of conditions that cover all learned negative examples, and they are called *Generalized Hypotheses*. With more positive examples being learned, the specialized hypotheses become more general. When more negative examples are learned, the generalized hypotheses become stricter. After learning all labeled examples (including positive ones and negative ones), the two sets finally converge into one set in Version Space.

The Version Space learning process can be used to build an interlinking pattern which is able to precisely justify all links across the data sets. The interlinking pattern is the hypothesis in the interlinking process. More specifically, given a set of sample links assessed as either *positive* or *negative* by users, it is required to construct an interlinking pattern with attribute correspondences. Consequently, the interlinking pattern is compatible with all of positive links (i.e., assessed correct links) and conflicting to all of negative links (i.e., assessed incorrect links).

Since the learning process is to put the set of potential attribute correspondences of each sample link that is assessed by users into the interlinking pattern, the set of potential attribute correspondences of each sample link should be discovered. There are two instances that form a sample link. If two attribute values of two linked instances is the same, we assume that these two attributes form a potential attribute correspondence in the sample link. For each pair of instances i and i' that forms a sample link, we use an l-tuple $(B^{(j)})$, $j = 1, \ldots, l$ with a sequence of bit values to denote the similarity value between their attribute values, where the bit value (denoted by $B^{(j)}$) is defined in Formula (7.1). In Formula (7.1), $\equiv$ and $\not\equiv$ refer to similarity and non-similarity between two property values $i.p_j$ and $i'.p'_j$ respectively. Each attribute comparison is determined by some *similarity metric* (such as "Levenshtein" [Levenshtein 1966] and "Jaccard" [Jaccard 1912]). The bit value is equal to 0 or 1 (denoted by 0|1 in the following text), representing non-similarity and similarity respectively. For example, if there are three attribute correspondences across two corresponding classes, l is equal to 3, and then there will be eight possible tuples of similarity values, (0,0,0),(0,0,1),$\cdots$,(1,1,1) for all instance pairs between these two classes.

$$B^{(j)} = \begin{cases} 1 & i.p_j \equiv i'.p'_j \\ 0 & i.p_j \not\equiv i'.p'_j \end{cases} \text{ where } j = 1, \ldots, l \tag{7.1}$$

For each potential attribute correspondence across two corresponding classes via the K-medoids clustering, its similarity value is either 1 if the two attribute values v and v' on a user-assessed link are similar based on a similarity metric or 0 otherwise. Then, a ***binary similarity sequence*** (**BSS**) for the attribute pairs is shown below.

$$\begin{array}{rl} & (AC_1, \quad AC_2, \quad \ldots, AC_l) \\ \text{Binary Similarity Sequence} = & (0|1, \quad 0|1, \quad \ldots, 0|1) \end{array} \tag{7.2}$$

In Formula (7.2), $AC_1, AC_2, \ldots, AC_l$ stand for l potential attribute correspondences generated by the clustering method of Chapter 5. So **the input of the Version Space learning step** is a set of binary similarity sequences. **The output of the Version Space learning step** is an interlinking pattern composed of binary similarity sequences, which can cover all positive links and filter out all negative links simultaneously. In machine learning [Mitchell 1982], the input and output of a supervised learning algorithm are expressed with *instance language* and

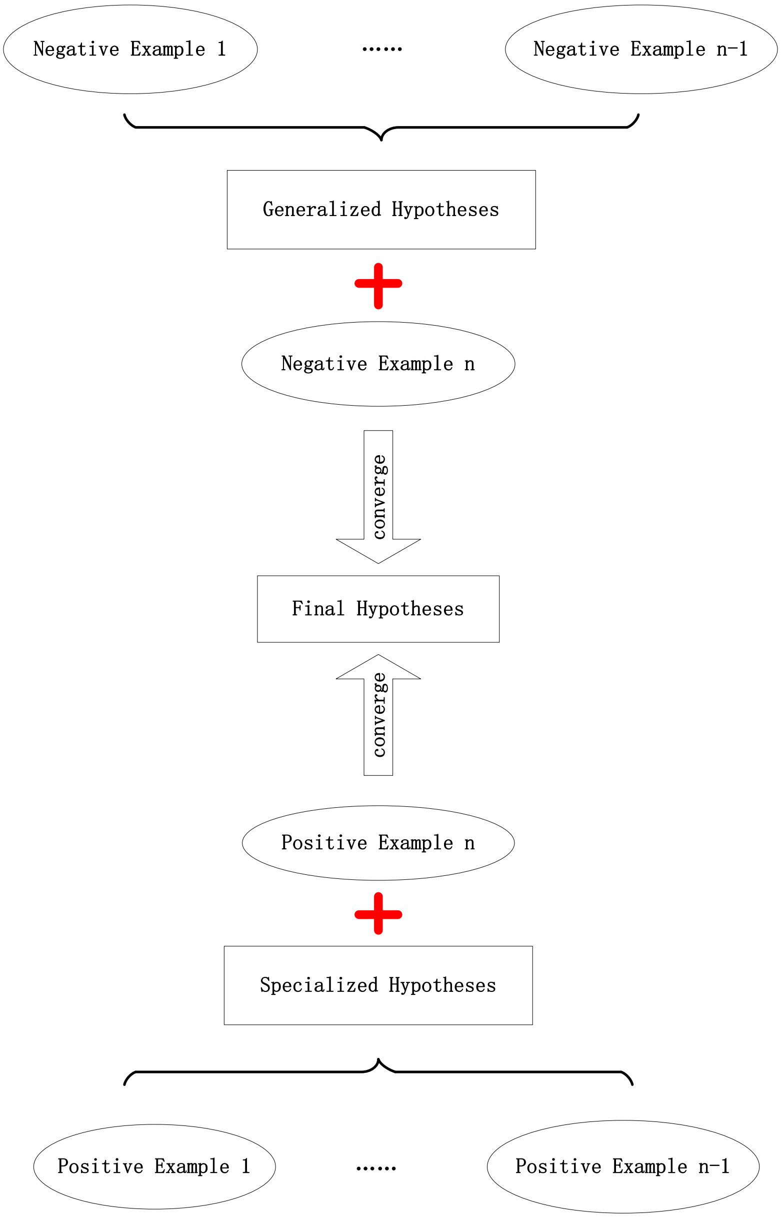


Figure 7.1: Learning Process of Version Space

generalization language respectively. With respect to the interlinking problem of this thesis, the instance language of Version Space is a binary similarity sequence. Each bit of the instances' binary similarity sequences is either 0 or 1. The generalization language is also a binary similarity sequence. Each bit of such a binary similarity sequence is 0, 1 or ×, where × means either 0 or 1.

In general, the computed similarity of two attribute values may not exactly be equal to 0 or 1, but another decimal in the range [0,1], such as 0.75. In our design, we always convert the similarity value to a binary value with a threshold T. If the computed similarity value is larger than T, we assume that the similarity is 1. If the computed similarity value is equal to or lower than T, we assume that the similarity is 0.

7.2 Pros and Cons of Version Space

The key advantage of adopting Version Space is that it can precisely find out an interlinking pattern that distinguishes the positive and negative links. As for the interlinking process, the Version Space method traverses all hypotheses that can satisfy the labeled examples (or assessed links) [Mitchell 1982], and creates two sets of patterns. The first one is called a set of *specialized patterns* (denoted as Pat_S), each of which is composed of as many restrictions as possible. The other one is called a set of *generalized patterns* (denoted as Pat_G), each of which just takes into account the least-restricted conditions. More specifically, Pat_S only covers all positive links, while Pat_G is able to covers any links except negative links. Each Pat_S must be subsumed by one Pat_G. Each Pat_G must subsume one Pat_S. Both of the two sets of patterns can cover all positive links, and do not cover any negative link. For the Version Space method, the two sets of patterns will eventually converge to a set of binary similarity sequences after learning all assessed links, which is called a set of *conjunctive patterns*, if there exist several qualified binary similarity sequences, or converge to a null pattern otherwise. The conjunctive pattern is a conjunction of attribute correspondences across two corresponding classes, which is represented as a binary similarity sequence with bit values 0, 1 and ×.

In practice, there are many cases in which a set of conjunctive patterns cannot be built, i.e., it is possible that there is no conjunctive pattern that can cover all positive links and filter all negative links. In this situation, some conjunctive patterns can be built and each of them can only cover a portion of the positive links and filter all negative links, while none of them can cover all positive links. Thus, generalized patterns for such cases can be built by combining all these conjunctive patterns, so that all positive links can be covered. An example is shown below, where $AC_1, AC_2, AC_3, AC_4, AC_5$ denote potential attribute correspondences resulted from

K-medoids clustering.

$$\begin{array}{ccccccc}
\text{Link No.} & \text{Type} & AC_1 & AC_2 & AC_3 & AC_4 & AC_5 \\
1 & \text{Positive} & (0, & 1, & 1, & 0, & 0) \\
2 & \text{Positive} & (0, & 0, & 1, & 1, & 0) \\
3 & \text{Negative} & (1, & 0, & 1, & 0, & 0) \\
4 & \text{Negative} & (0, & 0, & 1, & 0, & 0)
\end{array} \tag{7.3}$$

In this example, with the two positive links, there is only one specialized pattern can be produced, which is (0,×,1,×,0), where $\times = 0|1$. Given a negative link, we can derive generalized patterns by traversing all binary similarity sequences that are conflicting to the negative link with at least one bit (e.g., attribute correspondence AC_i). In Example (7.3), based on the third assessed link (a negative link), the candidate generalized patterns are binary similarity sequences: (0,×,×,×,×), (×,1,×,×,×), (×,×,0,×,×), (×,×,×,1,×), (×,×,×,×,1), where ×=0|1. Then, the generalized pattern with the first three links is supposed to be (0,×,×,×,×), in that only (0,×,×,×,×) contains the specialized pattern. When combining the fourth assessed link, however, a conflict will be induced, because there will be no feasible binary similarity sequence that can cover all the positive links and filter out all the negative links simultaneously.

Disjunctive Version Space (DVS) [Mitchell 1997, Sebag 1996] adopts disjunctive constructor to solve the above problem. The instance language of Disjunctive Version Space is a binary similarity sequence, which is the same with the one of Version Space. Each bit of the instances' binary similarity sequences is either 0 or 1. The generalization language is a disjunction of these binary similarity sequences. However, its output is represented in a more complicated format with a large number of binary similarity sequences, because the binary similarity sequences in the pattern cannot induce a more concise expression, if there is only one bit difference between them. In contrast, we design an Extended Version Space, which can not only cover all positive links and filter all negative links simultaneously, but we show that its expression is in a very concise representation.

7.3 Extended Version Space

We present how to build an interlinking pattern by Extended Version Space in this section in details.

The instance language of Extended Version Space is a binary similarity sequence, which is the same with the ones of Version Space and Disjunctive Version Space. Each bit of the instances' binary similarity sequences is either 0 or 1. The generalization language is a disjunction of binary similarity sequences. Each bit of the instances' binary similarity sequences is 0, 1 or ×.

A disjunctive pattern that is generated by Extended Version Space can be recursively represented as Formula (7.4), where $(B^{(j)})$ represents a binary similarity sequence with l bits, where $j = 1, \ldots, l$.

$$Pattern \ ::= \ (B^{(j)}) \mid Pattern \cup (B^{(j)}) \tag{7.4}$$

In this formula, ∪ denotes an union operation (or disjunction) in the set theory. This formula means that a pattern is either a binary similarity sequence, or a union of such sequences. We give a simple example based on the definition. Assume that there are two data sets which are represented in English and French respectively, and we are given two sample links, each of which connects two instances and there are three potential attribute correspondences generated by the K-medoids clustering step (name↔nom, sex↔gender, and supervisor↔directeur) to compare, i.e., three bits per binary similarity sequence. Assume that these two links can be represented as binary similarity sequences (1,1,0) and (1,1,1) respectively, and they are both assessed as *positive* by users, implying that they are correct links. Then, the disjunctive pattern that satisfies these two links simultaneously can be written as $(1,1,0) \cup (1,1,1)$, or in a concise format $(1,1,\times)$ where $\times = 0|1$. A disjunctive pattern can be represented with a disjunction of multiple binary similarity sequences, as defined in Formula (7.4). Such a pattern can take over the cases Version Space cannot handle. As for Example (7.3), the disjunctive expression of the generalized pattern can be written as $(0,1,\times,\times,\times) \cup (0,0,1,1,\times)$, where $\times$ represents $0|1$. This pattern can cover all positive links and filter all negative links. This solution is called Extended Version Space. Note that neither the specialized pattern nor the generalized pattern of Extended Version Space is a set of interlinking patterns, each one maintains only one specialized pattern and one generalized pattern whenever a new assessed link is learned. The reason is that when the disjunction operator is applied, we can always find out the most strict/general pattern whenever an assessed link is learned. The pseudo-code of the Extended Version Space method is shown in Algorithm 3 and the description appears thereafter.

Algorithm 3 aims at generating a disjunctive pattern in the form of Formula (7.4). It separately processes assessed links one by one. The whole algorithm is split into two parts, based on whether the checked sample link is assessed as positive or not. The details are described below. Note that whatever the given assessed link is (either line 5-9 or line 10-19), the first step always checks whether it is included by current patterns. The pattern Pat_{Ori} is the generalized pattern of Disjunctive Version Space whose binary similarity sequences are induced into a concise format. It is not the generalized pattern of Extended Version Space. It is used to evaluate the F-measure and running time of merged Disjunctive Version Space in the experiments that are described in Chapter 8.

- If the current link is assessed as a **positive** one, the algorithm will check if it is covered by the current specialized pattern (Pat_S). If yes, the specialized pattern and the current generalized pattern (Pat_G) will stay unchanged. If not, we will merge the link represented in the form of binary similarity sequence (denoted as BSS_{link} in the following text) into Pat_S such that the newly added link can be included by Pat_S. In the meantime, Pat_S will be reconstructed

```
Input: Binary Similarity Sequences of All Assessed Links
Output: The Generalized Pattern $Pat_G$
$Pat_S$=$\varnothing$; /*empty set*/
$Pat_G$=$U$=$(\varphi^{(i)})$, where $\varphi^{(i)}$=$\times$, $i = 1, 2, \ldots, l$; /*universal set*/
$Pat_{Ori}$=$U$=$(\varphi^{(i)})$, where $\varphi^{(i)}$=$\times$, $i = 1, 2, \ldots, l$;
for (each assessed link (denoted as $BSS_{link}$)) do
  if ($BSS_{link}$ is the binary similarity sequence of a positive link) then
    if $BSS_{link} \notin Pat_S$ then
      $Pat_S$=MERGE($Pat_S$, $BSS_{link}$);
    if $BSS_{link} \notin Pat_G$ then
      $Pat_G$=MERGE($Pat_G$, $BSS_{link}$);
  if ($BSS_{link}$ is the binary similarity sequence of a negative link) then
    if $BSS_{link} \in Pat_{Ori}$ then
      if ($\exists$ $BSS$, $BSS_{link} \subseteq BSS$, $BSS \in Pat_{Ori}$) then
        $Pat_{Ori} - BSS$;
        for ($BSS_{new} \in$ COM($BSS$, $BSS_{link}$)) do
          $Pat_{Ori}$=MERGE($Pat_{Ori}$, $BSS_{new}$);
    if ($Pat_S \subseteq Pat_{Ori}$) then
      $Pat_G$ = RESELECT($Pat_{Ori}$, $Pat_S$);
    else
      $Pat_G$ = $\emptyset$.
```

Algorithm 3: Building Pattern via Extended Version Space

if the newly added link can induce a more concise expression, for example, replacing (0,1,1,1,0) and (1,1,1,1,0) by ($\times$,1,1,1,0). Note that we should also revise Pat_G to make sure $Pat_S \subseteq Pat_G$. So we should also merge BSS_{link} into Pat_G, if it cannot be included by Pat_G. Finally, we reconstruct Pat_G to be more concise as the way that Pat_S is done.

- If the current link is assessed as a **negative** one, we need to check if it can be included by Pat_{Ori}. If not, Pat_S and Pat_{Ori} stay the same. We delete BSS_{link} from Pat_{Ori}, if the negative link can be included by Pat_{Ori}. The delete operation is fulfilled by picking out the complement of BSS_{link}, which is a set of all possible binary similarity sequences that have at least one bit difference with BSS_{link} and are contained in Pat_{Ori}. It is not a binary similarity sequence that is created by negating each bit of BSS_{link}. The complement operation is executed below. 1) pick out BSS, which is one binary similarity sequence in Pat_{Ori} that contains BSS_{link}. 2) delete BSS from Pat_{Ori}. $-$ in the line 13 is the *difference* operation on sets that removes BSS from the set of binary similarity sequences that are represented by Pat_{Ori}. 3) the complement of BSS_{link} is computed with regard to BSS. It is a set of all binary similarity sequences in BSS except BSS_{link}. Afterwards, we can merge the complement of BSS_{link} into Pat_{Ori} such that BSS_{link} can be excluded by the new Pat_{Ori} without losing other binary similarity sequences. Then, Pat_{Ori} should be reconstructed further into a more concise expression (if possible). Finally, a RESELECT step is executed to remove from Pat_{Ori} the binary similarity sequences that cannot cover any binary similarity sequence in Pat_S, otherwise the number of binary similarity sequences in Pat_G would increase exponentially when more negative links are learned. Silk will perform quite slowly if too many binary similarity sequences are transformed in the script.

In this algorithm, we assume that there is no intersection between all positive links and all negative links. It means that users never mark wrongly. Therefore, all positive links are included by Pat_{Ori}, no matter whether they are included by Pat_S. Similarly, all negative links are not included by Pat_S, no matter whether they are included by Pat_{Ori}. Therefore, when a positive link is not included by Pat_S, we do not need to merge the link into Pat_{Ori}. Similarly, when a negative link is included by Pat_{Ori}, we do not need to merge its complement into Pat_S.

As follows, a discussion of why Pat_S is not used as the final pattern is given. Note that if Pat_S is used in the Silk script, the link set produced by the Silk script is only the links that have the same binary similarity sequences with the ones of positive links that have been learned. However, if Pat_G is used for generating links, other correct links that do not have the same binary similarity sequences with the ones of positive links that have been learned are also generated. Since the operation RESELECT picks out binary similarity sequences in Pat_{Ori} that contains any binary similarity sequences in Pat_S, the correct links that have the same binary similarity sequences with the learned links can also be covered by Pat_G of Extended Version Space. Therefore, the generated link set of Pat_G contains more correct links than the one of Pat_S.

The key operations appearing in Algorithm 3 are described below.

7.3.1 Complement operation

The Complement operation (denoted as Com(BSS,BSS_{link})) aims to generate a set of binary similarity sequences that are contained in BSS, each of which are in conflict with the negative link's binary similarity sequence BSS_{link}. It can be formally defined as follows, where BSS_{link} is referred to as the given negative link, BSS is referred to as the binary similarity sequence in the generalized pattern Pat_{Ori} that contains the negative link's binary similarity sequence BSS_{link}. BSS_{new} is the binary similarity sequence that is contained in BSS and conflict with the negative link's binary similarity sequence BSS_{link}. $\alpha^{(k)}$ represents the kth bit in the BSS_{new} $(k = 1, 2, \cdots, l)$. $\beta^{(k)}$ represents the kth bit in the BSS_{link} $(k = 1, 2, \cdots, l)$. $\delta^{(k)}$ represents the kth bit in the BSS $(k = 1, 2, \cdots, l)$.

$$\textsc{Com}(BSS, BSS_{link}) = \\ \begin{cases} \textsc{Com}(BSS_{link}), & \textbf{if } Pat{=}U{=}(\varphi^{(i)}), \text{ where } \varphi^{(i)}{=}\times,\ i = 1, 2, \ldots, l \\ \varnothing, & \textbf{if } BSS{=}BSS_{link} \\ \bigcup(\gamma^{(k)}), & \textbf{if } BSS_{new}{\in}\textsc{Com}(BSS_{link}),\ BSS{\cap}BSS_{new}{\neq}\varnothing \\ & \quad \text{where } \gamma^{(k)} = \begin{cases} \delta^{(k)}, & \textbf{if } \delta^{(k)}{\neq}\times,\ \alpha^{(k)}{=}\times \\ \alpha^{(k)}, & \text{other conditions} \end{cases} \end{cases} \tag{7.5}$$

$$\begin{aligned} \textsc{Com}(BSS_{link}) = {} & \textsc{Com}(BSS_{link}, \beta^{(1)}) \cup \textsc{Com}(BSS_{link}, \beta^{(2)}) \cup \ldots \cup \\ & \textsc{Com}(BSS_{link}, \beta^{(l)}), \\ \textit{where } \textsc{Com}(BSS_{link}, \beta^{(k)}) & = (\beta^{(1)}, \ldots, \beta^{(k-1)}, (1 - \beta^{(k)}), \times, \ldots, \times) \end{aligned} \tag{7.6}$$

The Complement function (denoted as Com(BSS_{link})) is used to compute the complement of a negative link, so that it can be merged into Pat_{Ori} (as shown in Algorithm 3). For instance, as for the fourth negative link BSS_{link} = (0,0,1,0,0) in Example 7.3. Com(BSS_{link}) can be represented as follows.

$$\begin{aligned} \textsc{Com}(BSS_{link}) = {} & (1, \times, \times, \times, \times)\cup \\ & (0, 1, \times, \times, \times)\cup \\ & (0, 0, 0, \times, \times)\cup \\ & (0, 0, 1, 1, \times)\cup \\ & (0, 0, 1, 0, 1) \end{aligned} \tag{7.7}$$

Obviously, the Complement representation shown in Formula (7.7) complies with all the possible links that are conflicting to the given negative link (0,0,1,0,0) on at least one bit.

When learning the negative link BSS_{link} = (0,0,1,0,0) in Example (7.3), there is one binary similarity sequence in Pat_{Ori} that contains such a binary similarity sequence. It is BSS=(0,×,×,×,×). Thus, Com(BSS,BSS_{link}) is computed as below.

$$\begin{aligned}\textsc{Com}(BSS, BSS_{link}) = {} & (0,\ 1,\ \times,\ \times,\ \times)\cup \\ & (0,\ 0,\ 0,\ \times,\ \times)\cup \\ & (0,\ 0,\ 1,\ 1,\ \times)\cup \\ & (0,\ 0,\ 1,\ 0,\ 1)\end{aligned} \tag{7.8}$$

The Complement representation shown in Formula (7.8) complies with all the possible links that are contained in Pat_{Ori} and conflicting to the given negative link's binary similarity sequence (0,0,1,0,0) on at least one bit. Such an expression can be merged into Pat_{Ori}, so that Pat_{Ori} can filter the fourth negative link. The merge operation is introduced in the next subsection.

7.3.2 Merge operation

The Merge operation (denoted by Merge(Pat, BSS_{link})) is used to check if the newly added BSS_{link} is supposed to be inserted and merged with some binary similarity sequences in the pattern (denoted by BSS^i_{Pat}), such that the whole pattern can be represented in a more succinct way. In order to formally define this operation, two other operations should be introduced.

- The first one is denoted by BSSMerge(BSS^i_{Pat}, BSS^j_{Pat}), with respect to two binary similarity sequences, such as BSS^i_{Pat} and BSS^j_{Pat}. Its formal definition is shown below.

$$\begin{aligned}\textsc{BSSMerge}(BSS^i_{Pat}, BSS^j_{Pat}) &= (b_1, b_2, \cdots, b_l) \\ \textit{where } b_k &= \textsc{BitMerge}(\beta_i^{(k)}, \beta_j^{(k)}) \\ &= \begin{cases} \beta_i^{(k)}, & \textbf{if } \beta_i^{(k)} = \beta_j^{(k)} \\ \times, & \textbf{if } \beta_i^{(k)} \neq \beta_j^{(k)} \end{cases}\end{aligned} \tag{7.9}$$

  where $\beta_i^{(k)}$ denotes the kth bit of the pattern BSS^i_{Pat}. BitMerge in the above formula is a critical operation. Two examples are given to illustrate it here. (1) If BSS^i_{Pat} = (×,1,1,1) and BSS_{link} = (0,1,1,1), then BitMerge(BSS^i_{Pat}, BSS_{link}) = (×,1,1,1). (2) If BSS^i_{Pat} = (0,×,1,1) and BSS^j_{Pat} = (1,×,1,1), then BitMerge(BSS^i_{Pat}, BSS^j_{Pat}) = (×,×,1,1). Obviously, the core of BitMerge is to construct a concise pattern which is compatible with the given binary similarity sequences.

- The second one is called Diff operation, also with respect to two binary similarity sequences, such as BSS^i_{Pat} and BSS^j_{Pat}. Diff(BSS^i_{Pat},BSS^j_{Pat}) is a function to compute the number of different bits between the two binary similarity sequences. For example, Diff((0,1,1,1),(0,1,0,1)) = 1 and Diff((0,0,×,1),(0,1,0,0)) = 3.

Based on the definitions of BSSMerge and Diff, Merge(Pat, BSS_{link}) can be formally represented as a recursive function below. Some examples are given to illustrate it later on.

$$\textsc{Merge}(Pat, BSS_{link}) = \begin{cases} Pat, \ \ \textbf{if} \ \exists BSS^i_{Pat} \in Pat, \ \textsc{Diff}(BSS^i_{Pat}, BSS_{link}) = 0 \\ Pat \cup BSS_{link}, \\ \qquad \textbf{if} \ \forall BSS^i_{Pat} \in Pat, \ \textsc{Diff}(BSS^i_{Pat}, BSS_{link}) \geq 2 \\ \textsc{Merge}(Pat - BSS^i_{Pat}, \mathrm{BSS}\textsc{Merge}(BSS^i_{Pat}, BSS_{link})), \\ \qquad \textbf{if} \ \exists BSS^i_{Pat} \in Pat, \ \textsc{Diff}(BSS^i_{Pat}, BSS_{link}) = 1 \\ Pat \cup BSS_{link}, \ \ \textbf{if} \ Pat = \varnothing \end{cases} \tag{7.10}$$

In Formula (7.10), the notation "−" means the *difference* operation (a.k.a., taking away) on sets. In the above formula, if there exists a binary similarity sequence in the pattern which is equal to the link's binary similarity sequence, then the pattern stays unchanged. This means that the link is covered by the pattern. If all binary similarity sequences in the pattern that have more than one different bit with the link's binary similarity sequence, then the link should be inserted into the pattern. If there exists one binary similarity sequence in the pattern that has one different bit with the link's binary similarity sequence, then the link should be merged with such a binary similarity sequence. The newly produced binary similarity sequence should be further merged with other binary similarity sequences in the pattern if possible. If the pattern is an empty set, the link should be inserted into the pattern directly. The computation is recursively performed until there are no binary similarity sequences to merge. Finally, the new produced binary similarity sequence is inserted into the pattern.

Here, several examples are given to illustrate the computation of $\textsc{Merge}(Pat, BSS_{link})$, supposing the current pattern Pat = (0,1,1,×,×)∪(1,0,1,0,0).

- If BSS_{link} = (1,0,1,0,0), since there exists one binary similarity sequence (1,0,1,0,0) in the current pattern such that $\textsc{Diff}((1,0,1,0,0), BSS_{link})=0$, the returned new pattern stays unchanged (i.e., the first situation in Formula (7.10)).

- If BSS_{link} = (1,1,1,0,1), since $\textsc{Diff}$ value is always no less than 2 for any binary similarity sequence in the current pattern, the new pattern should be represented as $Pat \cup BSS_{link}$ = (0,1,1,×,×)∪(1,0,1,0,0)∪(1,1,1,0,1).

- If BSS_{link} = (1,1,1,0,0), since $\textsc{Diff}((1,0,1,0,0), BSS_{link})$ = $\textsc{Diff}$((1,0,1,0,0),(1,1,1,0,0)) = 1, the new pattern should be (0,1,1,×,×)∪(1,×,1,0,0).

- If BSS_{link} = (0,1,1,1,×), it also belongs to the third situation because $\textsc{Diff}$((0,1,1,×,×),(0,1,1,1,×))=1, then the new pattern can be derived as:

$$\begin{aligned} \textsc{Merge}(Pat, BSS_{link}) &= \textsc{Merge}(Pat - (0,1,1,\times,\times), (0,1,1,\times,\times)) \\ &= Pat \\ &= (0,1,1,\times,\times) \cup (1,0,1,0,0). \end{aligned}$$

7.3.3 Reselect operation

The Reselect operation selects from Pat_{Ori} the binary similarity sequences that cover any binary similarity sequence in Pat_S. It can be formally defined as follows, where $BSS_{Pat_{Ori}}$ and BSS_{Pat_S} are referred to as the binary similarity sequence in Pat_{Ori} and the binary similarity sequence in Pat_S respectively.

$$\begin{aligned}&\textsc{Reselect}(Pat_{Ori}, Pat_S) =\\&\{BSS_{Pat_{Ori}} \mid \exists\ BSS_{Pat_S}, BSS_{Pat_S} \subseteq BSS_{Pat_{Ori}}\} \cup\\&\{BSS_{Pat_S} \mid \nexists\ BSS_{Pat_{Ori}}, BSS_{Pat_S} \subseteq BSS_{Pat_{Ori}}\}\end{aligned} \tag{7.11}$$

The Reselect operation is used to reduce the size of Pat_G: the binary similarity sequence in Pat_{Ori} can be maintained if and only if it covers at least one binary similarity sequence belonging to Pat_S. Although the existence of other irrelevant binary similarity sequences in Pat_{Ori} does not break the relationship $Pat_S \subseteq Pat_{Ori}$, too many irrelevant binary similarity sequences will degrade the Silk interlinking speed. Hence, it is necessary to intensively pick the relevant binary similarity sequences in Pat_{Ori} and filter out less informative ones meanwhile. Besides, the binary similarity sequences in Pat_S that are not subsumed by any binary similarity sequence in Pat_{Ori} should also be added into Pat_G, in order to keep the relationship $Pat_S \subseteq Pat_G$. This operation is the key difference between Extended Version Space and Disjunctive Version Space, which makes the interlinking pattern be represented in a concise format.

In the end of Algorithm 3, the converged generalized pattern will act as the final output. The optimality and correctness of Algorithm 3 is proved as follows.

Theorem 1. *The output pattern of Algorithm 3 covers* ***all*** *positive links and excludes* ***all*** *negative links, with a concise representation format.*

Proof. This proof shows the soundness of the Extended Version Space algorithm. In the proof, we first prove that the output pattern Pat_G of Algorithm 3 is a complete pattern which covers **all** positive links and excludes **all** negative links, and then prove its representation is in a concise format.

To prove that the output Pat_G ***covers all positive links***: On one hand, the algorithm traverses all positive links (line 5 in Algorithm 3). On the other hand, the four separate situations in the Merge function (i.e., Formula (7.10)) covers all cases of the pattern's representation. Note that the operation in each of the four cases always makes sure $BSS_{link} \in Pat$, hence, the returned Pat_G must include the positive link.

To prove that the output Pat_G ***excludes all negative links***: We just need to prove that the lines 14-15 in Algorithm 3 must be able to exclude the negative link BSS_{link}. This can be derived from the definition of Complement, i.e., Formula (7.5) and Formula (7.6). A good example is illustrated in Formula (7.7) and Formula (7.8) which is easy to understand.

To prove that the output Pat_G ***is in a concise representation***: The third situation in MERGE function (Formula (7.10)) will merge the similar binary similarity sequences that only have one bit difference into one binary similarity sequence. Therefore, the size of each pattern Pat_{EVS} (i.e., Pat_G in Algorithm 3) in Extended Version Space must be equal or smaller than the size of each pattern Pat_{DVS} in Disjunctive Version Space. It means that $|Pat_{EVS}| \leq |Pat_{DVS}|$, where $|Pat|$ is the number of binary similarity sequences in the pattern Pat. As for Example (7.3), the interlinking pattern (i.e., Pat_G) of Extended Version Space is $(0,1,\times,\times,\times) \cup (0,0,1,1,\times)$. $|Pat_{EVS}|$ is 2. In contrast, the interlinking pattern of Disjunctive Version Space is $(0,0,0,0,0) \cup (0,0,0,0,1) \cup (0,0,0,1,0) \cup (0,0,0,1,1) \cup (0,0,1,0,1) \cup (0,0,1,1,0) \cup (0,0,1,1,1) \cup (0,1,0,0,0) \cup (0,1,0,0,1) \cup (0,1,0,1,0) \cup (0,1,0,1,1) \cup (0,1,1,0,0) \cup (0,1,1,0,1) \cup (0,1,1,1,0) \cup (0,1,1,1,1) \cup (1,0,0,0,0) \cup (1,0,0,0,1) \cup (1,0,0,1,0) \cup (1,0,0,1,1) \cup (1,0,1,0,1) \cup (1,0,1,1,0) \cup (1,0,1,1,1) \cup (1,1,0,0,0) \cup (1,1,0,0,1) \cup (1,1,0,1,0) \cup (1,1,0,1,1) \cup (1,1,1,0,0) \cup (1,1,1,0,1) \cup (1,1,1,1,0) \cup (1,1,1,1,1)$. $|Pat_{DVS}|$ is 30. Obviously, $|Pat_{EVS}| < |Pat_{DVS}|$. □

The above proof proves the soundness of the algorithm. We did not prove whether if there exist a pattern, the algorithm will find it (completeness). We did not encounter a case when doing interlinking experiments that are described in Chapter 8, in which the algorithm does not find a pattern while there exist one.

The time complexity of Algorithm 3 is analyzed as follows. Assume that there are n_P positive links and n_N negative links assessed. The complexity of MERGE(Pat_S, BSS_{link}) can be derived as $O(m_S^2)$, where m_S denotes the maximum number of binary similarity sequences in the specialized pattern Pat_S. The derivation of $O(m_S^2)$ is given as follows. When a new correct link is learned, this link has to be compared with each binary similarity sequence in the Pat_S to determine which of the four situations it belongs to in Formula (7.10). The number of comparisons is m_S. If the newly assessed link is merged with one binary similarity sequence of Pat_S, then the newly generated binary similarity sequence has to be compared with each of the rest binary similarity sequences in Pat_S, and the number of comparisons is m_S-1. The merging procedure continues until there is no newly generated binary similarity sequence or there is only one binary similarity sequence left in Pat_S. So, the time complexity of MERGE(Pat_S, BSS_{link}) is equal to the maximum total number of comparisons $\frac{m_S \cdot (m_S+1)}{2}$, i.e., $O(m_S^2)$. Similarly, the complexity of MERGE(Pat_G, BSS_{link}) is equal to $O(m_G^2)$, where m_G denotes the maximum number of binary similarity sequences in the generalized pattern Pat_S.

Let us compute the values of m_S and m_G as follows. Since each binary similarity sequence has l bits and any two binary similarity sequences (e.g., BSS_i and BSS_j) with $\text{DIFF}(BSS_i, BSS_j) = 1$ has to be merged as one binary similarity sequence in Pat_S, there must be at most 2^{l-1} binary similarity sequences in each Pattern Pat_S. That is, $m_S = 2^{l-1}$. Here are two examples to illustrate m_S. If $l=2$, the most complicated pattern is $(0,1) \cup (1,0)$, that is, $m_S = 2^{2-1} = 2$; if $l=3$, the most complicated pattern is $(0,0,1) \cup (0,1,0) \cup (1,1,1) \cup (1,0,0)$, that is $m_S = 2^{3-1} = 4$. Similarly, m_G is also 2^{l-1}.

On the other hand, the complexity of Reselect operation is the maximum number of binary similarity sequences in the pattern Pat_{Ori}, which is denoted as m_{Ori}, and the maximum number of binary similarity sequences in the pattern Pat_S. Similarly, m_{Ori} is also 2^{l-1}. The complexity of Complement operation is the number of potential attribute correspondences (i.e., l). Then, the total complexity of Algorithm 3 is equal to $O(n_P{\cdot}(m_S^2{+}m_G^2){+}n_N{\cdot}(l^2{\cdot}m_{Ori}^2{+}m_{Ori}{+}m_S))$. So the computational complexity of Extended Version Space is $O(2^{2l})$.

It is obvious that the number of potential attribute correspondences that are generated by K-medoids is a key factor to impact the computational complexity of Extended Version Space. That is, the effectiveness of K-medoids clustering is supposed to be helpful to reduce the computational complexity of Extended Version Space.

In comparison to the Extended Version Space, the functions Merge and Reselect are not required in Disjunctive Version Space. The computational complexity of Disjunctive Version Space is $O(n_P{+}n_N{\cdot}l^2)$. So, the computation of Extended Version Space should be more expensive than that of Disjunctive Version Space.

7.4 Link Generation with Extended Version Space

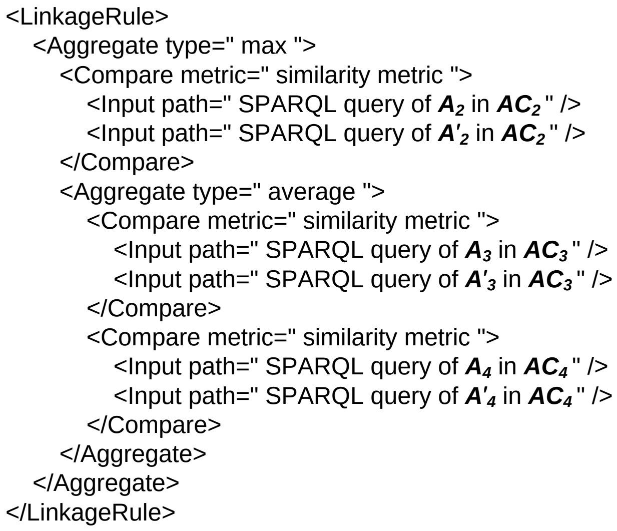
In the implementation, each interlinking pattern (Formula (7.4)) is represented in the form of Silk script and executed by Silk. The potential attribute correspondences, that are discovered by K-medoids in Step 2 and 3 shown in Figure 4.1, will be organized and put into the interlinking pattern in the following way: 1) Attribute correspondences in each binary similarity sequence, whose similarity is 1, will be aggregated by the aggregation method "average" to generate links that have the same set of attribute correspondences with the learned binary similarity sequence. If there is only one attribute correspondence in a binary similarity sequence, there is no need to aggregate the attribute correspondence with such an aggregation method. 2) all binary similarity sequences in the interlinking pattern will be aggregated by the aggregation method "max". It means that two compared instances that form any binary similarity sequence in the interlinking pattern will be treated as a link.

After generating the interlinking pattern, we can convert the interlinking pattern into a Silk script with the transformation rules in Chapter 6. As for Example (7.3), the linkage rule segment of the Silk script can be expressed in Figure 7.2.

In Figure 7.2, AC_2 is a correspondence between the attribute A_2 and the attribute A'_2. AC_3 is a correspondence between the attribute A_3 and the attribute A'_3. AC_4 is a correspondence between the attribute A_4 and the attribute A'_4. The Silk script means that two compared instances can form a link only if the two instances satisfy one of two conditions. One condition is that attribute values of A_2 and A'_2 are similar. The other condition is that not only attribute values of A_3 and A'_3 are similar, but also attribute values of A_4 and A'_4 are similar.

After learning with Extended Version Space, the interlinking pattern for the example in Section 2.1 of Chapter 2 is below

```
<?xml version="1.0" encoding="utf-8" ?>
```



```
<LinkageRule>
  <Aggregate type=" max ">
    <Compare metric=" similarity metric ">
      <Input path=" SPARQL query of A2 in AC2 " />
      <Input path=" SPARQL query of A'2 in AC2 " />
    </Compare>
    <Aggregate type=" average ">
      <Compare metric=" similarity metric ">
        <Input path=" SPARQL query of A3 in AC3 " />
        <Input path=" SPARQL query of A'3 in AC3 " />
      </Compare>
      <Compare metric=" similarity metric ">
        <Input path=" SPARQL query of A4 in AC4 " />
        <Input path=" SPARQL query of A'4 in AC4 " />
      </Compare>
    </Aggregate>
  </Aggregate>
</LinkageRule>
```

Figure 7.2: Linkage Rule Segment in Silk Script of Example (7.3)

```
<Silk>
    <Prefixes>
        <Prefix id="rdf" namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
        <Prefix id="owl" namespace="http://www.w3.org/2002/07/owl#" />
        <Prefix id="id2" namespace=
        "http://ec.europa.eu/eurostat/ramon/ontologies/geographic.rdf#" />
        <Prefix id="id1" namespace="http://rdf.insee.fr/geo/" />
    </Prefixes>

    <DataSources>
        <DataSource id="id1" type="file">
            <Param name="file" value=
            "C:/Zhengjie/study/eclipse3/SILKscript2/regions-2010.rdf"/>
            <Param name="format" value="RDF/XML" />
        </DataSource>

        <DataSource id="id2" type="file">
            <Param name="file" value=
            "C:/Zhengjie/study/eclipse3/SILKscript2/nuts2008_complete.rdf"/>
            <Param name="format" value="RDF/XML" />
        </DataSource>
    </DataSources>

    <Interlinks>
        <Interlink id="no1">
            <LinkType>owl:sameAs</LinkType>
            <SourceDataset dataSource="id1" var="e1">
                <RestrictTo>
                ?e1 rdf:type id1:Departement .
                </RestrictTo>
            </SourceDataset>
            <TargetDataset dataSource="id2" var="e2">
```

```
                <RestrictTo>
                ?e2 rdf:type id2:NUTSRegion . ?e2 id2:level ?ov1 . FILTER (?ov1=3) .
                </RestrictTo>
            </TargetDataset>
            <LinkageRule>
                <Aggregate type="max">
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:name" />
                        </TransformInput>
                    </Compare>
                </Aggregate>
            </LinkageRule>

            <Filter />

            <Outputs>
                <Output type="file">
                    <Param name="file" value=
                    "C:/Zhengjie/study/eclipse3/trunk-test/accepted_links.xml"/>
                    <Param name="format" value="alignment"/>
                </Output>
            </Outputs>
        </Interlink>

    </Interlinks>
</Silk>
```

According to the Silk script above, there is only one binary similarity sequence in the generalized pattern of Extended Version Space in this example. Actually, it is the concise expression for the generalized pattern of Disjunctive Version Space. The linkage rule segment in the Silk script of Disjunctive Version Space is below.

```
                <Aggregate type="max">
                    <Aggregate type="average">
                        <Compare metric="jaccard">
                            <TransformInput function="tokenize">
                                <Input path="?e1/id1:nom" />
                            </TransformInput>
                            <TransformInput function="tokenize">
                                <Input path="?e2/id2:name" />
                            </TransformInput>
                        </Compare>
                        <Compare metric="jaccard">
                            <TransformInput function="tokenize">
                                <Input path="?e1\ id1:subdivision/id1:nom" />
                            </TransformInput>
                            <TransformInput function="tokenize">
                                <Input path="?e2/id2:hasParentRegion/id2:name" />
                            </TransformInput>
                        </Compare>
                    </Aggregate>
                    <Compare metric="jaccard">
                        <TransformInput function="tokenize">
                            <Input path="?e1/id1:nom" />
                        </TransformInput>
                        <TransformInput function="tokenize">
                            <Input path="?e2/id2:name" />
                        </TransformInput>
                    </Compare>
                </Aggregate>
```

It means that each correct link should either have two attribute correspondences, *insee:nom*↔*eurostat:name* and *insee:subdivision*$^{-1}$*/insee:nom*↔*eurostat:hasParentRegion/eurostat:name*, or have one attribute correspondence *insee:nom*↔*eurostat:name.* Thus, no matter what, each correct link should have the attribute correspondence *insee:nom*↔*eurostat:name.*

7.5 Conclusion

This chapter introduces the Extended Version Space to learn the interlinking pattern that can cover all assessed correct links while excluding all assessed incorrect links. The Extended Version Space has the following feature:

- Concise expression of the interlinking pattern leads to short interlinking time The *MERGE* operation and the *RESELECT* operation of Extended Version Space make the interlinking pattern more concise than the one of Disjunctive Version Space, so that the interlinking time that is required by Silk become shorter. The reason is that each attribute correspondence in the interlinking pattern will be transformed into a query in the Silk script, which demands an I/O operation. The less attribute correspondences a Silk script have, the shorter the interlinking time is. If the interlinking pattern is expressed in a concise expression, there are fewer attribute correspondences being included in the Silk script. Consequently, the interlinking time of Silk is shorter.

The next chapter will show the experimental results of the clustering effect of K-medoids and comparisons between Extended Version Space, Disjunctive Version Space and other interlinking methods.

Chapter 8

Experiments

Contents

This chapter evaluates the interlinking method of this thesis. First, experimental setup is presented. Second, K-medoids is evaluated on its clustering effect. Third, Extended Version Space is evaluated by comparing the F-measure and running time of Extended Version Space with the ones of Disjunctive Version Space and other related works.

8.1 Experimental Setup

The interlinking method in this thesis is evaluated based on some well-known public data sets mainly downloaded from the IM@OAEI 2010[1] (data sets *Person1* and *Person2*), IM@OAEI 2012[2] (data sets *Sandbox*) and CKAN[3] (geographical data sets *INSEE* and *EUROSTAT*).

Table 8.1 shows the statistical information of each interlinking task. For each interlinking task in the table, two statistics are shown. One is the number of attributes in each class of a class correspondence, the other is the number of instances in each class of a class correspondence. For example, as for the interlinking task on the data sets *INSEE* and *EUROSTAT*, there is a class correspondence *insee:Departement*↔*eurostat:NUTSRegion*. The interlinking task of

[1] http://oaei.ontologymatching.org/2010/im/index.html

[2] http://www.instancematching.org/oaei/

[3] http://datahub.io/

Interlinking Task	Corresponding Classes	Number of Attributes	Number of Instances
INSEE and *EUROSTAT*	*insee:Departement*	4	100
	eurostat:NUTSRegion	12	2008
Person1	*Person1:Person*	8	500
	Person2:Person	8	500
Person2	*Person1:Person*	8	600
	Person2:Person	8	400
Sandbox001	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	24	367
Sandbox002	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	24	367
Sandbox003	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	24	367
Sandbox006	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	24	367
Sandbox010	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	25	367
Sandbox011	*owl:NamedIndividual*	24	363
	owl:NamedIndividual	23	367

Table 8.1: Statistics of Each Interlinking Task

the two data sets is finding out similar instances across two corresponding classes *insee:Departement* and *eurostat:NUTSRegion*. There are 4 attributes and 100 instances in the class *insee:Departement*. There are 12 attributes and 2008 instances in the class *eurostat:NUTSRegion*. When interlinking data sets *INSEE* and *EUROSTAT*, properties, attribute paths of the form relation/property and attribute paths of the form relation^{-1}/property are clustered and used for constructing the interlinking pattern. When interlinking other data sets, only properties are clustered and used for constructing the interlinking pattern. Because these properties can help gain a high level F-measure for the other interlinking tasks. In the table, prefix *person1* is *http://www.okkam.org/ontology_person1.owl#*. Prefix *person2* is *http://www.okkam.org/ontology_person2.owl#*. Prefix *owl* is *http://www.w3.org/2002/07/owl#*.

A set of sample links are created for constructing and improving the interlinking pattern. These sample links are generated by combining all discovered potential attribute correspondences into a disjunction of discovered potential attribute correspondences. These links are assessed and used to improve the interlinking pattern afterwards. Moreover, another set of links are generated by combining all discovered potential attribute correspondences into a conjunction of discovered potential attribute correspondences. Empirically, this set of links are almost all correct links throughout each interlinking task. These links are also used to construct and improve the interlinking pattern, but they do not need to be assessed. Furthermore, all links that are formed by the same instances of these correct links are deleted from the set of sample links, so as to reduce the number of links to be assessed by users. All binary similarity sequence groups that have the same binary similarity sequence with the correct links are also deleted.

There is a *reference link set* provided in each pair of interlinking data sets of IM@OAEI. For the interlinking task on data sets *INSEE* and *EUROSTAT*, reference links are created by running several hand-made Silk scripts.

According to the strategy of Active Learning, the interlinking pattern will be improved faster with unlearned binary similarity sequences. Thus in each learning round, different binary similarity sequence groups which have not been learned before will be selected. In each group, a random link is selected as a sample link to assess and improve the interlinking pattern. In practice, the sample link will be assessed by users. In the experiment, such a procedure is simulated as follows: the sample link will be assessed as positive if it exists in the reference link set, or negative otherwise.

The F-measure and running time are computed every 10 assessed links. If the interlinking procedure stops before learning 100 assessed links, we assume that the F-measure and running time keep at the same level with the ones when the procedure stops, in order to evaluate our interlinking method with other works when the learning process stops with more than 100 assessed links.

The interlinking data sets are stored in RDF files. The experiments are performed on each data set 5 times in 4 threads and allocated maximally 2GB of RAM. Since Active Learning is applied to select sample links to be assessed by users, there are different links that are used to improve the interlinking pattern for different experiments. Thus, the Precisions and Recalls of generated link sets and running time of each interlinking task to be shown are the average values under 5 experiments. F-measures of generated link sets are computed with regard to the average precisions and average recalls of each interlinking task.

All algorithms are implemented in Java 1.6+, and the experiments are performed on a desktop computer (2.5GHz 8-core CPU, memory size=32GB, 64-bit operating system). The experimental results are stored in RDF files.

The interlinking method is evaluated in two phases. One is the evaluation of the K-medoids clustering effect. The other is the evaluation on Extended Version Space. The chapter compares the F-measures and running time of Extended Version Space with two well-known related works. One is the Disjunctive Version Space algorithm [Mitchell 1997]. The other one is the approach proposed by Ngonga Ngomo et al. [Ngonga Ngomo 2013, Ngonga Ngomo 2012b] which exhibits the best results before our work. As shown in [Ngonga Ngomo 2013], its generated pattern exhibits prominently high F-measures.

	Ratio of Discovered Potential Attribute Correspondences on Attribute Pairs	Ratio of Attribute Correspondences on Attribute Pairs
INSEE and *EUROSTAT*	0.25	0.06
Person1	0.37	0.16
Person2	0.37	0.16
Sandbox001	0.49	0.08
Sandbox002	0.49	0.08
Sandbox003	0.49	0.08
Sandbox006	0.49	0.08
Sandbox010	0.49	0.08
Sandbox011	0.49	0.08

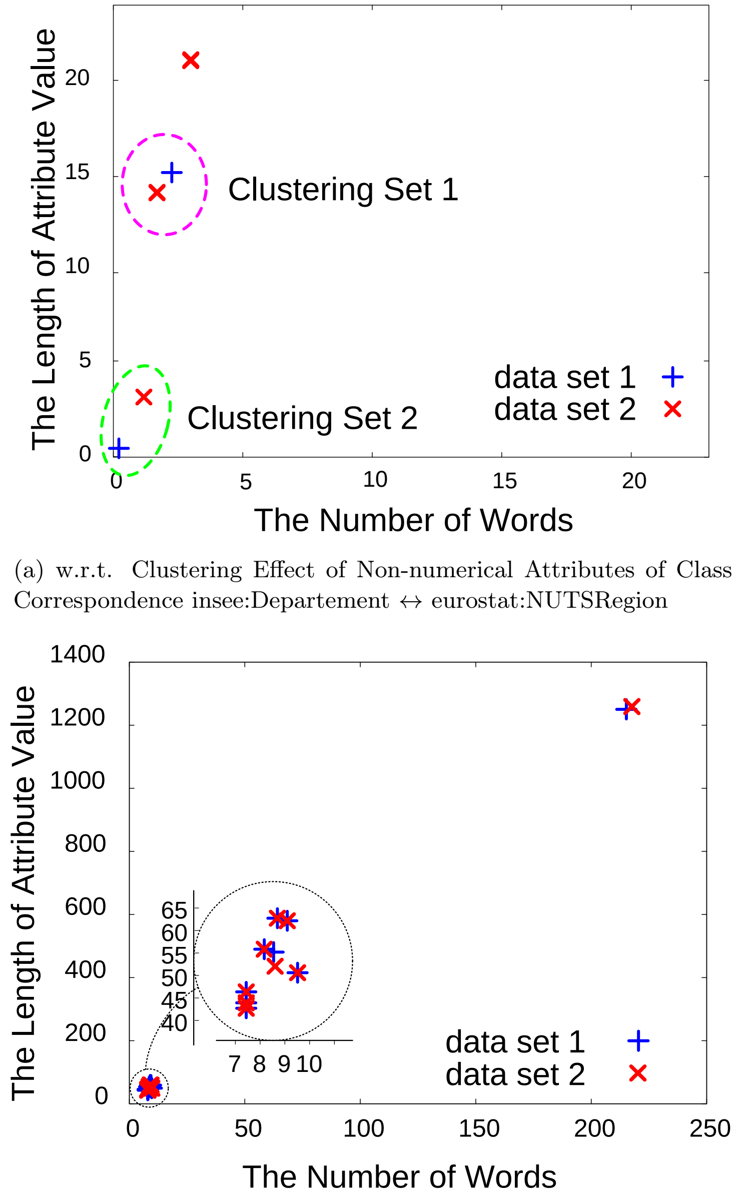
Table 8.2: Two Ratios of Attribute Correspondences

8.2 Evaluation on K-medoids Clustering

In this section, the clustering effect is investigated by using the K-medoids clustering method. For the simplicity of presentation, Figure 8.1 presents the clustering effect of two statistical features (*average number of words* and *average length of attribute values*) from among the totally four statistical features, with respect to the non-numerical attributes of the class correspondence *insee:Departement↔eurostat:NUTSRegion* for the two data sets *INSEE* and *EUROSTAT* and the class correspondence *owl:NamedIndividual↔owl:NamedIndividual* for the two data sets *Sandbox* and *Sandbox 001.* With these two statistical features, a clear classification of attributes over data sets is already observed. Each coordinate in the figures represents the center of a clustered group in a class. In Figure 8.1 (a), there are two sets of matched centers, whose centers are located around (1.00, 2.00) and (2.50, 15.00) respectively. In Figure 8.1 (b), there are two sets of matched groups, whose centers are located around (8.50, 52.00) and (220.00, 1250.00) respectively. As a consequence, it is easy to build potential attribute correspondences across the matched clustered groups from two corresponding classes for each interlinking task.

Table 8.2 shows two ratios that are related to the potential attribute correspondences that are discovered by the K-medoids clustering step. The ratio of discovered potential attribute correspondences on attribute pairs is the ratio of the number of potential attribute correspondences that are discovered by the K-medoids clustering step on the number of attribute pairs across two corresponding classes. The ratio of attribute correspondences on attribute pairs is the ratio of the number of attribute correspondences that exist in the correct links on the number of attribute pairs across two corresponding classes.

The two ratios are formulated as below. Assume that we are going to find



(a) w.r.t. Clustering Effect of Non-numerical Attributes of Class Correspondence insee:Departement ↔ eurostat:NUTSRegion

(b) w.r.t. Clustering Effect of Non-numerical Attributes of Class Correspondence owl:NamedIndividual ↔ owl:NamedIndividual in Sandbox001

Figure 8.1: K-medoids Clustering Effect on Different Data Sets

similar instances of two corresponding classes across two RDF data sets. There are n attributes of one corresponding class, which are clustered by K-medoids. There are n' attributes of the other corresponding class, which are also clustered by K-medoids. So the number of attribute pairs is $n \times n'$. There is a set A, each element is a discovered potential attribute correspondence. Thus, the ratio of discovered potential attribute correspondences on attribute pairs is $\frac{|A|}{n \times n'}$. Assume that there is a set L of correct links across the two corresponding classes. For each link l in the set L, there is a similarity tuple $(B^{(i)})$, $i = 1, \ldots, m$, where m is the number of potential attribute correspondences that are discovered by the K-medoids clustering step. Each item $B^{(i)}$ of the similarity tuple is a similarity value of two attribute values across two instances that form the link l. If the similarity value is larger than a predefined threshold T, we assume the two attributes form an attribute correspondence in the link l. We state that the link l has such an attribute correspondence, or such an attribute correspondence is contained in the link l. Here, we set $T = 0.5$, in that according to the analysis of Section 8.3.2, the interlinking running time of the threshold 0.5 is shorter than the one of other thresholds. Moreover, the F-measure of the threshold 0.5 is more stable and higher than the ones of other thresholds. Thus, the number of attribute correspondences across two corresponding classes is the size of a set C of attribute correspondences that exist in any correct link. Consequently, the ratio of attribute correspondences on attribute pairs is $\frac{|C|}{n \times n'}$.

From the table, we observe that many attribute pairs that are not corresponding are not generated by the K-medoids clustering step. We also observe that there are still some attribute pairs that are not corresonding being recognized as potential attribute correspondences by the K-medoids clustering step. According to the first column of the table, all ratios of discovered potential attribute correspondences on attribute pairs are below 0.5. It means that the K-medoids clustering step can reduce at least 50% attribute pairs that cannot help compare instances and produce links for each interlinking task. The computational complexity of Extended Version Space is largely reduced, because the computational complexity of Extended Version Space is influenced by the number of potential attribute correspondences that are discovered by the K-medoids clustering step. As for the second column of the table, each ratio of attribute correspondences on attribute pairs are smaller than the ratio of discovered potential attribute correspondences on attribute pairs in the same row. It means that some discovered potential attribute correspondences are not attribute correspondences. It also means that there are still a lot of attribute pairs which are considered to be attribute correspondences by the K-medoids clustering step. They will increase the computational complexity of Extended Version Space.

To conclude, K-medoids can effectively cluster attributes so that attribute correspondences of each interlinking task are discovered. The K-medoids clustering step largely reduces the number of attribute pairs that are not corresponding for comparing instances, but there are still some attribute pairs that are not corresponding are considered to be attribute correspondences.

Attribute Correspondence	Coverage Probability
insee:nom↔eurostat:name	1.00
insee:subdivision^{-1}/insee:nom↔ eurostat:hasParentRegion/eurostat:name	0.90

Table 8.3: Coverage Probabilities of Attribute Correspondences in *INSEE* and *EUROSTAT*

8.3 Evaluation on Extended Version Space

This section presents three evaluations. First, it presents the necessity of applying a disjunctive pattern for most of interlinking tasks. Second, it evaluates the F-measure and running time of Extended Version Space when the binary similarity sequences are built with different thresholds. Third, it compares the F-measure and running time of Extended Version Space with the ones of Disjunctive Version Space. Fourth, it compares the F-measure and running time of Extended Version Space with the ones of other related works.

8.3.1 Coverage Probability of Attribute Correspondence

This section illustrates the necessity to express the interlinking pattern into a disjunctive pattern (i.e., disjunction of conjunctions of attribute correspondences) rather than a conjunctive pattern (i.e., conjunction of attribute correspondences) for most of interlinking tasks. We compute the coverage probability of each attribute correspondence in each interlinking task, which is a ratio of the number of correct links that contain the attribute correspondence on the number of all correct links in the interlinking task. If there is no attribute correspondence whose coverage probability is 100%, a disjunctive pattern is needed. In other words, there is no attribute correspondence that exists in all correct links. Therefore, a conjunctive pattern cannot be used for distinguishing correct links and incorrect links. In the following, we will show the coverage probabilities of attribute correspondences in each interlinking task.

The interlinking task *INSEE* and *EUROSTAT* does not need a disjunctive pattern. In this interlinking task, there are two attribute correspondences which are contained in the correct links. They are, *insee:nom↔eurostat:name* and *insee:subdivision*$^{-1}$*/insee:nom↔eurostat:hasParentRegion/eurostat:name*. According to Table 8.3, the coverage probability of *insee:nom↔eurostat:name* is 100%. The coverage probability of *insee:subdivision*$^{-1}$*/insee:nom↔eurostat:hasParentRegion/eurostat:name* is 90%. Since there is an attribute correspondence whose coverage probability is 100%, a disjunctive pattern is not needed for this interlinking task.

The interlinking task *Person1* does not need a disjunctive pattern. In contrast, the interlinking task *Person2* needs a disjunctive pattern. The coverage probabilities in the interlinking tasks *Person1* and *Person2* are shown in

Attribute Correspondence	AC_1	AC_2	AC_3	AC_4	AC_5	AC_6
Coverage Probability in *Person1*	1.00	0.90	0.79	0.98	0.98	0.94
Coverage Probability in *Person2*	0.93	0.78	0.59	0.86	0.86	0.86

Table 8.4: Coverage Probabilities of Attribute Correspondences in *Person1* and *Person2*

Table 8.4. In the table, AC_1 stands for the attribute correspondence *person1:soc_sec_id↔person2:soc_sec_id*. AC_2 stands for the attribute correspondence *person1:date_of_birth↔person2:date_of_birth*. AC_3 stands for the attribute correspondence *person1:age↔person2:age*. AC_4 stands for the attribute correspondence *person1:given_name↔person2:given_name*. AC_5 stands for the attribute correspondence *person1:surname↔person2:surname*. AC_6 stands for the attribute correspondence *person1:phone_number↔person2:phone_number*. Prefix *person1* is *http://www.okkam.org/ontology_person1.owl#*. Prefix *person2* is *http://www.okkam.org/ontology_person2.owl#*. Since there is one attribute correspondence whose coverage probability is 100% in the interlinking task *Person1*, so a disjunctive pattern is not needed for such a interlinking task. Since there is no attribute correspondence whose coverage probability is 100% in the interlinking task *Person2*, a disjunctive pattern is needed for such a interlinking task.

The interlinking tasks of *Sandbox* all need a disjunctive pattern. The coverage probabilities in the interlinking tasks *Sandbox001*, *Sandbox002*, *Sandbox003*, *Sandbox006*, *Sandbox010*, and *Sandbox011* are shown in Table 8.5. AC_1 stands for the attribute correspondence *sandbox1:amount↔sandbox2:amount*. AC_2 stands for the attribute correspondence *sandbox1:size↔sandbox2:size*. AC_3 stands for the attribute correspondence *sandbox1:calling_code↔sandbox2:calling_code*. AC_4 stands for the attribute correspondence *sandbox1:article↔sandbox2:article*. AC_5 stands for the attribute correspondence *sandbox1:gender↔sandbox2:gender*. AC_6 stands for the attribute correspondence *sandbox1:name↔sandbox2:name*. AC_7 stands for the attribute correspondence *sandbox1:date_of_birth↔sandbox2:date_of_birth*. AC_8 stands for the attribute correspondence *sandbox1:form_of_government↔sandbox2:form_of_government*. AC_9 stands for the attribute correspondence *sandbox1:currency↔sandbox2:currecy*. AC_{10} stands for the attribute correspondence *sandbox1:religion↔sandbox2:religion*. AC_{11} stands for the attribute correspondence *sandbox1:iso_639_1_code↔sandbox2:iso_639_1_code*. Both prefix *sandbox1* and prefix *sandbox2* are *http://oaei.ontologymatching.org/2012/IIMBTBOX/*. Since there is no attribute correspondence whose coverage probability is 100% in each interlinking task of the table, a disjunctive pattern is needed for each interlinking task.

To conclude, a disjunctive pattern is needed to express the interlinking pattern for most of the interlinking tasks. There are totally 9 interlinking tasks being computed the coverage probabilities of attribute correspondences. Among all these 9 interlinking tasks, 7 interlinking tasks require disjunctive interlinking patterns for

Attribute Correspondence	AC_1	AC_2	AC_3	AC_4	AC_5	AC_6	AC_7	AC_8	AC_9	AC_{10}	AC_{11}
Coverage Probability in *Sandbox001*	0.02	0.09	0.06	0.88	0.30	0.43	0.19	0.16	0.16	0.02	0.01
Coverage Probability in *Sandbox002*	0.02	0.09	0.06	0.88	0.30	0.43	0.01	0.17	0.16	0.02	0.01
Coverage Probability in *Sandbox003*	0.02	0.09	0.06	0.88	0.30	0.01	0.01	0.16	0.16	0.02	0.01
Coverage Probability in *Sandbox006*	0.02	0.09	0.06	0.86	0.30	0.91	0.19	0.15	0.16	0.02	0.01
Coverage Probability in *Sandbox010*	0.02	0.09	0.06	0.88	0.30	0.98	0.19	0.16	0.16	0.02	0.01
Coverage Probability in *Sandbox011*	0.02	0.09	0.06	0.88	0.30	0.98	0.19	0.16	0.16	0.02	0.01

Table 8.5: Coverage Probabilities of Attribute Correspondences in *Sandbox001*, *Sandbox002*, *Sandbox003*, *Sandbox006*, *Sandbox010*, and *Sandbox011*

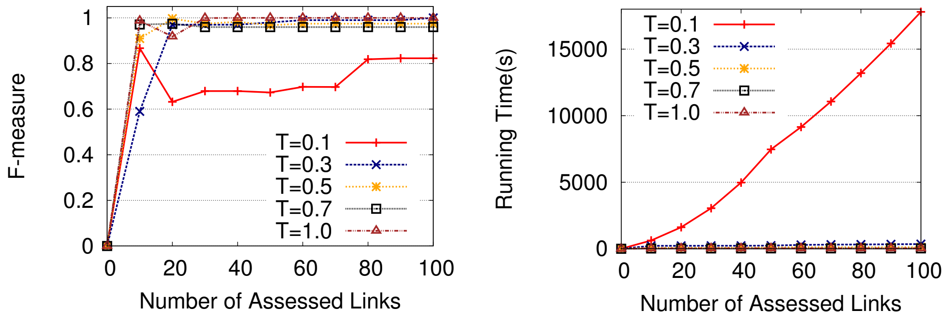
generating links. Therefore, a disjunctive pattern is highly required for interlinking various kinds of RDF data sets.

8.3.2 Threshold for Building Binary Similarity Sequence

This section evaluates the thresholds for creating binary similarity sequences of sample links. According to the illustration of Section 6.3 in Chapter 6, a set of sample links are created for constructing and improving the interlinking pattern by Extended Version Space. In order to construct and improve the interlinking pattern, a binary similarity sequence is created to represent the set of potential attribute correspondences in each sample link to be learned by Extended Version Space. The binary similarity sequence is transferred from a similarity tuple $(B^{(i)})$, $i = 1, \ldots, m$, where m is the number of potential attribute correspondences that are discovered by the K-medoids clustering step. Each item of the similarity tuple $(B^{(i)})$ is the similarity of two attributes values from two instances that form a sample link. If the similarity is above a pre-defined threshold T, we assume that the potential attribute correspondence exists in the sample link. Thus, the threshold T influences the learning effect of Extended Version Space. It is a parameter of function f that converts a decimal in the range $[0, 1]$ to a binary value $\{0, 1\}$. This function maps the similarity values of each sample link's similarity tuple into a binary similarity sequence.

$$f(x) = \begin{cases} 1 & \text{if } x > T \text{ or } x = 1.0 \\ 0 & \text{if } x \leq T \end{cases} \tag{8.1}$$

Since there are countless decimals in the range $[0, 1]$, we only pick out a few decimals to be the threshold T in order to evaluate the F-measure and running time of each interlinking task. They are $T = 0.1, 0.3, 0.5, 0.7, 1.0$.



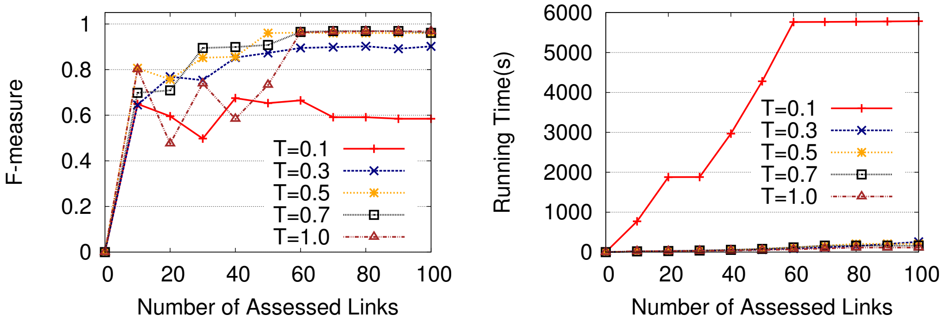
(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.2: F-measure and Running Time of *Person1* with Different Thresholds

Figure 8.2 shows the F-measures and running time of the interlinking task on the data set *Person1*. According to Figure 8.2 (a), the threshold 0.5 is the best choice in order to achieve a stable and high F-measure. F-measures of the thresholds 0.3, 0.5, 0.7, and 1.0 are quite similar. F-measures of the threshold 1.0 are much lower than the ones of other thresholds. The F-measures of the thresholds 0.5, 0.7 and 1.0 converge faster than the thresholds 0.3 and 1.0. Among the three thresholds 0.5, 0.7 and 1.0, the threshold 1.0 achieves the highest F-measure when the interlinking process finishes. However, it has an obvious fall when there are 20 assessed links. It means that F-measures of the threshold 1.0 is less stable than the ones of the thresholds 0.5 and 0.7. Furthermore, the threshold 0.5 achieves a higher F-measure than the threshold 0.7 when the interlinking process finishes. Therefore, the threshold 0.5 is the best choice for gaining a stable and high F-measure for the interlinking task *Person1*.

According to Figure 8.2 (b), the threshold 0.5 is the best choice in order to achieve a stable and high F-measure with a relatively shorter running time. The running time of the threshold 0.1 is the longest. Furthermore, the running time of the threshold 0.1 increases faster than the one of other thresholds. The running time of the thresholds 0.3, 0.5, 0.7 and 1.0 is similar. The thresholds that have the shortest running time are 1.0 and 0.7. The total running time of the two thresholds is 30 seconds and 28 seconds respectively. The threshold 0.5 spends 90 more seconds to finish the interlinking process than the threshold 1.0, but the growth of F-measure with the threshold 0.5 is more stable than the one of the threshold 1.0. Furthermore, F-measure of the threshold 0.5 is higher than the one of the threshold 0.7 when the interlinking process finishes. Thus, the threshold 0.5 is a best choice to obtain a stable and high F-measure with a relatively shorter running time.

To conclude, the interlinking effect of the interlinking task *Person1* is the best when the threshold is 0.5.



(a) Comparison on F-measure

(b) Comparison on Running Time

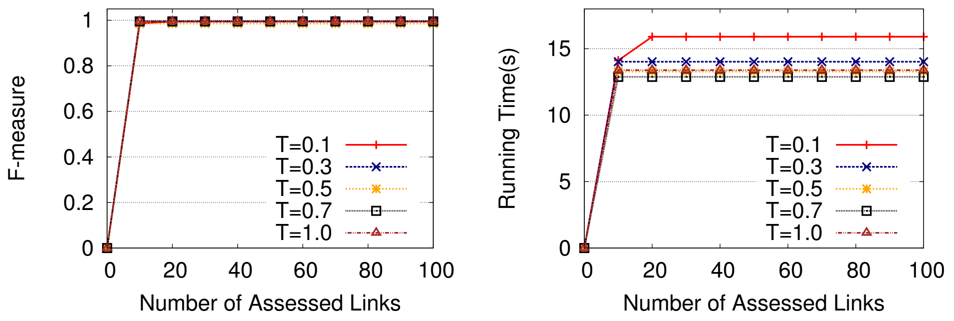
Figure 8.3: F-measure and Running Time of *Person2* with Different Thresholds

Figure 8.3 shows the F-measures and running time of the interlinking task on data sets *Person2*. According to Figure 8.3 (a), the threshold 0.7 is the best choice in order to achieve a stable and high F-measure. In detail, the growth of F-measure of the threshold 1.0 is the most unstable one among all thresholds, in that there are two falls when there are 20 and 40 assessed links respectively. The threshold 0.1 has the lowest F-measure of all thresholds when the interlinking process finishes. The threshold 0.3 has a relatively lower F-measure than the thresholds 1.0, 0.7 and 0.5 when the interlinking process finishes. In contrast, F-measures of the thresholds

0.5 and 0.7 grow approximately more stably and higher than the ones of other thresholds. The growth of F-measure when the threshold is 0.7 are better than the one of the threshold 0.5, because there is a little fall of the threshold 0.5 when there are 20 assessed links. So the threshold 0.7 is the best choice to gain a stable and high F-measure.

As for the running time of the five thresholds in Figure 8.3 (b), the threshold 0.1 costs the longest running time than other thresholds throughout the whole interlinking process. The running time of the thresholds 0.3, 0.5, 0.7 and 1.0 is similar. They spend 258 seconds, 205 seconds, 162 seconds and 114 seconds respectively when the interlinking process finishes.

Therefore, the interlinking effect reaches the best when the threshold is 0.7 for the interlinking task *Person2*.



(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.4: F-measure and Running Time of *INSEE* and *EUROSTAT* with Different Thresholds

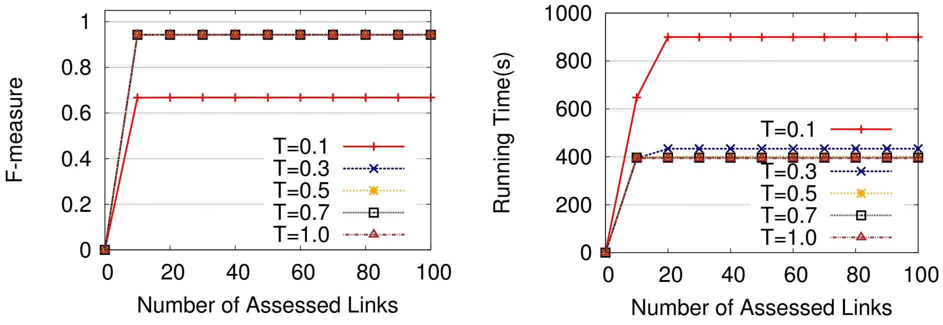
Figure 8.4 shows the F-measures and running time of the interlinking task on data sets *INSEE* and *EUROSTAT*. In Figure 8.4 (a), F-measure reaches 0.99 when there are 10 assessed links no matter which threshold is used. F-measure stay at the same level when more assessed links are obtained and used to improved the interlinking pattern.

In Figure 8.4 (b), the running time of the five thresholds is quite close. The threshold 0.1 costs the longest time, 16 seconds, for interlinking. The thresholds 0.1, 0.3, 0.5 and 0.7 cost fewer seconds than the threshold 0.1, but the difference is only a few seconds.

Therefore, the interlinking effects of all thresholds are roughly the same for this interlinking task.

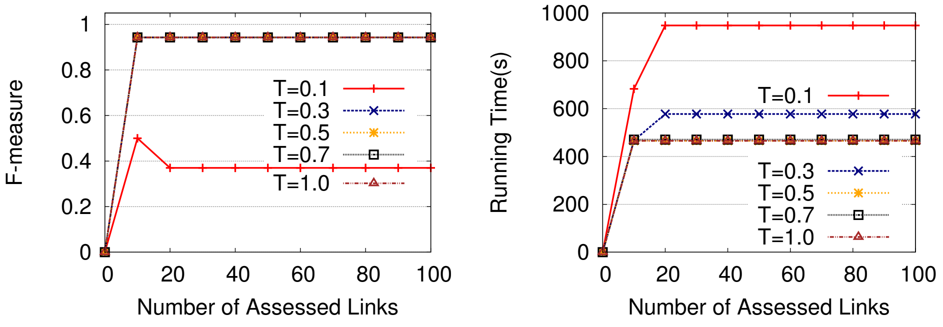
Figure 8.5 and Figure 8.6 show F-measures and running time of the interlinking tasks *Sandbox001* and *Sandbox002*. As for the F-measures of *Sandbox001* and *Sandbox002* in Figure 8.5 (a) and Figure 8.6 (a), the thresholds 0.3, 0.5, 0.7 and 1.0 outperform the threshold 0.1. The threshold 0.1 achieves much lower F-measures than other thresholds.

According to Figure 8.5 (b) and Figure 8.6 (b), the running time of thresholds 0.1 and 0.3 is longer than the one of the rest thresholds no matter how many assessed



(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.5: F-measure and Running Time of *Sandbox001* with Different Thresholds

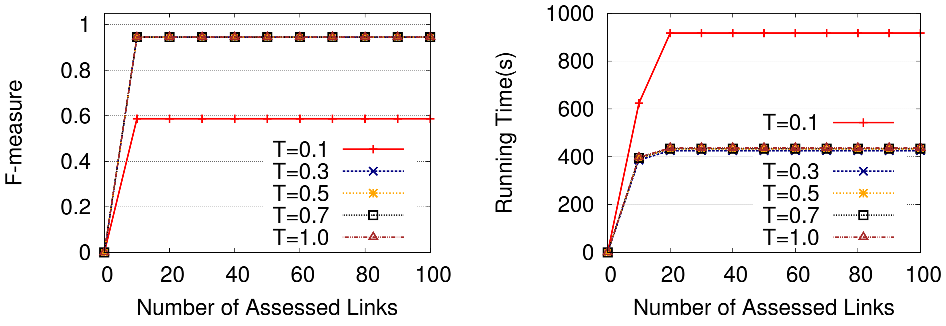


(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.6: F-measure and Running Time of *Sandbox002* with Different Thresholds

links are used to improve the interlinking pattern. In particular, the running time of threshold 0.1 is roughly two times of the running time of the thresholds 0.5, 0.7 and 1.0 after learning 20 assessed links. Even with longer running time, the threshold 0.1 cannot enhance F-measures to the same level with the ones of the thresholds 0.5, 0.7 and 1.0.

Therefore, the interlinking effect of the interlinking tasks *Sandbox001* and *Sandbox002* is the best when the thresholds are 0.5, 0.7 and 1.0.



(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.7: F-measure and Running Time of *Sandbox003* with Different Thresholds

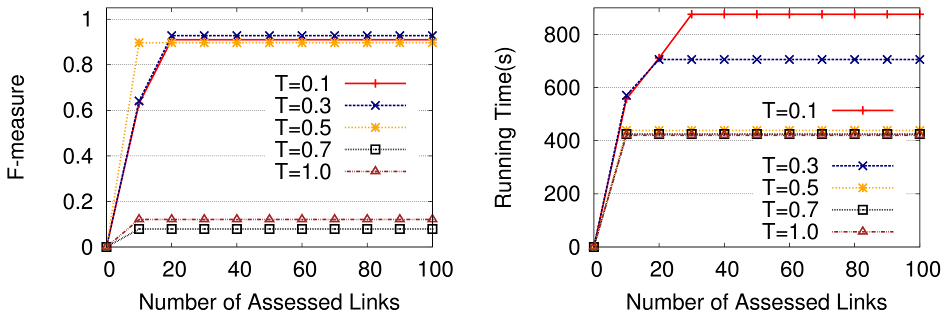
Figure 8.7 shows F-measures and running time of the interlinking task *Sandbox003*. As for the F-measures of *Sandbox003* in Figure 8.7 (a), the thresholds 0.3, 0.5, 0.7 and 1.0 outperform the threshold 0.1. F-measures of the four thresholds are 0.94 when there are 10 assessed links. In contrast, the F-measure of the threshold 0.1 is only 0.59.

As for Figure 8.7 (b), the running time of the thresholds 0.3, 0.5, 0.7 and 1.0 is much lower than the one of the threshold 0.1. They are 425 seconds, 432 seconds, 434 seconds and 437 seconds respectively when the interlinking process finishes. The threshold 0.3 has the shortest running time. However, the difference between these running time is only a few seconds.

Hence, for the interlinking task *Sandbox003*, the thresholds 0.3, 0.5, 0.7 and 1.0 are the best choices.

Figure 8.8 shows F-measures and running time of the interlinking task *Sandbox006*. As for the F-measures of *Sandbox006* in Figure 8.8 (a), the threshold 0.1, 0.3 and 0.5 outperform the other two thresholds. In detail, the three thresholds enhance the F-measure up to roughly 0.9 when there are 20 assessed links. In contrast, F-measure stays at the level of roughly 0.1 when the thresholds are 0.7 and 1.0.

As for the running time shown in Figure 8.8 (b), the thresholds 0.5, 0.7 and 1.0 are the best choices. The threshold 0.1 costs the longest running time after learning 30 assessed links. It uses 876 seconds for interlinking the data sets. The threshold 0.3 costs the second longest running time after learning 30 assessed links. It spends 706 seconds. The thresholds 0.5, 0.7 and 1.0 only cost 439 seconds, 425 seconds and 420 seconds respectively after learning 30 assessed links. Therefore, the thresholds

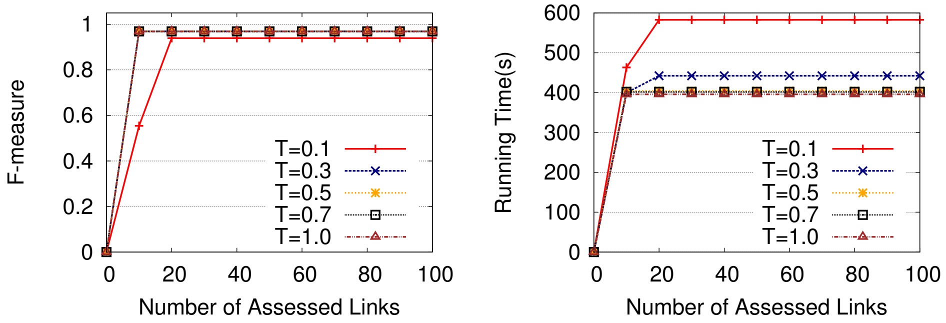


(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.8: F-measure and Running Time of *Sandbox006* with Different Thresholds

0.5, 0.7 and 1.0 perform the best from the perspective of the running time.

By considering both the F-measure and the running time, the threshold 0.5 perform the best in the interlinking task *Sandbox006*. Because it uses a relatively shorter running time to achieve a relatively high F-measure when the interlinking process finishes.



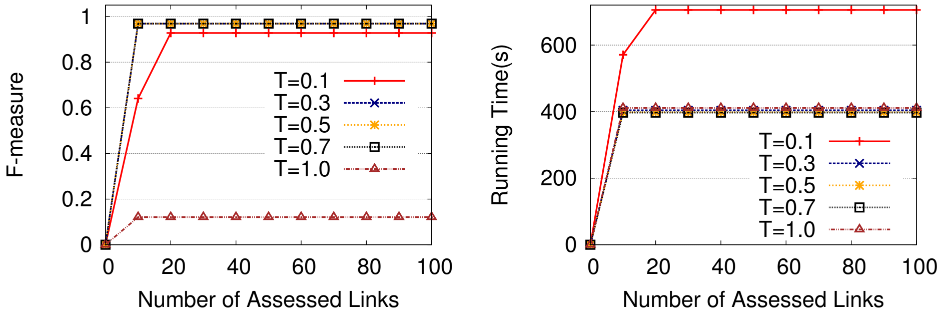
(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.9: F-measure and Running Time of *Sandbox010* with Different Thresholds

Figure 8.9 shows F-measures and running time of the interlinking task *Sandbox010*. As for the F-measures of *Sandbox010* in Figure 8.9 (a), the F-measures are highest when the thresholds are 0.3, 0.5, 0.7 and 1 after learning 10 assessed links. They reach 0.97 when there are only 10 assessed links. The F-measure of the threshold 1.0 is less higher, which is 0.94.

According to Figure 8.9 (b), the thresholds 0.5, 0.7 and 1.0 cost shorter running time than the thresholds 0.1 and 0.3 after learning 20 assessed links. In detail, the threshold 0.1 costs 80 more seconds than the thresholds 0.5, 0.7 and 1.0 after learning 20 assessed links. Moreover, the threshold 0.3 costs 40 more seconds than the thresholds 0.5, 0.7 and 1.0 when there are 20 assessed links. The thresholds 0.5, 0.7 and 1.0 cost roughly the same amount of time throughout the whole interlinking process. They spend 404 seconds, 401 seconds and 396 seconds respectively when the interlinking process finishes.

Therefore, the interlinking effect of the interlinking task *Sandbox010* is the best when the thresholds are 0.5, 0.7 and 1.0.



(a) Comparison on F-measure (b) Comparison on Running Time

Figure 8.10: F-measure and Running Time of *Sandbox011* with Different Thresholds

Figure 8.10 shows F-measures and running time of the interlinking task *Sandbox011*. As for the F-measures of *Sandbox011* in Figure 8.10 (a), the thresholds 0.3, 0.5 and 0.7 perform the best. Their F-measures reach 0.97 when there are only 10 assessed links. With the same amount of assessed links, F-measures of the thresholds 1.0 and 0.1 are 0.12 and 0.93 respectively. The highest F-measures of the thresholds 1.0 and 0.1 are 0.12 and 0.93 respectively when the interlinking process finishes.

From Figure 8.10 (b), the threshold 0.1 costs much more seconds than other thresholds. In particular, it costs 706 seconds to fulfill this interlinking task. The thresholds 0.3, 0.5, 0.7 and 1.0 require 404 seconds, 398 seconds, 397 seconds and 411 seconds respectively to fulfill the interlinking task. The difference is only a few seconds.

Therefore, the interlinking effect are the best when interlinking the data sets *Sandbox011* with the thresholds 0.3, 0.5 and 0.7.

To conclude, based on the analysis above, Extended Version Space performs the best when the threshold is 0.5, because the threshold 0.5 is the best choice for 8 interlinking tasks among all 9 interlinking tasks. The reason of good performance when the threshold is 0.5 is analyzed as follows.

If the threshold is lower than 0.5, there will be lots of dissimilar attribute values that are recognized as “similar”. Thereafter, a lot of attribute pairs that are not corresponding will be treated as attribute correspondences and added in the interlinking pattern. They will cause the interlinking pattern to cover more incorrect links, which in turn will reduce the F-measure of the generated link set. Furthermore, the running time of the interlinking pattern will increase, in that more attribute pairs that are not corresponding are added into the interlinking pattern. The more attribute correspondences are contained in the interlinking pattern, the more I/O operations are required for obtaining attribute values by the Silk script. For example, both F-measure and running time are not satisfiable in the interlinking tasks *Person1*, *Perons2*, *Sandbox001*, *Sandbox002* and *Sandbox003* when the threshold is

0.1. Figure 8.11 shows the average precisions, average recalls and F-measures of the interlinking task *Sandbox001* when the threshold is 0.1. The average precision is only 0.52 when the interlinking process finishes, though the average recall is as high as 0.92. Consequently, the F-measure is only 0.67. The running time of the interlinking task *Sandbox001* with the threshold 0.1 is shown in Figure 8.5 (b), which is much longer than the one of other thresholds when the interlinking process finishes.

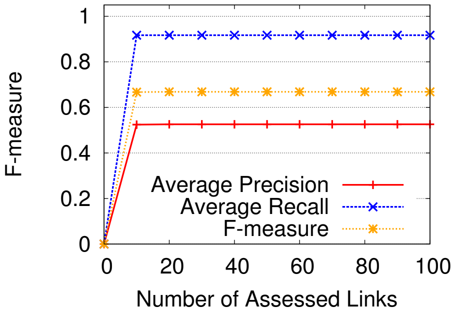


Figure 8.11: Interlinking Effect of *Sandbox001* with the Threshold $T = 0.1$

If the threshold is higher than 0.5, there will be lots of similar attribute values that are recognized as “dissimilar”. Thereafter, a lot of attribute correspondences will not be treated as attribute correspondences and not added in the interlinking pattern. Consequently, there are fewer attribute correspondences in the interlinking pattern for generating a correct link. It means that the number of restrictions for generating a correct link is fewer, which will make the interlinking pattern unable to filter some incorrect links. Therefore, the F-measure of the generated link set will decrease accordingly. The running time of the interlinking pattern will also decrease, because there are less attribute correspondences in the interlinking pattern, which require less I/O operations for obtaining attribute values by the Silk script. For example, F-measure is relatively low in the interlinking tasks *Sandbox006* and *Sandbox011* when the threshold is 1.0. Figure 8.12 shows the average precisions, average recalls and F-measures of the interlinking task *Sandbox011* when the threshold is 1.0. The average precision is only 0.06 when the interlinking process finishes, though the average recall is as high as 0.99. So the F-measure is only 0.12. The running time of the interlinking task *Sandbox011* with the threshold 1.0 is shown in Figure 8.10 (b). The running time of the threshold 1.0 is as short as the one of other thresholds except the threshold 0.1 after learning 10 assessed links.

Therefore, according to the evaluation of this section, in the following evaluation sections, we show the F-measures and running time of Extended Version Space by creating binary similarity sequences with the threshold $T = 0.5$ for each sample link.

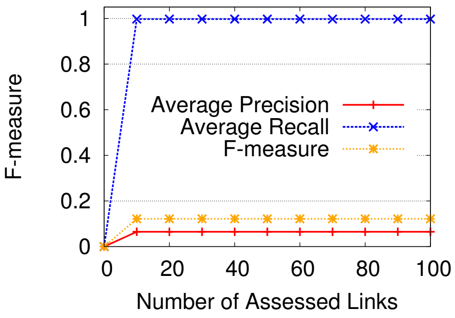


Figure 8.12: Interlinking Effect of *Sandbox001* with the Threshold $T = 1.0$

8.3.3 Comparison between Extended Version Space and Disjunctive Version Space

This section compares F-measures and running time of Extended Version Space and the ones of Disjunctive Version Space on different interlinking tasks.

The F-measures and running time of Extended Version Space and Disjunctive Version Space are computed by executing interlinking with the generalized patterns of both learning methods when the assessed links are transferred into binary similarity sequences with the threshold 0.5. There is an exception. If there is no assessed incorrect link's binary similarity sequence being learned, the generalized pattern is a universal expression which contains all correct and incorrect links. Thus, it cannot be used for generating links. In this case, we compute the F-measures by executing interlinking with the specialized patterns of both methods.

Since the number of binary similarity sequences in the generalized pattern of Disjunctive Version Space is usually very huge, the running time of executing interlinking with such a generalized pattern will be extremely long. For instance, each interlinking task of *Sandbox* has 70 potential attribute correspondences that are discovered by the K-medoids clustering step. There are at most 2^{70} binary similarity sequences in the generalized pattern of Disjunctive Version Space. As for the interlinking tasks *Person1* and *Person2*, each of them has around 10 potential attribute correspondences. There are at most 2^{10} binary similarity sequences in the generalized pattern of Disjunctive Version Space. Therefore, if we generate links with such a large pattern, there will be a lot of I/O operations that are required by the Silk script. The reason is that each attribute correspondence of a binary similarity sequence is transferred into a pair of queries in the Silk script. Each query gets an attribute value from the interlinking RDF data set, which is an I/O operation for the computer. Consequently, the more binary similarity sequences in the interlinking pattern, the more I/O operations are required by Silk.

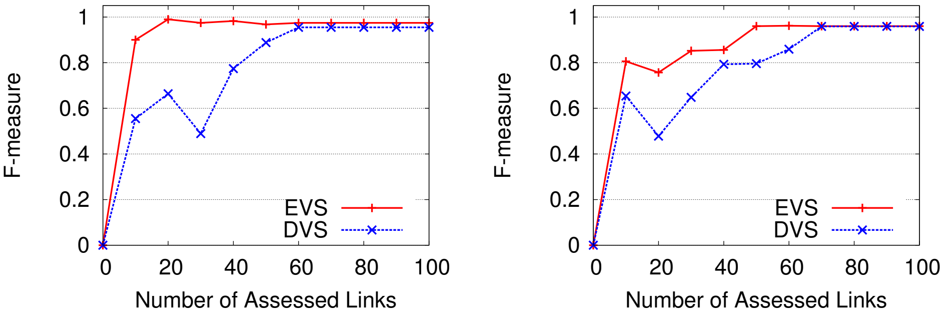
We do not generate links with the generalized pattern of Disjunctive Version Space but with a generalized pattern of Disjunctive Version Space after the operation MERGE of Extended Version Space. The operation is defined in Section 7.3.2 of Chapter 7. On the one hand, the F-measures of the generated link set by executing the generalized pattern after merging are the same with the one before merging. On the other hand, the running time of the generalized pattern before merging is definitely longer than the one of the generalized pattern after merging. If we can show that the running time of the generalized pattern after merging is longer than the one of Extended Version Space, the running time of Extended Version Space will definitely be shorter than the one of Disjunctive Version Space.

We implement a merged Disjunctive Version Space as a by-product when implementing Extended Version Space, which is the Pat_{Ori} in Algorithm 3. There are two operations that distinguish the generalized pattern of Disjunctive Version Space from the generalized pattern of Extended Version Space. They are MERGE and RESELECT. We maintain a copy of the generalized pattern after the binary similarity sequences are merged while before being reselected into a generalized pattern of Extended Version Space when a new assessed link's binary similarity sequence is learned. This pattern is used to evaluate the F-measures and running time of the merged Disjunctive Version Space.

In the figures of this section, Extended Version Space is denoted as EVS. The merged Disjunctive Version Space is denoted as DVS.

8.3.3.1 F-measure

This subsection compares the F-measures of Extended Version Space and the ones of merged Disjunctive Version Space in all 9 interlinking tasks.

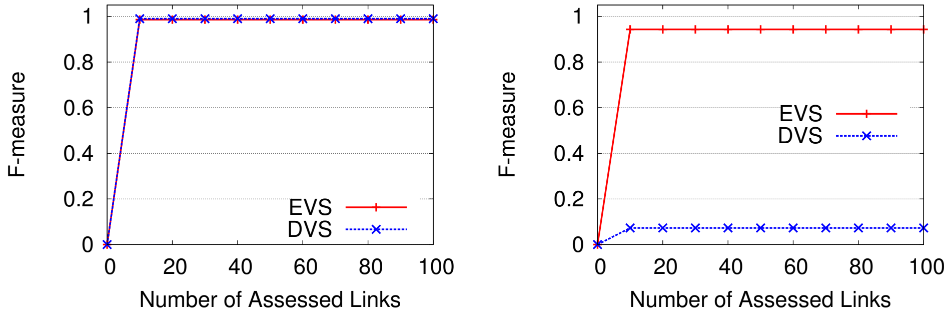


(a) Person1 (b) Person2

Figure 8.13: F-measure of *Person1* and *Person2* with EVS and DVS when $T = 0.5$

Extended Version Space converges faster than merged Disjunctive Version Space when interlinking *Person1* and *Person2*. In Figure 8.13 (a), Extended Version Space converges faster than merged Disjunctive Version Space. The F-measure of Extended Version Space is 0.9 when there are 10 assessed links. In contrast, the F-measure of merged Disjunctive Version Space is 0.56. Afterwards, the F-measure of Extended Version Space increases to 0.98 when there are 20 assessed links. While the F-measure of merged Disjunctive Version Space reaches the similar F-measure as Extended Version Space when there are 60 assessed links. Extended Version Space reaches a higher F-measure much quicker than merged Disjunctive Version

Space. In Figure 8.13 (b), Extended Version Space converges faster than merged Disjunctive Version Space, too. F-measure of Extended Version Space reaches 0.96 when there are 50 assessed links, but merged Disjunctive Version Space should learn 70 assessed links to gain the same F-measure. Hence, in this figure, the Extended Version Space converges faster than merged Disjunctive Version Space.



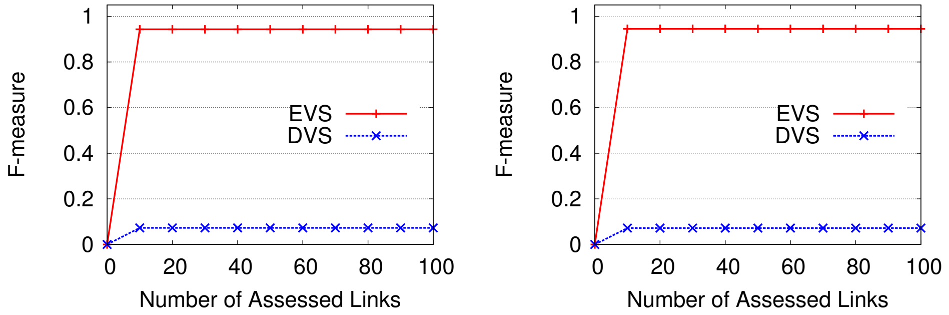
(a) INSEE and EUROSTAT

(b) Sandbox001

Figure 8.14: F-measure of *INSEE* vs. *EUROSTAT* and *Sandbox001* with EVS and DVS when $T = 0.5$

The two learning methods converge at the same speed when interlinking the data sets *INSEE* and *EUROSTAT*. In Figure 8.14 (a), Extended Version Space converges at the same speed with merged Disjunctive Version Space. Both of their F-measure reach 0.99 when there are 10 assessed links.

Extended Version Space converges faster than merged Disjunctive Version Space when interlinking *Sandbox001*. In Figure 8.14 (b), F-measure of merged Disjunctive Version Space is 0.07 no matter how many assessed links are learned. In contrast, the F-measure of Extended Version Space reaches 0.94 when there are 10 assessed links. In this figure, Extended Version Space converges faster to a much higher F-measure than merged Disjunctive Version Space.

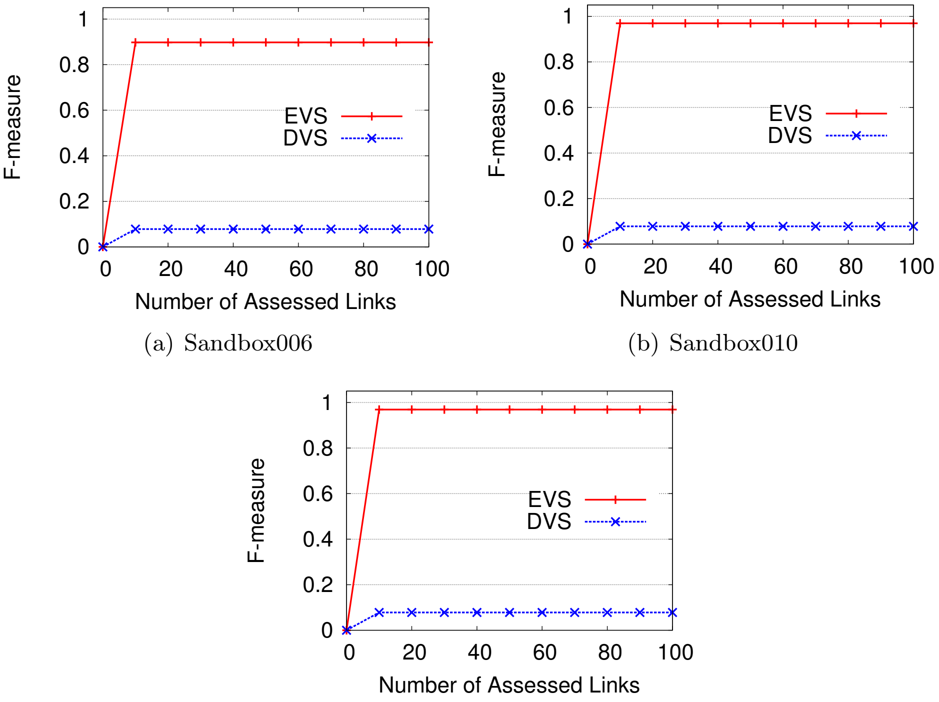


(a) Sandbox002

(b) Sandbox003

Figure 8.15: F-measure of *Sandbox002* and *Sandbox003* with EVS and DVS when $T = 0.5$

In the interlinking tasks *Sandbox002*, *Sandbox003*, *Sandbox006*, *Sandbox010* and



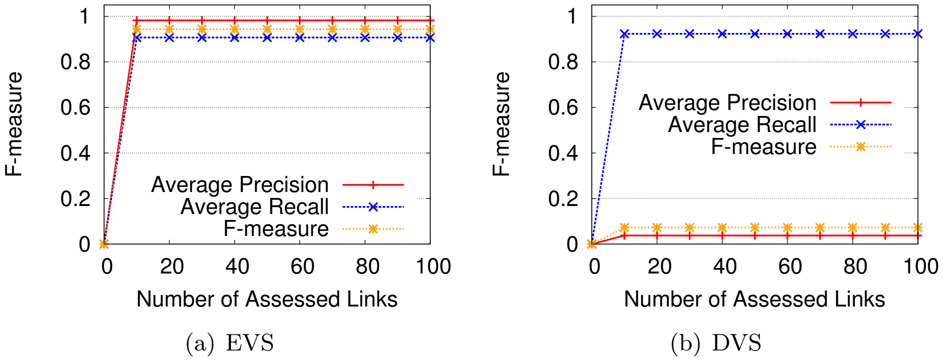
(a) Sandbox006

(b) Sandbox010

(c) Sandbox011

Figure 8.16: F-measure of *Sandbox006*, *Sandbox010* and *Sandbox011* with EVS and DVS when $T = 0.5$

Sandbox011, Extended Version Space converges faster with a much higher F-measure than merged Disjunctive Version Space. In Figure 8.15 and Figure 8.16, Extended Version Space's F-measures are higher than 0.9 when the interlinking process finishes, while merged Disjunctive Version Space's F-measures are only 0.07.



(a) EVS

(b) DVS

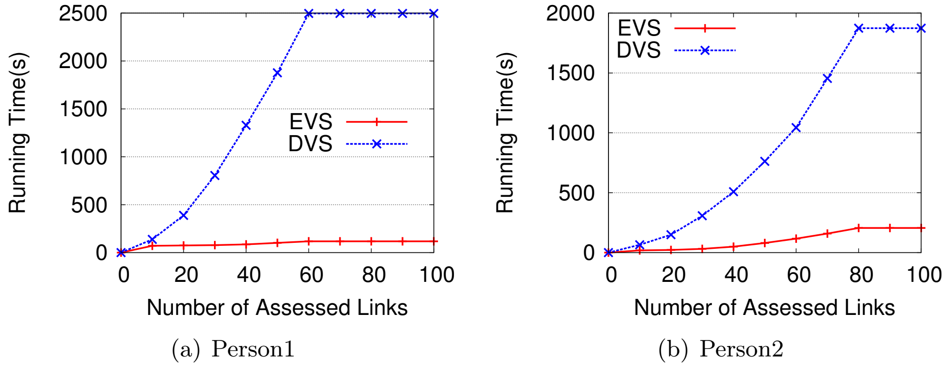
Figure 8.17: Precision, Recall and F-measure of Sandbox001 with EVS and DVS when $T = 0.5$

To conclude, Extended Version Space converges faster than merged Disjunctive Version Space in 8 interlinking tasks of all 9 interlinking tasks. In other words, the F-measure of Extended Version Space reaches a high level with fewer assessed links than the one of merged Disjunctive Version Space. The reason is that, the interlinking pattern of Extended Version Space is able to filter more incorrect links than the one of merged Disjunctive Version Space. For example, when interlinking *Sandbox001*, the average precisions, average recalls and F-measures of merged Disjunctive Version Space and Extended Version Space are shown in Figure 8.17. According to Figure 8.17 (a), average precisions and average recalls of Extended Version Space are both relatively more than 0.9 when learning different amount of assessed links. In contrast, average precisions and average recalls of merged Disjunctive Version Space are quite different in Figure 8.17 (b). Average recalls of merged Disjunctive Version Space are very high. It means that the interlinking pattern of merged Disjunctive Version Space covers most of the correct links of two RDF data sets. Average precisions of merged Disjunctive Version Space are quite low, because there are many incorrect links being produced. The generalized pattern of merged Disjunctive Version Space usually contain more binary similarity sequences than the one of Extended Version Space. Because there is a RESELECT operation in Extended Version Space that is not required in Disjunctive Version Space, which picks out binary similarity sequences from the merged generalized pattern so that each covers at least one binary similarity sequence in the specialized pattern. It also means that only binary similarity sequences that are relevant to assessed correct links are maintained in the generalized pattern of Extended Version Space. However, the generalized pattern of merged Disjunctive Version Space contains not only relevant binary similarity sequences but also irrelevant binary similarity sequences that cover many incorrect links. This is the reason why merged Disjunctive Version

Space converges slower than Extended Version Space, because it may produce more incorrect links than Extended Version Space. Only when more assessed incorrect links are learned, the generalized pattern of merged Disjunctive Version Space can filter the same amount of incorrect links with the generalized pattern of Extended Version Space. Moreover, since the F-measure of generated link set that is produced by executing the generalized pattern of Disjunctive Version Space is the same with the one of merged Disjunctive Version Space, the F-measure of Extended Version Space converges faster than Disjunctive Version Space' F-measure.

8.3.3.2 Running Time

This section compares the running time of Extended Version Space with the one of merged Disjunctive Version Space in all 9 interlinking tasks.



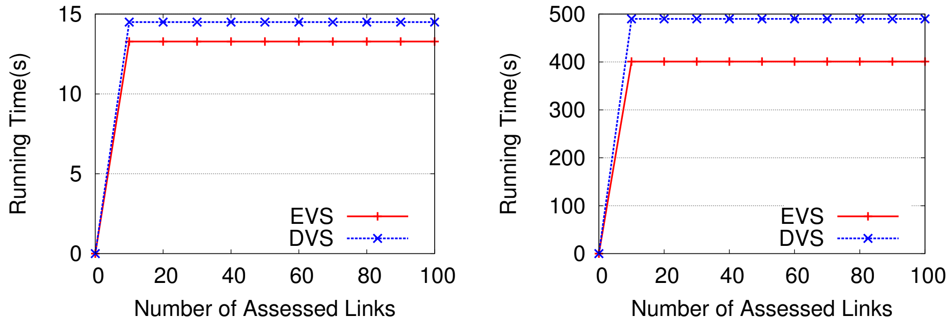
(a) Person1

(b) Person2

Figure 8.18: Running Time of *Person1* and *Person2*

Figure 8.18 confirms that Extended Version Space outperforms Disjunctive Version Space with respect to the running time when interlinking *Person1* and *Person2*. As for both interlinking tasks, Disjunctive Version Space spends far longer running time than Extended Version Space. By referring to Figure 8.13, Disjunctive Version Space costs much longer running time in order to reach the same level of F-measure with Extended Version Space. According to Figure 8.13 (a) and Figure 8.18 (a), Disjunctive Version Space spends 2300 more seconds to reach the same F-measure of Extended Version Space when there are 60 assessed links for interlinking *Person1*. According to Figure 8.13 (b) and Figure 8.18 (b), Disjunctive Version Space spends 1300 more seconds to reach the same F-measure of Extended Version Space when there are 70 assessed links for interlinking *Person2*. Thus, this figure shows that Extended Version Space converges more quickly to a higher F-measure with shorter running time than merged Disjunctive Version Space.

The running time of Extended Version Space and the one of merged Disjunctive Version Space are quite close when interlinking data sets *INSEE* and *EUROSTAT* in each learning round. In Figure 8.19 (a), the running time of Extended Version Space and the one of merged Disjunctive Version Space are almost the same when

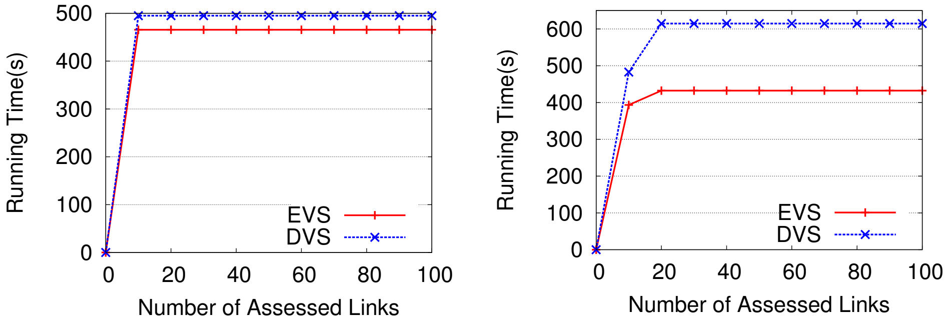


(a) INSEE and EUROSTAT (b) Sandbox001

Figure 8.19: Running Time of *INSEE* vs. *EUROSTAT* and *Sandbox001*

there are 10 assessed links. Both of them spend around 14 seconds to find out correct links.

There is roughly 100-second difference between the running time of Extended Version Space and the one of merged Disjunctive Version Space when interlinking data sets *Sandbox001* in each learning round. In Figure 8.19 (b), merged Disjunctive Version Space is slower than Extended Version Space by nearly 90 seconds in each learning round. By referring to Figure 8.14 (b), F-measure of Extended Version Space is always much higher than the one of merged Disjunctive Version Space in each learning round, although merged Disjunctive Version Space spends longer time to improve the interlinking pattern. Thus, Figure 8.19 (b) also shows that Extended Version Space converges more quickly to a higher F-measure with shorter running time than merged Disjunctive Version Space.

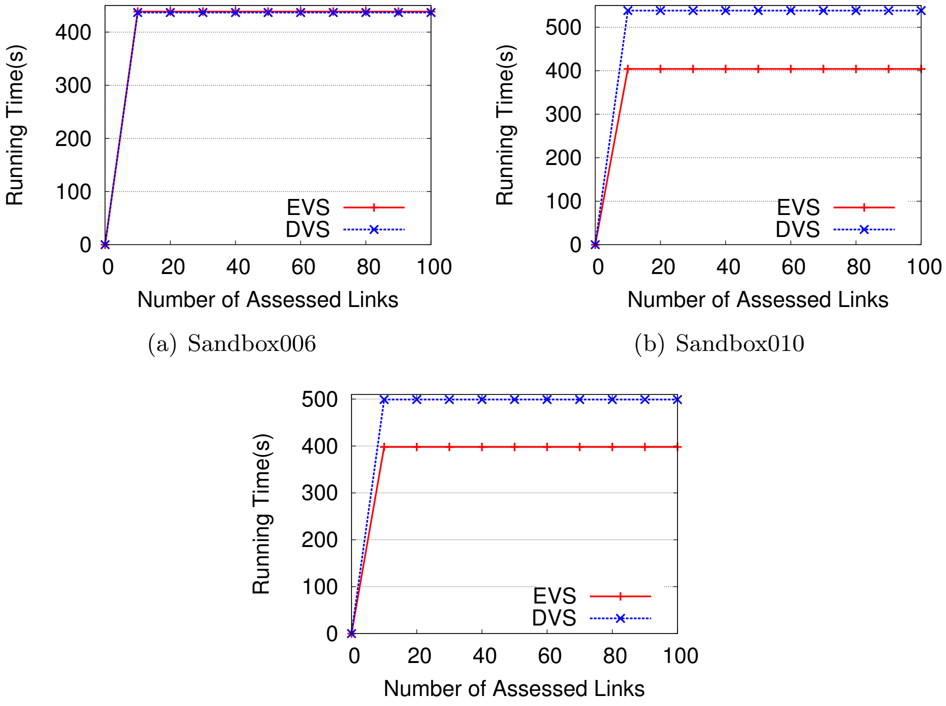


(a) Sandbox002 (b) Sandbox003

Figure 8.20: Running Time of *Sandbox002* and *Sandbox003*

The running time of Extended Version Space and the one of merged Disjunctive Version Space are quite close when interlinking *Sandbox002* in each learning round. There is roughly 180-second difference between the running time of Extended Version Space and the one of merged Disjunctive Version Space when interlinking data sets *Sandbox003* in each learning round. As for the interlinking task of *Sandbox002* in

Figure 8.20 (a), merged Disjunctive Version Space spends almost the same running time with Extended Version Space to generate correct links in each learning round. However, its F-measure is much lower than the one of Extended Version Space in Figure 8.15 (a). For the interlinking task of *Sandbox003* in Figure 8.20 (b), merged Disjunctive Version Space is slower than Extended Version Space by roughly 180 seconds since there are 20 assessed links. Whereas, F-measure of Extended Version Space is far higher than the one of merged Disjunctive Version Space in Figure 8.15 (b) in each learning round. Thus, this figure also shows that Extended Version Space converges more quickly to a higher F-measure with shorter running time than merged Disjunctive Version Space.



(a) Sandbox006

(b) Sandbox010

(c) Sandbox011

Figure 8.21: Running Time of *Sandbox006*, *Sandbox010* and *Sandbox011*

The running time of Extended Version Space and the one of merged Disjunctive Version Space are quite close when interlinking *Sandbox006* in each learning round. There is roughly 100-second difference between the running time of Extended Version Space and the one of merged Disjunctive Version Space when interlinking *Sandbox010* and *Sandbox011* in each learning round. As for the interlinking task of *Sandbox006* in Figure 8.21 (a), the running time of Extended Version Space is almost the same with the one of merged Disjunctive Version Space in each learning round, but the F-measure of Extended Version Space is much higher than the one of merged Disjunctive Version Space in Figure 8.16 (a) in each learning round. In Figure 8.21 (b) and (c), merged Disjunctive Version Space spends 100 more seconds than Extended Version Space. Whereas, F-measure of Extended Version Space is much higher than the one of merged Disjunctive Version Space in Figure 8.16 (b) and (c) in each learning round. Thus, this figure also shows that Extended Version Space converges more quickly to a higher F-measure with shorter running time than merged Disjunctive Version Space.

To conclude, Extended Version Space converges more quickly to a higher F-measure with shorter running time than Disjunctive Version Space. From the analysis of this section, the running time of Extended Version Space is shorter than the one of merged Disjunctive Version Space except the interlinking task *Sandbox006*. Even in the interlinking task *Sandbox006*, Extended Version Space only spends 1 more second than merged Disjunctive Version Space in each learning round. Since we are using a merged generalized pattern of Disjunctive Version Space for evaluation, the unmerged generalized pattern of Disjunctive Version Space will definitely spend much longer running time than Extended Version Space in all 9 interlinking tasks. The reason is that the interlinking pattern of Extended Version Space is usually more concise than the one of Disjunctive Version Space. The bigger an interlinking pattern is, the more I/O operations are required for generating links. Thus, the running time will increase consequently. Therefore, Extended Version S-pace requires shorter running time than Disjunctive Version Space. Moreover, with regards to the analysis of F-measure in Section 8.3.3.1, Extended Version Space achieves much higher F-measures than merged Disjunctive Version Space with the relatively short running time in most interlinking tasks. Hence, Extended Version Space converges more quickly to a higher F-measure with shorter running time than Disjunctive Version Space.

8.3.4 Comparison between Extended Version Space with other related works

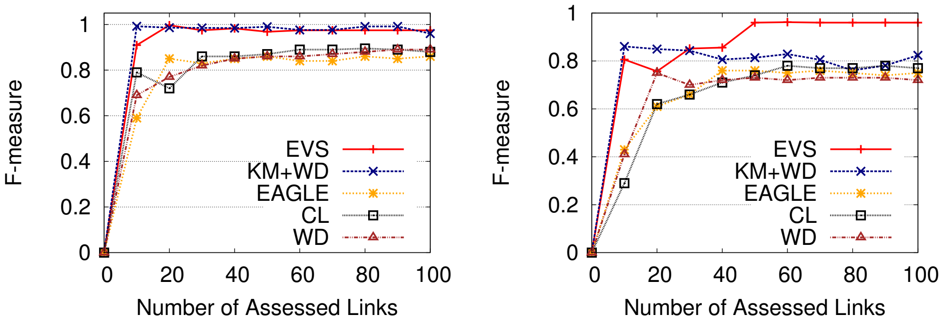
This section compares the F-measures and running time of Extended Version Space with related works [Ngonga Ngomo 2013, Ngonga Ngomo 2012b] that use Genetic Programming for interlinking. The protocol of the experiments on Extended Version Space in this section is the same as the one of Section 8.3.3, except that the experiments in this section are performed with a single thread.

Figure 8.22 and Figure 8.23 shows the comparisons on F-measure and running time. In both figures, our method is denoted as EVS. The three comparative methods of Ngonga Ngomo et al. are denoted as $EAGLE, CL, WD$ respectively. CL and WD are two interlinking methods that combine $EAGLE$ with two different Active Learning methods. Both Active Learning methods classify sample links into several groups according to the similarity vectors that are composed of similarities of attribute values according to each potential attribute correspondence. CL's Active Learning method selects informative links to be assessed by considering the intra-group correlations of sample links. WD's Active Learning method selects informative links to be assessed by considering both the intra-group correlations and the inter-group correlations of sample links. The figures show the best convergence results that are presented in their paper [Ngonga Ngomo 2013]. We compare F-measures and running time of Extended Version Space with their best F-measures when the size of initial population of Genetic Programming is 100. We do not implement *EAGLE*, *CL* and *WD* on our own, because the correspondence discovering method [Ngonga Ngomo 2011b] required by the three methods is not an open-source method.

Instead, we implement an interlinking method by combining K-medoids, a Genetic Programming method and the Active Learning method in *WD*. K-medoids is applied here to find out potential attribute correspondences. Genetic Programming is used to construct and improve the interlinking pattern. The Active Learning method in *WD* is used to select informative links to be assessed and learned by the interlinking pattern. We name such an interlinking method as *KM+WD*. The F-measure and running time of *KM+WD* are shown when the initial population of Genetic Programming is 20. The differences between *KM+WD* and *WD* of Ngonga Ngomo et al. [Ngonga Ngomo 2013] are below. First, we use K-medoids to find out potential attribute correspondences. The correspondence discovering method [Ngonga Ngomo 2011b] required by the three methods *EAGLE*, *CL* and *WD* extract correspondences from unstructured data of any content management system. The paper [Ngonga Ngomo 2011b] states that the usability of their approach relies heavily on the quality of the knowledge that returned by the automated means. It means that there are attribute pairs that are not corresponding and discovered by the correspondence discovering method of [Ngonga Ngomo 2011b]. Second, our genetic programming method uses only two aggregation methods "max" and "average". *WD* of Ngonga Ngomo et al. uses three aggregation methods "max", "min" and "add". Third, we do not set threshold in the Silk script for each attribute correspondence with the operations *mutation* and *crossover* .

Both Extended Version Space and *KM+WD* are implemented in Java 1.6+ with a single thread, and the experiments are performed on a desktop computer (2.5GHz 8-core CPU, memory size=32GB, 64-bit operating system) and allocated maximally 2GB of RAM. In contrast, *EAGLE*, *WD* and *CL* are implemented in Java 1.7+ with a single thread, and the experiments are performed on a server (2.0GHz 4-core CPU) and allocated maximally 2GB of RAM. The only difference on execution is that we use a CPU with a higher main frequency. Since most of running time is spent on I/O operations, which are required to extract data from data sets, the difference of CPU does not influence the running time of both interlinking methods.

Figure 8.22 shows the F-measure comparisons on two interlinking tasks *Person1* and *Person2*. With respect to the data set *Person1*, the final converged F-measures under *EVS* and *KM+WD* are much higher than the best ones of *EAGLE*, *CL* and *WD* (0.97 and 0.96 versus 0.86, 0.88 and 0.89 respectively). The F-measures of *EVS* and *KM+WD* stay on a high level when there are more than 20 assessed links. For the interlinking task *Person2*, the final F-measures of *EVS* and *KM+WD* are 0.96 and 0.82 respectively, while the ones of *EAGLE*, *CL* and *WD* are all roughly 0.77. When interlinking *Person2*, F-measure of Extended Version Space decreases after learning 20 assessed links. The reason is that some assessed links are classified



(a) Person1

(b) Person2

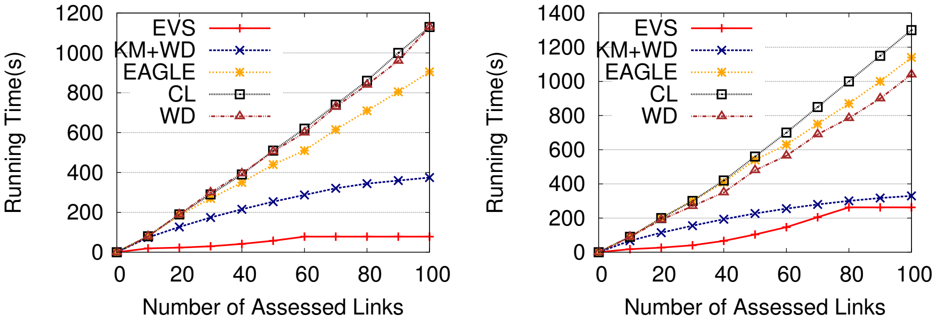
Figure 8.22: F-measures of Extended Version Space and Genetic Programming on *Person1* and *Person2*

wrongly into binary similarity sequence groups where most of the sample links in the group have the opposite marks with the assessed link. It means that either an assessed correct link is classified into a binary similarity sequence group where most of the sample links are incorrect links, or an assessed incorrect link is classified into a binary similarity sequence group where most of the sample links are correct links. With the wrongly classified assessed links, the positive (negative) binary similarity sequence will be recognized as a negative (positive) binary similarity sequence. Therefore, the improved interlinking pattern cannot cover/filter other sample links in the same binary similarity sequence group of the learned assessed link. Hence, F-measure of generated link set will decrease accordingly.

Both Extended Version Space and *KM+WD* achieve better F-measures in the interlinking task *Person1* and *Person2* than *EAGLE*, *CL* and *WD*. We can also observe that Extended Version Space and *KM+WD* converge faster than *EAGLE*, *CL* and *WD*. Extended Version Space has better F-measures than *EAGLE*, *CL* and *WD* by about 10% in both interlinking tasks when the interlinking process finishes. This is mainly due to the fact that Extended Version Space builds a disjunctive pattern, which can cover positive links (i.e., assessed correct links) and filter out negative links (i.e., assessed incorrect links) more comprehensively. However, the interlinking patterns of *EAGLE*, *CL* and *WD* are change by the operations *mutation* and *crossover* during each learning round, which may make their F-measures decrease accordingly. The two operations make some changes on some randomly-chosen parts of the interlinking pattern. This is also the reason that *KM+WD* sometimes decreases when more assessed links are learned. *KM+WD* also has better F-measures than *EAGLE*, *CL* and *WD* by about 10% in the interlinking task *Person1*. Moreover, it has better F-measures than *EAGLE*, *CL* and *WD* in the interlinking task *Person2* before learning 70 assessed links and after learning 90 assessed links. The reason that *KM+WD* achieves better F-measure than the other three works in both data sets is that K-medoids discovers more attribute correspondences and less attribute pairs that are not corresponding than the correspondence discovering method in

[Ngonga Ngomo 2011b]. Consequently, the interlinking pattern of *KM+WD* covers more correct link and filters more incorrect links than the ones of *EAGLE*, *CL* and *WD*.

Note that there are about 10,446 sample links and 6845 sample links for the two data sets respectively while there are 60 sample links and 80 sample links used by Extended Version Space for constructing and improving the interlinking pattern (only 1% of the total sample links), which means a quite high interlinking efficiency.



(a) Person1 (b) Person2

Figure 8.23: Running Times of Extended Version Space and Genetic Programming on *Person1* and *Person2*

Figure 8.23 shows the comparisons of the running time on two interlinking tasks *Person1* and *Person2*.

Extended Version Space reaches high F-measures with shorter running time than other related works on both interlinking tasks. As for the interlinking task *Person1*, Extended Version Space spends shorter time than other works when there are more than 10 assessed link. The interlinking procedure stops after learning 60 assessed links, because there is no more binary similarity sequence groups that have not been learned. There are totally 57 binary similarity sequence groups in the interlinking task *Person1*. The running time of Extended Version Space stays at 78 seconds with a higher F-measure 0.97 by referring to Figure 8.22 (a). Comparing to other interlinking methods, Extended Version Space reaches higher F-measures with a relatively shorter time. As for the data set *Person2*, the running time of Extended Version Space is shorter than the ones of other related works. The interlinking procedure stops after learning 80 assessed links, because there is no more binary similarity sequence groups that have not been learned. There are totally 71 binary similarity sequence groups in the interlinking task *Person2*. The reason for the efficiency of Extended Version Space is that it maintains a more concise and precise interlinking pattern that requires less I/O operations for interlinking data sets, so it can reach higher F-measures with shorter running time in both interlinking tasks.

We also can observe that Extended Version Space spends less time on the interlinking task *Person1* than the interlinking task *Person2*. Because there are fewer binary similarity sequence groups of the interlinking task *Person1* than the ones of the interlinking task *Person2*. Since the interlinking pattern is composed of different binary similarity sequences of assessed links, the size of the interlinking pattern is decided by the number of binary similarity sequence groups of correct links. In *Person1*, all correct links are transferred into 17 binary similarity sequence groups. While in *Person2*, all correct links are transferred into 46 binary similarity sequence groups. Thus, the interlinking pattern of *Person2* will definitely become larger than the one of *Person1* when more assessed links are learned. When the interlinking pattern is larger, there are more I/O operations that are required by a Silk script when generating links according to the interlinking pattern. Therefore, the running time of *Person2* becomes longer than the one of *Person1* when more assessed links are learned.

As for the running time of *KM+WD*, it is shorter than the ones of the works that use Genetic Programm. The reason is that the initial population of *KM+WD* is only 20. But the ones of *EAGLE*, *CL* and *WD* are 100, so more interlinking patterns should be evaluated during each learning round of *EAGLE*, *CL* and *WD* than *KM+WD*. Consequently, the running time of *EAGLE*, *CL* and *WD* are longer than the one of *KM+WD*. Comparing to Extended Version Space, *KM+WD* should evaluate several interlinking patterns in each learning round. While Extended Version Space only evaluates one interlinking pattern in each learning round. Therefore, the running time of Extended Version Space is shorter than the one of *KM+WD* in both interlinking tasks.

To conclude, Extended Version Space performs better than other related works because of the following reasons. Genetic Programming searches for the suitable interlinking pattern by evaluating more than one interlinking pattern during each learning round. The evaluation is realized by executing each interlinking pattern to generate a link set, which will increase the running time by requiring many I/O operations. Furthermore, crossover and mutation operations of Genetic Programming easily cause the instability of the interlinking pattern, which may make the F-measure decrease in the meanwhile. In contrast, Extended Version Space does not need to evaluate more than one interlinking pattern during each learning round. It does not change the interlinking pattern with crossover and mutation operations but with informative assessed links. Furthermore, the K-medoids clustering step of Extended Version Space discovers more attribute correspondences than the correspondence discovering method [Ngonga Ngomo 2011b] of *EAGLE*, *CL* and *WD*, which also helps increase the F-measure and reduce the running time of Extended Version Space. Therefore, it reaches high F-measure with shorter time than other related works.

8.4 Conclusion

This chapter evaluates the interlinking method of this thesis.

First, the chapter shows the clustering effect of K-medoids. K-medoids can cluster attributes of each class efficiently, so that a lot of attribute pairs that are not corresponding are not considered when constructing the interlinking pattern.

Second, the chapter presents the coverage probabilities of attribute correspondences for each interlinking task so as to show the necessity of applying a disjunctive pattern for most of interlinking tasks.

Third, the F-measure and running time of Extended Version Space are evaluated when different thresholds are used to create binary similarity sequence of sample links.

Fourth, Extended Version Space is compared with Disjunctive Version Space on various data sets. It shows that Extended Version Space can converge to high F-measures faster than Disjunctive Version Space. It also shows that Extended Version Space requires less time to interlink two data sets than Disjunctive Version Space, because the big size of the generalized pattern of Disjunctive Version Space will lead to many I/O operations by Silk script when generating links. In one sentence, Extended Version Space generates a concise disjunctive pattern, which can cover positive links and filter negative links more comprehensively with shorter running time than Disjunctive Version Space.

Fifth, Extended Version Space is compared with related works that use Genetic Programming for interlinking. Extended Version Space converges to high F-measures faster than these related works. The reason is that Genetic Programming searches for the interlinking pattern by evaluating more than one interlinking pattern during each learning round, which requires longer running time. Moreover, crossover and mutation operations of Genetic Programming easily cause instability of the interlinking pattern, which may lead to the decrease of F-measures. In contrast, Extended Version Space does not need to evaluate more than one interlinking pattern during each learning round. It does not change the interlinking pattern with crossover and mutation operations but with informative assessed links. Therefore, it reaches a high F-measure with shorter running time than other related works when the interlinking process finishes.

Sixth, we also implement a Genetic Programming-based interlinking method. It uses K-medoids to discover attribute correspondences between two corresponding classes. It also uses the Active Learning strategy that is applied in the related work *WD* to select links to be assessed and learned. The experiments show that K-medoids effectively finds out more attribute correspondences than the correspondence discovering method that is applied in *EAGLE*, *CL* and *WD*.

The next chapter will conclude the whole thesis.

Chapter 9
Conclusion

This thesis presents an interlinking method that generates links for two RDF data sets. It is made up of two steps. One step is to discover attribute correspondences by analyzing the statistical features of attribute values and clustering attributes of each class into groups with a K-medoids clustering. The other step is to build an interlinking pattern iteratively with Extended Version Space. These two steps effectively discover potential attribute correspondences and then build an interlinking pattern with these discovered potential attribute correspondences and sample links assessed by users. The thesis proves that the constructed interlinking pattern can distinguish correct and incorrect links efficiently. Moreover, the thesis proves that the generated interlinking pattern possesses a concise representation format. The key findings are summarized below.

- The clustering step helps reduce the number of attribute pairs that are not corresponding when building the interlinking pattern. It in turns reduce the computational complexity of Extended Version Space, because the computational complexity of Extended Version Space is influenced by the number of potential attribute correspondences. Yet, there are some attribute pairs that are not corresponding being discovered as attribute correspondences. Thus, the precision of potential attribute correspondences should be improved further.
- Extended Version Space builds the interlinking pattern more comprehensively than other related works. With only 1% of sample links, Extended Version Space effectively generates interlinking patterns for the interlinking tasks that do not have a conjunctive interlinking pattern. If the number of sample links is big, it may require some assessment work for users. By comparing Extended Version Space to Disjunctive Version Space and other recent interlinking works, experiments show Extended Version Space converges to a higher F-measure (at least 0.9) with shorter running time than other related works.

In the future, I plan to do the following works.

- First, discover more value features that can be used to distinguish attributes. This work is to reduce further the attribute pairs that are not corresponding and are recognized as potential attribute correspondences by the clustering step. According to the experiments, although the clustering step reduces a lot of attribute pairs that are not corresponding, there are still some attribute pairs that are incorrectly recognized as attribute correspondences. Thus, various value features need to be applied in the clustering step to distinguish attributes more precisely so that the number of attribute pairs that are not corresponding can be further reduced.

- Second, reduce the number of sample links that are sent to users to assess and optimize the whole interlinking process to be more automatic. Although only 1% links are required to improve the interlinking pattern, the number of links to be assessed may be large if the number of sample links is big. Some advanced Active Learning methods can be applied here to reduce the number of sample links to be assessed.

- Third, make the interlinking method of this thesis run on larger web data sets robustly and efficiently. An efficient interlinking method should be able to find out links for various data sets. Thus, more interlinking tests on larger web data sets will be executed to improve the interlinking method of this thesis.

- Fourth, implement the interlinking method in some ontology matchers so as to improve the results of ontology matchers. Interlinking data sets and Ontology Matching are two topics that can help improve the results of each other. Thus, I plan to implement the interlinking method in this thesis in an ontology matcher to improve the results of the Ontology Matcher.

Chapter 10

Appendix on Implementation

10.1 K-medoids

This clustering method is implemented as a java function called *AttriClustering*.

10.2 Transferring EDOAL Correspondence into Silk Script

These translations have been implemented as an alignment renderer following the `AlignmentVisitor` class of the Alignment API. The API provides the notion of a visitor of the alignment cells. These visitors are used in the implementation for rendering the alignments by traversing inductively EDOAL expressions.

The implementation follows the presentation of Chapter 6: a visitor is able to generate graph patterns which may be used in a variety of contexts, such as generating SPARQL queries, and further generate specific uses of the graph patterns. Some visitors implemented are listed below:

- The *EDOALRendererVisitor* class traverses EDOAL expressions and translate each EDOAL correspondence into graph patterns. It implements the T functions (see Section 6.2.1) of Chapter 6.
- The *SILKRendererVisitor* class finds out attribute correspondences across two corresponding classes by extending the *EDOALRendererVisitor* class and generates Silk script for interlinking the same instances across two corresponding classes in data sets (see Section 6.2.2, Section 6.2.3 and Section 6.2.4).

10.3 Extended Version Space

This Machine Learning method is implemented as a java function called *updateVersionSpace*.

Bibliography

[Ahuja 1993] Ravindra K. Ahuja, Thomas L. Magnanti and James B. Orlin. Network flows: Theory, algorithms, and applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. (Cited on page 26.)

[Alajlan 2014] Naif Alajlan, Edoardo Pasolli, Farid Melgani and Andrea Franzoso. *Large-Scale Image Classification Using Active Learning*. IEEE Geoscience and Remote Sensing Letters, vol. 11, no. 1, pages 259–263, 2014. (Cited on page 32.)

[Ananthakrishna 2002] Rohit Ananthakrishna, Surajit Chaudhuri and Venkatesh Ganti. *Eliminating Fuzzy Duplicates in Data Warehouses*. In Proceedings of the 28th International Conference on Very Large Databases (VLDB 2002), 2002. (Cited on page 18.)

[Antoniou 2008] Grigoris Antoniou and Frank van Harmelen. A semantic web primer, 2nd edition (cooperative information systems). The MIT Press, 2008. (Cited on page 5.)

[Atencia 2012] Manuel Atencia, Jérôme David and François Scharffe. *Keys and pseudo-keys detection for web datasets cleansing and interlinking*. In Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW), pages 144–153, Galway, Ireland, 2012. (Cited on pages 18, 19 and 20.)

[Auer 2009] Sören Auer, Jens Lehmann and Sebastian Hellmann. *LinkedGeoData: Adding a Spatial Dimension to the Web of Data*. In Proceedings of the 8th International Semantic Web Conference (ISWC'09), pages 731–746, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 17.)

[Baeza-Yates 1999] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. Modern information retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. (Cited on page 21.)

[Banzhaf 1998] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller and Peter Nordin. Genetic programming – an introduction; on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. (Cited on page 31.)

[Benjelloun 2009] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang and Jennifer Widom. *Swoosh: a generic approach to entity resolution*. The LDB Journal, vol. 18, no. 1, pages 255–276, January 2009. (Cited on page 18.)

[Berlin 2002] Jacob Berlin and Amihai Motro. *Database Schema Matching Using Machine Learning with Feature Selection.* In Proceedings of the Conference on Advanced Information Systems Engineering (CAiSE), pages 452–466, 2002. (Cited on pages 1, 25, 26, 29 and 30.)

[Bhatia 2010] Nitin Bhatia and Vandana. *Survey of Nearest Neighbor Techniques.* CoRR, vol. abs/1007.0085, 2010. (Cited on page 12.)

[Bilke 2005] Alexander Bilke and Felix Naumann. *Schema Matching using Duplicates.* In Proceedings of the 21st International Conference on Data Engineering, pages 69–80. IEEE Computer Society, 2005. (Cited on pages 1, 25, 27, 29 and 30.)

[Brabham 2008] Daren C. Brabham. *Crowdsourcing as a Model for Problem Solving: An Introduction and Cases.* Convergence: The International Journal of Research into New Media Technologies, vol. 14, no. 1, pages 75–90, 2008. (Cited on page 23.)

[Brain 2003] Damien Brain. *Learning from Large Data: Bias, Variance, Sampling, and Learning Curves.* PhD thesis, Deakin University, 2003. (Cited on page 32.)

[Broder 2014] Andrei Broder, Lluis Garcia-Pueyo, Vanja Josifovski, Sergei Vassilvitskii and Srihari Venkatesan. *Scalable K-Means by Ranked Retrieval.* In Proceedings of 7th ACM International Conference on Web Search and Data Mining, WSDM'14, pages 233–242, New York, NY, USA, 2014. ACM. (Cited on page 28.)

[Burges 1998] Chris Burges. *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery, vol. 2, no. 2, pages 121–167, 1998. (Cited on page 12.)

[Cao 2011] Yunbo Cao, Zhiyuan Chen, Jiamin Zhu, Pei Yue, Chin-Yew Lin and Yong Yu. *Leveraging unlabeled data to scale blocking for record linkage.* In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11) - Volume Volume Three, pages 2211–2217. AAAI Press, 2011. (Cited on page 18.)

[Chapelle 2006] Olivier Chapelle, Bernhard Schölkopf and Alexander Zien. Semi-supervised learning. MIT Press, 2006. (Cited on page 11.)

[Chow 2007] Tommy W. S. Chow. Neural networks and computing: Learning algorithms and applications. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007. (Cited on page 12.)

[Cohen 1998] William W. Cohen. *Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity.* In Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98), pages 201–212, New York, NY, USA, 1998. ACM. (Cited on page 27.)

[Cover 1991] Thomas M. Cover and Joy A. Thomas. Elements of information theory. Wiley-Interscience, New York, NY, USA, 1991. (Cited on page 26.)

[Cristianini 2000] Nello Cristianini and John Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, New York, NY, USA, 2000. (Cited on page 12.)

[Cudré-Mauroux 2009] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer and Hermann de Meer. *idMesh: Graph-based Disambiguation of Linked Data.* In Proceedings of the 18th International World Wide Web Conference (WWW), pages 591–600, Madrid, Spain, 2009. ACM. (Cited on pages 22 and 23.)

[Dagan 1995] Ido Dagan and Sean P. Engelson. *Committee-Based Sampling For Training Probabilistic Classifiers.* In Proceedings of the 12th International Conference on Machine Learning, pages 150–157. Morgan Kaufmann, 1995. (Cited on page 32.)

[David 2011] Jérôme David, Jérôme Euzenat, François Scharffe and Cássia Trojahn dos Santos. *The Alignment API 4.0.* Semantic Web Journal, vol. 2, no. 1, pages 3–10, 2011. (Cited on pages 41, 56 and 68.)

[Demartini 2012] Gianluca Demartini, Djellel Eddine Difallah and Philippe Cudré-Mauroux. *ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking.* In Proceedings of the 21st International Conference on World Wide Web (WWW'12), pages 469–478, New York, NY, USA, 2012. ACM. (Cited on pages 22 and 23.)

[Dempster 1977] Arthur P. Dempster, Nan M. Laird and Donald B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm.* Journal of the Royal Statistical Society, Series B (Methodological), vol. 39, no. 1, pages 1–38, 1977. (Cited on page 23.)

[Deutsch 2008] Alin Deutsch, Alan Nash and Jeffrey B. Remmel. *The chase revisited.* In Maurizio Lenzerini and Domenico Lembo, editeurs, Symposium on Principles of Database Systems (PODS), pages 149–158. ACM, 2008. (Cited on page 28.)

[Dhamankar 2004] Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy and Pedro Domingos. *iMAP: discovering complex semantic matches between database schemas.* In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pages 383–394. ACM Press, 2004. (Cited on pages 1, 25, 29 and 30.)

[Di 2013] Sheng Di, Derrick Kondo and Cappello Franck. *Characterizing Cloud Applications on a Google Data Center.* In 42nd International Conference on Parallel Processing (ICPP2013), Lyon, France, Oct 2013. (Cited on page 28.)

[Doan 2004] Anhai Doan, Jayant Madhavan, Pedro Domingos and Alon Halevy. *Ontology Matching: A Machine Learning Approach.* In Steffen Staab and Rudi Studer, editeurs, Handbook on Ontologies, pages 385–404. Springer Verlag, Berlin (DE), 2004. (Cited on page 25.)

[Dong 2005] Xin Dong, Alon Halevy and Jayant Madhavan. *Reference reconciliation in complex information spaces.* In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 85–96, New York, NY, USA, 2005. ACM. (Cited on page 28.)

[Druck 2011] Gregory Druck. *Generalized Expectation Criteria for Lightly Supervised Learning.* PhD thesis, University of Massachusetts Amherst, September 2011. (Cited on page 32.)

[Duan 2012] Songyun Duan, Achille Fokoue, Oktie Hassanzadeh, Anastasios Kementsietsidis, Kavitha Srinivas and Michael J. Ward. *Instance-Based Matching of Large Ontologies Using Locality-Sensitive Hashing.* In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein and Eva Blomqvist, editeurs, Proceedings of International Semantic Web Conference (1), volume 7649 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2012. (Cited on page 27.)

[Ebert 2012] Sandra Ebert, Mario Fritz and Bernt Schiele. *RALF: A reinforced active learning formulation for object class recognition.* In Proceedings of 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 0, pages 3626–3633, Los Alamitos, CA, USA, 2012. IEEE Computer Society. (Cited on page 32.)

[Euzenat 1994] Jérôme Euzenat. *Brief overview of T-tree: the Tropes Taxonomy building Tool.* In Philip Smith, Clare Beghtol, Raya Fidel and Barbara Kwasnik, editeurs, Proceedings of 4th ASIS SIG/CR Workshop on Classification Research, pages 69–87, New York, NY, USA, 1994. Advances in classification research 4, Information today, Medford. (Cited on page 25.)

[Euzenat 2007] Jérôme Euzenat and Pavel Shvaiko. Ontology matching. Springer-Verlag, Heidelberg (DE), 2007. (Cited on pages 1, 24, 25 and 27.)

[Euzenat 2008] Jérôme Euzenat, François Scharffe and Axel Polleres. *Processing Ontology Alignments with SPARQL (Position paper).* In Proceedings of IEEE International Workshop on Ontology Alignment and Visualization (OAaV), pages 913–917, Barcelona, Spain, 2008. (Cited on page 73.)

[Fan 2012] Zhengjie Fan. *Data linking with ontology alignment.* In Proceedings of 9th European semantic web conference (ESWC), pages 854–858, Heraklion (GR), 2012. (Cited on page 24.)

[Fausett 2013] Adam Fausett and M. Emre Celebi. *An Accelerated Nearest Neighbor Search Method for the K-Means Clustering Algorithm.* In Chutima Boonthum-Denecke and G. Michael Youngblood, editeurs, FLAIRS Conference, St. Pete Beach, Florida, 2013. AAAI Press. (Cited on page 28.)

[Forgy 1965] E. Forgy. *Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classification.* Biometrics, vol. 21, no. 3, pages 768–769, 1965. (Cited on page 53.)

[Fu 2013] Yifan Fu, Xingquan Zhu and Bin Li. *A survey on instance selection for active learning.* Knowledge and Information Systems, vol. 35, no. 2, pages 249–283, 2013. (Cited on page 32.)

[Ghahramani 2004] Zoubin Ghahramani. *Unsupervised learning.* In Advanced Lectures on Machine Learning, pages 72–112. Springer-Verlag, 2004. (Cited on page 11.)

[Gosavi 2009] Abhijit Gosavi. *Reinforcement Learning: A Tutorial Survey and Recent Advances.* INFORMS Journal on Computing, vol. 21, no. 2, pages 178–192, April 2009. (Cited on page 12.)

[Gravano 2003] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan and Divesh Srivastava. *Approximate String Joins in a Database (Almost) for Free - Erratum.* In Proceedings of the 29th International Conference on Very Large Databases (VLDB), pages 491–500, 2003. (Cited on page 18.)

[Guha 2004] Sudipto Guha, Nick Koudas, Amit Marathe and Divesh Srivastava. *Merging the Results of Approximate Match Operations.* In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, René J. Miller, José A. Blakeley and K. Bernhard Schiefer, editeurs, Proceedings of the 30th International Conference on Very Large Databases (VLDB), pages 636–647. Morgan Kaufmann, 2004. (Cited on page 18.)

[Guyon 2003] Isabelle Guyon and André Elisseeff. *An Introduction to Variable and Feature Selection.* Journal of Machine Learning Research, vol. 3, pages 1157–1182, March 2003. (Cited on page 26.)

[Guyon 2012] Isabelle Guyon, Gavin Cawley and Gideon Dror. Active learning challenge: Challenges in machine learning, volumn 6. Microtome Publishing, River Edge, NJ, USA, 2012. (Cited on page 32.)

[Hamerly 2002] Greg Hamerly and Charles Elkan. *Alternatives to the K-means algorithm that find better clusterings.* In Proceedings of the 8th International

Conference on Information and Knowledge Management (CIKM'02), pages 600–607, NY, USA, 2002. ACM. (Cited on page 53.)

[Hassanzadeh 2009] Oktie Hassanzadeh and Mariano Consens. *Linked Movie Data Base*. In Proceedings of the WWW 2009 Workshop on Linked Data on the Web (LDOW 2009), April 2009. (Cited on page 17.)

[Hastie 2001] Trevor Hastie, Robert Tibshirani and Jerome Friedman. The elements of statistical learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. (Cited on page 2.)

[Haykin 1998] Simon Haykin. Neural networks: A comprehensive foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd édition, 1998. (Cited on page 12.)

[Hinton 1999] Geoffrey E. Hinton and Terrence Joseph Sejnowski. Unsupervised learning: Foundations of neural computation. A Bradford Book. MIT Press, 1999. (Cited on page 11.)

[Hogan 2010] Aidan Hogan, Axel Polleres, Jürgen Umbrich and Antoine Zimmermann. *Some entities are more equal than others: statistical methods to consolidate Linked Data*. In Proceedings of the Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS 2010), Heraklion, Greece, 2010. (Cited on pages 18 and 23.)

[Hu 2011] Wei Hu, Jianfeng Chen and Yuzhong Qu. *A self-training approach for resolving object coreference on the semantic web*. In Proceedings of the 20th International World Wide Web Conference (WWW), 2011. (Cited on pages 18, 19, 23, 24, 29, 30 and 31.)

[Ichise 2003] Ryutaro Ichise, Hiedeaki Takeda and Shinichi Honiden. *Integrating Multiple Internet Directories by Instance-based Learning*. In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), pages 22–28, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. (Cited on page 25.)

[Ichise 2004] Ryutaro Ichise, Masahiro Hamasaki and Hideaki Takeda. *Discovering Relationships Among Catalogs*. In Einoshin Suzuki and Setsuo Arikawa, editeurs, Discovery Science, volume 3245 of *Lecture Notes in Computer Science*, pages 371–379. Springer, 2004. (Cited on page 25.)

[Isele 2011a] Robert Isele and Christian Bizer. *Learning linkage rules using genetic programming*. In Proceedings of the 6th Ontology Matching Workshop at the International Semantic Web Conference (OM@ISWC), volume 814 of *CEUR Workshop Proceedings*, Bonn, Germany, 2011. (Cited on pages 31, 32 and 33.)

[Isele 2011b] Robert Isele, Anja Jentzsch and Christian Bizer. *Efficient Multidimensional Blocking for Link Discovery without Losing Recall.* In 14th International Workshop on the Web and Databases (WebDB 2011), Athens, 2011. (Cited on page 34.)

[Isele 2013] Robert Isele and Christian Bizer. *Active learning of expressive linkage rules using genetic programming.* Web Semantics: Science, Services and Agents on the World Wide Web, vol. 23, no. 0, 2013. (Cited on pages 25, 29, 30, 31, 32, 33, 34 and 43.)

[Jaccard 1912] Paul Jaccard. *The Distribution of the Flora in the Alpine Zone.* New Phytologist, vol. 11, no. 2, pages 37–50, February 1912. (Cited on page 94.)

[Jain 1999] A. K. Jain, M. N. Murty and P. J. Flynn. *Data Clustering: A Review.* ACM Computing Surveys, vol. 31, no. 3, pages 264–323, September 1999. (Cited on page 28.)

[Jentzsch 2012] Anja Jentzsch. *Link Specification Language*, 2012. (Cited on page 70.)

[Jiang 2007] Liangxiao Jiang, Zhihua Cai, Dianhong Wang and Siwei Jiang. *Survey of Improving K-Nearest-Neighbor for Classification.* In J. Lei, editeur, Fuzzy Systems and Knowledge Discovery, pages 679–683. IEEE Computer Society, 2007. (Cited on page 12.)

[John 1996] George John and Pat Langley. *Static Versus Dynamic Sampling for Data Mining.* In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pages 367–370. AAAI Press, 1996. (Cited on page 32.)

[Kaelbling 1996] Leslie Pack Kaelbling, Michael L. Littman and Andrew W. Moore. *Reinforcement learning: a survey.* Journal of Artificial Intelligence Research, vol. 4, pages 237–285, 1996. (Cited on page 12.)

[Kang 2003] Jaewoo Kang and Jeffrey F. Naughton. *On Schema Matching with Opaque Column Names and Data Values.* In Proceedings of the 22nd International Conference on Management of Data (SIGMOD), pages 205–216. ACM Press, 2003. (Cited on pages 1, 25, 26, 29 and 30.)

[Kaufman 1987] Leonard Kaufman and Peter Rousseeuw. Clustering by means of medoids. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. Fac., Univ., 1987. (Cited on pages 2 and 50.)

[Kotsiantis 2007] Sotiris B. Kotsiantis. *Supervised Machine Learning: A Review of Classification Techniques.* In Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and

Pervasive Technologies, pages 3–24, Amsterdam, The Netherlands, 2007. IOS Press. (Cited on page 11.)

[Lacher 2001] Martin S. Lacher and Technische Universität München. *Facilitating the Exchange of Explicit Knowledge Through Ontology Mappings.* In Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference, pages 305–309. AAAI Press, 2001. (Cited on page 25.)

[Levenshtein 1966] Vladimir I. Levenshtein. *Binary Codes Capable of Correcting Deletions, Insertions and Reversals.* Soviet Physics Doklady, vol. 10, no. 8, pages 707–710, 1966. (Cited on page 94.)

[Lewis 1994] David D. Lewis and Jason Catlett. *Heterogeneous Uncertainty Sampling for Supervised Learning.* In Proceedings of the 11th International Conference on Machine Learning, pages 148–156. Morgan Kaufmann, 1994. (Cited on page 32.)

[Lu 2004] Yi Lu, Shiyong Lu, Farshad Fotouhi, Youping Deng and Susan J. Brown. *Incremental genetic K-means algorithm and its application in gene expression data analysis.* BMC Bioinformatics, vol. 5, no. 172, 2004. (Cited on page 28.)

[Lu 2012] Bin Lu and Fangyuan Ju. *An optimized genetic K-means clustering algorithm.* In Proceedings of 2012 International Conference on Computer Science and Information Processing (CSIP), pages 1296–1299, 2012. (Cited on page 28.)

[Marshall 1992] Andrew D. Marshall and Ralph R. Martin. Computer vision, models, and inspection. Robotics and Automated Systems, Vol. 4. World Scientific, 1992. (Cited on page 23.)

[McCallum 2000] Andrew McCallum, Kamal Nigam and Lyle H. Ungar. *Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching.* In Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00), pages 169–178, New York, NY, USA, 2000. ACM. (Cited on page 28.)

[Mitchell 1982] Tom M. Mitchell. *Generalization as Search.* Artificial Intelligence, vol. 18, no. 2, pages 203–226, 1982. (Cited on pages 2, 12, 93, 94 and 96.)

[Mitchell 1997] Tom M. Mitchell. Machine learning. McGraw-Hill, New York, 1997. (Cited on pages 2, 12, 93, 97 and 113.)

[Murthy 1997] Sreerama K. Murthy. *Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey.* Data Mining and Knowledge Discovery, vol. 2, pages 345–389, 1997. (Cited on page 12.)

[Ngo 2012] Duy Hoa Ngo and Zohra Bellahsene. *YAM++ - Results for OAEI 2012.* In Proceedings of International Semantic Web Conference, United States, 2012. (Cited on page 46.)

[Ngonga Ngomo 2011a] Axel-Cyrille Ngonga Ngomo. *A time-efficient hybrid approach to link discovery.* In Proceedings of the 6th Ontology Matching Workshop (OM), volume 814 of *CEUR Workshop Proceedings*, Bonn, Germany, 2011. (Cited on page 31.)

[Ngonga Ngomo 2011b] Axel-Cyrille Ngonga Ngomo, Norman Heino, Klaus Lyko, René Speck and Martin Kaltenböck. *SCMS - Semantifying Content Management Systems.* In Proceedings of Internationl Semantic Web Conference (ISWC 2011), 2011. (Cited on pages 137, 139 and 140.)

[Ngonga Ngomo 2011c] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer and Konrad Höffner. *RAVEN – Active Learning of Link Specifications.* In Pavel Shvaiko, Jérôme Euzenat, Tom Heath, Christoph Quix, Ming Mao and Isabel F. Cruz, editeurs, Proceedings of the 6th International Workshop on Ontology Matching, volume 814 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011. (Cited on pages 2, 25, 29, 31, 32, 33, 34 and 35.)

[Ngonga Ngomo 2012a] Axel-Cyrille Ngonga Ngomo. *Learning conformation rules for linked data integration.* In Proceedings of the 7th International Workshop on Ontology Matching, 2012. (Cited on page 32.)

[Ngonga Ngomo 2012b] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. *EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming.* In Elena Simperl, Philipp Cimiano, Axel Polleres, Óscar Corcho and Valentina Presutti, editeurs, Proceedings of the 9th Extended Semantic Web Conference, volume 7295 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2012. (Cited on pages 31, 32, 33, 35, 43, 113 and 136.)

[Ngonga Ngomo 2013] Axel-Cyrille Ngonga Ngomo, Klaus Lyko and Victor Christen. *COALA-correlation-Aware Active Learning of Link Specifications.* In Proceedings of the Extended Semantic Web Conference (ESWC), 2013. (Cited on pages 31, 32, 33, 35, 113, 136 and 137.)

[Nijssen 2003] Siegfried Nijssen and Joost N. Kok. *Efficient Frequent Query Discovery in FARMER.* In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel and Ljupco Todorovski, editeurs, Proceedings of the 7th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), volume 2838 of *Lecture Notes in Computer Science*, pages 350–362. Springer, 2003. (Cited on page 28.)

[Nikolov 2008] Andriy Nikolov, Victoria S. Uren, Enrico Motta and Anne N. De Roeck. *Handling instance coreferencing in the KnoFuss architecture.* In Proceedings of the Workshop: Identity and Reference on the Semantic Web at 5th European Semantic Web Conference (ESWC 2008), volume 422, 2008. (Cited on page 31.)

[Nikolov 2010] Andriy Nikolov, Victoria Uren and Enrico Motta. *Data linking: Capturing and utilising implicit schema-level relations.* In Proceedings of the 3rd Workshop on Linked Data on the Web (LDOW), Raleigh, USA, 2010. (Cited on pages 2, 6, 25, 29, 31 and 34.)

[Nikolov 2012] Andriy Nikolov, Mathieu d'Aquin and Enrico Motta. *Unsupervised Learning of Link Discovery Configuration.* In Proceedings of Extended Semantic Web Conference (ESWC), pages 119–133, 2012. (Cited on page 31.)

[Niu 2012] Xing Niu, Shu Rong, Haofen Wang and Yong Yu. *An effective rule miner for instance matching in a web of data.* In Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM), pages 1085–1094. ACM, 2012. (Cited on pages 18, 23, 25, 29, 30, 31 and 34.)

[Nottelmann 2005] Henrik Nottelmann. *sPLMap: A probabilistic approach to schema matching.* In Proceedings of the 27th European Conference on Information Retrieval, pages 81–95. Springer Verlag, 2005. (Cited on pages 1, 25, 27, 29 and 30.)

[Nottelmann 2006] Henrik Nottelmann and Umberto Straccia. *A Probabilistic, Logic-based Framework for Automated Web Directory Alignment.* In Zongmin Ma, editeur, Soft Computing in Ontologies and the Semantic Web, volume 204 of *Studies in Fuzziness and Soft Computing*, pages 47–77. Springer Verlag, 2006. (Cited on page 25.)

[Okabe 2000] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara and Sung Nok Chiu. Spatial tessellations: Concepts and applications of Voronoi diagrams. Probability and Statistics. Wiley, NY, USA, 2nd édition, 2000. (Cited on page 50.)

[Oyelade 2010] O. J. Oyelade, O. O. Oladipupo and I. C. Obagbuwa. *Application of K-Means Clustering Algorithm for Prediction of Students Academic Performance.* CoRR, vol. abs/1002.2425, 2010. (Cited on page 28.)

[Pernelle 2013] Nathalie Pernelle, Fatiha Saïs and Danai Symeonidou. *An automatic key discovery approach for data linking.* Journal of Web Semantics, vol. 23, pages 16–30, 2013. (Cited on page 19.)

[Piatetsky-Shapiro 1991] G. Piatetsky-Shapiro. *Discovery, analysis and presentation of strong rules.* In G. Piatetsky-Shapiro and W. J. Frawley, editeurs, Knowledge Discovery in Databases, pages 229–248. AAAI Press, 1991. (Cited on page 12.)

[Qin 2007] Han Qin, Dejing Dou and Paea LePendu. *Discovering Executable Semantic Mappings Between Ontologies.* In Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I

(OTM'07), volume 4803, pages 832–849, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited on pages 1, 25, 28, 29 and 30.)

[Raimond 2008] Yves Raimond, Christopher Sutton and Mark Sandler. *Automatic Interlinking of Music Datasets on the Semantic Web.* In Proceedings of WWW Linking Data on the Web Workshop, 2008. (Cited on page 17.)

[Rashidi 2011] Parisa Rashidi and Diane J. Cook. *Ask me better questions: active learning queries based on rule induction.* In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11), pages 904–912, New York, NY, USA, 2011. ACM. (Cited on page 32.)

[Ray 1999] Siddheswar Ray and Rose H. Turi. *Determination of number of clusters in K-means clustering and application in colour image segmentation.* In Proceedings of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques (ICAPRDT'99), pages 137–143, 1999. (Cited on page 28.)

[Rivero 2011] Carlos R. Rivero, Inma Hernández, David Ruiz and Rafael Corchuelo. *Generating SPARQL executable mappings to integrate ontologies.* In Proceedings of the 30th International Conference on Conceptual Modeling (ER'11), pages 118–131, Berlin, Heidelberg, 2011. Springer-Verlag. (Cited on pages 28, 29 and 73.)

[Rong 2012] Shu Rong, Xing Niu, Evan Wei Xiang, Haofen Wang, Qiang Yang and Yong Yu. *A Machine Learning Approach for Instance Matching Based on Similarity Metrics.* In Proceedings of International Semantic Web Conference (1), volume 7649 of *Lecture Notes in Computer Science*, pages 460–475. Springer, 2012. (Cited on page 31.)

[Russell 1996] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik and Douglas D. Edwards. Artificial intelligence: A modern approach. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. (Cited on page 25.)

[Saar-tsechansky 2004] Maytal Saar-tsechansky and Foster Provost. *Active Sampling for Class Probability Estimation and Ranking.* Machine Learning, no. 2, pages 153–178, February 2004. (Cited on page 32.)

[Safavian 1991] S. Rasoul Safavian and David Landgrebe. *A Survey of Decision Tree Classifier Methodology.* IEEE Transactions on Systems, Man and Cybernetics, vol. 21, no. 3, pages 660–674, 1991. (Cited on page 12.)

[Scharffe 2009] François Scharffe. *Correspondence Patterns Representation.* PhD thesis, University of Innsbruck, 2009. (Cited on pages 21, 41, 56 and 68.)

[Scharffe 2012] François Scharffe, Laurent Bihanic, Gabriel Képéklian, Ghislain Atemezing, Raphaël Troncy, Franck Cotton, Fabien Gandon, Serena Villata, Jérôme Euzenat, Zhengjie Fan, Bénédicte Bucher, Fayçal Hamdi, Pierre-Yves Vandenbussche and Bernard Vatant. *Enabling Linked Data Publication with the Datalift Platform*. In Proceedings of AAAI Workshops, 2012. (Cited on pages 4, 19 and 20.)

[Schopman 2012] Balthasar A. C. Schopman, Shenghui Wang, Antoine Isaac and Stefan Schlobach. *Instance-Based Ontology Matching by Instance Enrichment*. Journal on Data Semantics, vol. 1, no. 4, pages 219–236, 2012. (Cited on page 27.)

[Sebag 1996] Michèle Sebag. *Delaying the Choice of Bias: A Disjunctive Version Space Approach*. In Proceedings of the 13th International Conference on Machine Learning, pages 444–452. Morgan Kaufmann, 1996. (Cited on page 97.)

[Settles 2009] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. (Cited on page 32.)

[Seung 1992] H. Sebastian Seung, M. Opper and Haim Sompolinsky. *Query by committee*. In Proceedings of the 5th Annual workshop on Computational Learning Theory (COLT'92), pages 287–294, New York, NY, USA, 1992. ACM. (Cited on page 32.)

[Shen 2005] Warren Shen, Xin Li and AnHai Doan. *Constraint-Based Entity Matching*. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2005. (Cited on pages 23 and 31.)

[Sleeman 2010] Jennifer Sleeman and Tim Finin. *A Machine Learning Approach to Linking FOAF Instances*. In Proceedings of the AAAI Spring Symposium on Linked Data Meets Artificial Intelligence. AAAI Press, January 2010. (Cited on page 17.)

[Song 2011] Dezhao Song and Jeff Heflin. *Automatically Generating Data Linkages Using a Domain-Independent Candidate Selection Approach*. In Proceedings of the 10th International Semantic Web Conference (ISWC), Bonn, Germany, 2011. (Cited on pages 18, 19, 20, 21 and 31.)

[Song 2013] Dezhao Song and Jeff Heflin. *Domain-Independent Entity Coreference for Linking Ontology Instances*. Journal of Data and Information Quality, vol. 4, no. 2, page 7, 2013. (Cited on pages 18, 19, 21 and 35.)

[Sree 2014] Pokkuluri Kiran Sree and Inampudi Ramesh Babu. *Identification of Protein Coding Regions in Genomic DNA Using Unsupervised FMACA Based Pattern Classifier*. CoRR, vol. abs/1401.6484, 2014. (Cited on page 28.)

[Stergiou 1996] Christos Stergiou and Dimitrios Siganos. *Neural Networks*. Rapport technique, Department of Computing - Imperial College London, 1996. (Cited on page 12.)

[Stumme 2001] Gerd Stumme and Alexander Maedche. *FCA-MERGE: Bottom-Up Merging of Ontologies.* In Proceedings of the 17th International Joint Conference on Artificial Intellligence (IJCAI), pages 225–230, 2001. (Cited on page 25.)

[Suchanek 2012] Fabian M. Suchanek, Serge Abiteboul and Pierre Senellart. *PARIS: Probabilistic Alignment of Relations, Instances, and Schema.* Proceedings of the VLDB Endowment, vol. 5, no. 3, pages 157–168, 2012. (Cited on pages 1 and 23.)

[Tejada 2001] Sheila Tejada, Craig A. Knoblock and Steven Minton. *Learning Object Identification Rules for Information Integration.* Information Systems, vol. 26, pages 607–635, 2001. (Cited on page 18.)

[Tejada 2002] Sheila Tejada, Craig A. Knoblock and Steven Minton. *Learning domain-independent string transformation weights for high accuracy object identification.* In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02), pages 350–359, New York, NY, USA, 2002. ACM. (Cited on page 18.)

[Tran 2011] Quang-Vinh Tran, Ryutaro Ichise and Bao-Quoc Ho. *Cluster-based similarity aggregation for ontology matching.* In Pavel Shvaiko, Jérôme Euzenat, Tom Heath, Christoph Quix, Ming Mao and Isabel F. Cruz, editeurs, Proceedings of Internationl Semantic Web Conference Ontology Matching Workshop, volume 814 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011. (Cited on pages 1, 25, 27 and 29.)

[Wang 2005] Xiaoying Wang and Jon M. Garibaldi. *A Comparison of Fuzzy and Non-Fuzzy Clustering Techiques in Cancer Diagnosis.* In Proceedings of the 2nd International Conference in Computational Intelligence in Medicine and Healthcare, BIOPATTERN Conference, Costa da Caparica, Lisbon, Portugal, 2005. (Cited on page 28.)

[Whang 2009] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald and Hector Garcia-Molina. *Entity resolution with iterative blocking.* In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (SIGMOD'09), pages 219–232, New York, NY, USA, 2009. ACM. (Cited on page 18.)

[Winkler 2006] William E. Winkler. *Overview of record linkage and current research directions.* Rapport technique, Bureau of the Census, 2006. (Cited on page 18.)

[Witten 2005] Ian H. Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques. Data Management Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd édition, 2005. (Cited on page 12.)

[Xing 2003] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart Russell. *Distance Metric Learning, With Application To Clustering With Side-Information*. In Advances in Neural Information Processing Systems 15, pages 505–512. MIT Press, 2003. (Cited on page 28.)

[Xu 2005] Rui Xu and Donald Coolidge Wunsch. *Survey of clustering algorithms*. IEEE Transactions on Neural Networks, vol. 16, no. 3, pages 645–678, 2005. (Cited on page 11.)

[Zhu 2005] Xiaojin Zhu. *Semi-Supervised Learning Literature Survey*. Rapport technique 1530, Computer Sciences, University of Wisconsin-Madison, 2005. (Cited on page 11.)