A Generalization Approach for Automatic Link Discovery

Abstract

A significant portion of the evolution of Linked Data datasets lies in updating the links to other datasets. An important challenge when aiming to update these links automatically under the open-world assumption is the fact that usually only positive examples for the links exist. We address this challenge by presenting and evaluating WOMBAT, a novel approach for the discovery of links between knowledge bases that relies exclusively on positive examples. WOMBAT is based on generalisation via an upward refinement operator to traverse the space of link specification. We study the theoretical characteristics of WOMBAT and evaluate it on 8 different benchmark datasets. Our evaluation suggests that WOMBAT outperforms state-of-the-art supervised approaches while relying on less information. Moreover, our evaluation suggests that WOMBAT's pruning algorithm allows it to scale well even on large datasets.

1 Introduction

The Linked Open Data Cloud has grown from a mere 12 datasets at its beginning to a compendium of more than 9,000 public RDF data sets.[1] In addition to the number of the datasets published growing steadily, we also witness the size of single datasets growing with each new edition. For example, *DBpedia* has grown from 103 million triples describing 1.95 million things (DBpedia 2.0) to 583 million triples describing 4.58 million things (DBpedia 2014) within 7 years. This growth engenders an increasing need for automatic support when maintaining evolving datasets. One of the most crucial tasks when dealing with evolving datasets lies in updating the links from these data sets to other data sets. While supervised approaches have been devised to achieve this goal, they assume the provision of both positive and negative examples for links (Auer et al. 2013). However, the links available on the Data Web only provide positive examples for relations and no negative examples, as the open-world assumption underlying the Web of Data suggests that the non-existence of a link between two resources cannot be understood as stating these two resources are not related. Consequently, state-of-the-art supervised learning approaches for link discovery can only be employed if the end users are willing to provide the algorithms with information that is generally not available on the Linked Open Data Cloud, i.e., with negative examples.

We address this drawback by proposing the first approach for learning links based on positive examples only. Our approach, dubbed WOMBAT, is inspired by the concept of generalisation in quasi-ordered spaces. Given a set of positive examples we aim to find a classifier that covers a large number of positive examples (i.e., achieves a high recall on the positive examples) while still achieving a high precision. We use link specifications (LS, see Section 2) as classifier (Auer et al. 2013; Isele, Jentzsch, and Bizer 2012; Ngonga Ngomo 2012). A main challenge while learning is that LS can use various similarity metrics, acceptance thresholds and nested logical combinations of those.

The contributions of this paper are: 1) We provide the first approach for learning LS that is able to learn links from positive examples only. 2) Our approach is based on an upward refinement operator for which we analyse its theoretical characteristics. 3) We use the characteristics of our operator to devise a pruning approach and improve the scalability of WOMBAT. 4) We evaluate WOMBAT on 8 benchmark datasets and show that in addition to needing less training data, it also outperforms the state of the art in most cases.

2 Preliminaries

The aim of link discovery (LD) is to discover the set $\{(s,t) \in S \times T : Rel(s,t)\}$ provided an input relation Rel, two sets S (source) and T (target) of RDF resources. To achieve this goal, declarative LD frameworks rely on LS, which describe the conditions under which $Rel(s,t)$ can be assumed to hold for a pair $(s,t) \in S \times T$. Several grammars have been used for describing LS in previous works (Ngonga Ngomo and Lyko 2012; Isele, Jentzsch, and Bizer 2011; Nikolov, d'Aquin, and Motta 2012). In general, these grammars assume that LS consist of two types of atomic components: *similarity measures* m, which allow comparing property values of input resources and *operators* op, which can be used to combine these similarities to more complex specifications. Without loss of generality, we define a similarity measure m as a function $m : S \times T \rightarrow [0,1]$.We use *mappings* $M \subseteq S \times T$ to store the results of the application of a similarity function to $S \times T$ or subsets thereof. We denote the set of all mappings as $\mathcal{M}$ and the set of all LS as $\mathcal{L}$. We define

[1] http://lodstats.aksw.org

Table 1: Link Specification Syntax and Semantics

LS	$[[LS]]_M$
$f(m,\theta)$	$\{(s,t)\|(s,t) \in M \wedge m(s,t) \geq \theta\}$
$L_1 \sqcap L_2$	$\{(s,t)\|(s,t) \in [[L_1]]_M \wedge (s,t) \in [[L_2]]_M\}$
$L_1 \sqcup L_2$	$\{(s,t)\|(s,t) \in [[L_1]]_M \vee (s,t) \in [[L_2]]_M\}$
$L_1 \backslash L_2$	$\{(s,t)\|(s,t) \in [[L_1]]_M \wedge (s,t) \notin [[L_2]]_M\}$

a *filter* as a function $f(m,\theta)$. We call a specification *atomic* when it consists of exactly one filtering function. A complex specification can be obtained by combining two specifications L_1 and L_2 through an *operator* that allows merging the results of L_1 and L_2. Here, we use the operators $\sqcap$, $\sqcup$ and $\backslash$ as they are complete and frequently used to define LS.

We define the semantics $[[L]]_M$ of a LS L w.r.t. a mapping M as given in Table 1. Those semantics are similar to those used in languages like SPARQL, i.e., they are defined extensionally through the mappings they generate. The mapping $[[L]]$ of a LS L with respect to $S \times T$ contains the links that will be generated by L. A LS L is *subsumed* by L', denoted by $L \sqsubseteq L'$, if for all mappings M, we have $[[L]]_M \subseteq [[L']]_M$. Two LS are *equivalent*, denoted by $L \equiv L'$ iff $L \sqsubseteq L'$ and $L' \sqsubseteq L$. Subsumption ($\sqsubseteq$) is a partial order over $\mathcal{L}$.

3 Constructing and Traversing Link Specifications

The goal of our learning approach is to learn a specification L that generalizes a mapping $M \subseteq S \times T$ which contains a set of pairs (s,t) for which $Rel(s,t)$ holds. Our approach consists of two main steps. First, we aim to derive initial atomic specifications A_i that achieve the same goal. In a second step, we combine these atomic specifications to the target complex specification L by using the operators $\sqcap$, $\sqcup$ and $\backslash$. In the following, we detail how we carry out these two steps.

3.1 Learning Atomic Specifications

The goal here is to derive a set of initial atomic specifications $\{A_1, \ldots, A_n\}$ that achieves the highest possible F-measure given a mapping $M \subseteq S \times T$ which contains all known pairs (s,t) for which $Rel(s,t)$ holds. Given a set of similarity functions m_i, the set of properties P_s of S and the set of properties P_t of T, we begin by computing the subset of properties from S and T that achieve a coverage above a threshold $\tau \in [0,1]$, where the coverage of a property p for a knowledge base K is defined as

$$coverage(p) = \frac{|\{s : (s,p,o) \in K\}|}{|\{s : \exists q : (s,q,o) \in K\}|}. \quad (1)$$

Now for all property pairs $(p,q) \in P_s \times P_t$ with $coverage(p) \geq \tau$ and $coverage(q) \geq \tau$, we compute the mappings $M_{ij} = \{(s,t) \in S \times T : m_{ij}(s,t) \geq \theta_j\}$, where m_{ij} compares s and t w.r.t. p and q and M_{ij} is maximal w.r.t. the F-measure it achieves when compared to M. To this end, we apply an iterative search approach. Finally, we select M_{ij} as the atomic mapping for p and q. Thus, we return as many atomic mappings as property pairs with sufficient coverage. Note that this approach is not quintessential for WOMBAT and can thus be replaced with any approach of choice which returns a set of initial LS that is to be combined.

3.2 Combining Atomic Specifications

After deriving atomic LS as described above, WOMBAT computes complex specifications by using an approach based on generalisation operators. The basic idea behind these operators is to perform an iterative search through a solution space based on a score function. Formally, we rely on the following definitions:

Definition 1 ((Refinement) Operator)**.** *In the quasi-ordered space $(\mathcal{L}, \sqsubseteq)$, we call a function from $\mathcal{L}$ to $2^{\mathcal{L}}$ an (LS)* operator. *A* downward (upward) refinement operator *ρ is an operator, such that for all $L \in \mathcal{L}$ we have that $L' \in \rho(L)$ implies $L' \sqsubseteq L$ ($L \sqsubseteq L'$). L' is called a* specialisation *(*generalisation*) of L. $L' \in \rho(L)$ is usually denoted as $L \rightsquigarrow_\rho L'$.*

Definition 2 (Refinement Chains)**.** *A* refinement chain *of a refinement operator ρ of length n from L to L' is a finite sequence $L_0, L_1, \ldots, L_n$ of LS, such that $L = L_0, L' = L_n$ and $\forall i \in \{1 \ldots n\}, L_i \in \rho(L_{i-1})$. This refinement chain* goes through *L'' iff there is an i ($1 \leq i \leq n$) such that $L'' = L_i$. We say that L'' can be reached from L by ρ if there exists a refinement chain from L to L''. $\rho^*(L)$ denotes the set of all LS which can be reached from L by ρ. $\rho^m(L)$ denotes the set of all LS which can be reached from L by a refinement chain of ρ of length m.*

Definition 3 (Properties of refinement operators)**.** *An operator ρ is called (1) **(locally) finite** iff $\rho(L)$ is finite for all LS $L \in \mathcal{L}$; (2) **redundant** iff there exists a refinement chain from $L \in \mathcal{L}$ to $L' \in \mathcal{L}$, which does not go through (as defined above) some LS $L'' \in \mathcal{L}$ and a refinement chain from L to L' which does go through L''; (3) **proper** iff for all LS $L \in \mathcal{L}$ and $L' \in \mathcal{L}$, $L' \in \rho(L)$ implies $L \not\equiv L'$. An LS upward refinement operator ρ is called **weakly complete** iff for all LS $\bot \sqsubset L$ we can reach a LS L' with $L' \equiv L$ from $\bot$ (most specific LS) by ρ.*

We designed two different operators for combining atomic LS to complex specifications: The first operator takes an atomic LS and uses the three logical connectors to append further atomic LS. Assuming that $(A_1, \ldots, A_n)$ is the set of atomic LS found, φ can be defined as follows:

$$\varphi(L) = \begin{cases} \bigsqcup_{i=1}^{n} A_i & \text{if } L = \bot \\ \left(\bigsqcup_{i=1}^{n} L \sqcup A_i\right) \sqcup \left(\bigsqcup_{i=1}^{n} L \sqcap A_i\right) \sqcup \left(\bigsqcup_{i=1}^{n} L \backslash A_i\right) & \text{otherwise} \end{cases}$$

This naive operator is not a refinement operator (neither upward nor downward). Its main advantage lies in its simplicity allowing for a very efficient implementation. However, it cannot reach all specifications, e.g., a specification of the form $(A_1 \sqcup A_2) \sqcap (A_3 \sqcup A_4)$ cannot be reached. Examples of chains generated by φ are as follows:

1. $\bot \rightsquigarrow_\varphi A_1 \rightsquigarrow_\varphi A_1 \sqcup A_2 \rightsquigarrow_\varphi (A_1 \sqcup A_2) \backslash A_3$
2. $\bot \rightsquigarrow_\varphi A_2 \rightsquigarrow_\varphi A_2 \sqcap A_3 \rightsquigarrow_\varphi (A_2 \sqcap A_3) \backslash A_4$

The second operator, ψ, uses a more sophisticated expansion strategy in order to allow learning arbitrarily nested LS and is shown in Figure 1. Less formally, the operator works

$$\psi(L)=\begin{cases}\{A_{i_1}\setminus A_{j_1}\sqcap\cdots\sqcap A_{i_m}\setminus A_{j_m}\,|\,A_{i_k},A_{j_k}\in P & \\ \quad\text{for all } 1\leq k\leq m\} & \text{if } L=\bot\\ \{L\sqcup A_i\setminus A_j\,|\,A_i\in P,A_j\in P\} & \text{if } L=A\,(atomic)\\ \{L_1\}\cup\{L\sqcup A_i\setminus A_j\,|\,A_i\in P,A_j\in P\} & \text{if } L=L_1\setminus L_2\\ \{L_1\sqcap\cdots\sqcap L_{i-1}\sqcap L'\sqcap L_{i+1}\sqcap\cdots\sqcap L_n\,|\,L'\in\psi(L_i)\} & \\ \quad\cup\{L\sqcup A_i\setminus A_j\,|\,A_i\in P,A_j\in P\} & \text{if } L=L_1\sqcap\cdots\sqcap L_n (n\geq 2)\\ \{L_1\sqcup\cdots\sqcup L_{i-1}\sqcup L'\sqcup L_{i+1}\sqcup\cdots\sqcup L_n\,|\,L'\in\psi(L_i)\} & \\ \quad\cup\{L\sqcup A_i\setminus A_j\,|\,A_i\in P,A_j\in P\} & \text{if } L=L_1\sqcup\cdots\sqcup L_n (n\geq 2)\end{cases}$$

Figure 1: Definition of the refinement operator ψ.

as follows: It takes a LS as input and makes a case distinction on the type of LS. Depending on the type, it performs the following actions:

- The $\bot$ link specification is refined to the set of all combinations of $\setminus$ operations. This set can be large and will only be built iteratively (as required by the algorithm) with at most approx. n^2 refinements per iteration (see the next section for details).
- In LS of the form $A_1 \setminus A_2$, ψ can drop the second part in order to generalise.
- If the link specification is a conjunction or disjunction, the operator can perform a recursion on each element of the conjunction or disjunction.
- For LS of any type, a disjunction with an atomic LS can be added.

Below are two example refinement chains of ψ:

1. $\bot \rightsquigarrow_\psi A_1 \setminus A_2 \rightsquigarrow_\psi A_1 \rightsquigarrow_\psi A_1 \sqcup A_2 \setminus A_3$
2. $\bot \rightsquigarrow_\psi A_1 \setminus A_2 \sqcap A_3 \setminus A_4 \rightsquigarrow_\psi A_1 \sqcap A_3 \setminus A_4 \rightsquigarrow_\psi A_1 \sqcap A_3 \rightsquigarrow_\psi (A_1 \sqcap A_3) \sqcup (A_5 \setminus A_6)$

ψ is an upward refinement operator with the following properties.[2]

Proposition 1. *ψ is weakly complete.*

Proposition 2. *ψ is finite, not proper and redundant.*

Naturally, the restrictions of ψ (being redundant and not proper) raise the question whether there are LS refinement operators satisfying all theoretical properties:

Proposition 3. *There exists a weakly complete, finite, proper and non-redundant refinement operator in $\mathcal{L}$.*

The existence of an operator which satisfies all considered theoretical criteria of a refinement operator is an artifact of only finitely many semantically inequivalent LS existing in $\mathcal{L}$. This set is however extremely large and not even small fractions of it can be evaluated in all but very simple cases. For example, the operator α as $\alpha(\bot) = C$ and $\alpha(L) = \emptyset$ for all $L \neq \bot$ is trivially non-redundant and it is proper by definition. Such an operator α is obviously not useful as it does not help *structuring the search space*. Providing a useful way to structure the search space is the main reason for refinement operators being successful for learning in other complex languages as it allows to gradually converge towards useful solutions while being able to prune other paths which cannot lead to promising solutions (explained in the next section). This is the major reason why we sacrificed properness and redundancy for a better structure of the search space.

[2]Proofs are given in the appendix.

4 The WOMBAT Algorithm

Algorithm 1: WOMBAT Learning Algorithm

Input: Sets of resources S and T; examples $E \subseteq S \times T$; property coverage threshold τ; set of similarity functions **F**

```
1  A ⟵ null (the list of initial atomic metrics);
2  i ⟵ 1 ;
3  foreach property p_s ∈ S do
4      if coverage(p_s) ≥ τ then
5          foreach property p_t ∈ T do
6              if coverage(p_t) ≥ τ then
7                  Find atomic metric m(p_s, p_t) that leads
                   to highest F-measure;
8                  Optimize similarity threshold for
                   m(p_s, p_t) to find best mapping A_i;
9                  Add A_i to A;
10                 i ⟵ i + 1;
11 Γ ⟵ ⊥ (initiate search tree Γ to the root node ⊥);
12 F_best ⟵ 0, L_best ⟵ null;
13 while termination criterion not met do
14     Choose the node with highest scoring LS L in Γ;
15     if L == ⊥ then
16         foreach A_i, A_j ∈ A, where i ≠ j do
17             Only add refinements of form A_i \ A_j;
18     else
19         Apply operator to L;
20         if L is a refinement of ⊥ then
21             foreach A_i, A_j ∈ A, where i ≠ j do
22                 In addition to refinements, add
                   conjunctions with specifications of the
                   form A_i \ A_j as siblings;
23     foreach refinement L' do
24         if L' is not already in the search tree Γ then
25             Add L' to Γ as children of the node
               containing L;
26     Update F_best and L_best;
27     if F_best has increased then
28         foreach subtree t ∈ Γ do
29             if F_best > F_max(t) then
30                 Delete t;
31 Return L_best;
```

We have now introduced all ingredients necessary for defining the WOMBAT algorithms. The first algorithm,

which we refer to as *simple* version, uses the operator φ, whereas the second algorithm, which we refer to as *complete*, uses the refinement operator ψ. The complete algorithm has the following specific characteristics: First, while ψ is finite, it would generate a prohibitively large number of refinements when applied to the $\bot$ concept. For that reason, those refinements will be computed stepwise as we will illustrate below. Second, as ψ is an upward refinement operator it allows to prune parts of the search space, which we will also explain below. We only explain the implementation of the complex WOMBAT algorithm as the other is a simplification excluding those two characteristics.

Algorithm 1 shows the individual steps required. It takes the source dataset S, the target dataset T, examples $E \subseteq S \times T$ as well as the property coverage threshold and the set of considered similarity functions as input. In Line 3, the property matches are computed by optimizing the threshold for properties that have the minimum coverage (Line 7) as previously described. The main loop starts in Line 13 and runs until a termination criterion is satisfied, e.g. 1) a fixed number of LS has been evaluated, 2) a certain time has elapsed, 3) the best F-score has not changed for a certain time or 4) a perfect solution has been found. Line 14 states that a heuristic-based search strategy is employed. By default, we employ the F-score directly. More complex heuristics introducing a bias towards specific types of LS could be encoded here. In Line 15, we make a case distinction: Since the number of refinements of $\bot$ is extremely high and not feasible to compute in most cases, we perform a stepwise approach: In the first step, we only add simple LS of the form $A_i \setminus A_j$ as refinements (Line 17). Later, in Line 22, we add more complex conjunctions if the simpler forms are promising. Apart from this special case, we apply the operator directly. Line 24 updates the search tree by adding the nodes obtained via refinement. Moreover, it contains a redundancy elimination procedure: We only add those nodes to the search tree which are not already contained in it.

The subsequent part starting from Line 26 defines our *pruning procedure*: Since ψ is an upward refinement operator, we know that the set of links generated by a child node is a superset of or equal to the set of links generated by its parent. Hence, while both precision and recall can improve in subsequent refinements, they cannot rise arbitrarily. Precision is bound as false positives cannot disappear during generalisation. Furthermore, the achievable recall r_{max} is that of the most general constructable LS, i.e., $\mathcal{A} = \bigcup A_i$ This allows to compute an upper bound on the achievable F-score. In order to do so, we first build a set S' with those resources in S occurring in the input examples E as well as a set T' with those resources in T occurring in E. The purpose of those is to restrict the computation of F-score to the fragment $S' \times T' \subseteq S \times T$ relevant for example set E. We can then compute an upper bound of precision of a link specification L as follows:

$$p_{max}(L) = \frac{|E|}{|E| + |\{(s,t) \mid (s,t) \in [[L]], s \in S' \text{ or } t \in T'\} \setminus E|}$$

F_{max} is then computed as the F-measure obtained with recall r_{max} and precision p_{max}, i.e., $F_{max} = \frac{2 p_{max} r_{max}}{p_{max} + r_{max}}$. It is an upper bound for the maximum achievable F-measure of any node reachable via refinements. We can disregard all nodes in the search tree which have a maximum achievable F-score that is lower than the best F-score already found. This is implemented in Line 28. The pruning is conservative in the sense that no solutions are lost. In the evaluation, we give statistics on the effect of pruning. WOMBAT ends by returning L_{best} as the best LS found, which is the specification with the highest F-score. In case of ties, we prefer shorter specifications over long ones. Should the tie persist, then we prefer specifications that were found early.

Proposition 4. *WOMBAT is complete, i.e., it will eventually find the LS with the highest F-measure within $\mathcal{L}$.*

5 Evaluation

We evaluated our approach using 8 benchmark datasets. Five of these benchmarks were real-world datasets while three were synthetic. The real-world interlinking tasks used were those in (Köpcke, Thor, and Rahm 2010). The synthetic datasets were from the OAEI 2010 benchmark[3]. All experiments were carried out on a 4-core 2.80 GHz PC running *OpenJDK* 64-Bit Server 1.7.0_75 on *Ubuntu* 14.04.2 LTS. Each experiment was assigned 7 GB RAM. For testing WOMBAT against the benchmark datasets in both its simple and complete version, we used the `jaccard` and `trigrams` similarity measures. As termination criterion, we used a maximum number of refinement nodes of 2000. The coverage threshold τ was set to 0.6.

To be comparable, we used the evaluation protocol in (Kejriwal and Miranker 2015). For the first series of experiments, we split each input dataset into 10 parts of the same and we used 3 randomly selected parts (30%) of the data as training data and the remaining 7 parts (70%) as test data. For the comparison with (Kejriwal and Miranker 2015), a random selection of 2% of the reference data was used as training data while the rest of the data was used for testing. For each of the datasets we ran each of the algorithms 25 times and use the results of the best 5 runs.

We report the average of the best F-measures achieved by each algorithm in Table 2. Our results suggest that we achieve higher F-measures than the state of the art on all datasets. Interestingly, the simple operator φ achieves the best results. While ψ is guaranteed to be able to achieve the same results, it requires more iterations to achieve this goal.

In the second set of experiments, we measured the effect of increasing the amount of training data on the precision, recall and F-score achieved by both simple and complete versions of WOMBAT. The results are presented in Figure 2. Our results suggest that the complete version of WOMBAT converges faster towards the best solution that it can find. This suggests that once trained on a dataset, our approach can be used on subsequent versions of real datasets, where a small number of novel resources is added in each new version, which is the problem setup considered in this paper. On the other hand, the simple version is able to find better specifications as it can explore longer sequences of mappings.

[3] `http://oaei.ontologymatching.org/2010/`

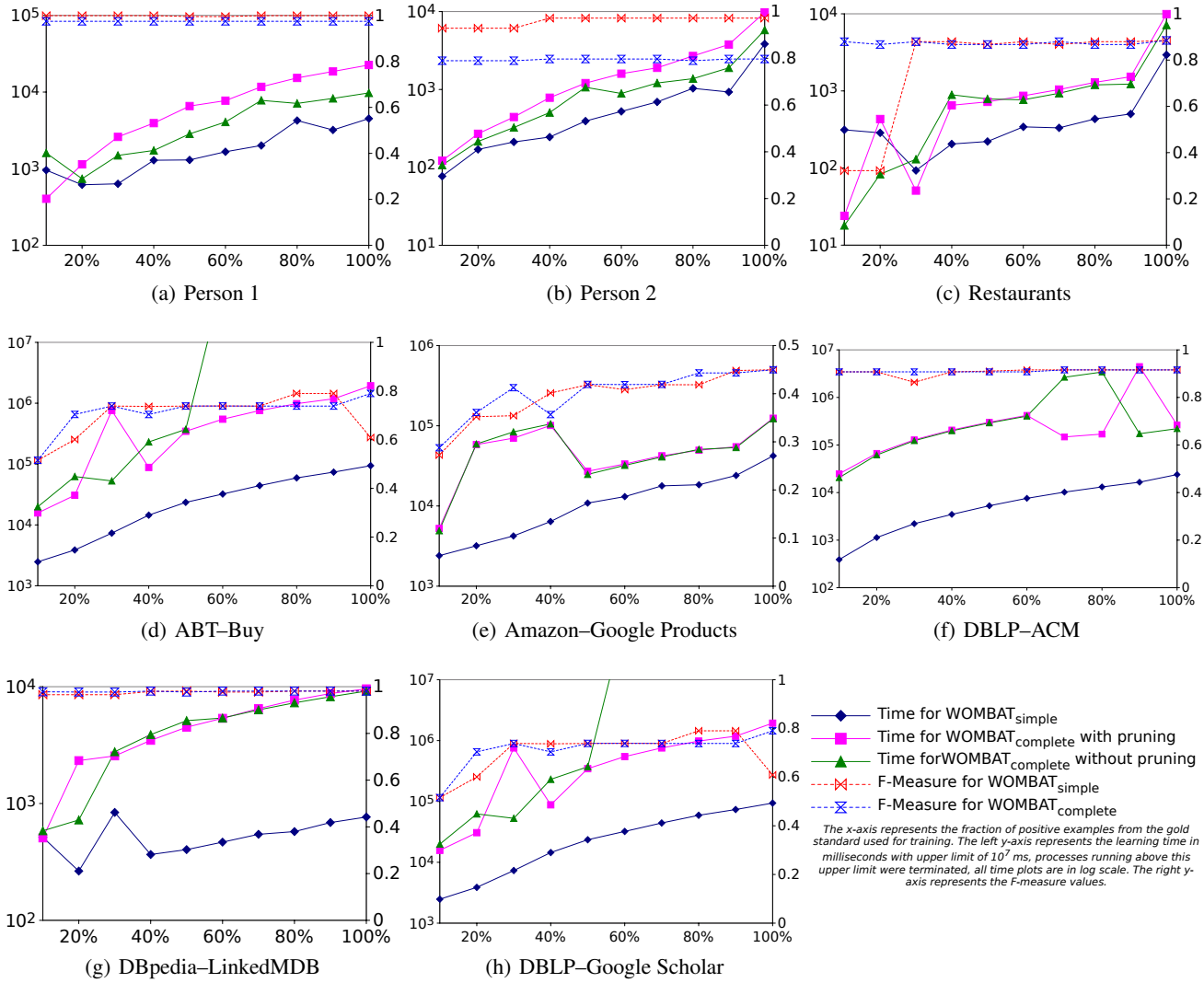

Figure 2: Runtime and F-measure results of WOMBAT.

Table 2: A comparison of WOMBAT F-Measure against 4 state-of-the-art approaches on 8 benchmarks using 30% of the original data as training data.

Dataset	Eagle	Euclid linear	Euclid conj.	Euclid dis.	WOMBAT Complete	WOMBAT Simple
Person 1	0.994	0.948	0.961	0.991	0.974	**1.000**
Person 2	0.886	0.803	0.823	0.883	0.797	**0.984**
Restaurants	0.885	0.871	0.837	0.885	0.885	**0.994**
DBLP-ACM	0.905	0.876	0.892	0.905	0.908	**0.961**
Abt-Buy	0.293	0.289	0.289	0.289	0.385	**0.477**
Amazon-GP	0.333	0.308	0.305	0.318	0.416	**0.565**
DBP-LMDB	0.973	0.967	0.964	0.975	0.978	**0.991**
DBLP-GS	0.720	0.827	0.760	0.744	0.718	**0.916**

In the third set experiments, we measured the learning time for each of the benchmark datasets. The results are presented in Figure 2. As expected, the simple approach is time-efficient to run even without any optimization. While the complete version of WOMBAT without pruning is significantly slower (up to 1 order of magnitude), the effect of pruning can be clearly seen as it reduces the runtime of the algorithm while also improving the total space that the complete version of WOMBAT can explore. These results are corroborated by our fourth set of experiments, in which we evaluated the pruning technique of the complete version of WOMBAT. In those experiments, for each of aforementioned benchmark datasets we computed what we dubbed as *pruning factor*. The pruning factor is the number of searched nodes (search tree size plus pruned nodes) divided by the maximum size of the search tree (which we set to 2000 nodes in all experiments). The results are presented in Table 4. Our average *pruning factor* of 2.55 shows that we can discard more than 3000 nodes while learning specifications.

Overall, our results show that ψ and φ are able to learn high-quality link specifications using only positive examples. When combined with our pruning algorithm, the complete version of ψ achieves runtimes that are comparable to those of φ. However, given its completeness, ψ can reach

specifications that simply cannot be learned by φ.

In the final set of experiments, we compared the two versions of WOMBAT against the 2 systems proposed in (Kejriwal and Miranker 2015), where we followed the same experimental setup in the paper. The results (presented in Table 3) suggests that WOMBAT is capable of achieving slightly inferior F-measures although it should be noted that the competing systems are optimised for a low number of examples and they also get negative examples as input.

6 Related Work

There is a significant body of related work on *positive only learning*, which we can only briefly cover here. The work presented by (Muggleton 1997) showed that logic programs are *learnable* with arbitrarily low expected error from positive examples only. (Nigam et al. 2000) introduced an algorithm for learning from labeled and unlabeled documents based on the combination of Expectation Maximization (EM) and a naive Bayes classifier. (Denis, Gilleron, and Letouzey 2005) provides an algorithm for learning from positive and unlabeled examples for statistical queries. The pLSA algorithm (Zhou et al. 2010) extends the original probabilistic latent semantic analysis, which is a purely unsupervised framework, by injecting a small amount of supervision information from the user.

For learning with *refinement operators*, significant previous work exists in the area of Inductive Logic Programming and more generally concept learning which we only briefly sketch here. A milestone was the Model Inference System in (Shapiro 1991). Shapiro describes how refinement operators can be used to adapt a hypothesis to a sequence of examples. Afterwards, refinement operators became widely used as a learning method. In (van der Laag and Nienhuys-Cheng 1994) some general results regarding refinement operators in quasi-ordered spaces were published. In (Esposito et al. 2004) and later (Iannone, Palmisano, and Fanizzi 2007), algorithms for learning in description logics (in particular for the language $\mathcal{ALC}$) were created which also make use of refinement operators. Recent studies of refinement operators include (Lehmann and Hitzler 2007; Lehmann and Hitzler 2010) which analysed properties of $\mathcal{ALC}$ and more expressive description logics. A constructive existence proof for ideal (complete, proper and finite) operators in the lightweight $\mathcal{EL}$ description logics has been shown in (Lehmann and Haase 2009).

Table 3: Comparison of WOMBAT F-Measure against 2 the approaches proposed in (Kejriwal and Miranker 2015) on 6 benchmarks using 2% of the original data as training data.

Dataset	Pessimistic	Re-weighted	Simple	Complete
Persons 1	1.000	1.000	0.974	0.999
Persons 2	0.971	0.971	0.797	0.797
Restaurants	0.946	0.946	0.880	0.885
DBLP-ACM	0.934	0.946	0.943	0.916
Amazon-GP	0.391	0.429	0.361	0.416
Abt-Buy	0.362	0.371	0.350	0.320

Table 4: The *pruning factor* of the benchmark datasets. The first column contains the size of the training data as a percentage of its relative original dataset, D_1 stands for *Person 1* dataset, D_2 stands for *Person 2* dataset, D_3 stands for *Restaurant* dataset, D_4 stands for *DBLP–ACM* dataset, D_5 stands for *Abt–Buy* dataset, D_6 stands for *Amazon–GoogleProducts* dataset, D_7 stands for *DBpedia–LinkedMDB* dataset and D_8 stands for *DBLP–GoogleScholar*.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
10%	1.57	1.29	1.17	6.23	3.38	1.14	1.00	1.79
20%	2.13	1.29	1.45	5.58	3.00	1.38	1.86	1.93
30%	1.85	1.57	1.17	6.79	3.00	1.33	2.86	2.01
40%	2.13	1.57	1.45	6.85	3.39	1.37	1.86	2.36
50%	2.13	1.57	1.45	6.85	3.39	1.38	1.86	2.45
60%	2.13	1.57	1.45	6.85	3.39	1.45	2.33	1.66
70%	2.13	1.57	1.45	6.79	1.79	1.54	2.36	2.44
80%	2.13	1.57	1.45	6.79	3.39	1.59	2.36	2.26
90%	2.13	1.57	1.45	6.93	3.39	1.60	2.36	1.97
100%	2.13	1.57	1.45	6.79	3.39	1.60	2.36	2.05

Most LD approaches for *learning LS* developed are supervised. One of the first approaches to target this goal was presented in (Isele and Bizer 2011). While this approach achieves high F-measures, it also requires large amounts of training data. Hence, methods based on active learning have also been developed (see, e.g., (Isele, Jentzsch, and Bizer 2012; Ngonga Ngomo, Lyko, and Christen 2013)). In general, these approaches assume some knowledge about the type of links that are to be discovered. For example, unsupervised approaches such as PARIS (Suchanek, Abiteboul, and Senellart 2011) aim to discover exclusively `owl:sameAs` links. Newer unsupervised techniques for learning LS include approaches based on probabilistic models (Suchanek, Abiteboul, and Senellart 2011) and genetic programming (Nikolov, d'Aquin, and Motta 2012; Ngonga Ngomo and Lyko 2013), which all assume that a 1-to-1 mapping is to be discovered. To the best of out knowledge, this paper presents the first LD approach designed to learn from positive examples only.

7 Conclusions and Future Work

We presented the (to the best of our knowledge) first approach to learn LS from positive examples via generalisation over the space of LS. We presented a simple operator φ that aims to achieve this goal as well as the complete operator ψ. We evaluated φ and ψ against state-of-the-art link discovery approaches and showed that we outperform them on benchmark datasets. We also considered scalability and showed that ψ can be brought to scale similarly to φ when combined with the pruning approach we developed. In future work, we aim to parallelize our approach as well as extend it by trying more aggressive pruning techniques for better scalability.

References

[Auer et al. 2013] Auer, S.; Lehmann, J.; Ngonga Ngomo, A.-C.; and Zaveri, A. 2013. Introduction to linked data and its lifecycle on the web. In *Reasoning Web*, 1–90.

[Denis, Gilleron, and Letouzey 2005] Denis, F.; Gilleron, R.; and Letouzey, F. 2005. Learning from positive and unlabeled examples. *Theoretical Computer Science* 348(1):70 – 83. Algorithmic Learning Theory (ALT 2000) 11th International Conference, Algorithmic Learning Theory 2000.

[Esposito et al. 2004] Esposito, F.; Fanizzi, N.; Iannone, L.; Palmisano, I.; and Semeraro, G. 2004. Knowledge-intensive induction of terminologies from metadata. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, 441–455. Springer.

[Iannone, Palmisano, and Fanizzi 2007] Iannone, L.; Palmisano, I.; and Fanizzi, N. 2007. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence* 26(2):139–159.

[Isele and Bizer 2011] Isele, R., and Bizer, C. 2011. Learning Linkage Rules using Genetic Programming. In *Sixth International Ontology Matching Workshop*.

[Isele, Jentzsch, and Bizer 2011] Isele, R.; Jentzsch, A.; and Bizer, C. 2011. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*.

[Isele, Jentzsch, and Bizer 2012] Isele, R.; Jentzsch, A.; and Bizer, C. 2012. Active learning of expressive linkage rules for the web of data. In *ICWE*, 411–418.

[Kejriwal and Miranker 2015] Kejriwal, M., and Miranker, D. P. 2015. Semi-supervised instance matching using boosted classifiers. In *The Semantic Web. Latest Advances and New Domains*. Springer. 388–402.

[Köpcke, Thor, and Rahm 2010] Köpcke, H.; Thor, A.; and Rahm, E. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.* 3(1-2):484–493.

[Lehmann and Haase 2009] Lehmann, J., and Haase, C. 2009. Ideal downward refinement in the EL description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.

[Lehmann and Hitzler 2007] Lehmann, J., and Hitzler, P. 2007. Foundations of refinement operators for description logics. In *ILP*, volume 4894 of *Lecture Notes in Computer Science*, 161–174. Springer.

[Lehmann and Hitzler 2010] Lehmann, J., and Hitzler, P. 2010. Concept learning in description logics using refinement operators. *Machine Learning journal* 78(1-2):203–250.

[Muggleton 1997] Muggleton, S. 1997. Learning from positive data. In *Inductive logic programming*. Springer. 358–376.

[Ngonga Ngomo and Lyko 2012] Ngonga Ngomo, A.-C., and Lyko, K. 2012. Eagle: Efficient active learning of link specifications using genetic programming. In *Proceedings of ESWC*.

[Ngonga Ngomo and Lyko 2013] Ngonga Ngomo, A.-C., and Lyko, K. 2013. Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *Proceedings of the Ontology Matching Workshop*.

[Ngonga Ngomo, Lyko, and Christen 2013] Ngonga Ngomo, A.-C.; Lyko, K.; and Christen, V. 2013. Coala – correlation-aware active learning of link specifications. In *Proceedings of ESWC*.

[Ngonga Ngomo 2012] Ngonga Ngomo, A.-C. 2012. Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *Proceedings of ISWC*.

[Nigam et al. 2000] Nigam, K.; McCallum, A. K.; Thrun, S.; and Mitchell, T. 2000. Text classification from labeled and unlabeled documents using em. *Machine learning* 39(2-3):103–134.

[Nikolov, d'Aquin, and Motta 2012] Nikolov, A.; d'Aquin, M.; and Motta, E. 2012. Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications*. Springer. 119–133.

[Shapiro 1991] Shapiro, E. Y. 1991. Inductive inference of theories from facts. In Lassez, J. L., and Plotkin, G. D., eds., *Computational Logic: Essays in Honor of Alan Robinson*. The MIT Press. 199–255.

[Suchanek, Abiteboul, and Senellart 2011] Suchanek, F. M.; Abiteboul, S.; and Senellart, P. 2011. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB* 5(3):157–168.

[van der Laag and Nienhuys-Cheng 1994] van der Laag, P. R. J., and Nienhuys-Cheng, S.-H. 1994. Existence and nonexistence of complete refinement operators. In Bergadano, F., and Raedt, L. D., eds., *ECML*, volume 784 of *Lecture Notes in Artificial Intelligence*, 307–322. Springer-Verlag.

[Zhou et al. 2010] Zhou, K.; Gui-Rong, X.; Yang, Q.; and Yu, Y. 2010. Learning with positive and unlabeled examples using topic-sensitive plsa. *Knowledge and Data Engineering, IEEE Transactions on* 22(1):46–58.