Induction of Link Specifications using Refinement Operators

Klaus Lyko, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, and Mofeed Hassan

Department of Computer Science, University of Leipzig
lyko|lehmann|ngonga|mounir@informatik.uni-leipzig.de

Abstract. Links between RDF knowledge bases are a key building block of the Web of Data. However, the size and complexity of the knowledge bases at hand usually render the manual detection of links impracticable. These characteristics of the Web of Data have led to substantial research on machine learning approaches for the automatic detection of links. In this article, we develop a learning algorithm targeting the automatic discovery of links between RDF resources. In contrast to the state of the art, our algorithm is based on constructing a partial order over link specifications and the definition of a refinement operator, which allows the traversal of this partially ordered space. We describe the theoretical properties of this operator and define a learning algorithm tailored to those properties. The algorithm is compared to state-of-the-art approaches on eight datasets. Our results show that we already achieve comparable results as state-of-the-art algorithms in terms of F-measure and fast convergence towards good results.

1 Introduction

Over the last years, the Linked Data Web has evolved to encompass billions of triples distributed over more than 10,000 knowledge bases.[1] Establishing links between these knowledge bases is of central importance to realize the vision of the Data Web. Moreover, links between knowledge bases are used in manifold applications including federated query [17], keyword search and question answering engines [19] as well as data fusion and query engines. Detecting links between knowledge bases is most commonly carried out by using Link Discovery (LD) frameworks, of which most rely on link specifications (LS) to compute links.

In general, a LS can be regarded as describing a similarity condition that needs to hold between two resources to be considered to be in a given relation. Two problems are engendered by this approach to LD: First, naïve LD approaches have a quadratic time complexity. This problem has been addressed by algorithms such as blocking and the combination of specialized strategies for specific algorithms for certain metrics [1]. Second, determining accurate LS for a given linking task is usually non-trivial. Determining a good specification is then equivalent to determining the LS which maximizes a given objective function. Several supervised machine-learning approaches have been devised to address the problem to determining accurate LS. Recently unsupervised approaches have been designed. Most rely on genetic programming [14, 16] or hierar-

[1] http://stats.lod2.eu

chical search [14]. Most unsupervised LD approaches rely on a so-called pseudo-F-measure (PFM), which is used to determine the quality of a given LS.

However, state-of-the-art unsupervised approaches present two main drawbacks: First, most of these approaches are non-deterministic [14, 16]. They frequently generate suboptimal specifications and can be less time-efficient than deterministic approaches. Yet, they can use all operators available to constructing complex LS. Second, current deterministic approaches are most commonly limited to using a single type of operator to build a LS. In this paper, we provide a novel approach dubbed LION (**Li**nk Specification Induct**ion**), which can be used to learn LS. Our approach relies on a novel refinement operator which allows the *efficient generation* of specifications containing *several operators* while *remaining deterministic*. Overall, our contributions are:

- The definition of a link specification refinement operator and an investigation of its theoretical properties.
- A learning algorithm, which is able to use redundant refinement operators and iteratively explore the search space via the ability to expand the same node several times.
- The derivation of upper bounds for the objective function used by the approach in unsupervised interlinking. This upper bound is used to improve the efficiency of the learning algorithm.
- 8 experiments comparing LION's performance with the state of the art in terms of F-Measure.

The extended version of this paper presents the complete formalism and evaluation of LION.[2] The rest of this paper is organized as follows: We begin by introducing the link specification language $\mathcal{LS}$ and derive the syntax and semantics of LS. Thereafter, we present our approach. In particular, we present a refinement operator, the upper bound for the objective function we rely upon and combine these two into an approach to detecting LS. In the subsequent section, we evaluate our approach on eight different datasets and compare it with state-of-the-art approaches for learning LS. We round up the paper with a summary of the main results.

2 Preliminaries: Syntax and Semantics of Link Specifications

The goal of this section is to present LS formally. We begin by giving a short overview of RDF, the data representation language for the Web of data. Thereafter, we present the syntax and semantics of LS in detail.

RDF is based on the notion of using URIs[3] to label entities from the real world uniquely within knowledge bases. Entities from the real world are called resources. Formally, let K be a finite RDF knowledge base. K can be regarded as a set of triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{L} \cup \mathcal{B})$, where $\mathcal{R}$ is the set of all RDF resources, $\mathcal{B}$ is the set of all blank nodes ($\mathcal{B} \cap \mathcal{R} = \emptyset$), $\mathcal{P}$ the set of all predicates ($\mathcal{P} \subseteq \mathcal{R}$) and

[2] Available at `http://svn.aksw.org/papers/2016/ISWC_LION/LION_Technical_Report_public_2016.pdf`

[3] `http://www.w3.org/Addressing/`

$\mathcal{L}$ the set of all literals ($\mathcal{L} \cap \mathcal{R} = \emptyset$). The aim of LD is to discover the set $\{(s,t) \in S \times T : Rel(s,t)\}$ provided an input relation Rel, a set $S \subseteq \mathcal{R}$ of source resources (for example descriptions of persons) and a set $T \subseteq \mathcal{R}$ of target 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 [10, 4, 16]. 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.

LS	[[LS]]
$f(m,\theta,M)$	$\{(s,t,m(s,t))\|(s,t) \in M \wedge m(s,t) \geq \theta\}$
$LS_1 \sqcap LS_2$	$\{(s,t,r) \mid (s,t,r_1) \in [[L_1]] \wedge (s,t,r_2) \in [[L_2]] \wedge r = \min(r_1,r_2)\}$
$LS_1 \sqcup LS_2$	$\left\{(s,t,r) \mid \begin{cases} r = r_1 \text{ if } \exists(s,t,r_1) \in [[L_1]] \wedge \neg(\exists r_2 : (s,t,r_2) \in [[L_2]]), \\ r = r_2 \text{ if } \exists(s,t,r_2) \in [[L_2]] \wedge \neg(\exists r_1 : (s,t,r_1) \in [[L_1]]), \\ r = \max(r_1,r_2) \text{ if } (s,t,r_1) \in [[L_1]] \wedge (s,t,r_2) \in [[L_2]]. \end{cases}\right\}$

Table 1: Link Specification Syntax and Semantics

Without loss of generality, we define a similarity measure m as a function $m : S \times T \rightarrow [0,1]$ (e.g., the edit similarity dubbed `edit`[4]) 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. and write $m(p_s,p_t)$. We use *mappings* $M \subseteq S \times T \times [0,1]$ to store the results of the application of a similarity function to $S \times T$ or subsets thereof. The set of all mappings is denoted by $\mathcal{M}$. We define a *filter* a function $f(m,\theta,M)$ with $f(m,\theta,M) = \{(s,t,m(s,t))|(s,t) \in M \wedge (m(s,t) \geq \theta)\}$.

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 by an *operator* that allows merging the results of L_1 and L_2. In the following, we limit ourselves to the operators $\sqcap$ and $\sqcup$ which are frequently used in link specifications. We define the semantics of link specifications as given in Table 1. Those semantics are similar to those used in query languages like SPARQL, i.e., they are defined extensionally through the mappings they generate. The mapping $[[LS]]$ of a link specification LS with respect to $S \times T$ and a threshold θ contains the links that will be generated by LS. Those mappings are later used to define a partial order over link specifications.

Based on the defined grammar of link specifications, we can regard all LS as *trees* $L = (V(L), E(L))$ which abide by the following restrictions: (1) The leaves of L are always filter nodes that run on $S \times T$. (2) All other nodes of L are operator nodes. An example of a complex LS is given in Figure 1.

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

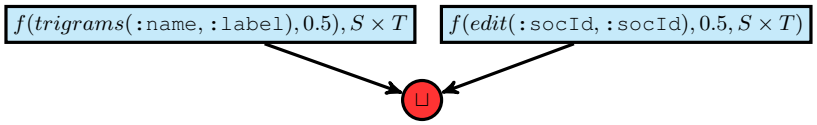

Fig. 1: Example of a complex link specification. The filter nodes are rectangles while the operator nodes are circles. `:socID` stands for social security number.

3 A Refinement Operator for Link Specifications

In our approach, we consider the language $\mathcal{LS}$, which contains all LS as defined above with the two logical operators $\sqcap$ and $\sqcup$. We also assume a fixed set of similarity metrics SM and a finite set Θ of threshold values for each similarity measure. The following definitions will introduce the refinement operator for link specifications.

Definition 1 (Refinement Operator). *A* quasi-ordering *is a reflexive and transitive relation. In a quasi-ordered space $(S, \preceq)$ a* downward (upward) refinement operator *ρ is a mapping from S to 2^S, such that for all $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a* specialisation *(*generalisation*) of C. $C' \in \rho(C)$ is usually denoted as $C \rightsquigarrow_\rho C'$.*

Definition 2 (Linkspec Subsumption). *A link specification LS is subsumed by a link specification LS', denoted by $LS \sqsubseteq LS'$, if for all S and T, we have $(s, t, r) \in [[LS]]_{S,T}$ implies $(s, t, r') \in [[LS']]_{S,T}$.*

Following definition 1 and 2, we call a refinement operator in $(\mathcal{LS}, \sqsubseteq)$, i.e., the space of link specifications and the introduced subsumption relation, a *link specification refinement operator*.

Definition 3 (Refinement Chains). *A* refinement chain *of a link specification refinement operator ρ of length n from a link specification LS to a link specification LS' is a finite sequence $LS_0, LS_1, \ldots, LS_n$ of link specs, such that $LS = LS_0, LS_1 \in \rho(LS_0), LS_2 \in \rho(LS_1), \ldots, LS_n \in \rho(LS_{n-1}), LS' = LS_n$. This refinement chain* goes through *LS'' iff there is an i ($1 \leq i \leq n$) such that $LS'' = LS_i$. We say that LS'' can be reached from LS by ρ if there exists a refinement chain from LS to LS''. $\rho^*(LS)$ denotes the set of all link specs, which can be reached from LS by ρ. $\rho^m(LS)$ denotes the set of all link specs, which can be reached from LS by a refinement chain of ρ of length m.*

Definition 4 (properties of LS operators). *An $\mathcal{LS}$ link specification refinement operator ρ is called*

- (locally) finite *iff $\rho(LS)$ is finite for all link specs $LS \in \mathcal{LS}$;*
- redundant *iff there exists a refinement chain from a link specification $LS \in \mathcal{LS}$ to a link specification $LS' \in \mathcal{LS}$, which does not go through (as defined above) some link specification $LS'' \in \mathcal{LS}$ and a refinement chain from LS to a link specification equal to LS', which does go through LS'';*

- proper *iff for all link specs* $LS \in \mathcal{LS}$ *and* $LS' \in \mathcal{LS}$, $LS' \in \rho(LS)$ *implies* $LS \not\equiv LS'$*;*
- ideal *iff it is finite, complete (see below), and proper.*

An link specification upward refinement operator ρ *is called*

- complete *iff for all link specs* $LS \in \mathcal{LS}, LS' \in \mathcal{LS}$ *with* $LS \sqsubset LS'$ *we can reach a link spec* $LS'' \in \mathcal{LS}$ *with* $LS'' \equiv LS$ *from* LS' *by* ρ*;*
- weakly complete *iff for all link specs* $\bot \sqsubset LS$ *we can reach a link specification* LS' *with* $LS' \equiv LS$ *from* $\bot$ *(most specific link spec) by* ρ*.*

We will now define the refinement operator ρ for link specifications. Let the following be given: SM is the set of similarity metrics. $\bot$ is the empty link spec. $\Delta = S \times T \times 0$. A function $dt : SM \times [0,1] \rightarrow [0,1]$ (decrease threshold) is assumed to exist, which takes a similarity measure σ and a value between 0 and 1 as input and decreases the value. We assume an implementation of dt, which allows to reach all relevant threshold values for each similarity measure.

The used refinement operator ρ^{cl} is then defined as follows: $LS' \in \rho^{cl}(C)$ iff there exists a refinement chain $LS \rightsquigarrow_\rho LS_1 \rightsquigarrow_\rho \ldots \rightsquigarrow_\rho LS_n = LS'$ such that $LS \not\equiv LS'$ and $LS_i \equiv LS$ for $i \in \{1, \ldots, n-1\}$ with ρ defined as in Figure 2.

$$\rho(LS) = \begin{cases} \{f(m_1, 1, \Delta) \sqcap \cdots \sqcap f(m_n, 1, \Delta) \mid m_i \in SM, 1 \leq i \leq n, n \leq 2^{|SM|}\} \\ \quad \text{if } LS = \bot \\ f(m, dt(\theta), M) \cup LS \sqcup f(m', 1, M)(m \in SM, m \neq m') \\ \quad \text{if } LS = f(m, \theta, M) \text{ (atomic)} \\ LS_1 \sqcap \cdots \sqcap LS_{i-1} \sqcap LS' \sqcap LS_{i+1} \sqcap \cdots \sqcap LS_n \text{ with } LS' \in \rho(LS_i) \\ \quad \text{if } LS = LS_1 \sqcap \cdots \sqcap LS_n (n \geq 2) \\ LS_1 \sqcup \cdots \sqcup LS_{i-1} \sqcup LS' \sqcup LS_{i+1} \sqcup \cdots \sqcup LS_n \text{ with } LS' \in \rho(LS_i) \\ \quad \text{if } LS = LS_1 \sqcup \cdots \sqcup LS_n (n \geq 2) \quad \cup LS \sqcup f(m, 1, M) \\ \quad (m \in SM, m \text{ not used in } LS) \end{cases}$$

Fig. 2: Definition of the refinement operator ρ.

The operator ρ makes a case distinction on the input it receives. We can differentiate 4 cases. In case of the bottom link specification $\bot$, it creates arbitrary conjunctions with thresholds of 1. We will later explain how those are constructed stepwise for efficiency reasons. In case $\bot$ is refined the first time ($n = 1$) it creates all atomic link specifications. Second, if the operator refines an atomic link specification, it performs two kinds of operations: a) modify the threshold of the link specification or b) create a disjunction. If the input link specification is complex we can differentiate two cases: In case the input is a conjunction of link specifications, the operator can recursively be applied to each element of the conjunction. In case of a disjunction, it can also be recursively applied to elements of the disjunction, but in addition it can also extend the

disjunction with an atomic link specification which is not yet used in any other element of the disjunction.

Proposition 1. *ρ is an upward refinement operator which is weakly complete, redundant and finite. ρ is redundant, not complete and not proper.*

In the extended version of this paper, six proofs / counterexamples for each claim in this proposition are provided in the appendix. Figure 3 shows an example refinement chain generated by ρ. First, it compares social security numbers via edit distance and then proceeds to take the name into account using the trigram similarity measure.

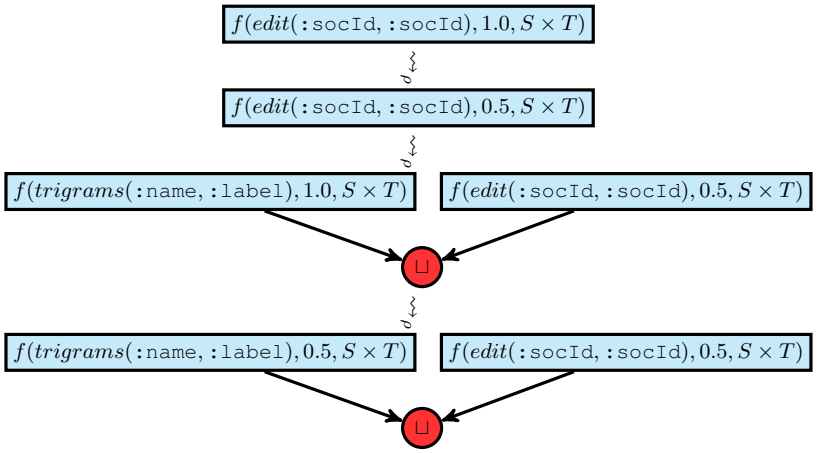

Fig. 3: Example for a ρ refinement chain. In this case, a link specification comparing social security IDs using the edit distance similarity metric and threshold 1 (only equal IDs accepted) is iteratively refined to a link specification using a disjunction of two atomic similarity measures in which the left part compars the value of the property `name` in the source dataset with the value of property `label` in the target dataset using the trigram similarity measure.

The design of the operator is inspired by similar techniques for induction of logic programs and description logic concepts [8]. In fact, the formal approach we took in this effort allows to carry over ideas from those disciplines and we believe it will also give rise to further optimisations on top of those proposed herein.

4 Objective Function and its Upper Bound

In order to optimise the efficiency of the learning algorithm, it is beneficial to be able to compute a so-called *upper bound* for the objective function value of a link specification. The objective function is the criterion, which the learning algorithm aims to maximise – in our case standard F-measure in case of supervised learning and pseudo

F-measure in case of unsupervised learning. Obviously, each link specification LS that is investigated by the learning algorithm can be assigned an objective function value. Since we perform upward refinement using ρ^{cl}, we know that any link specification obtained via refinements of LS is more general (with respect to our partial order $\sqsubseteq$). This allows us to compute upper bounds on what objective function values can be achieved in refinements of LS.

The idea behind unsupervised approaches to learning link specifications is to refrain from using any training data. Instead, unsupervised approaches aim to optimize an objective function. The objective functions we consider herein approximate the value of the F-measure achieved by a specification and are thus dubbed pseudo-F-measure (PFM) [16]. In this work we use the extended PFM definition of [14]. Our pseudo-precision $\mathcal{P}$ computes the fraction of links that stand for one-to-one links while our pseudo-recall $\mathcal{R}$ computes the fraction of the total number of resources from those that are involved in at least one link. Let λ_S be the subset of S whose elements are linked to at least one element of T.

Formally $\mathcal{P}$ is computed by dividing the sum of λ_S and λ_T by the total number of links in M:

$$\mathcal{P}(M) = \frac{|\lambda_S| + |\lambda_T|}{2 \cdot M}, \mathcal{R}(M) = \frac{|\lambda_S| + |\lambda_T|}{S + T}. \tag{1}$$

Finally, the PFM $\mathcal{F}_\beta$, is defined as

$$\mathcal{F}_\beta = (1 + \beta^2)\frac{\mathcal{P}\mathcal{R}}{\beta^2\mathcal{P} + \mathcal{R}}. \tag{2}$$

In order to improve the efficiency of the algorithm by ignoring parts of the search space which cannot improve PFM, we need to compute the maximum achievable PFM ($\mathcal{F}_\beta^{\max}$) for a link specification. Note that we have an upward refinement operator, which means that refinements of a link specification LS will always generate at least as many links as LS generated already. Therefore, we need to find the set M' with $M \subseteq M'$, which maximizes the pseudo F-Measure. Naturally, when the maximum achievable pseudo-F-measure of a link specification in the algorithm search tree is lower than an already achieved pseudo-F-measure, no further refinements of that link specification need to be performed and the search tree can be pruned at that point to improve efficiency. We can compute $\mathcal{F}_\beta^{\max}$ as follows: The unlinked resources in S for a mapping M can be computed as $S - \lambda_S$. Let us assume without loss of generality that the number of unlinked resources in S is less or equal to the number of unlinked resources in T. We construct a mapping M' as an extension of M such that all non-linked resources of S in M are now connected via one-to-one links until all are connected: $links(S, M') = S$ ("filling up"). This leads to pseudo-precision and pseudo-recall formula, which we can express using the same terms as in the definition of pseudo-precision and recall for the original mapping M:

$$\mathcal{P}(M') = \frac{2 \cdot |S| + |\lambda_T| - |\lambda_S|}{2 \cdot (|M| + |S| - |\lambda_S|)}, \tag{3}$$

$$\mathcal{R}(M') = \frac{2 \cdot |S| + |\lambda_T| - |\lambda_S|}{|S| + |T|}. \tag{4}$$

In M', all resources in S are linked and the remaining task is to observe the effect of connecting so far unlinked resources in T. Connecting those resources always increases pseudo-recall, but lets precision converge towards 50%, i.e. can either increase or decrease depending on its current value. Assume we create mappings M^x as extensions of M' in which x so far unlinked resources in T are linked to S. The maximum pseudo-F-measure obtained this way is $\mathcal{F}_\beta^{\max}$ (see Appendix B of the full technical report for the proof):

$$\mathcal{F}_\beta^{\max} = \max_{x=0}^{(|T|-|\lambda_T|)-(|S|-|\lambda_S|)} (\mathcal{F}_\beta(M^x)) \tag{5}$$

5 Learning Algorithm

The interlinking learning problem can be defined as follows:

Definition 5 (Learning Problem). *Given two sets of resources S and T, a (precision-recall-tradeoff) parameter β and a set of similarity measures, find $LS \in \mathcal{LS}$ such that $\mathcal{F}_\beta([[LS]])$ is maximal.*

A learning algorithm can be constructed as a combination of a refinement operator, which defines how the search tree can be built, and a search algorithm, which controls how the tree is traversed and which nodes have to be expanded. At the core, we follow similar algorithms in Inductive Logic Programming and Concept Learning in Description Logics.

We have to tackle two important problems for defining the learning algorithm:

1. ρ is redundant.
2. While ρ is finite, the number of refinements can be very large.

Redundancy For avoiding the negative effects of redundancy, we check for each node in the tree whether an equivalent node exists elsewhere in the search tree. If yes, then this node is ignored. Using this method, all link specs occur only once in the search tree. Note that this does not affect the completeness of the algorithm, since at least one refinement chain to a possible solution will not be blocked (see completeness construction in appendix). For performing the redundancy check efficiently, we convert all link specs to a normal form fixing the order of elements in conjunctions and disjunctions. Using binary search, checking whether a link specification exists in the search tree only requires logarithmically many syntactical comparisons of link specifications, which themselves can be carried out in linear time due to the normal form.

Iterative Refinement In our proposed learning algorithm, the refinement operator ρ is used for building the search tree, while a heuristic decides which nodes to expand. We can tackle the potentially large number of refinements generated by the operator by using an iterative approach, in particular by only considering refinements up to link specification size n. The size of a link specification is the number of its atomic measures. We call this the *size expansion* of a node in the search tree. This size limit for

each node is incrementally increased each time the refinement operator is applied on a node.

Formally, we define a *node* in a search tree to be a quadruple (LS, n, r, c), where LS is a link spec, $n \in \mathbb{N}$ is the size expansion, $r \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node and c is another boolean marker, which is true, when all refinements for that node have been generated.

The search heuristic selects the node with the highest score in the search tree, where the score of a node is defined as follows:

Definition 6 (score). *Let $N = (LS, n, r, c)$ be a node in the search tree. The score of N is $F_\beta[[LS]] - n \cdot \alpha$, where α is a penalty for the expansion of the node.*

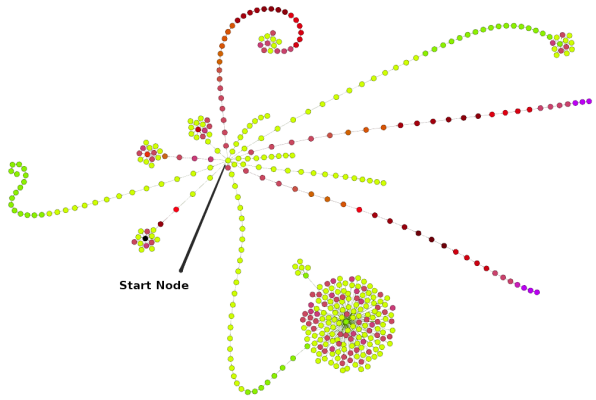

Fig. 4: Heatmap of a search tree in a LION algorithm run (nodes are link specifications and edges obtained via the refinement operator). Green nodes have low objectivw function values and red nodes high objective function values. This run started at the center at which atomic specifications with several different metrics are created. A sequence of threshold refinements are then applied to the same metric, which result in the longer "linear" sequences and, ultimately, in "clouds" of more complex link specifications. A frequent observation when iteratively constructing those heatmap visualisations is that LION explores several parts of the search space in parallel and, therefore, is not very prone to falling into local maxima.

Algorithm 1 shows how iterative refinement, redundancy checking and the upper bound for F-Measure are incorporated in the learning algorithm. The F-Measure bounds are used to ignore parts of the search tree, which cannot lead to an improvement of F-Measure (realised in Line 5 of the algorithms which only considers nodes which have a upper bound higher than the currently best F-Measure). Compared to other algorithms learning over structured data, this has the benefit that it adapts itself to the task at hand: Pruning is more aggressive for easier link tasks, i.e. when high F-Measures can be found early at algorithm runtime, whereas it will be less aggressive for harder tasks.

As a consequence of the weak completeness of the refinement operator, the algorithm is guaranteed to return the link specifications in $\mathcal{LS}$ given infinite resources. In practice, only small parts of $\mathcal{LS}$ can be explored and different termination criteria, e.g. a maximum runtime, can be used.

Algorithm 1: Learning algorithm

Input: Sets of resources S and T, β, SM
Result: Link Specification LS

ST (search tree) is set to the tree consisting only of the root node $(\top, 0, \textit{false}, \textit{false})$;
$F_{best} = 0$, $LS_{best} = null$;
Compute property matches;
while *termination criterion not met* **do**
    Choose a node $N = (LS, n, r, c)$ with highest score in ST, $r = \textit{false}$, $c = \textit{false}$ and $F_{\beta}^{\max}[[LS]] \geq F_{best}$;
    if *N==null, i.e. no such node exists* **then**
        return LS_{best}
    Expand N up to size $n + 1$;
    begin
        $ST.addAll(\{(LS', n, \textit{checkRed}(ST, LS'), false) | LS' \in \rho(LS)\})$,i.e. add nodes with refined link specifications as children of N;
        Update F_{best} and LS_{best};
        if F_{best} *has increased* **then**
            prune(ST);
        Evaluate created non-redundant nodes;
        $N = (LS, n + 1, r, c)$;
return LS_{best};

6 Experiments

We evaluated our approach on eight benchmark datasets with respect to quality of the generated link specification. We ran our experiments on five real-world and three synthetic interlinking tasks. The real-world interlinking tasks are DBLP - ACM, Abt-Buy, Amazon-Google Products, DBpedia - Linked Movie Database and DBLP - Google Scholar and were extracted from websites or databases (see [7]). The synthetic datasets were taken from the OAEI 2010 benchmark[5]. All experiments were carried out on a single machine with an 2GB allocated memory on a server with an AMD Opteron Quadcore Processor (2GHz) running Java 7. As references, we compared the F-measure achieved by our approach with the deterministic learning approach EUCLID [14] and the non-deterministic genetic-programming-based approach EAGLE [14, 13]. EUCLID regards link discovery as a classification problem and learns both linear and boolean

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

classifiers [12, 14]. The boolean classifiers are based either on conjunctions or disjunctions. Therefore, three different EUCLID approaches (linear, conjunctive, disjunctive) exist and we compare against each of those. EAGLE is an evolutionary approach for learning link specifications, which has previously shown to be competitive in terms of F-measure and more time-efficient than other learning algorithms [14]. For all four approaches we compare against, we used the Levenshtein, Trigrams, Cosine and Jaccard Index as similarity measures.

We conducted preliminary experiments to determine the most appropriate parameter setting. For LION we ran experiments using different parameters and choose the best performing parametrisation over all datasets. LION was tested with all expansion penalties in the interval of $[0.5, 1.0]$ using steps of 0.05. We concluded that an expansion penalty of 0.95 leads to the best results over all datasets. As suggested in [14] EAGLE was set to use a population of 20, mutation and crossover probability were set to 0.6 and was as LION restricted to only construct complex link specifications using the operators $\sqcap$ and $\sqcup$. EUCLID's granularity factor was set to 5.

Throughout all experiments we use a balanced precision - recall - tradeoff parameter of $\beta = 1$. We ran all experiments for ten minutes and logged the results (achieved F-Score and best specifications learned) of each approach for all iterations/loops or generations they were able to carry out in this time. Because EAGLE is non-deterministic its results are presented as average over 5 different runs.

7 Results

7.1 Unsupervised Learning

We will first discuss the results of the experiments on unsupervised learning. Table 2 shows the outcomes of the unsupervised learning approaches[6]. For each experiment and approach we present the F-Measure and corresponding pseudo-F-Measure of the best learned link specification after 10 minutes. Note, that the results for EAGLE are the means of 5 different runs.

The presented algorithm LION performed well on most benchmarks. It achieved the best results for three data sets and ties as best approach with EUCLID (linear) on two datasets. On the other data sets EUCLID linear and conjunctive achieved highest F-Measures. Generally, the achieved F-Measures are fairly close. EAGLE was outperformed by either algorithm, but as its results are means of 5 runs, it is able to learn high quality link specifications in theory during individual runs. As a matter of fact, single runs of EAGLE achieved results comparable to those of LION and EUCLID. Overall, LION was able to slightly outperform all other approaches and won against the second best algorithms EUCLID conjunctive by a margin of 1% F-Measure. While this result is already encouraging, the experiments revealed that subtle differences in thresholds can lead to significant performance differences. As a consequence, we plan to include and

[6] Note, we also conducted further experiments using a supervised learning approach. Results are available in the extended version of this paper, which can be found at our project website `http://aksw.org/Projects/LION.html`.

Dataset		LION	EAGLE	EUCLID lin.	EUCLID conj.	EUCLID disj.
Person 1	F	**100**	99.6	**100**	99.50	68.87
	PFM	100	99.6	100	99.49	100
Person 2	F	**95.69**	85.92	80.08	89.66	79.44
	PFM	54.27	50.79	55.20	50.42	54.86
Restaurant	F	**56.58**	51.22	**56.58**	**56.58**	55.80
	PFM	46.57	39.71	46.57	46.57	48.12
DBLP-ACM	F	**91.36**	90.62	90.99	91.09	91.10
	PFM	87.74	87.04	87.75	87.71	87.71
Abt-Buy	F	**34.93**	34.87	33.22	33.22	34.04
	PFM	41.06	41.10	39.61	39.61	42.00
Amazon-Google Product	F	33.60	35.09	**35.76**	33.30	31.36
	PFM	40.17	40.78	40.66	39.57	45.08
DBpedia-LinkedMDB	F	97.87	97.72	**98.21**	97.67	97.67
	PFM	97.72	97.60	97.82	97.57	97.57
DBLP-Google Scholar	F	49.86	33.04	**50.08**	**50.08**	42.94
	PFM	22.63	21.87	22.56	22.56	22.64
Overall	ø(F)	69.99	66.01	68.12	68.89	62.65
	ø(PFM)	61.27	59.81	61.27	60.44	62.25

Table 2: Results of 5 different unsupervised approaches on 8 benchmarks. For each dataset we present the F-Measure (F) and pseudo-F-Measure (PFM) of the best Link Specification learned after 10 minutes. The last row presents the mean F-Measure and PFM over all datasets.

test several black box parameter optimisation algorithms into LION in order to verify whether performance gains can be obtained by fine-tuning threshold.

Table 3 shows the area under the F-measure time curve (AUFTC) values. Greater AUFTC values indicate faster convergence of the algorithm towards high F-Measures. The table clearly shows that LION outperforms the state of the art w.r.t. its convergence to high F-measure.

For all data sets EUCLID disjunctive performed fewest iterations in 10 minutes and in three cases only one. Thus, its AUCFT values suffer. This is due to the fact that executing disjunctive link specifications on large mappings is computationally expensive.

7.2 Supervised Learning

Table 4 shows the supervised learning results. As the quality of the supervised algorithms is highly dependent on the training data, we compare all approaches on three randomly selected set of training data and present the results as means of these three runs in table 4. LION achieves the best overall performance together with EUCLID disjunctive. Both achieve the best result in 4 and 3 link discovery tasks, respectively. The consistently low F-Measures for two out of 8 datasets (Amazon-GoogleProducts and Abt-Buy) across all algorithms and learning modes suggests that satisfying link specifications for those tasks may not exist.

	EUCLID lin.	EUCLID conj.	EUCLID disj.	EAGLE	LION
Persons1	575	565	207	584	**597**
Persons2	481	526	237	524	**573**
Restaurants	338	338	331	**367**	347
DBLP-ACM	204	**551**	**551**	544	542
Abt-Buy	198	198	204	194	**208**
Amazon-GoogleProducts	258	246	243	284	**353**
DBLP-LinkedMDB	583	585	581	568	**587**
DBLP-GoogleScholar	247	234	129	99	**261**
Sum	2883	3243	2482	3164	**3469**

Table 3: Area under the F-measure time curve (AUFTC) of the unsupervised approaches on all datasets.

LION or EUCLID achieved the highest F-Measures every time. On average, they outperform EAGLE by about 5% F-Measure. LION performed particularly well for the Abt-Buy data set achieving 4% better F-Score then EUCLID (33.26 % vs. 28.91 %). The DBLP-Google Scholar dataset is the only time EUCLIDs linear classifier achieved the highest F-Measures (82.74%) and beat LION by 6%. For the other datasets LION and EUCLID share the best F-Measures within a 4% margin. Over all experiments both LION and EUCLID disjunctive best optimize the objective function at hand (compare the ø(PFM) row in table 2 and the ø(F) row of table 4). While on average both LION and EUCLID learn comparably good link specifications it is hard, given a specific interlinking task, to decide which algorithm to prefer. This questions remains open for future work. Furthermore, as many link specifications are sensitive to the threshold a more accurate threshold adjustment method may improve the results.

8 Related Work

This work is related to LD and refinement operators. LD frameworks aim to tackle two main problems: the runtime of LD and the learning of accurate LS. Time-efficiency has been addressed in a variety of ways in previous works. For example, the LIMES framework [11] provides time-efficient algorithms for running specific atomic measures (e.g., PPJoin+ [24] and $\mathcal{HR}^3$ [10]) and combines them by using set operators and filters. SILK [4] relies on the lossless blocking algorithm MultiBlock to execute LS efficiently. Multiblock allows mapping a whole link specification in a space that can be segmented to overlapping blocks. The similarity computations are then carried out within the blocks only. A similar approach is followed by the KnoFuss system [16]. Other time-efficient systems include [22] which present a lossy but time-efficient approach for the efficient processing of LS.

The learning of accurate link specification is the main topic of this work and was addressed by supervised and newly even unsupervised approaches. For example, the approach presented in [3] relies on large amounts of training data to detect accurate link specification using genetic programming. RAVEN [12] is (to the best of our knowledge)

Data set		LION	EAGLE	EUCLID lin.	EUCLID conj.	EUCLID disj.
Person 1	F	**99.81**	91.94	94.82	96.12	99.08
	$\sigma(F)$	1.19	11.27	4.73	4.2	1.42
Person 2	F	84.47	69.01	80.31	82.32	**88.30**
	$\sigma(F)$	0.72	4.98	10.70	6.06	7.35
Restaurant	F	87.61	87.69	87.06	83.70	**88.56**
	$\sigma(F)$	5.55	1.5	0.66	6.55	0
DBLP-ACM	F	**90.91**	85,26	87.65	89.23	90.52
	$\sigma(F)$	3.97	8.45	1.92	2.15	0.64
Abt-Buy	F	**33.26**	26.53	28.91	28.91	28.91
	$\sigma(F)$	0	2.57	3.51	3.51	3.51
Amazon-GoogleProducts	F	29.02	31.70	30.85	30.55	**31.81**
	$\sigma(F)$	0	1.76	0.47	0.12	0.79
DBpedia-LinkedMDB	F	**97.61**	88.28	96.75	96.48	97.59
	$\sigma(F)$	0.22	8.16	2.34	2.07	0.14
DBLP-Google Scholar	F	76.72	69.38	**82.74**	76.00	74.44
	$\sigma(F)$	0	2.58	0.86	1.85	0
Overall	ø(F)	74.93	68.72	73.64	72.91	74.90

Table 4: F-Measure results of 5 different approaches on 8 benchmarks using a batch learning approach with 30% trainings data, the results are the means of 3 different runs using different training data. Each row shows the mean F-Measure F and its standard deviation σ. The last row presents the mean F-Measure over all data sets and runs.

the first active learning technique for LD. The approach was implemented for linear or Boolean classifiers and shown to require a small number of queries to achieve high accuracy. While the first active genetic programming approach was presented in [2], similar approaches for LD were developed later [5, 13]. A newly presented semi-supervised instance matching approach only requires a minimal set of training data to bootstrap and uses the boosting machine learning paradigm to maximizing the performance of several weighted classifiers over several iterations [6]. Newly developed for learning LS (see, e.g., [16, 9, 14]) abide by the paradigm of unsupervised machine learning. Other systems descriptions can be found in the results of the Ontology Alignment Evaluation Initiative.[7]

Refinement Operators have been used mainly in machine learning. In the area of Inductive Logic Programming considerable efforts have been made to analyse the properties of refinement operators (for a comprehensive treatment, see e.g. [15]). Through the definition of a partial order based on the semantics of link specifications, we were able to draw on this work. In Shapiro's Model Inference System [18], he describes how refinement operators can be used to adapt a hypothesis to a sequence of examples. In the following years, refinement operators became widely used. [23] found some general properties of refinement operators in quasi-ordered spaces. Nonexistence conditions for ideal refinement operators relating to infinite ascending and descending refinement

[7] `http://ontologymatching.org`

chains and covers have been developed. The result has been used to show the non-existence of ideal refinement operators for clauses ordered by θ-subsumption. Within the last decade, several refinement operators for description logics have been investigated. The most fundamental work is [8], which shows for many description languages the maximal sets of properties which can be combined. Among other things, a non-ideality result for the languages $\mathcal{ALC}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$ is shown. Although more detailed investigation is required, those results potentially carry over to link specifications. [8] also outlines approaches for handling infinite and redundant operators, which influenced the design of our learning algorithm. [20] present a refinement operator to learn RDF data enrichment pipelines.

[21][8] performs link specification learning using refinement operators as we do in this submission. A major difference is that the paper focuses on positive only learning, i.e. only positive examples are given, whereas our focus is on unsupervised learning as well as supervised learning from positive and negative examples. The LS language used is different as it supports the intersection of link specifications. While this increases expressivity, it can affect the runtime and memory usage of the algorithm severely. It also leads to a different design and properties of the refinement operator. Our operator is finite and can in general derive longer link specifications. Apart from the language, this is also due to our approach using a more sophisticated pruning procedure by deriving upper bounds for pseudo F-measure.

9 Conclusions and Future Work

We presented a novel approach for finding a LS which maximises the F-Measure as objective function. Using upper bounds for the objective function, the algorithm is able to prune parts of the search space and thereby exploit the partial order defined over them. A series of experiments on 8 benchmark tasks has shown competitive results in terms of F-Measure and runtime. We believe this work lays a foundation for several lines of research starting from it: a) Experimenting with different scoring functions to ensure that the heuristic search in the algorithm has a good tradeoff between exploration and exploitation. b) The inclusion of black box optimisation techniques for fine-tuning the thresholds of link specifications. c) Integrating the algorithm into an active learning framework for learning link specifications.

References

1. S. Auer, J. Lehmann, A.-C. N. Ngomo, and A. Zaveri. Introduction to linked data and its lifecycle on the web. In *Reasoning Web*, pages 1–90, 2013.
2. J. de Freitas, G. Pappa, A. da Silva, M. Gonç andalves, E. Moura, A. Veloso, A. Laender, and M. de Carvalho. Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, 2010.
3. R. Isele and C. Bizer. Learning Linkage Rules using Genetic Programming. In *Sixth International Ontology Matching Workshop*, 2011.

[8] `http://svn.aksw.org/papers/2016/ECAI_WOMBAT/public.pdf`

4. R. Isele, A. Jentzsch, and C. Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.
5. R. Isele, A. Jentzsch, and C. Bizer. Active learning of expressive linkage rules for the web of data. In *ICWE*, pages 411–418, 2012.
6. M. Kejriwal and D. Miranker. Semi-supervised instance matching using boosted classifiers. In F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains*, volume 9088 of *Lecture Notes in Computer Science*, pages 388–402. Springer International Publishing, 2015.
7. H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.
8. J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.
9. K. Lyko, K. Höffner, R. Speck, A.-C. Ngonga Ngomo, and J. Lehmann. Saim - one step closer to zero-configuration link discovery. In *Proc. of the Extended Semantic Web Conference Posters & Demos*, 2013.
10. A.-C. Ngonga Ngomo. Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *Proceedings of ISWC*, 2012.
11. A.-C. Ngonga Ngomo. On link discovery using a hybrid approach. *Journal on Data Semantics*, 1:203 – 217, December 2012.
12. A.-C. Ngonga Ngomo, J. Lehmann, S. Auer, and K. Höffner. Raven: Active learning of link specifications. In *Proceedings of the Ontology Matching Workshop*, 2011.
13. A.-C. Ngonga Ngomo and K. Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *Proceedings of ESWC*, 2012.
14. A.-C. Ngonga Ngomo and K. Lyko. Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *Proceedings of the Ontology Matching Workshop*, 2013.
15. S.-H. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of Inductive Logic Programming*. Lecture Notes in Computer Science. Springer, 1997.
16. A. Nikolov, M. D'Aquin, and E. Motta. Unsupervised learning of data linking configuration. In *Proceedings of ESWC*, 2012.
17. M. Saleem and A.-C. N. Ngomo. Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In *The Semantic Web: Trends and Challenges*, pages 176–191. Springer, 2014.
18. E. Y. Shapiro. Inductive inference of theories from facts. In J. L. Lassez and G. D. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 199–255. The MIT Press, 1991.
19. S. Shekarpour, A.-C. N. Ngomo, and S. Auer. Question answering on interlinked data. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, and S. B. Moon, editors, *WWW*, pages 1145–1156. International World Wide Web Conferences Steering Committee / ACM, 2013.
20. M. Sherif, A.-C. Ngonga Ngomo, and J. Lehmann. Automating RDF dataset transformation and enrichment. In *12th Extended Semantic Web Conference, Portoroz, Slovenia, 31st May - 4th June 2015*. Springer, 2015.
21. M. A. Sherif, A.-C. Ngonga Ngomo, and J. Lehmann. A generalization approach for automatic link discovery. In *Technical Report*, 2016.
22. D. Song and J. Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *ISWC*, pages 649–664, 2011.
23. P. R. J. van der Laag and S.-H. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In *Proc. of 7th Europ. Conf. on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag, 1994.
24. C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.