DRAGON: Decision Tree Learning for Link Discovery

Daniel Obraczka[1(✉)] and Axel-Cyrille Ngonga Ngomo[2(✉)]

[1] University of Leipzig, 04109 Leipzig, Germany
obraczka@informatik.uni-leipzig.de
[2] University of Paderborn, 33098 Paderborn, Germany
axel.ngonga@upb.de

Abstract. The provision of links across RDF knowledge bases is regarded as fundamental to ensure that knowledge bases can be used joined to address real-world needs of applications. The growth of knowledge bases both with respect to their number and size demands the development of time-efficient and accurate approaches for the computation of such links. This is generally done with the aid of machine learning approaches, such as e.g. Decision Trees. While Decision Trees are known to be fast, they are generally outperformed in the link discovery task by the state-of-the-art in terms of quality, i.e. F-measure. In this work, we present DRAGON, a fast decision-tree-based approach that is both efficient and accurate. Our approach was evaluated by comparing it with state-of-the-art link discovery approaches as well as the common decision-tree-learning approach J48. Our results suggest that our approach achieves state-of-the-art performance with respect to its F-measure while being 18 times faster on average than existing algorithms for link discovery on RDF knowledge bases. Furthermore, we investigate why DRAGON significantly outperforms J48 in terms of link accuracy. We provide an open-source implementation of our algorithm in the LIMES framework.

Keywords: Link discovery · Decision trees · Machine learning · Entity resolution · Semantic web

1 Introduction

RDF is now ubiquitous on the Web. For example, more than 1 billion URLs embed RDF data in different serialization formats. A representative fragment of the RDF data available on the planet, the Linked Open Data Cloud, has grown from merely 12 to almost 10,000 knowledge bases [9] over the last decade. The growth in the number of knowledge bases is accompanied by a growth in size. For example, DBpedia–one of the most popular bases–has grown from $\approx 10^8$ triples (DBpedia 2.0) in 2007 to $\approx 10^{10}$ triples in 2016 (DBpedia 2016-04). This growth in the number and size of knowledge bases has led to an increase of the necessity for efficient and accurate link discovery (i.e., the computation of links between knowledge bases) approaches.

The original version of this chapter was revised: The project number of OPAL in the acknowlegdements section was corrected. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-19274-7_51

© Springer Nature Switzerland AG 2019
M. Bakaev et al. (Eds.): ICWE 2019, LNCS 11496, pp. 441–456, 2019.
https://doi.org/10.1007/978-3-030-19274-7_31

In this paper, we consider the *declarative link discovery setting* [15], in which the set of conditions under which two resources are to be linked is to be devised explicitly and subsequently executed to compute links across two (not necessarily distinct) sets of RDF resources S and T. The declarative link discovery problem has been shown to be challenging even for domain experts, since this set of conditions, which we call a *link specification*, can be very complex [11]. This complexity is due to

1. the plethora of similarity measures (e.g., edit distance, cosine similarity, Jaccard similarity) that are used to compare the property values of entities to find links and
2. the manifold means through which these similarities can be combined (e.g., min, max, linear combinations).

A large number of dedicated machine learning algorithms ranging from genetic programming [11] to refinement operators [25] has hence been devised to simplify the declarative link discovery process (see [15] for a survey). While the F-measure of these machine learning approaches has increased steadily, little attention has been paid to their time efficiency. Most commonly, the approaches are declared scalable by virtue of the bound similarity computation algorithms they rely upon, e.g., AllPairs [1], PPJoin+ [30], EdJoin [29]. First pruning approaches are developed in works such as [25] but solely for particularly slow versions of the algorithm. Given that the time efficiency of learning approaches for link discovery is critical for their usefulness in practical applications (e.g., in learning scenarios with humans in the loop), the primary aim of this work is hence *to improve the time efficiency of link discovery algorithms, while maintaining their classification performance* (i.e. maintaining the same F-measure). We achieve this goal with our novel approach DRAGON, a decision-tree-based approach for link discovery. The contributions of this work are as follows:

1. We devise an efficient and effective algorithm for learning link specifications within the decision tree paradigm.
2. We evaluate our algorithm on nine benchmark data sets against state-of-the-art link discovery and decision-tree-learning approaches. Our results show that DRAGON outperforms existing link discovery solutions significantly w.r.t. runtime, while achieving equally good performance w.r.t. the F-measure. Moreover, our approach outperforms generic solutions to learning decision trees significantly.
3. We investigate why our approach produces better link specifications than common decision-tree algorithms.

An open-source implementation of DRAGON is provided in the LIMES [18] framework[1].

This paper is structured as follows: We start by giving a formal definition of the key concepts underlying this work. Thereafter, we give a brief overview of the state of the art. We subsequently present our approach to link discovery, which we finally evaluate on synthetic and real-world datasets.

[1] https://github.com/dice-group/LIMES/.

2 Preliminaries

Definition 1 (Link Discovery). *Given two sets S (source) and T (target) of RDF resources and a relation R, compute the set M of pairs of instances $(s,t) \in S \times T$ such that $R(s,t)$.*

We call M a mapping. Commonly, the set S is a subset of the instances contained in the knowledge base $\mathcal{K}_S$. The same applies to the set T and a knowledge base $\mathcal{K}_T$. Note that neither S and T nor $\mathcal{K}_S$ and $\mathcal{K}_T$ are necessarily disjoint. Since computing M is a non-trivial task, most frameworks compute an approximation $M' = \{(s,t) \in S \times T : \sigma(s,t) \geq \theta\}$, where σ is a similarity function and θ is a similarity threshold. The relation $R(s,t)$ is then considered to hold if $\sigma(s,t) \geq \theta$. The similarity function σ and the threshold θ are expressed in a *link specification* (LS). Different grammars have been proposed to describe LSs [12,14,19]. We adopt the following formal setting, which is akin to that of [25]. This grammar is equivalent to that used in a large body of work [15].

We begin by defining the syntax of link specifications. To this end, we define a similarity measure m to be a function $m : S \times T \to [0,1]$. These functions commonly compare attributes (or sets of attributes) of pairs of resources to compute a similarity value. Mappings $M \subseteq S \times T$ are used to store the results of the application of a similarity function to $S \times T$. We define a $filter$ as a function $f(m,\theta)$ over the set of all mappings (i.e., the powerset of $S \times T$). A link specification is called *atomic* iff it comprises exactly a single filtering function. A complex specification L can be obtained by combining two link specifications L_1 and L_2 with the operators $\sqcap, \sqcup$ and $\backslash$.

We define the semantics $[[L]]_M$ of a LS L w.r.t a mapping M as follows:

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

Moreover, we write $[[L]]$ as a shorthand for $[[L]]_{S \times T}$.

Definition 2 (Link Discovery as Classification). *The goal of link discovery can be translated to finding a classifier $\mathcal{C} : S \times T \to \{-1, +1\}$ which maps non-matches (i.e. $(s,t) \in S \times T : \neg R(s,t)$) to the class -1 and matches to $+1$.*

The classifier returns $+1$ for a pair (s,t) *iff* $\sigma(s,t) \geq \theta$ for the corresponding link specification. In all other cases, the classifier returns -1. In this work, we use decision trees for link discovery. The attributes we use are similarity measures. Because these measures have numeric values, we compute decision trees with binary splits. An example tree is shown in Fig. 1.

3 The DRAGON Algorithm

In the following, we present how we use decision trees to learn accurate LSs efficiently. We begin by giving a brief overview of our approach. Thereafter, we show how we tailor the construction of decision trees to the LS learning problem. Finally, we show how to prune trees to avoid overfitting.

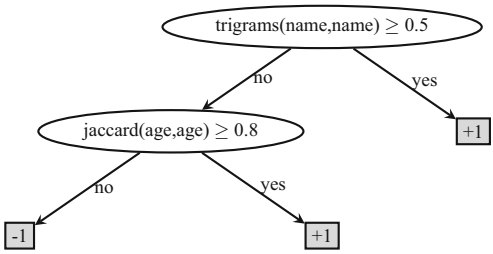

Fig. 1. A decision tree learned by DRAGON for the dataset of which a part is shown in Table 1. The classification decisions are shown in the gray nodes.

Table 1. Example datasets containing persons and link candidates with appropriate labeling showing corresponding similarity values using cosine similarity on name attributes. We represent the data as tables and omit namespaces for the sake of legibility.

URI	name	age
:S1	"Hans-Peter"	26
:S2	"Heiko Kurt"	13
:S3	"Amata"	52
:S4	"Ariane"	25
:S5	"Sib"	56

(a) Source Dataset

URI	name	age
:T1	"Hans Peter"	26
:T2	"Heiko"	13
:T3	"Amata K."	52
:T4	"Ariane"	25
:T5	"Ffion"	12

(b) Target Dataset

Pair	Label	sim
(S2, T2)	+1	0.707
(S2, T3)	-1	0.0
(S3, T3)	+1	0.816
(S4, T4)	+1	1.0
(S5, T5)	-1	0.0

(c) Link candidates

3.1 Overview

Algorithms 1 and 2 give an overview of our approach depending on the measure we use. We assume that we are given a maximum height for the tree. To determine the root of the tree we detect the best split attribute. We considered two measures for this purpose: the **global F-measure** aims to maximize the total F-measure achieved by the decision tree while the **Gini** measure acts locally and aims to ensure that the split attribute is optimal. We elaborate on how we use these fitness functions to find good split attributes in the section "Determining Split Attributes". DRAGON continues recursively by constructing the left and right child of the tree if it found a split attribute and has not yet reached the maximum height. For **Gini**, it updates the training data depending on the split attribute by removing any link pairs that are not accepted by the split attribute. For example, for the decision tree shown in Fig. 1, the training data for the right child of the root node would not include the pair (S5, T5) from Table 1(c), because the similarity value of the qgrams measure on the name attribute is lower than 0.4. The training data for the left child (ergo, the "no" child, see Fig. 1) is the global training data used to initialize our approach minus the training data for the right child of the root node. For example, the pair (S4, T4) would not be contained in the training data for the left child, since the similarity value of the qgrams measure on the name attribute is 1.0, and it

therefore belongs to the training data of the right child. For **global F-measure**, the training data does not need to be changed, since we try to optimize globally.

In the following, we take a closer look at how the split attributes are computed.

Algorithm 1. Overview of DRAGON using Gini Index

```
1  constructTree(int maxHeight, int currentHeight, boolean rightNode, data
     trainingData)
2  splitAttribute = getBestSplitAttribute(trainingData)
3  if splitAttribute ≠ null AND currentHeight < maxHeight then
4      currentHeight++
5      rightData = updateTrainingData(splitAttribute,trainingData)
6      leftData = trainingData \ rightData
7      rightChild = constructTree(maxHeight, currentHeight,
8      TRUE, rightData)
9      leftChild = constructTree(maxHeight, currentHeight,
10     FALSE, leftData)
11 return this;
```

Algorithm 2. Overview using global F-measure

```
1  constructTree(int maxHeight, int currentHeight, data trainingData)
2  splitAttribute = getBestSplitAttribute(trainingData)
3  if splitAttribute ≠ null AND currentHeight < maxHeight then
4      currentHeight++
5      rightChild = constructTree(maxHeight, currentHeight,
6      trainingData)
7      leftChild = constructTree(maxHeight, currentHeight,
8      trainingData)
9  return this;
```

3.2 Initialization

The goal of the initialization is to determine the set of mappings that will be used during the construction of the decision tree. We assume that we are given a training data set TS with $TS \subseteq S \times T \times \{+1, -1\}$ (see Table 1 for an example). Every triple $(s, t, +1) \in TS$ is a positive example, while every triple $(s, t, -1) \in TS$ is a negative example. We begin by calculating the subsets $S' \subseteq S$ and $T' \subseteq T$ that can be found in our training dataset as follows:

$$S' = \{s \in S : \exists t \in T \text{ with } (s, t, +1) \in TS \vee (s, t, -1) \in TS\},$$

$$T' = \{t \in T : \exists s \in S \text{ with } (s, t, +1) \in TS \vee (s, t, -1) \in TS\}.$$

We use the sets S' and T' as test sets for our algorithm.

If we use the **global F-measure** as fitness function, we adopt the approach used by [25] by computing the mapping $M_i = \{(s, t) \in S' \times T' : m_i(s, t) \geq \theta_i\}$

which achieves the highest F-measure on the training dataset TS for each of the similarity measures m_i available to our approach. We determine the threshold θ_i by lowering the value of the said threshold from 1 to a lower bound λ by a given rate $\tau \in [0, 1)$.

For the **Gini Index**, we begin by calculating the similarity of each entity pair $(s, t) \in S' \times T'$ using all the measures m_i available to our learner. Any similarity values below the lower bound λ are disregarded i.e., set to 0. For each measure $m \in m_i$ we now have a list of similarity values $[sim_{i1}, ..., sim_{in}]$ ordered from lowest to highest, with the corresponding entity pair $(s, t) \in S' \times T'$.

3.3 Determining Split Attributes

We build our decision tree using top-down induction [24]. We implement two measures to decide on the attribute to use for the splits: *Global F-measure* and *Gini Index*. With the global F-measure, we target the improvement of the overall performance of the decision tree we learn. In contrast, the Gini Index aims to improve the local performance of a given leaf. The performance of the two strategies is compared in the evaluation section.

Learning with the Global F-Measure. The trees we learn with the global F-measure are a combination of the atomic LS computed during the initialization step. Let k be the number of these atomic LSs. We set the atomic link specification which leads to the mapping with the highest F-measure over all measures m_i as our first split attribute. After this initialization of the tree, our tree consists of a root and 2 leaves. In the example shown in Fig. 1, the root would be the node $jaccard(name, name) \geq 0.4$. We first remove the root from the set of LSs that can be added to the tree, leading to $k - 1$ LSs still being contained in the set of candidate LSs. We sequentially position every of the remaining $k - 1$ LSs at every of the 2 leaf nodes, hence generating $2(k - 1)$ trees. We then compute the resulting mapping and select the tree with the best F-measure. After removing the LS used from the set of candidate LSs, we iterate the addition approach until we cannot improve the F-measure achieved by the tree or until we have used all atomic LS we computed during the initialization step.

Formally, at iteration $i \in \{1, \ldots, k\}$, we have $(k - i + 1)$ trees to try out and i nodes in the decision tree where an atomic LS can be added. Hence, the maximal number of trees we need to generate is given by the following:

$$\sum_{i=1}^{k} i(k - i + 1) = \frac{k(k + 1)(k - 2)}{6} \in O(k^3). \tag{1}$$

Our algorithm is hence clearly polynomial in the number of trees computed. In contrast, generating all possible decision trees which can be created using k atomic link specifications is exponential in complexity.

Learning with the Gini Index. We use the similarity values we determined in the initialization in combination with the Gini Index as follows: We determine which measure will be our splitting attribute by calculating the average Gini Index of all measures still available as follows:

$$\overline{G}(\mathcal{N}, sim_{ij}) = \frac{|\mathcal{N}_l|}{|\mathcal{N}|} \times G(\mathcal{N}_l) + \frac{|\mathcal{N}_r|}{|\mathcal{N}|} \times G(\mathcal{N}_r), \tag{2}$$

where $|\mathcal{N}|$ is the number of pairs accepted by the node $\mathcal{N}$ of the decision tree. In the first iteration, $\mathcal{N}$ contains the training data. $\mathcal{N}_l$ accepts the pairs with similarity values below sim_{ij}, $\mathcal{N}_r$ permits the pairs with values above or equal to sim_{ij}. For each m_i we calculate the average Gini Index for all sim_{ij}, with $j \in \{1, \cdots, n\}$, to determine the best split point for each measure.

The Gini Index is

$$G(\mathcal{N}) = 1 - \left(\left(\frac{|\mathcal{N}^+|}{|\mathcal{N}|} \right)^2 + \left(\frac{|\mathcal{N}^-|}{|\mathcal{N}|} \right)^2 \right), \tag{3}$$

where $|\mathcal{N}^+|$ is the number of positive examples and $|\mathcal{N}^-|$ the number of negative examples accepted by $\mathcal{N}$. The splitting attribute will be the measure m with the corresponding threshold $\theta = sim_{ij}$. Common decision tree algorithms will set the threshold in the splitting attribute to be $\theta = (sim_{ij} + sim_{i(j-1)})/2$, whereas we set it to the higher value (i.e. sim_{ij}). In the evaluation section we will see that this choice improves the quality of our decision tree considerably.

After finding the root node the training data is provided as seen in Algorithm 1, we stop when we either have reached the maximum tree height or measure with an average Gini Index below 1 can be found.

For example, imagine we were to use the example in Table 1 to learn a tree using Gini Index. $S' \times T'$ are the link pairs from the training data, therefore the first step will be to determine the similarity value for these pairs using all available measures on all attributes. We provided the cosine similarity on the name attributes in Table 1(c). To test how well the cosine similarity performs we have to find the ideal split point. We start with the lowest similarity value, that is bigger than 0, which in our case is 0.707. $|\mathcal{N}_l| = 2$, since only two links have a similarity value below 0.707. $|\mathcal{N}_r| = 3$ containing the remaining links. To calculate the average Gini Index we need to determine the Gini Indices for the left and right node. Both are 0 since the left node only contains negative and the right only positive examples. Since our split attribute perfectly divides the examples to the appropriate classes we have a pure leaf with the average Gini Index of 0 and our tree induction is finished. Bear in mind that we choose $\theta = 0.707$ in our link specification in contrast to common decision tree algorithms that would take $\theta = (0.707 + 0.0)/2 = 0.3535$.

3.4 Pruning

Previous works (e.g., [24]) have shown that pruning decision trees can improve their performance significantly. Given our approach to decision tree generation,

Table 2. Characteristics of the used datasets.

Label	#Attributes	$\|S\| \times \|T\|$	|reference dataset|
Person1	11	250,000	500
Person2	11	240,000	400
Restaurants	5	72,433	112
DBLP-ACM	4	6,000,000	2,224
DBLP-Scholar	4	168,100,000	5,347
Amazon-GP	4	4,400,000	1,300
Abt-Buy	4	1,200,000	1,097
Drugs	1	1,090,000	1,047
Movies	2	1,090,000	1,047

we devised a *Global F-measure pruning* and implemented the *error estimate pruning* used by [24] for the sake of comparison. Given a tree $\mathcal{N}$ with height h, we start our pruning process by iterating over the nodes $\mathcal{N}_i$ at height $h-2$, with $i \in \{1, \ldots, n\}$, where n is the number of nodes at height $h-2$. We compute the F-measure of $\mathcal{N}$, after we pruned the left, right and both leaves of $\mathcal{N}_i$ respectively. The tree with the best F-measure is kept, i.e. $\mathcal{N}$ will be overwritten by it. After repeating this process for all nodes at height $h-2$, we continue at height $h-3$ and so on until we reach the root node and terminate.

If we were to prune the tree from Fig. 1, we would subsequently compute the F-measure of the whole tree and after we removed the node $jaccard(age, age) \geq 0.8$.

4 Evaluation

The aim of our evaluation was to evaluate the effectiveness (i.e., the F-measure) and the efficiency (i.e., the runtime) achieved by DRAGON. We were especially interested in the performance of DRAGON on real datasets. Hence, we evaluated DRAGONon the four real-world datasets from [14]. These datasets were obtained by manually curating data harvested from the Web and determining links between these datasets manually. In addition, we aimed to compute the performance of DRAGON on synthetic datasets. We selected three datasets from OAEI 2010 benchmark[2] and the two datasets *Drugs* and *Movies*, used in [19]. Table 2 presents some details pertaining to these datasets. We ran two series of experiments. In the first series on experiments, we tested whether our approach to setting thresholds in decision trees is superior to that followed by other decision-tree learning approaches. To this end, we compared different ways of setting thresholds in our algorithm. We also used this experiment to determine the default settings for subsequent experiments. In our second series of experiments, we compared DRAGON with state-of-the-art link discovery algorithms.

[2] http://oaei.ontologymatching.org/2010/im/.

In all experiments we used a ten-fold cross validation setting were the training folds consist of 50% positive and 50% negative examples. We present the average results achieved by all algorithms over the ten runs of the crossvalidation setting. To make a comparison between algorithms over all datasets easier we also calculated the average rank of the approaches given the performance measure (time or f-measure). To achieve this the approaches are ranked by their performance per dataset, for ties the mean of the ranks are assigned, and the ranks are averaged columnwise to get the final value. This average rank is added as last row in the tables we present. All experiments were carried out on a 64-core 2.3 GHz Server running Oracle Java 1.8.0_77 on Ubuntu 14.04.4 LTS, with each experiment assigned 20 GB of RAM.

4.1 Parameter Discovery

To check whether our approach to discovering settings is better than that followed by other decision-tree-based approaches, we ran our Gini approach combined with the Global F-measure pruning (G) and the error estimate pruning (E). We set λ between 0.05 and 0.8. Our results are displayed in Table 3. In our experiments, $\lambda = 0.4$ achieves the best rank and is therefore an appropriate setting for our algorithm. We hence selected this value as default.

Table 3. F-measure for 10-fold cross validation averaged over 10 results. UP indicates choosing the upper value as split point while MP uses the middle between two similarity values as split point in the node. The best, second- and third-best value in a row are highlighted using colored cells of decreasing strength.

Data	UP, $\lambda = .05$		UP, $\lambda = .2$		UP, $\lambda = .4$		UP, $\lambda = .8$		MP, $\lambda = .05$		MP, $\lambda = .4$	
	G	E	G	E	G	E	G	E	G	E	G	E
Movies	0.969	0.969	0.975	0.975	0.969	0.969	0.994	0.832	0.967	0.967	0.734	0.905
Person1	0.978	0.970	0.966	0.965	0.980	0.983	0.985	0.986	0.945	0.931	0.940	0.947
Person2	1.000	1.000	0.999	0.999	1.000	1.000	0.999	0.999	0.980	0.980	0.980	0.980
Drugs	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.978	0.978	0.938	0.938
Restaurants	0.976	0.976	0.986	0.986	0.976	0.976	1.000	1.000	0.961	0.944	1.000	1.000
DBLP-ACM	0.881	0.918	0.872	0.912	0.895	0.961	0.974	0.966	0.858	0.894	0.388	0.939
Abtbuy	0.441	0.496	0.573	0.545	0.644	0.617	0.209	0.208	0.442	0.498	0.661	0.661
DBLP-GS	0.928	0.694	0.938	0.695	0.972	0.970	0.871	0.905	0.906	0.830	0.185	0.711
AMAZON-GP	0.429	0.641	0.467	0.711	0.761	0.704	0.351	0.343	0.429	0.581	0.682	0.717
Avg Rank	6.625	6.438	6.188	6.188	3.938	3.938	5.688	5.813	9.438	9.250	8.250	6.250

We then compared our approach to setting thresholds in split points with that implemented by classical decision-tree-learning approaches such as J48 (see two right-most columns of Table 3). Note that we used $\lambda = 0.05$ to test if a higher threshold to calculate the initial similarity values improves the quality of the constructed decision tree. While choosing the middle between similarity values as split point seems to be beneficial on some datasets (e.g., Abtbuy), it is clear that choosing the upper value bears better results overall. A comparison of the difference in F-measure achieved by the settings $\lambda = 0.05$ (first column in

Table 3) and $\lambda = 0.4$ (third column in Table 3) further reveals that preventing a decision tree from learning measures with a low threshold value can improve the quality up to 43% (Amazon-GP).

4.2 Comparison with Other Approaches

Our choice of state-of-the-art link discovery algorithm to compare with DRAGON was governed by the need to provide a fair evaluation. Firstly, the algorithms we were to test against needed to learn link specifications explicitly, since our approach tackles the task of *declarative link discovery*. Secondly, we needed approaches able to perform supervised learning, since our approach would have an unfair advantage over unsupervised classifiers. We hence chose WOMBAT since it fits these requirements and to perform as well as [13] w.r.t. the F-measure it achieves. In addition, WOMBAT was shown to be robust w.r.t. the number of examples used for learning. We also ran our experiments with EAGLE [19] and EUCLID [20]. We selected EAGLE [19] because it was shown to outperform MARLIN [2] and FEBRL [4] significantly in terms of runtime, while achieving comparable F-measure [19]. EUCLID's major advantage over EAGLE is the fact that it is deterministic and reaches similar F-measure to EAGLE. In our evaluation we use the supervised linear version of EUCLID. All the chosen approaches and especially DRAGON are contained in the open-source LIMES link discovery framework and are free to use. We decided not to compare DRAGON with classifiers from e.g. SILK [28], since LIMES has been shown [18] to be significantly faster than SILK and therefore runtime comparisons would not be fair. We also compare our approaches with J48, an often used implementation of the C4.5 algorithm [10].

To test our hypothesis that common decision tree approaches choose threshold values that are too low, we also implemented *J48opt*. In this approach, we took the decision tree we get from *J48*, parsed it into a LS, raised all its thresholds by δ and calculated the F-measure it achieved. We repeated this process until all thresholds equal 1. We took the LS with the threshold setting that resulted in the highest F-measure. In our experiments, we set δ to 0.05. For DRAGON, we tested *Global F-measure* (in the following referred to as DRAGON_{GL}) and *Gini Index* (DRAGON_{GI}), as well as the two pruning algorithms *Global F-measure pruning* and *error estimate pruning*. We will also indicate the pruning method in subscript as well. Hence, $\text{DRAGON}_{GL \cdot E}$ is a configuration of DRAGON which was achieved by the use of *Global F-measure* for finding split attributes and *error estimate pruning*. In contrast, $\text{DRAGON}_{GL \cdot G}$ is DRAGON with global F-measure and global F-measure pruning. We set the maximum tree height to 3. For the Gini configurations, we set λ to 0.4. For the global F-measure configurations, we set λ to 0.6. We discovered that each approach performs best with these parameters through empirical tests. The termination criteria for WOMBAT was either finding a link specification with F-measure of 1 or a refinement depth of 10. The coverage threshold was set to 0.6 and the similarity measures used were

Table 4. Averaged results for 10-fold cross validation. We highlight the best, second- and third-best value in a row using colored cells of decreasing strength.

Data	DRAGON				WOMBAT	EUCLID	EAGLE	J48	J48opt
	$GI \cdot G$	$GI \cdot E$	$GL \cdot G$	$GL \cdot E$					
Movies	0.971	0.971	0.994	0.994	0.993	0.984	0.989	0.835	0.989
Person1	0.984	0.985	0.980	0.978	1.000	0.983	0.993	0.855	0.957
Person2	1.000	1.000	0.998	1.000	0.991	0.823	0.952	0.924	1.000
Drugs	0.998	0.998	0.998	0.998	0.998	0.996	0.996	0.938	0.998
Restaurants	1.000	1.000	0.980	1.000	0.994	0.953	0.952	0.905	0.962
DBLP-ACM	0.934	0.932	0.970	0.983	0.983	0.978	0.980	0.768	0.860
Abtbuy	0.679	0.642	0.529	0.555	0.647	0.003	0.555	0.429	0.582
DBLP-GS	0.973	0.973	0.906	0.934	0.963	0.899	0.945	0.880	0.955
Amazon-GP	0.741	0.711	0.620	0.354	0.761	0.712	0.731	0.408	0.450
Avg Rank	3.333	3.889	5.111	4.278	2.778	6.611	5.056	8.667	5.278

(a) F-measure

Data	DRAGON				WOMBAT	EUCLID	EAGLE	J48	J48opt
	$GI \cdot G$	$GI \cdot E$	$GL \cdot G$	$GL \cdot E$					
Movies	152	134	1652	1652	1623	4414	122539	170	204
Person1	473	488	21433	21783	12527	210140	1343	272	991
Person2	114	119	3144	3050	2350	39109	1080	148	337
Drugs	48	41	307	297	246	215	1047	60	88
Restaurants	19	20	487	478	631	9721	1013	27	51
DBLP-ACM	2626	2618	40614	40562	46002	308044	94873	680	23109
Abtbuy	1619	1574	6419	6310	19002	31065	1243	510	34762
DBLP-GS	5623	5517	28454	28541	143973	496948	110168	948	45036
Amazon-GP	3126	3102	21593	21741	56651	325759	26638	1331	134693
Avg Rank	2.3	2	6.2	5.9	6.6	8.3	6.6	1.9	5.2

(b) Time in ms

the same as in DRAGON: *jaccard, trigrams, cosine* and *qgrams*. EAGLE was configured to run 100 generations. The mutation and crossover rates were set to 0.6 as in [19]. EUCLID was set to the default parameters. For *J48*, we discovered, that using reduced error pruning with 5 folds delivered the best results overall. Otherwise we used the default parameters found in the Weka framework [10].

To determine the significant differences between classifier performances usually a pair-wise Wilcoxon signed-ranks test is performed. However, this would lead to a multiple testing problem in our case, since we compare more than two classifiers. We therefore follow the recommendations given in [7] to use a Friedman test to determine if the average ranks of the algorithms are significantly different and, if this is true, perform a Nemenyi test to compare all classifiers. In Table 4 the results are presented. Concerning F-measure we have significant differences between the algorithms (Friedman p-value = 0.009).

We can see that WOMBAT and DRAGON$_{GI\cdot G}$ produce the best results regarding F-measure and there is no significant difference in performance (Nemenyi p-value = 0.999). WOMBAT achieves a higher F-measure than DRAGON$_{GI\cdot G}$ on four datasets (Movies, Person1, DBLP-ACM, AMAZON-GP) and is tied with DRAGON on the Drugs dataset. DRAGON outperforms WOMBAT on the remaining four datasets. The only significant difference in F-measure is between WOMBAT and j48 (Nemenyi p-value = 0.007) and between DRAGON and j48 (Nemenyi p-value = 0.023). We can also see that *J48opt* performs better than *J48*, albeit not significantly better (Nemenyi p-value = 0.326).

In Table 4b, we present the average runtimes of the approaches. We can determine, that there is a significant difference in runtime-efficiency between the algorithms (Friedman p-value = 1.127×10^{-8}). It is evident, that DRAGON(specifically DRAGON$_{GI}$) and *J48* are the fastest approaches. We determined no significant difference between them (Nemenyi p-value = 0.999). The DRAGON$_{GI}$ configurations of our approach are on average 18 times faster than WOMBAT[3] , while DRAGON$_{GL}$ is roughly as efficient as WOMBAT. The performance advantage of DRAGON$_{GI}$ is due to the fact that, after calculating the initial similarity values of the entity pairs, it does not need the costly calculations of mappings.[4] Overall, our results suggest that DRAGON performs as well as the state of the art w.r.t. the quality of the LSs it generates, but clearly outperforms the state of the art w.r.t. its runtime[5], making it more conducive to practical application.

4.3 Efficiency of Pruning

Since pruning is an important factor in decision tree learning, we recorded the effect pruning had in the second experiment. The results are displayed in Fig. 2.

First note that the configuration DRAGON$_{GL\cdot GL}$ was omitted in the figures since the unpruned and pruned trees were identical. This is not surprising because the measure for building the tree is the same as for pruning it. Therefore, any leaves that would be pruned simply do not appear in the tree in the first place. For the other configurations, we can observe, that pruning has on average a positive impact on F-measure. The exceptions are datasets for which LS of the size 1 are learned in the first place (such as Movies and Person2 for DRAGON$_{GI}$) and Amazon-GP, where pruning has a negative effect.

[3] This difference is significant with a p-value = 0.0297 for the Nemenyi test.

[4] Note that in DRAGON$_{GL}$ these costs are linear for each mapping computation, since we only need to use set-operations on the mappings from the initial atomic specifications.

[5] DRAGON$_{GI}$ is also significantly faster than EUCLID (Nemenyi p-value = 0.0001) and EAGLE (Nemenyi p-value = 0.0297).

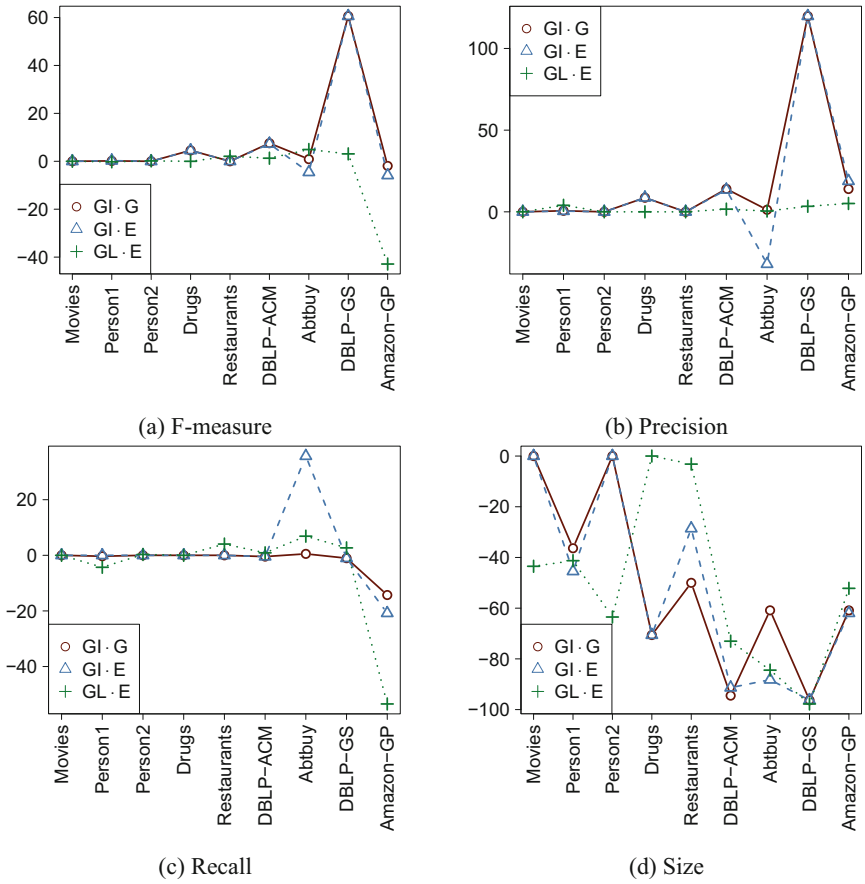

Fig. 2. Percentage change between unpruned and pruned trees in the averaged results of the ten-fold crossvalidation

5 Related Work

A plethora of approaches for link discovery has been developed recently. We give a brief overview of existing approaches and refer the reader to the corresponding surveys and comparisons [6,15,26] for further details. A popular link discovery framework is SILK [28], which uses multi-dimensional blocking to achieve lossless link discovery. Another lossless framework is LIMES [17], which combines similarity measures using a set-theoretical approach. An overview and a comparison of further link discovery frameworks can be found in [15].

As we have seen in definition 2, link discovery is a binary classification problem and therefore can be tackled with classical machine learning approaches such as *support vector machines* [2], *artificial neural networks* [16] and *genetic programming* [11]. [26] give an overview of these approaches and compare them. Most dedicated approaches to learning link specifications are supervised tech-

niques implemented as batch learning or as active learning. For example, EAGLE [19], COALA [21] and GenLink [11] support both active and batch learning within the genetic programming paradigm. Other approaches such as EUCLID [20] and KnoFuss [22] support unsupervised learning. WOMBAT [25] uses an upward refinement operator to implement positive-only learning.

Record linkage is a closely related field of link discovery and there are several approaches which use decision trees. Early works by [5] aimed at matching two databases containing customer records. They used manually generated training data to train a CART [3] classifier and pruned it to reduce complexity and make it more robust. [8] incorporated ID3 [23] into their record linkage toolbox TAILOR. They implemented two approaches: In the first they manually labeled record pairs and used the comparison vector to train a decision tree. The second approach tried to overcome the manual labeling effort by applying a clustering algorithm on their comparison vectors to get three clusters: matches, non-matches, potential matches. Another notable approach is Active Atlas [27], which combines active learning with the C4.5 [24] decision tree learning algorithm.

Our approach extends these approaches in several ways. By learning an operator tree our approach is able to learn much more expressive models than linear classifiers (e.g. [2]), since we can express conjunction and disjunction. Furthermore, we are more expressive than boolean classifiers such as [4,16], because we can model negations. We additionally combine the generation of expressive link specifications with a deterministic approach, providing more reliability than comparable non-deterministic approaches [11,19,21,22]. DRAGON differs further by relying on a decision-tree-based learning paradigm, which allows it to not have to recompute mappings, leading to a more time-efficient approach (see evaluation section). It shares some similarity with the state of the art by virtue of its iterative approach to addressing the generation of link specifications (see, e.g., [25]).

6 Conclusion and Future Work

In this work, we presented DRAGON, a decision tree learning approach for link discovery. We evaluated our approach on nine benchmark datasets against three state-of-the-art approaches WOMBAT, EUCLID and EAGLE. Our approach delivers state-of-the-art performance w.r.t. the F-measure it achieves while on average being more than 18 times faster than the state of the art. We investigated why our approach outperforms other decision tree approaches by optimizing the choice of thresholds in similarity measures. Interestingly, our Global F-measure approach is biased towards precision while using Gini leads to a bias towards recall. In future work, we will investigate how we can use these biases for ensemble learning. We will also look into the possibility to perform active learning with DRAGON, in particular with the use of incremental decision trees.

Acknowledgements. This work has been supported by the BMVI projects LIMBO (project no. 19F2029C) and OPAL (project no. 19F2028A).

References

1. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: 16th International Conference on World Wide Web. WWW 2007, pp. 131–140. ACM, New York (2007). https://doi.org/10.1145/1242572.1242591
2. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: International Conference on Knowledge Discovery and Data Mining. KDD 2003, pp. 39–48. ACM, New York (2003). https://doi.org/10.1145/956750.956759
3. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Chapman & Hall, New York (1984)
4. Christen, P.: Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In: International Conference on Knowledge Discovery and Data Mining. KDD 2008, pp. 1065–1068. ACM, New York (2008). https://doi.org/10.1145/1401890.1402020
5. Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.E.: Efficient data reconciliation. Inf. Sci. **137**(1–4), 1–15 (2001). https://doi.org/10.1016/S0020-0255(00)00070-0
6. Daskalaki, E., Flouris, G., Fundulaki, I., Saveta, T.: Instance matching benchmarks in the era of linked data. J. Web Sem. **39**, 1–14 (2016). https://doi.org/10.1016/j.websem.2016.06.002
7. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006). http://www.jmlr.org/papers/v7/demsar06a.html
8. Elfeky, M.G., Elmagarmid, A.K., Verykios, V.S.: Tailor: a record linkage tool box. In: International Conference on Data Engineering, pp. 17–28 (2002). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=994694
9. Ermilov, I., Lehmann, J., Martin, M., Auer, S.: LODStats: the data web census dataset. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 38–46. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46547-0_5
10. Holmes, G., Donkin, A., Witten, I.: Weka: a machine learning workbench. In: Proceedings of the Second Australia and New Zealand Conference on Intelligent Information Systems, pp. 357–361. Brisbane, Australia (1994). http://www.cs.waikato.ac.nz/~ml/publications/1994/Holmes-ANZIIS-WEKA.pdf
11. Isele, R., Bizer, C.: Learning expressive linkage rules using genetic programming. Proc. VLDB Endow. **5**(11), 1638–1649 (2012). https://doi.org/10.14778/2350229.2350276
12. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall. In: 14th International Workshop on the Web and Databases (2011). http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/IseleJentzschBizer-WebDB2011.pdf
13. 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). https://doi.org/10.1007/978-3-319-18818-8_24
14. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proc. VLDB Endow. **3**(1), 484–493 (2010). https://doi.org/10.14778/1920841.1920904
15. Nentwig, M., Hartung, M., Ngomo, A.N., Rahm, E.: A survey of current link discovery frameworks. Semant. Web **8**(3), 419–436 (2017). https://doi.org/10.3233/SW-150210

16. Ngomo, A.N., Lehmann, J., Auer, S., Höffner, K.: RAVEN - active learning of link specifications. In: Ontology Matching Workshop, pp. 25–36 (2011). http://ceur-ws.org/Vol-814/om2011_Tpaper3.pdf
17. Ngonga Ngomo, A.C.: On link discovery using a hybrid approach. J. Data Semant. **1**, 203–217 (2012). https://doi.org/10.1007/s13740-012-0012-y
18. Ngonga Ngomo, A.C., Auer, S.: Limes: a time-efficient approach for large-scale link discovery on the web of data. In: ICJAI. IJCAI 2011, pp. 2312–2317. AAAI Press (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-385
19. 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). https://doi.org/10.1007/978-3-642-30284-8_17
20. Ngonga Ngomo, A.C., Lyko, K.: Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: Ontology Matching Workshop, pp. 25–36 (2013). http://ceur-ws.org/Vol-1111/om2013_Tpaper3.pdf
21. 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). https://doi.org/10.1007/978-3-642-38288-8_30
22. Nikolov, A., d'Aquin, 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). https://doi.org/10.1007/978-3-642-30284-8_15
23. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986). https://doi.org/10.1023/A:1022643204877
24. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
25. Sherif, M.A., Ngonga Ngomo, A.C., Lehmann, J.: WOMBAT – a generalization approach for automatic link discovery. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10249, pp. 103–119. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58068-5_7
26. Soru, T., Ngonga Ngomo, A.C.: A comparison of supervised learning classifiers for link discovery. In: 10th International Conference on Semantic Systems, pp. 41–44. ACM (2014). https://doi.org/10.1145/2660517.2660532
27. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Inf. Syst. **26** (2001). https://doi.org/10.1016/S0306-4379(01)00042-4
28. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk - A link discovery framework for the web of data. In: Workshop on Linked Data on the Web, LDOW (2009). http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf
29. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. Proc. VLDB Endow. **1**(1), 933–944 (2008). https://doi.org/10.14778/1453856.1453957
30. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst. **36**(3), 15 (2011). https://doi.org/10.1145/2000824.2000825