Learning Concise Pattern for Interlinking with Extended Version Space

Zhengjie Fan
INRIA & LIG
655, avenue de l'Europe
Montbonnot Saint Martin
38334 Saint-Ismier, France
Email: zjfanster@gmail.com

Jérôme Euzenat
INRIA & LIG
655, avenue de l'Europe
Montbonnot Saint Martin
38334 Saint-Ismier, France
Email: Jerome.Euzenat@inria.fr

François Scharffe
LIRMM
161 rue Ada
34095 Montpellier Cedex 5, France
Email: francois.scharffe@lirmm.fr

***Abstract*—Many data sets on the web contain analogous data which represent the same resources in the world, so it is helpful to interlink different data sets for sharing information. However, finding correct links is very challenging because there are many instances to compare. In this paper, an interlinking method is proposed to interlink instances across different data sets. The input is class correspondences, property correspondences and a set of sample links that are assessed by users as either "positive" or "negative". We apply a machine learning method, Version Space, in order to construct a classifier, which is called interlinking pattern, that can justify correct links and incorrect links for both data sets. We improve the learning method so that it resolves the no-conjunctive-pattern problem. We call it Extended Version Space. Experiments confirm that our method with only 1% of sample links already reaches a high F-measure (around 0.96-0.99). The F-measure quickly converges, being improved by nearly 10% than other comparable approaches.**

I. Introduction

Nowadays, many organizations and individuals publish RDF data sets on the web[1], which 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, leading to response delay and mistakes.

Many existing solutions are being used for interlinking. (1) One straight-forward idea is to compare the property values of instances for identifying links [1], yet it is impossible to compare all possible pairs of property values. (2) Another common strategy is to compare instances' property values with respect to property correspondences found by instance-based ontology matching [2], [3], [4], [5], [6], [7], [8], which can generate property correspondences based on instances. However, it is hard to identify the same instances across data sets, because there are the same instances whose property values of some property correspondences are not the same. (3) Many existing solutions [9], [10], [11] leverage Genetic Programming to construct interlinking patterns for comparing instances, however they spend long running times for gaining high F-measures.

In order to link instances across two data sets precisely and efficiently, we design an interlinking method, which employs a supervised learning method, called Version Space [12], to generate a classifier for the interlinking task, which can cover all assessed correct links (or positive links) and exclude all assessed incorrect links (or negative links) meanwhile. This classifier is called an interlinking pattern. Assume that we are going to find out the same instances that refer to the same people from two data sets. One data set is in English, the other data set is in French. All pairs of the same instances contain two property correspondences $name \leftrightarrow nom$ and $sex \leftrightarrow gender$. The interlinking pattern can be expressed as a conjunction of these two correspondences. Such a conjunctive formula can help classify pairs of instances into two classes. The pairs of instances that satisfy such a formula will be recognized as a link. The ones that do not satisfy such a formula will not be recognized as a link.

The Version Space method is able to construct an interlinking pattern from a set of assessed links and a set of property correspondences across two corresponding classes as input. Nevertheless, it suffers from a serious limitation. It can only build the interlinking pattern that is represented by a conjunctive pattern, a conjunction of property correspondences that satisfies all positive links. If there is not such a conjunctive pattern, Version Space cannot produce any interlinking pattern, but a null result instead. Disjunctive Version Space is one model designed to overcome such a limitation. It is able to express the cases that cannot be represented by a conjunctive pattern, via a disjunction of conjunctive patterns. When there are several conjunctive patterns each of which satisfies only some but not all positive links, the interlinking pattern can be expressed into a disjunction of conjunctive patterns by Disjunctive Version Space. Whereas, the interlinking pattern it learns is often a long expression, which will significantly increase the interlinking running time. To this end, we propose an Extended Version Space algorithm, which includes (excludes) all positive links (negative links), with a much more concise expression than Disjunctive Version Space, which speeds up interlinking in turn.

We evaluate the proposed interlinking method and other comparable methods using a set of large-scale data sets and make comparisons on the F-measure and running time. Experiments confirm that our interlinking method 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 interlinking approaches.

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

The remainder of the paper is organized as follows. We formulate the interlinking problem in Section II, and introduce related works in Section III. In Section IV and Section V, we describe Version Space and the Extended Version Space algorithm respectively. Section VI presents the experimental results based on different data sets. Finally, we conclude the paper with the future work in Section VII.

II. The Data Interlinking Problem

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

We assume that a set of class correspondences and all property correspondences across each pair of corresponding classes are available for two interlinking data sets. Since an interlinking pattern can be transferred into an interlinking script, which is executed by the semi-automatic interlinking tool Silk[2] for generating links across two RDF data sets, our work will focus on leveraging the given correspondences and a set of sample user-assessed links to build a concise and precise interlinking pattern for the two data sets.

III. Related Work

There are a group of related works that use Machine Learning [13], [14], [15], [10], [11] for learning the interlinking pattern.

Genetic Programming [16] is a supervised learning method that is used for learning interlinking patterns in some interlinking methods [9], [11], [10], [13]. A genetic programming algorithm is proposed in the interlinking method of [9]. The method improves the interlinking patterns by changing the aggregation methods of combining property correspondences, so as to let the patterns be able to classify more assessed links. One of the most recent works on interlinking is proposed by Ngonga Ngomo et al. [10]. The method uses not only Genetic Programming but also Active Learning [17] to construct and improve the interlinking pattern. Active Learning is a learning strategy that actively chooses unlabeled examples which bring the biggest information and change to the classifier [17]. It helps speed up the learning process when there are a large amount of assessed links to be learned. The interlinking approach in this paper outperforms many other solutions like [13] and [9]. The F-measure of generated link set grows higher with the same amount of assessed links than [13] and [9]. Another recent work on interlinking by learning the interlinking pattern is proposed by Isele et al. [11]. 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.

Nevertheless, Genetic Programming is not a method suitable for interlinking at runtime. In each learning round, more than one interlinking pattern should be evaluated. The process of evaluating each interlinking pattern is to interlink the data sets with the interlinking pattern and compute the Precision, Recall and F-measure of the generated link set. When interlinking data sets with each interlinking pattern, the computer should query each data set at least once in order to get instances' property values. The queries require I/O operations of the computer, which take a lot of running times.

[14] is another interlinking work that uses Machine Learning for interlinking. The authors design a boolean classifier to distinguish correct links and incorrect links that are assessed by users. The boolean classifier is a conjunction of several property correspondences. However, there are many interlinking tasks whose links do not share the same set of property correspondences. Therefore, this method cannot interlink the data sets that cannot be classified by such a boolean classifier.

To conclude, a learning method that costs shorter time and is able to generate interlinking patterns for all interlinking tasks is required.

IV. Version Space

Version Space [12], [18] is a supervised learning method that constructs and improves a set of classifiers by learning labeled examples one by one. 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 a set of the most strict compositions of the conditions that cover all positive examples being learned, and they are called *Specialized Hypotheses*. The other is a set of the most general compositions of conditions that cover all cases which exclude the negative examples being learned, 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 learning process of Version Space can build several interlinking patterns (i.e., hypotheses for the interlinking task) which are able to precisely distinguish all assessed links across the data sets. More specifically, given a set of sample links assessed as either *positive* or *negative* by users, it can construct several interlinking patterns with several property correspondences being compatible with all of positive links and conflicting to all of negative links. For each pair of instances i and i' that builds an assessed link, we apply an l-tuple $(B^{(j)})$, $j = 1, \ldots, l$ with a sequence of bit values to denote the similarities between property values of l pairs of corresponding properties. The bit value (denoted by $B^{(j)}$) is defined in Formula (1), where $\equiv$ and $\not\equiv$ refer to similarity and non-similarity between two property values $i.p_j$ and $i'.p'_j$ respectively. Each property comparison is determined by some *similarity metric* (such as "Levenshtein" [19] and "Jaccard" [20]). 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 property correspondences across two corresponding classes, then 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 the two classes.

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

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

For each pair of corresponding properties between two corresponding classes, its similarity value is either 1 if the two property values $i.p_j$ and $i'.p'_j$ on an assessed link are the same based on a similarity metric or 0 otherwise. Then, a ***binary similarity sequence*** (**BSS**) for the property pair is shown in Formula (2), where $PC_1, PC_2, \ldots, PC_l$ stand for l property correspondences, each either being equal to 1 (equal) or 0 (non-equal).

$$\begin{array}{ll} & (PC_1, \quad PC_2, \quad \ldots, PC_l) \\ \text{Binary Similarity Sequence} = & (0|1, \quad 0|1, \quad \ldots, 0|1) \end{array} \quad (2)$$

In general, the computed similarity of two property values may not exactly be equal to 0 or 1, but another decimal number in [0,1], such as 0.75. In our design, we always convert the similarity 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 smaller than T, we assume that the similarity is 0.

The input of the Version Space learning process is a set of binary similarity sequences. **The output of the Version Space learning process** is one or several interlinking patterns that cover all positive links and filter all negative links simultaneously, each of which is a conjunction of property correspondences. In machine learning, the input and output of a supervised learning algorithm are expressed with *instance language* and *generalization language* respectively [12]. With respect to the interlinking problem, 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, called a conjunctive pattern, but each bit of such a binary similarity sequence is 0, 1 or ×, where × = 0|1.

A. Problem of Version Space

In practice, there are many interlinking problem cannot be built an interlinking pattern by Version Space. In other words, it is possible that there is no conjunctive pattern that can cover all positive links and filter all negative links. An example is shown below: there are four assessed links, each having five property correspondences $PC_1, PC_2, PC_3, PC_4, PC_5$.

Link No.	Type	PC_1	PC_2	PC_3	PC_4	PC_5
1	Positive	(0,	1,	1,	0,	0)
2	Positive	(0,	0,	1,	1,	0)
3	Negative	(1,	0,	1,	0,	0)
4	Negative	(0,	0,	1,	0,	0)

(3)

In this example, with the two positive links, there is only one specialized pattern can be produced, which is (0,×,1,×,0). Given a negative link, we can derive a generalized pattern by traversing all binary similarity sequences that are conflicting to the negative link with at least one bit (e.g., property correspondence PC_i). For the above example, based on the third assessed link (a negative link), the candidate generalized patterns are binary similarity sequences: (0,×,×,×,×), (×,1,×,×,×), (×,×,0,×,×), (×,×,×,1,×), (×,×,×,×,1). Then, the generalized pattern with the first three links is supposed to be (0,×,×,×,×), in that only (0,×,×,×,×) contains all positive links that have been learned. When combining the fourth assessed link, however, a conflict arises, because there is no feasible conjunctive pattern that covers all the positive links and filters all the negative links simultaneously.

Disjunctive Version Space [18] adopted disjunctive constructor to solve the above problem. The instance language of Disjunctive Version Space is a binary similarity sequence. 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. The generalized pattern of Disjunctive Version Space of Example 3 is (0,0,0,0,0)∪(0,0,0,0,1)∪(0,0,0,1,0)∪(0,0,0,1,1)∪(0,0,1,0,1)∪(0,0,1,1,0)∪(0,0,1,1,1)∪(0,1,0,0,0)∪(0,1,0,0,1)∪(0,1,0,1,0)∪(0,1,0,1,1)∪(0,1,1,0,0)∪(0,1,1,0,1)∪(0,1,1,1,0)∪(0,1,1,1,1)∪(1,0,0,0,0)∪(1,0,0,0,1)∪(1,0,0,1,0)∪(1,0,0,1,1)∪(1,0,1,0,1)∪(1,0,1,1,0)∪(1,0,1,1,1)∪(1,1,0,0,0)∪(1,1,0,0,1)∪(1,1,0,1,0)∪(1,1,0,1,1)∪(1,1,1,0,0)∪(1,1,1,0,1)∪(1,1,1,1,0)∪(1,1,1,1,1). In contrast, Extended Version Space, 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. Extended Version Space deploys two operations MERGE and RESELECT, which are not used in Disjunctive Version Space, to construct a more concise interlinking pattern.

V. CONSTRUCTION OF AN INTERLINKING PATTERN VIA EXTENDED VERSION SPACE

The instance language of Extended Version Space is a binary similarity sequence. Each bit of the instances' binary similarity sequences is either 0 or 1. The generalization language is a disjunction of binary similarity sequences, called a disjunctive pattern. Each bit of the binary similarity sequences in the generalization language can be 0, 1 or ×.

A disjunctive pattern can be recursively represented as Formula (4), where $(B^{(j)})$ represents a binary similarity sequence with l bits, $j = 1, \ldots, l$.

$$Pattern \ ::= \ (B^{(j)}) \mid Pattern \cup (B^{(j)}) \qquad (4)$$

In this formula, ∪ denotes the union operation (i.e. disjunction) in 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 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 property correspondences (name↔nom, sex↔gender, and supervisor↔directeur) to compare, i.e., three bits per binary similarity sequence. Assume 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 both of the two links can be written as (1,1,0)∪(1,1,1), or in a concise format (1,1,×). A disjunctive pattern defined in Formula (4) can take over the cases Version Space cannot handle. As for Example 3, the disjunctive pattern can be written as (0,1,×,×,×)∪(0,0,1,1,×). This pattern can cover all positive links and filter all negative links, with a concise format. This method 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 interlinking 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 1.

Algorithm 1 EXTENDED VERSION SPACE

```
Input: Binary Similarity Sequences of All Assessed Links
Output: A Disjunctive Generalized Pattern $Pat_G$
 1: $Pat_S$=∅; /*empty set*/
 2: $Pat_G$=$(\beta^{(i)})_{l\times 1}$, where $\beta^{(i)}$=×, $i = 1, 2, \ldots, l$; /*universal set*/
 3: $Pat_{Ori}$=$(\varphi^{(i)})$, where $\varphi^{(i)}$=×, $i = 1, 2, \ldots, l$;
 4: for (each assessed link (denoted by $BSS_{link}$)) do
 5:   if ($BSS_{link}$ is marked as "positive") then
 6:     MERGE($Pat_S$, $BSS_{link}$);
 7:     MERGE($Pat_G$, $BSS_{link}$);
 8:   end if
 9:   if ($BSS_{link}$ is marked as "negative") then
10:     if (∃ BSS, $BSS_{link}$⊆BSS, BSS∈$Pat_{Ori}$) then
11:       $Pat_{Ori}$ = $Pat_{Ori}$−BSS;
12:       for ($BSS_{new}$∈(BSS−$BSS_{link}$)) do
13:         MERGE($Pat_{Ori}$, $BSS_{new}$);
14:       end for
15:     end if
16:     $Pat_G$ = RESELECT($Pat_{Ori}$, $Pat_S$);
17:   end if
18: end for
```

Algorithm 1 aims at generating a disjunctive pattern in the form of Formula (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.

- If the current link is assessed as a **positive** one, the algorithm will check if it is covered by the current patterns (Pat_S and Pat_G). Such a checking step is defined in the MERGE function. 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 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 (×,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_{Ori} stays the same. If yes, we need to delete the BSS_{link} from Pat_{Ori} such that the negative link can be excluded by the new Pat_{Ori}. We should find out the *BSS* that is contained in Pat_{Ori}, which contains BSS_{link}. Then, we need to delete BSS_{link} from *BSS*. We also need to delete *BSS* from Pat_{Ori}. Afterwards, we should add the rest binary similarity sequences that are contained in *BSS* into Pat_{Ori}. Then, Pat_{Ori} should be reconstructed further into a more concise expression (if possible). Finally, a RESELECT step removes 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 with more negative links learned. In general, this is not a must-do step, but 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 remove it from Pat_S.

The key operations appearing in Algorithm 1 are described below.

1) 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. This operation is one key difference between Extended Version Space and Disjunctive Version Space, which makes the interlinking pattern be represented in a more concise format. 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.

$$\begin{aligned} &\text{BSSMERGE}(BSS^i_{Pat}, BSS^j_{Pat}) = (b_1, b_2, \cdots, b_l) \\ &where\ b_k = \text{BITMERGE}(\beta_i^{(k)}, \beta_j^{(k)}) \\ &\qquad = \begin{cases} \beta_i^{(k)}, & \textbf{if } \beta_i^{(k)} = \beta_j^{(k)} \\ \times, & \textbf{if } \beta_i^{(k)} \neq \beta_j^{(k)} \end{cases} \end{aligned} \tag{5}$$

  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. Some examples are given to illustrate it later.

$$\text{MERGE}(Pat, BSS_{link}) = \begin{cases} Pat, \ \textbf{if } \exists BSS^i_{Pat} \in Pat, \ \text{DIFF}(BSS^i_{Pat}, BSS_{link}) = 0 \\ Pat \cup BSS_{link}, \\ \qquad \textbf{if } \forall BSS^i_{Pat} \in Pat, \ \text{DIFF}(BSS^i_{Pat}, BSS_{link}) \geq 2 \\ \text{MERGE}(Pat - BSS^i_{Pat}, \text{BSSMERGE}(BSS^i_{Pat}, BSS_{link})), \\ \qquad \textbf{if } \exists BSS^i_{Pat} \in Pat, \ \text{DIFF}(BSS^i_{Pat}, BSS_{link}) = 1 \\ Pat \cup BSS_{link}, \quad \textbf{if } Pat = \varnothing \end{cases} \quad (6)$$

In Formula (6), the notation "−" means the *difference* operation 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. 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. If the pattern is empty, then the link should be inserted into the pattern.

Here, several examples are given to illustrate the computation of 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 DIFF(BSS_{link},(1,0,1,0,0))=0, the returned new pattern stays unchanged (i.e., the first situation in Formula (6)).
- If BSS_{link} = (1,1,1,0,1), since 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∪BSS_{link} = (0,1,1,×,×)∪(1,0,1,0,0)∪(1,1,1,0,1).
- If BSS_{link} = (1,1,1,0,0), since DIFF((1,0,1,0,0),BSS_{link}) = DIFF((1,0,1,0,0),(1,1,1,0,0)) = 1, the new pattern should be (0,1,1,×,×)∪(1,×,1,0,0).

2) RESELECT *operation:* The RESELECT operation removes from Pat_{Ori} the binary similarity sequences that cannot cover any binary similarity sequence in Pat_S. This operation is the other key difference between Extended Version Space and Disjunctive Version Space, which also makes the interlinking pattern be represented in a more concise format. 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.

$$\text{RESELECT}(Pat_{Ori}, Pat_S) = \\ \{BSS_{Pat_{Ori}} \mid \exists \ BSS_{Pat_S}, BSS_{Pat_S} \subseteq BSS_{Pat_{Ori}}\} \cup \\ \{BSS_{Pat_S} \mid \nexists \ BSS_{Pat_{Ori}}, BSS_{Pat_S} \subseteq BSS_{Pat_{Ori}}\} \quad (7)$$

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 $Pat_S \subseteq Pat_{Ori}$, these binary similarity sequences will degrade the Silk interlinking speed. Hence, it is necessary to maintain the relevant binary similarity sequences in Pat_{Ori} and remove irrelevant 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 the end of Algorithm 1, the converged generalized pattern will act as the final output. The algorithm will terminate when there is no assessed link to be learned. The soundness of the algorithm has been proved [21]. But it is not sure 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 Section VI, in which the algorithm does not find a pattern while there exist one.

VI. EVALUATION

The interlinking method in this paper is evaluated based on public data sets mainly downloaded from IM@OAEI 2010[3] (data sets *Person1* and *Person2*), IM@OAEI 2012[4] (data sets *Sandbox001*) and CKAN[5] (geographical data sets *INSEE* and *EUROSTAT*). Silk is used here to generate a link set across two RDF data sets by executing an interlinking script that is transferred from an interlinking pattern, because it is an open source software.

A. Experimental Setting

A set of sample links are created for constructing and improving the interlinking pattern by Extended Version Space. We use K-medoids clustering method to cluster the properties of every class for each data set based on the statistics (e.g., the average value of properties), and then determine which pairs of properties between two classes across data sets can be regarded as potential property correspondences. The sample links are generated by Silk according to a pattern that combines all property correspondences into a disjunction of property correspondences. It means that any instance pair that have the same property values of any potential property correspondences will be linked as a sample link. These sample links are assessed by users and used to construct and improve an interlinking pattern afterwards.

The F-measure and runtime are computed every 10 assessed links. If the interlinking procedure stops before learning 100 assessed links, we assume that the F-measure and runtime keep on the same level as the one when the procedure stopped, in order to evaluate our interlinking method with other works when the learning process stops with 100 assessed links.

The experiments in Section VI-B1 are performed on each data set 5 times with a single thread and allocated maximally

[3]http://oaei.ontologymatching.org/2010/im/index.html
[4]http://www.instancematching.org/oaei/
[5]http://datahub.io/

2GB of RAM. The experiments in Section VI-B2 are performed on each data set 5 times with 4 threads and allocated maximally 2GB of RAM. Active Learning [22] is applied to select sample links to be assessed by users for reducing the number of assessed links. 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 times 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. The F-measures and running times of Extended Version Space are computed by executing interlinking with interlinking patterns when the assessed links are transferred into binary similarity sequences with the threshold T=0.5. According to our experiment, this threshold can lead to a relatively higher F-measures and lower running times of Extended Version Space than other thresholds.

Our interlinking method is implemented by 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.

B. Evaluation Results

1) Comparison with Genetic Programming: This section compares the F-measure and running time of Extended Version Space with related works [10], [9] that use Genetic Programming for interlinking.

Through Fig. 1 and Fig. 2, we can observe that the interlinking method of this paper performs better than the approaches proposed by Ngonga Ngomo et al. [10], [9], based on the same two data sets *Person1* and *Person2*. In both figures, our method is denoted as EVS. The three comparable methods of Ngonga Ngomo et al. are denoted as *EAGLE*, *CL*, *WD* respectively. *CL* and *WD* are two interlinking methods that combines *EAGLE* with two different Active Learning methods. Both Active Learning methods classify sample links into several groups according to the similarity sequence that are composed of similarities of property values according to each property correspondence. The figures show the best convergence results that are presented in their paper [10]. We compare F-measures and running times of Extended Version Space with the ones of the related works when they achieve their best F-measures with 100 initial population of Genetic Programming. We do not implement *EAGLE*, *CL* and *WD*, because the correspondence discovering method [23] required by the three methods is not a open-source method. The only difference on execution is that Extended Version Space is executed with a CPU of 2.5GHz, while the other three works are executed with a CPU of 2.0GHz. 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 times of both interlinking methods.

Fig. 1 shows the F-measure comparisons on two interlinking tasks *Person1* and *Person2*. With respect to the data set *Person1*, the final converged F-measure under *EVS* is much higher than the best ones of *EAGLE*, *CL* and *WD* (0.99 versus 0.86, 0.88 and 0.89 respectively). The F-measure of *EVS* stays on a high level when there are more than 20 assessed links. For the interlinking task *Person2*, the final converged F-measure of *EVS* is 0.96, while the ones of *EAGLE*, *CL* and *WD* are all roughly 0.77.

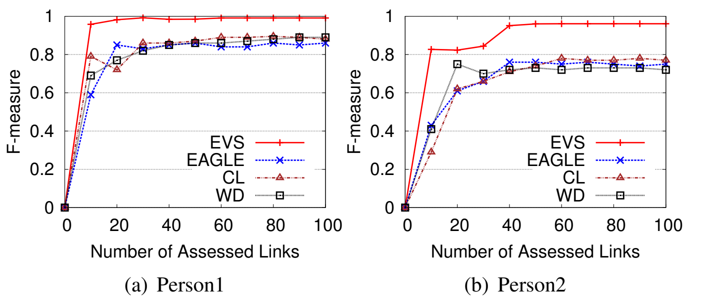


Fig. 1. F-measures of Extended Version Space and Genetic Programming

Extended Version Space achieves better F-measures in the interlinking task *Person1* and *Person2* than *EAGLE*, *CL* and *WD*. We can also observe that Extended Version Space converges faster than *EAGLE*, *CL* and *WD*. Extended Version Space has better F-measures than Ngonga Ngomo et al.'s work by about 10% in both interlinking tasks. This is mainly due to the fact that Extended Version Space builds a disjunctive pattern, which can cover positive links and filter out negative links more comprehensively. However, the operations *mutation* and *crossover* of Genetic Programming make some changes on some randomly-chosen parts of the interlinking pattern. These two operations may cause the interlinking pattern to cover less positive links and filter less negative links, so that F-measures decrease accordingly.

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 improving the interlinking pattern (only 1% of the total sample links), which means a quite high interlinking efficiency.

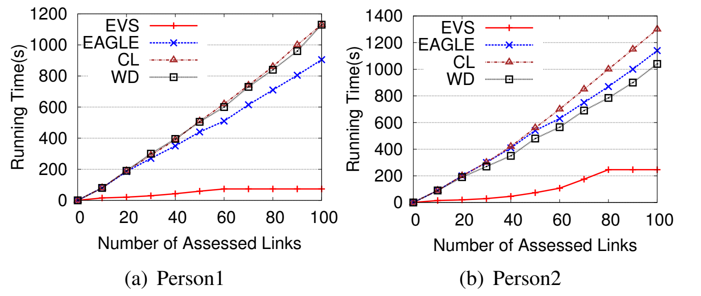


Fig. 2. Running Times of Extended Version Space and Genetic Programming

Fig. 2 shows the comparisons of the running time on two interlinking tasks.

Extended Version Space reaches high F-measures with shorter running times 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 sequences that have not been learned. There are totally 57 different binary similarity sequences in the interlinking task *Person1*. The running time of Extended

Version Space stays at 73 seconds with a higher F-measure 0.99 by referring to Fig. 1 (a). Thus, comparing to other interlinking methods, Extended Version Space reaches higher F-measures with a relatively shorter time in *Person1*. As for data set *Person2*, the running times of Extended Version Space are shorter than other related works after learning 10 assessed links. The interlinking procedure stops after learning 80 assessed links, in that there is no more binary similarity sequences that have not been learned. There are totally 71 different binary similarity sequences in the interlinking task *Person2*. The reason for the efficiency of Extended Version Space is that it executes only one interlinking pattern in each round, but Genetic Programming executes more than one interlinking pattern, which requires more running times. Thus, Extended Version Space can reach high F-measures with shorter time.

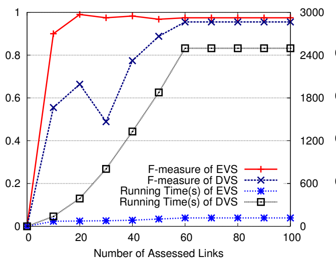
We also can observe that Extended Version Space spends less time on the interlinking task *Person1* than the interlinking task *Person2*. Because there are fewer different binary similarity sequences of the interlinking task *Person1* than the ones of the interlinking task *Person2*. Since the interlinking pattern is composed of different binary similarity sequences of assessed links, the size of the interlinking pattern is decided by the number of different binary similarity sequences. In *Person1*, all correct links are transferred into 17 different binary similarity sequence groups. While in *Person2*, all correct links are transferred into 46 different 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 Silk script when generating links. Therefore, the running times of *Person2* become longer than the ones of *Person1* when more assessed links are learned.

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 patterns during each learning round. The evaluation is realized by executing each interlinking pattern to generate a link set, which will increase a lot of running time with many I/O operations. Furthermore, crossover and mutation operations of Genetic Programming easily cause the instability of the interlinking pattern, which will 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. Therefore, it reaches high F-measure with shorter time than other related works.

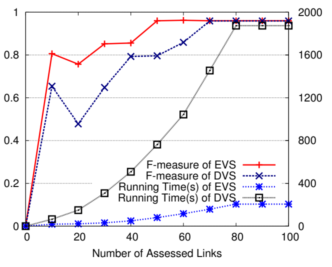
2) Comparison with Disjunctive Version Space: This section compares F-measures and running times of Extended Version Space and the ones of Disjunctive Version Space on different data sets.

The F-measures and running times 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 T=0.5. There is an exception. If there is no negative 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.

We do not generate links with the generalized pattern of Disjunctive Version Space but with the one of Disjunctive Version Space after the operation MERGE of Extended Version Space. The operation is defined in Section V-1. 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 times of the generalized pattern before merging are definitely longer than the ones of the generalized pattern after merging. If we can show that the running times of the generalized pattern after merging are longer than the ones of Extended Version Space, the running times of Extended Version Space will definitely be shorter than the ones of Disjunctive Version Space.



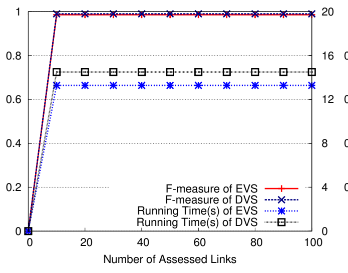
(a) Person1



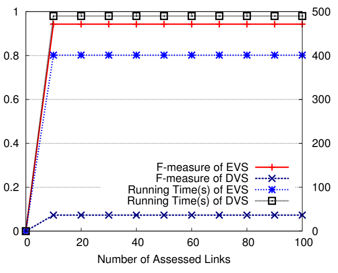
(b) Person2

Fig. 3. F-measure and Running Time of Extended Version Space and Merged Disjunctive Version Space on *Person1* and *Person2*

Extended Version Space converges faster than merged Disjunctive Version Space when interlinking *Person1* and *Person2* in Fig. 3. In Fig. 3(a), the F-measure of Extended Version Space increases to 0.98 when there are 20 assessed links. While merged Disjunctive Version Space should learn 60 assessed links so as to reach the similar level of F-measure. As for running time, Disjunctive Version Space spends 2300 more seconds to reach the similar F-measure as Extended Version Space when there are 60 assessed links. In Fig. 3(b), the 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. As for running time, Disjunctive Version Space spends 1300 more seconds to reach the same F-measure of Extended Version Space when there are 70 assessed links.



(a) INSEE and EUROSTAT



(b) Sandbox001

Fig. 4. F-measure and Running Time of Extended Version Space and Merged Disjunctive Version Space on *INSEE* vs. *EUROSTAT* and *Sandbox001*

Extended Version Space converges faster than merged Disjunctive Version Space when interlinking the data sets *INSEE* and *EUROSTAT* and *Sandbox001* in Fig. 4. In Fig. 4(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. But merged Disjunctive Version Space spends longer running time than Extended Version Space. In Fig. 4(b), the 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. 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 nearly 90 seconds more to improve the interlinking pattern.

To conclude, Extended Version Space converges faster to a higher F-measure with shorter running times than Disjunctive Version Space. Since there is a RESELECT operation in Extended Version Space, only binary similarity sequences that are relevant to assessed correct links are maintained in the generalized pattern of Extended Version Space (i.e., Pat_G). However, the generalized pattern of merged Disjunctive Version Space (i.e., Pat_{Ori}) contains not only relevant binary similarity sequences but also irrelevant ones that cover many incorrect links. Furthermore, the generalized pattern of Extended Version Space is usually more concise than the one of merged Disjunctive Version Space, which requires less I/O operations for querying data sets when generating links. Thus, the running time of Extended Version Space is less than the one of merged Disjunctive Version Space. Since the generalized pattern of Disjunctive Version Space definitely spends longer running time than the one of merged Disjunctive Version Space, Extended Version Space converges to a higher F-measure with shorter running times than Disjunctive Version Space.

VII. Conclusion and Future Work

In this paper, we proposed Extended Version Space to construct an interlinking pattern across two RDF data sets. The advantage of our design is its high F-measure as well as processing efficiency with relatively short running time. We evaluate different solutions using various public data sets. Experiments confirm that our method with only 1% of sample links already reaches a high F-measure (around 0.96-0.99). The F-measure quickly converges in our solution, which improves by nearly 10% on other interlinking approaches. In the future, we plan to evaluate our interlinking method with more data sets, which have different qualities on class/property correspondences. We also plan to implement our interlinking method in an ontology matching tool so as to help improve the generated ontology alignment.

Acknowledgment

This work is partially supported by ANR project Datalift (2010 CORD 009).

References

[1] F. M. Suchanek, S. Abiteboul, and P. Senellart, "Paris: Probabilistic alignment of relations, instances, and schema," *Proceedings of the VLDB Endowment*, vol. 5, no. 3, pp. 157–168, 2012.

[2] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, "imap: discovering complex semantic matches between database schemas," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. ACM Press, 2004, pp. 383–394.

[3] J. Berlin and A. Motro, "Database schema matching using machine learning with feature selection," in *Proceedings of the Conference on Advanced Information Systems Engineering*, 2002, pp. 452–466.

[4] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *Proceedings of 22nd International Conference on Management of Data*. ACM Press, 2003, pp. 205–216.

[5] A. Bilke and F. Naumann, "Schema matching using duplicates," in *Proceedings of the 21st International Conference on Data Engineering*. IEEE Computer Society, 2005, pp. 69–80.

[6] H. Nottelmann, "splmap: A probabilistic approach to schema matching," in *Proceedings of the 27th European Conference on Information Retrieval*. Springer Verlag, 2005, pp. 81–95.

[7] Q.-V. Tran, R. Ichise, and B.-Q. Ho, "Cluster-based similarity aggregation for ontology matching," in *Proceedings of Internationl Semantic Web Conference Ontology Matching workshop*, vol. 814. CEUR-WS.org, 2011.

[8] H. Qin, D. Dou, and P. 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*, vol. 4803, 2007, pp. 832–849.

[9] A.-C. Ngonga Ngomo and K. Lyko, "Eagle: Efficient active learning of link specifications using genetic programming," in *Proceedings of the 9th Extended Semantic Web Conference*, ser. Lecture Notes in Computer Science, vol. 7295. Springer, 2012, pp. 149–163.

[10] A.-C. Ngonga Ngomo, K. Lyko, and V. Christen, "Coala-correlation-aware active learning of link specifications," in *Proceedings of the Extended Semantic Web Conference*. Springer, 2013, pp. 442–456.

[11] R. Isele and C. 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.

[12] T. M. Mitchell, "Generalization as search," *Artif. Intell.*, vol. 18, no. 2, pp. 203–226, 1982.

[13] R. Isele and C. Bizer, "Learning linkage rules using genetic programming," in *Proceedings of 6th ontology matching workshop at the International Semantic Web Conference (OM@ISWC), Bonn (DE)*, ser. CEUR Workshop Proceedings, vol. 814, 2011.

[14] A.-C. Ngonga Ngomo, J. Lehmann, S. Auer, and K. Höffner, "Raven – active learning of link specifications," in *Proceedings of the 6th International Workshop on Ontology Matching*, vol. 814. CEUR-WS.org, 2011.

[15] A.-C. Ngonga Ngomo, "A time-efficient hybrid approach to link discovery," in *Proceedings of 6th ontology matching workshop (OM), Bonn (DE)*, ser. CEUR Workshop Proceedings, vol. 814, 2011.

[16] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin, *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[17] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[18] T. M. Mitchell, *Machine Learning*. McGraw-Hill, New York, 1997.

[19] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[20] P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[21] Z. Fan, "Concise pattern learning for rdf data sets interlinking," Ph.D. dissertation, University of Grenoble, 2014.

[22] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," *Knowledge and Information Systems*, vol. 35, no. 2, pp. 249–283, 2013.

[23] A.-C. Ngonga Ngomo, N. Heino, K. Lyko, R. Speck, and M. Kaltenböck, "Scms - semantifying content management systems," in *Proceedings of Internationl Semantic Web Conference*, ser. Lecture Notes in Computer Science, vol. 7032. Springer, 2011, pp. 189–204.