WOMBAT – A Generalization Approach for Automatic Link Discovery

Mohamed Ahmed Sherif[1](✉), Axel-Cyrille Ngonga Ngomo[1,2], and Jens Lehmann[3,4]

[1] R&D Department II, Computing Center, University of Leipzig, 04109 Leipzig, Germany
{sherif,ngonga}@informatik.uni-leipzig.de
[2] Data Science Group, University of Paderborn, Pohlweg 51, 33098 Paderborn, Germany
ngonga@upb.de
[3] Computer Science Institute, University of Bonn, Römerstr. 164, 53117 Bonn, Germany
jens.lehmann@cs.uni-bonn.de
[4] Fraunhofer IAIS, Schloss Birlinghoven, 53757 Sankt Augustin, Germany
jens.lehmann@iais.fraunhofer.de

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.

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

E. Blomqvist et al. (Eds.): ESWC 2017, Part I, LNCS 10249, pp. 103–119, 2017.
DOI: 10.1007/978-3-319-58068-5_7

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 [1]. 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 Sect. 2) as classifiers [1,7,14]. We are thus faced with the challenge of using various similarity metrics, acceptance thresholds and nested logical combinations of those when learning. 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 and 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 [6,15,19]. 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 LS. Without loss of generality, we define a similarity measure m as a function $m : S \times T \rightarrow [0,1]$. An example of a similarity measure is the edit similarity dubbed `edit`[2] which allows computing the similarity of a pair $(s,t) \in S \times T$ with respect to the properties p_s of s and p_t of t. 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

[2] We define the edit similarity of two strings s and t as $(1 + lev(s,t))^{-1}$, where lev stands for the Levenshtein distance.

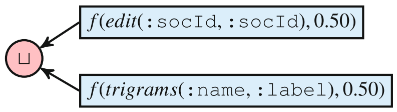

⊔
f(edit(:socId, :socId), 0.50)
f(trigrams(:name, :label), 0.50)

Fig. 1. Example of a complex LS. The filter nodes are rectangles while the operator nodes are circles. `:socID` stands for social security number.

Table 1. Link specification syntax and semantics

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

of all LS as $\mathcal{L}$. We define 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. An example of a complex LS is given in Fig. 1.

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

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

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:

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

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

Fig. 2. Definition of the refinement operator ψ.

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

The second operator, ψ, uses a more sophisticated expansion strategy in order to allow learning arbitrarily nested LS and is shown in Fig. 2. Less formally, the operator works 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 $\bot$ LS 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 LS 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.

Proposition 1. *ψ is an upward refinement operator.*

Proof. For an arbitrary LS L, we have to show for any element $L' \in \psi(L)$ that $L \sqsubseteq L'$ holds. The proof is straightforward by showing that L' cannot generate less links than L via case distinction and structural induction over LS:

- $L = \perp$: Trivial.
- L is atomic: Adding a disjunction cannot result in less links (this also holds for the cases below).
- L is of the form $L_1 \setminus L_2$: $L' = L_1$ cannot result in less links.
- L is a conjunction / disjunction: L' cannot result in less links by structural induction. □

Proposition 2. *ψ is weakly complete.*

Proof. To show this, we have to show that an arbitrary LS L can be reached from the $\perp$ LS. First, we convert everything to negation normal form by pushing $\setminus$ inside, e.g. LS of the form $L_1 \setminus (L_2 \sqcap L_3)$ are rewritten to $(L_1 \setminus L_2) \sqcup (L_1 \setminus L_3)$ and LS of the form $L_1 \setminus (L_2 \sqcup L_3)$ are rewritten to $(L_1 \setminus L_2) \sqcap (L_1 \setminus L_3)$ exhaustively. We then further convert the LS to conjunction normal including an exhaustive application of the distribute law, i.e., conjunctions cannot be nested within disjunctions. The resulting LS is dubbed L' and equivalent to L. We show that L' can always be reached from $\perp$ via induction over its structure:

- $L' = \perp$: Trivial via the empty refinement chain.
- $L' = A$ (atomic): Reachable via $\perp \rightsquigarrow_\psi A \setminus A' \rightsquigarrow_\psi A$.
- $L' = A_1 \setminus A_2$ (atomic negation): Reachable directly via $\perp \rightsquigarrow_\psi A_1 \setminus A_2$.
- L' is a conjunction with m elements: $\perp \rightsquigarrow_\psi A_{i_1} \setminus A_{j_1} \sqcap \cdots \sqcap A_{i_m} \setminus A_{j_m}$ where an element $A_{i_k} \setminus A_{j_k}$ is chosen as follows: Let the k-th element of conjunction L' be L''.
  - If L'' is an atomic specification A, then $A_{i_k} = A$ (A_{j_k} can be arbitrarily).
  - If L'' is an atomic negation $A_1 \setminus A_2$, then $A_{i_k} = A$ and $A_{j_k} = A_2$.
  - If L'' is a disjunction, the first element of this disjunction falls into one the above two cases and A_{i_k} and A_{j_k} can be set as described there.

  Each element of L'' is then further refined to L' as follows:
  - If L'' is an atomic specification A: $A \setminus A_{j_k}$ is refined to A.
  - If L'' is an atomic negation $A_1 \setminus A_2$: No further refinements are necessary.
  - If L'' is a disjunction. The first element of the disjunction is first treated according to the two cases above. Subsequent elements of the disjunction are either atomic LS or atomic negation and can be added straightforwardly as the operator allows adding disjunctive elements to any non-$\perp$ LS.

Please note that the case distinction is exhaustive as we assume L' is in conjunctive negation normal form, i.e., there are no disjunctions on the outer level, negation is always atomic, conjunctions are not nested within other conjunction and elements of disjunctions within conjunctions cannot be conjunctions. □

Proposition 3. *ψ is finite, not proper and redundant.*

Proof. Finiteness: There are only finitely many atomic LS. Hence, there are only finitely many atomic negations and, consequently, finitely many possible conjunctions of those. Consequently, $\psi(\perp)$ is finite. The finiteness of $\psi(L)$ with $L \neq \perp$ is straightforward.

Properness: The refinement chain $\bot \rightsquigarrow_{\psi}^{*} A_1 \sqcap A_2 \rightsquigarrow_{\psi}^{*} (A_1 \sqcup A_2) \sqcap A_2$ is a counterexample.

Redundancy: The two refinement chains $A_1 \sqcap A_3 \rightsquigarrow_{\psi}^{*} (A_1 \sqcup A_2) \sqcap A_3 \rightsquigarrow_{\psi}^{*} (A_1 \sqcup A_2) \sqcap (A_3 \sqcup A_4)$ and $A_1 \sqcap A_3 \rightsquigarrow_{\psi}^{*} A_1 \sqcap (A_3 \sqcap A_4) \rightsquigarrow_{\psi}^{*} (A_1 \sqcup A_2) \sqcap (A_3 \sqcup A_4)$ are a counterexample. $\square$

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

Proposition 4. *There exists a weakly complete, finite, proper and non-redundant refinement operator in $\mathcal{L}$.*

Proof. Let C be the set of LS in $\mathcal{L}$ in conjunctive negation normal form without any LS equivalent to $\bot$. We define the operator α as $\alpha(\bot) = C$ and $\alpha(L) = \emptyset$ for all $L \neq \bot$. α is obviously complete as any LS has an equivalent in conjunctive negation normal form. It is finite as S can be shown to the finite with an extended version of the argument in the finiteness proof of ψ. α is trivially non-redundant and it is proper by definition. $\square$

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 a reason why we sacrificed properness and redundancy for a better structure of the search space.

4 The WOMBAT Algorithm

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 of WOMBAT complete. Our approach takes the source dataset S, the target dataset T, examples $E \subseteq S \times T$ as well as

Algorithm 1. WOMBAT Learning Algorithm

```
   Input: Sets of resources $S$ and $T$; examples $E \subseteq S \times T$; property coverage
          threshold $\tau$; set of similarity functions $\mathbf{F}$
1  $\mathbf{A} \longleftarrow null$ (the list of initial atomic metrics);
2  $i \longleftarrow 1$ ;
3  foreach property $p_s \in S$ do
4      if $coverage(p_s) \geq \tau$ then
5          foreach property $p_t \in T$ do
6              if $coverage(p_t) \geq \tau$ 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 $\mathbf{A}$;
10                 $i \longleftarrow i + 1$;
11 $\Gamma \longleftarrow \perp$ (initiate search tree $\Gamma$ to the root node $\perp$);
12 $F_{best} \longleftarrow 0$, $L_{best} \longleftarrow null$;
13 while termination criterion not met do
14     Choose the node with highest scoring LS $L$ in $\Gamma$;
15     if $L == \perp$ then
16         foreach $A_i, A_j \in \mathbf{A}$, where $i \neq j$ do
17             Only add refinements of form $A_i \setminus A_j$;
18     else
19         Apply operator to $L$;
20         if $L$ is a refinement of $\perp$ then
21             foreach $A_i, A_j \in \mathbf{A}$, where $i \neq j$ do
22                 In addition to refinements, add conjunctions with specifications
                     of the form $A_i \setminus A_j$ as siblings;
23     foreach refinement $L'$ do
24         if $L'$ is not already in the search tree $\Gamma$ then
25             Add $L'$ to $\Gamma$ as children of the node containing $L$;
26     Update $F_{best}$ and $L_{best}$;
27     if $F_{best}$ has increased then
28         foreach subtree $t \in \Gamma$ do
29             if $F_{best} > F_{max}(t)$ then
30                 Delete $t$;
31 Return $L_{best}$;
```

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 described in Sect. 3.1. 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 $\perp$ 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). 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. For redundancy elimination, 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., $\mathscr{A} = \bigcup A_i$ 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 LS 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., $F_{max} = \frac{2p_{max}r_{max}}{p_{max}+r_{max}}$. 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 5. WOMBAT *is complete, i.e., it will eventually find the LS with the highest F-measure within $\mathcal{L}$.*

Proof. This is a consequence of the weak completeness of ψ and the fact that the algorithm will eventually generate all refinements of ψ. For the latter, we have to look at the refinement of $\perp$ as a special case since otherwise a straightforward application of ψ is used. For the refinements of $\perp$ it is easy to show via induction over the number of conjunctions in refinements that any element in $\psi(\perp)$ can be reached via the algorithm. (The pruning is conservative and only prunes nodes never leading to better solutions.) □

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 [9]. The synthetic datasets were from the OAEI 2010 benchmark[3]. All experiments were carried out on a 64-core 2.3 GHz PC running *OpenJDK* 64-Bit Server 1.7.0_75 on *Ubuntu* 14.04.2 LTS. Each experiment was assigned 20 GB RAM.

For testing Wombat against the benchmark datasets in both its simple and complete version, we used the `jaccard`, `trigrams`, `cosine` and `qgrams` similarity measures. We used two termination criteria: Either a LS with F-measure of 1 was found or a maximal depth of refinement (10 resp. 3 for the simple resp. complete version) was reached. This variation of the maximum refinement trees sizes between the simple and complete version was because Wombat complete adds a larger number of nodes to its refinement tree in each level. The coverage threshold τ was set to 0.6. A more complete list of evaluation results are available at the project web site.[4] Altogether, we carried out 6 sets of experiments to evaluate Wombat.

In *the first set of experiments*, we compared the average F-Measure achieved by the simple and complete versions of Wombat to that of four other state-of-the-art LS learning algorithms within a 10-fold cross validation setting. The other four LS learning algorithms were Eagle [15] as well as the *linear*, *conjunctive* and *disjunctive* versions of Euclid [16]. Eagle was configured to run 100 generations. The mutation and crossover rates were set to 0.6 as in [15]. To address the non-deterministic nature of Eagle, we repeated the whole process of 10-fold cross validation 5 time and present the average results. Euclid's grid size was set to 5 and 100 iterations were carried out as in [16]. The results of the evaluation are presented in Table 2. The simple version of Wombat was able to outperform the state-of-the-art approaches in 4 out of the 8 data sets and came in the second position in 2 datasets. Wombat complete was able to achieve the best F-score in 4 data sets and achieve the second best F-measure in 3 datasets. On average, both versions of Wombat were able to achieve an F-measure of 0.9, by which Wombat outperforms the three version of Euclid by an average of 11%. While Wombat was able to achieve the same performance of Eagle in average, Wombat is still to be preferred as (1) Wombat only requires positive examples and (2) Eagle is indeterministic by nature.

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

[4] https://github.com/AKSW/LIMES/tree/master/evaluationsResults/wombat.

Table 2. 10-fold cross validation F-measure results.

Dataset	WOMBAT Simple	WOMBAT Complete	EUCLID Linear	EUCLID Conjunction	EUCLID Disjunction	EAGLE
Person 1	**1.00**	**1.00**	0.64	0.97	**1.00**	0.99
Person 2	**1.00**	0.99	0.22	0.78	0.96	0.94
Restaurants	**0.98**	0.97	0.97	0.97	0.97	0.97
DBLP-ACM	0.97	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**
Abt-Buy	0.60	0.61	0.06	0.06	0.52	**0.65**
Amazon-GP	0.70	0.67	0.59	0.71	**0.73**	0.71
DBP-LMDB	0.99	**1.00**	0.99	0.99	0.99	0.99
DBLP-GS	**0.94**	**0.94**	0.90	0.91	0.91	0.93
Average	**0.90**	**0.90**	0.67	0.80	0.88	**0.90**

For *the second set of experiments*, we implemented an evaluation protocol based on the assumptions made at the beginning of this paper. Each input dataset was split into 10 parts of the same size. Consequently, we used 3 parts (30%) of the data as training data and the rest 7 parts (70%) for testing. This was to implement the idea of the dataset growing and the specification (and therewith the links) for the new version of the dataset having to be derived by learning from the old dataset. During the learning process, the score function was the F-measure achieved by each refinement of the portion of the training data related to $S \times T$ selected for training (dubbed $S' \times T'$ previously). The F-measures reported are those achieved by LS on the test dataset. We used the same settings for EAGLE and EUCLID as in the experiments before. The results (see Table 3) show clearly that our simple operator outperforms all other approaches in this setting. Moreover, the complete version of WOMBAT reaches the best F-measure on 2 datasets and the second-best F-measure on 3 datasets. This result of central importance as it shows that WOMBAT is well suited for the task for which it was designed. Interestingly, our approach also outperforms the approaches that rely on negative examples (i.e. EUCLID and EAGLE). The complete version of WOMBAT seems to perform worse than the simple version because it can only explore a tree of depth 3. However, this limitation was necessary to test both implementations using the same hardware.

In the *third 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 Fig. 3. Our results suggest that the complete version of WOMBAT is partly more stable in its results (see ABT-Buy and DBLP-Google Scholar) and 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

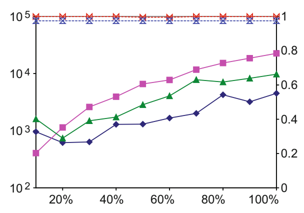
Table 3. A comparison of WOMBAT F-measure against 4 state-of-the-art approaches on 8 different benchmark datasets using 30% of the original data as training data.

Dataset	WOMBAT Simple	WOMBAT Complete	EUCLID Linear	EUCLID Conjunction	EUCLID Disjunction	EAGLE
Person 1	**1.00**	**1.00**	0.95	0.96	0.99	0.92
Person 2	**0.99**	0.79	0.80	0.82	0.88	0.69
Restaurants	**0.97**	0.88	0.87	0.84	0.89	0.88
DBLP-ACM	**0.95**	0.91	0.88	0.89	0.91	0.85
Abt-Buy	**0.44**	0.40	0.29	0.29	0.29	0.27
Amazon-GP	**0.54**	0.41	0.31	0.30	0.32	0.32
DBP-LMDB	**0.98**	**0.98**	0.97	0.96	0.97	0.89
DBLP-GS	**0.91**	0.74	0.83	0.76	0.74	0.69
Average	**0.85**	0.76	0.74	0.73	0.75	0.69

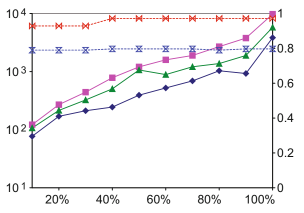
is the problem setup considered in this paper. On the other hand, the simple version is able to find better LS as it can explore longer sequences of mappings.

In the *fourth set experiments*, we measured the learning time for each of the benchmark datasets. The results are also presented in Fig. 3. 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 *fifth 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 this set of experiments). The results are presented in Table 5. Our average *pruning factor* of 2.55 shows that we can discard more than 3000 nodes while learning specifications.

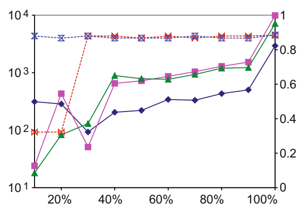
In *a final set of experiments*, we compared the two versions of WOMBAT against the 2 systems proposed in [8]. To be comparable, we used the same evaluation protocol in [8], where 2% of the gold standard was used as training data and the remaining 98% of the gold standard as test data. The results (presented in Table 4) suggests that WOMBAT is capable of achieving better or equal performance in 4 out of the 6 evaluation data sets. While WOMBAT achieved inferior F-measures for the other 2 data sets, it should be noted that the competing



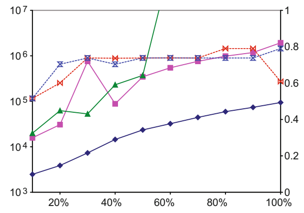
(a) Person 1



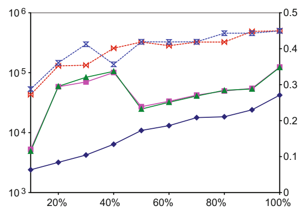
(b) Person 2



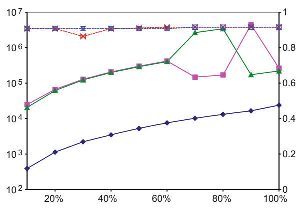
(c) Restaurants



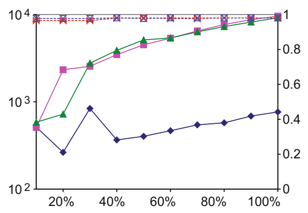
(d) ABT–Buy



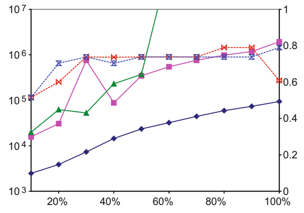
(e) Amazon–Google Products



(f) DBLP–ACM



(g) DBpedia–LinkedMDB



(h) DBLP–Google Scholar

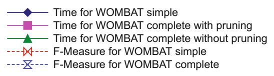


Fig. 3. Runtime and F-measure results of WOMBAT. The x-axis represents the fraction of positive examples from the gold standard used for training. The left y-axis represents the learning time in milliseconds with time out limit of 10^7 ms, processes running above this upper limit were terminated, all time plots are in log scale. The right y-axis represents the F-measure values.

systems are optimised for a low number of examples and they also get negative examples as input. Overall, these results suggest that our approach can generalise a small number of examples to a sensible LS.

Overall, our results show that ψ and φ are able to learn high-quality LS using only positive examples. When combined with our pruning algorithm, the complete version of ψ achieves runtimes that are comparable to those of φ. Given its completeness, ψ can reach specifications that simply cannot be learned by φ (see Fig. 4 for an example of such a LS). However, for practical applications, φ seems to be a good choice.

Table 4. Comparison of WOMBAT F-measure against the approaches proposed in [8] on 6 benchmarks using 2% of the original data as training data.

Dataset	Pessimistic	Re-weighted	Simple	Complete
Persons 1	**1.00**	**1.00**	**1.00**	**1.00**
Persons 2	0.97	**1.00**	0.80	0.84
Restaurants	0.95	0.94	**0.98**	0.88
DBLP-ACM	0.93	**0.95**	0.94	0.94
Amazon-GP	0.39	0.43	**0.53**	0.45
Abt-Buy	0.36	**0.37**	**0.37**	0.36
Average	0.77	**0.78**	0.77	0.74

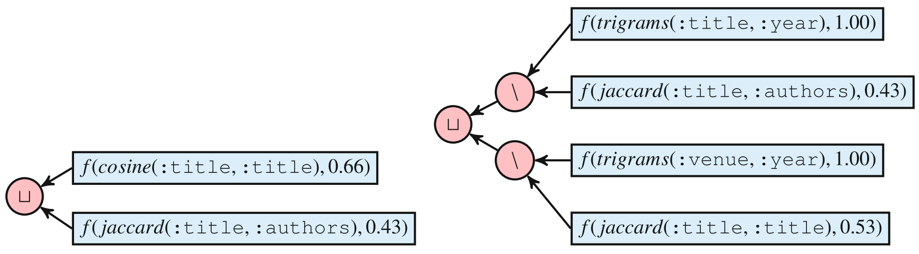
Table 5. The *pruning factor* of the benchmark datasets.

Dataset	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Person 1	1.57	2.13	1.85	2.13	2.13	2.13	2.13	2.13	2.13	2.13
Person 2	1.29	1.29	1.57	1.57	1.57	1.57	1.57	1.57	1.57	1.57
Restaurant	1.17	1.45	1.17	1.45	1.45	1.45	1.45	1.45	1.45	1.45
DBLP-ACM	6.23	5.58	6.79	6.85	6.85	6.85	6.79	6.79	6.93	6.79
Abt-Buy	3.38	3.00	3.00	3.39	3.39	3.39	1.79	3.39	3.39	3.39
Amazon-GP	1.14	1.38	1.33	1.37	1.38	1.45	1.54	1.59	1.60	1.60
DBP-LMDB	1.00	1.86	2.86	1.86	1.86	2.33	2.36	2.36	2.36	2.36
DBLP-GS	1.79	1.93	2.01	2.36	2.45	1.66	2.44	2.26	1.97	2.05

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 [13] showed that logic programs are *learnable* with arbitrarily low expected error from positive examples only. [18] introduced an algorithm for learning from labeled and unlabeled documents based on the combination of Expectation Maximization (EM) and a naive Bayes classifier. [2] provides an algorithm for learning from positive and unlabeled examples for statistical queries. The pLSA algorithm [24] extends the original unsupervised probabilistic latent semantic analysis, 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 [20]. 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 [23] some general results regarding refinement operators in quasi-ordered spaces were published. In [3] and later [4], algorithms for learning in description logics (in particular for the language $\mathcal{ALC}$) were created



⊔
f(cosine(:title, :title), 0.66)
f(jaccard(:title, :authors), 0.43)
(a) WOMBAT simple LS
⊔
\
\
f(trigrams(:title, :year), 1.00)
f(jaccard(:title, :authors), 0.43)
f(trigrams(:venue, :year), 1.00)
f(jaccard(:title, :title), 0.53)
(b) WOMBAT complete LS

Fig. 4. Best LS learned by WOMBAT for the *DBLP-GoogleScholar* data set.

which also make use of refinement operators. Recent studies of refinement operators include [11,12] which analysed properties of $\mathcal{ALC}$ and more expressive description logics. A constructive existence proof for ideal operators in the lightweight $\mathcal{EL}$ description logics has been shown in [10]. DEER [21] uses refinement operators for automatic datasets enrichment.

Most LD approaches for *learning LS* developed are supervised. One of the first approaches to target this goal was presented in [5]. 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., [7,17]). In general, these approaches assume some knowledge about the type of links that are to be discovered. For example, unsupervised approaches such as PARIS [22] aim to discover exclusively `owl:sameAs` links. Newer unsupervised techniques for learning LS include approaches based on probabilistic models [22] and genetic programming [16,19], 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.

Acknowledgments. This work has been supported by H2020 projects SLIPO (GA no. 731581) and HOBBIT (GA no. 688227) as well as the DFG project LinkingLOD (project no. NG 105/3-2) and the BMWI Project GEISER (project no. 01MD16014).

References

1. Auer, S., Lehmann, J., Ngonga Ngomo, A.-C., Zaveri, A.: Introduction to linked data and its lifecycle on the web. In: Reasoning Web, pp. 1–90 (2013)
2. Denis, F., Gilleron, R., Letouzey, F.: Learning from positive and unlabeled examples. Theoret. Comput. Sci. **348**(1), 70–83 (2005). Algorithmic Learning Theory 2000
3. Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-intensive induction of terminologies from metadata. In: McIlraith, S.A., Plexousakis, D., Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30475-3_31
4. Iannone, L., Palmisano, I., Fanizzi, N.: An algorithm based on counterfactuals for concept learning in the semantic web. Appl. Intell. **26**(2), 139–159 (2007)
5. Isele, R., Bizer, C.: Learning linkage rules using genetic programming. In: Sixth International Ontology Matching Workshop (2011)
6. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall. In: WebDB (2011)
7. Isele, R., Jentzsch, A., Bizer, C.: Active learning of expressive linkage rules for the web of data. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 411–418. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31753-8_34
8. Kejriwal, M., Miranker, D.P.: Semi-supervised instance matching using boosted classifiers. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 388–402. Springer, Cham (2015). doi:10.1007/978-3-319-18818-8_24
9. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proc. VLDB Endow. **3**(1–2), 484–493 (2010)
10. Lehmann, J., Haase, C.: Ideal downward refinement in the EL description logic. In: 19th International Conference on Inductive Logic Programming, Leuven, Belgium (2009)
11. Lehmann, J., Hitzler, P.: Foundations of refinement operators for description logics. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 161–174. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78469-2_18
12. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Mach. Learn. J. **78**(1–2), 203–250 (2010)
13. Muggleton, S.: Learning from positive data. In: Muggleton, S. (ed.) ILP 1996. LNCS, vol. 1314, pp. 358–376. Springer, Heidelberg (1997). doi:10.1007/3-540-63494-0_65
14. Ngonga Ngomo, A.-C.: Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012. LNCS, vol. 7649, pp. 378–393. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35176-1_24
15. Ngonga Ngomo, A.-C., Lyko, K.: EAGLE: efficient active learning of link specifications using genetic programming. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 149–163. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30284-8_17
16. Ngonga Ngomo, A.-C., Lyko, K.: Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: Proceedings of the Ontology Matching Workshop (2013)

17. Ngomo, A.-C.N., Lyko, K., Christen, V.: COALA – correlation-aware active learning of link specifications. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 442–456. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38288-8_30
18. Nigam, K., McCallum, A.K., Thrun, S., Mitchell, T.: Text classification from labeled and unlabeled documents using EM. Mach. Learn. **39**(2–3), 103–134 (2000)
19. Nikolov, A., dAquin, M., Motta, E.: Unsupervised learning of link discovery configuration. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 119–133. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30284-8_15
20. Shapiro, E.Y.: Inductive inference of theories from facts. In: Lassez, J.L., Plotkin, G.D. (eds.) Computational Logic: Essays in Honor of Alan Robinson. The MIT Press (1991)
21. Sherif, M.A., Ngomo, A.-C.N., Lehmann, J.: Automating RDF dataset transformation and enrichment. In: Gandon, F., Sabou, M., Sack, H., dAmato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 371–387. Springer, Cham (2015). doi:10.1007/978-3-319-18818-8_23
22. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: probabilistic alignment of relations, instances, and schema. PVLDB **5**(3), 157–168 (2011)
23. Laag, P.R.J., Nienhuys-Cheng, S.-H.: Existence and nonexistence of complete refinement operators. In: Bergadano, F., Raedt, L. (eds.) ECML 1994. LNCS, vol. 784, pp. 307–322. Springer, Heidelberg (1994). doi:10.1007/3-540-57868-4_66
24. Zhou, K., Gui-Rong, X., Yang, Q., Yu, Y.: Learning with positive and unlabeled examples using topic-sensitive PLSA. IEEE Trans. Knowl. Data Eng. **22**(1), 46–58 (2010)