Active Learning of Expressive Linkage Rules using Genetic Programming

Robert Isele\textsuperscript{a,*}, Christian Bizer\textsuperscript{a}

\textsuperscript{a}Research Group Data and Web Science, University of Mannheim, B6 26, 68131 Mannheim, Germany

Abstract

A central problem in the context of the Web of Linked Data as well as in data integration in general is to identify entities in different data sources that describe the same real-world object. Many existing methods for matching entities rely on explicit linkage rules, which specify the conditions which must hold true for two entities in order to be interlinked. As writing good linkage rules by hand is a non-trivial problem, the burden to generate links between data sources is still high. In order to reduce the effort and expertise required to write linkage rules, we present the ActiveGenLink algorithm which combines genetic programming and active learning to generate expressive linkage rules interactively. The ActiveGenLink algorithm automates the generation of linkage rules and only requires the user to confirm or decline a number of link candidates. ActiveGenLink uses a query strategy which minimizes user involvement by selecting link candidates which yield a high information gain. Our evaluation shows that ActiveGenLink is capable of generating high quality linkage rules based on labeling a small number of candidate links and that our query strategy for selecting the link candidates outperforms the query-by-vote-entropy baseline.

Keywords: Entity Matching, Duplicate Detection, Link Discovery, Active Learning, Genetic Programming, Linkage Rules, ActiveGenLink

1. Introduction

The goal of the Linked Data movement is to extend the Web with a global data space by making data sets accessible according to a set of best practices and by setting RDF links between data sources [4]. While the amount of data that is accessible as Linked Data has grown significantly over the last years, most data sources are still not sufficiently interlinked. Out of the over 31 billion RDF statements published as Linked Data less than 500 million represent RDF links between data sources [5]. Analysis of the Linking Open Data cloud confirms that it represents a weakly connected graph with most publishers only linking to one other data source [5].

A number of link discovery tools have been developed, which generate RDF links between entities in different data sets that represent the same real-world object. Unfortunately, fully automatic link discovery tools do not achieve a satisfying accuracy on many data sets [16]. For this reason, several semi-automatic link discovery tools - such as Silk [23] or LIMES [30] - have been developed. These tools compare entities in different Linked Data sources based on user-provided linkage rules which specify the conditions that must hold true for two entities in order to be interlinked.

Writing good linkage rules by hand is a non-trivial problem as the rule author needs to have detailed knowledge about the structure of the data sets: First of all, the author needs to choose discriminative properties of the entities to be interlinked together with a distance measure and an appropriate distance threshold. For data sets which are noisy or use different data formats, the property values need to be normalized by employing data transformations prior to comparison. As comparing entities by a single property usually is not sufficient to decide whether both entities describe the same real-world object, the linkage rule has to aggregate the similarity of multiple property comparisons using appropriate aggregation functions.

We illustrate this point with the example of a data set about movies: Even within this simple example the linkage rule author faces a couple of challenges. First of all, a comparison solely by film title fails for cases when movies with the same title actually represent distinct movies that have been released in different years. Therefore, the linkage rule needs to compare, at the very least, the titles of the movies as well as their release dates and combine both similarities with an appropriate aggregation function. As data sets can be noisy (e.g., the release dates might be off by a couple of days), the rule author also needs to choose suitable distance measures together with appropriate distance thresholds. Linkage rules must also cover data heterogeneities. For instance, a data source may contain some person names that are formatted as \texttt{⟨first name⟩⟨last name⟩} while others are formatted as \texttt{⟨last name⟩, ⟨first name⟩}. Finding such heterogeneities and adding the specific data transformations to avoid mismatches is often very tedious. Thus, writing a linkage rule is not only a cumbersome but...
also a time consuming task.

Supervised Learning A way to reduce this effort is to use supervised learning to generate links from existing reference links, which contain pairs of entities that have been labeled as matches or non-matches. Creating such reference links is much easier than to write linkage rules as it requires no previous knowledge about similarity computation techniques or the specific linkage rule format used by the system. Usually, reference links are created by domain experts who confirm or reject the equivalence of a number of entity pairs from the data sets. For instance, reference links for locations in a geographical data set can be created by labeling pairs of locations as correct or incorrect. Figure 1 shows an example of an entity pair. In the given example, the pair is to be declined as both entities represent different real-world locations.

Active Learning In order for the supervised learning algorithms to perform well on unknown data, reference links need to include all relevant corner cases. We illustrate this point by having a second look at the example in Figure 1: While for many cities a comparison by label is sufficient to determine if two entities represent the same real-world city, the given example shows the corner case of distinct cities sharing the same name. If the entity pairs to be labeled by the user are just selected randomly from the data sets, the user has to label a very large number of pairs in order to include these rare corner cases reliably. As manually labeling link candidates is time-consuming, methods to reduce the number of candidates which need to be labeled are desirable.

The fundamental idea of active learning in the context of entity matching is to reduce the number of link candidates which need to be labeled by actively selecting the most informative candidate for being labeled by the user.

Our Contribution In this article, we present ActiveGenLink, an algorithm for learning linkage rules interactively using active learning and genetic programming. ActiveGenLink learns a linkage rule by asking the user to confirm or reject a number of link candidates which are actively selected by the algorithm. Compared to writing linkage rules by hand, ActiveGenLink lowers the required level of expertise as the task of generating linkage rules is automated by the genetic programming algorithm while the user only has to verify a set of link candidates. The employed query strategy for selecting link candidates minimizes user involvement by selecting the links with the highest information gain for manual verification. Within our experiments, ActiveGenLink outperformed state-of-the-art unsupervised approaches after manually labeling a few link candidates (less than 5 within our experiments).

In addition, ActiveGenLink is capable of generating linkage rules with a comparable performance than the supervised GenLink algorithm [22] by labeling a much smaller number of link candidates (between 15 and 50 within our experiments). ActiveGenLink chooses which properties to compare, it chooses appropriate distance measures, aggregation functions, and thresholds, as well as data transformations that are applied to normalize data prior to comparison.

This article makes the following contributions:

1. We propose the ActiveGenLink algorithm which applies genetic programming and active learning to the problem of learning linkage rules for generating RDF links in the context of the Web of Data.
2. The learned rules are more expressive than the linkage rules learned in previous work on active learning of linkage rules as our algorithm combines different similarity measures non-linearly and also determines the data transformations that should be employed to normalize data prior to comparison.
3. We propose a query strategy that minimizes the number of links that need to be labeled by the user and outperforms the query-by-vote-entropy strategy, which has been used in previous work.
4. We have implemented the proposed approach in the Silk Workbench, a web application which can be used by Linked Data publishers and consumers to set RDF links. The Silk Workbench is part of the Silk Link Discovery Framework and is available for download under the terms of the Apache License.

This article builds on our previous work presented at two occasions:

- In [22], we present the GenLink algorithm for learning expressive linkage rules from a set of existing reference links using genetic programming.
- In [25], we present an approach which combines genetic programming and active learning to generate expressive linkage rules interactively.

This article extends the previously presented active learning approach with a novel query strategy, which further minimizes the number of links that need to be labeled by the user as well as a more extensive experimental evaluation. Although in this paper we focus on interlinking data sources in the context of the Web of Linked Data, our approach is not limited to this use case and can be applied to entity matching in other areas - such as in the context of relational databases - as well.

Outline This article is organized as follows: Section 2 formalizes the entity matching problem. Section 3 introduces our linkage rule representation. Based on that, Section 4 describes the ActiveGenLink workflow. Section 5 describes the genetic programming approach used for learning linkage rules from existing training data. Section 6 describes the proposed query strategy for selecting...
link candidates in detail. Section 7 introduces our approach of building the initial pool of unlabeled links. Section 8 presents the results of our experimental evaluation. Section 9 discusses related work. Section 10 presents the implementation of ActiveGenLink in the Silk Workbench.

2. Problem Definition

We consider the problem of matching entities between two data sources $A$ and $B$. The objective is to determine which entities in $A$ and $B$ identify the same real-world object.

The general problem of entity matching can be formalized as follows [18]:

**Definition 1 (Entity Matching).** Given two data sources $A$ and $B$, find the subset of all pairs of entities for which a relation $\sim_R$ holds:

$$M = \{(a, b); a \sim_R b, a \in A, b \in B\}$$

Similarly, we define the set of all pairs for which $\sim_R$ does not hold:

$$U = (A \times B) \setminus M$$

The purpose of relation $\sim_R$ is to relate all entities which represent the same real-world object.

In some cases a subset of $M$ and $U$ is already known prior to matching. Such reference links can, for instance, originate from previous data integration efforts. Alternatively, they can be created by domain experts who simply need to confirm or reject the equivalence of entity pairs from the data sets.

**Definition 2 (Reference Links).** A set of positive reference links $R_+ \subseteq M$ contains pairs of entities for which relation $\sim_R$ is known to hold (i.e. which identify the same real-world object). Analogously, a set of negative reference links $R_- \subseteq U$ contains pairs of entities for which relation $\sim_R$ is known to not hold (i.e. which identify different real-world objects).

Reference links can serve two purposes: Firstly, they can be used to evaluate the quality of a linkage rule. But more importantly, they can also be used to infer a linkage rule which specifies the conditions which must hold true for a pair of entities to be part of $M$:

**Definition 3 (Linkage Rule).** A linkage rule $l$ assigns a similarity value to each pair of entities:

$$l : A \times B \rightarrow [0, 1]$$

The set of matching entities is given by all pairs for which the similarity according to the linkage rule exceeds a threshold $\theta$:

$$M_l = \{(a, b); l(a, b) \geq \theta, a \in A, b \in B\}$$

As the linkage rule may return arbitrary values in the range $[0, 1]$, we fix the value of the threshold $\theta$ to 0.5 without loss of generality.

3. Linkage Rule Representation

In the context of this work, we represent a linkage rule as a tree which is built from four types of operators. We group the operators into *similarity operators*, which return a similarity score for two given entities, and *value operators*, which return a set of values for a given entity. First we introduce the value operators:

**Property Operator:** Retrieves all values of a specific property of each entity, such as its label property.

**Transformation Operator:** Transforms the values of a set of property or transformation operators according to a specific data transformation function. Examples of common transformation functions include case normalization, tokenization and concatenation of values from multiple operators. Multiple transformation operators can be nested in order to apply a sequence of transformations.

Now we take a closer look at the similarity operators:

**Comparison Operator:** Evaluates the similarity between the values of two input operators according to a specific distance measure, such as Levenshtein, Jaccard, or geographic distance. Allowed input operators are property operators and transformation operators. A user-specified threshold specifies the maximum distance. The threshold is used to normalize the distance measure to the interval $[0, 1]$.

**Aggregation Operator:** Due to the fact that, in most cases, the similarity of two entities cannot be determined by evaluating a single comparison, an aggregation operator combines the similarity scores from multiple comparison or aggregation operators into a single score according to a specific aggregation function. Examples of common aggregation functions include the weighted average or yielding the minimum score of all operators. Aggregation operators can be nested in order to create non-linear hierarchies.

Figure 2 specifies the valid structure of a linkage rule. The resulting linkage rule forms a tree where the terminal
nodes are represented by property operators and the internal nodes are represented by transformation, comparison and aggregation operators.

Our approach is independent of any specific aggregation functions, distance measures or data transformations. Thus, it can learn linkage rules with any functions provided to it. Table 1 lists the used distance measures, which we employed in our experiments. The used aggregation functions are listed in Table 2. Table 3 lists the transformation functions.

3.1. Example

Figure 3 shows a simple example of a linkage rule for interlinking cities. In this example, the linkage rule compares the labels as well as the coordinates of the entities. The labels are normalized by converting them to lower case prior to comparing them with the Levenshtein distance. The thresholds of the comparison operators normalize the similarity score to the range $[0, 1]$. The similarity score of the labels is then aggregated with the geographic similarity score into a single score by using the minimum aggregation i.e. both values must exceed the threshold of 0.5 in order to generate a link.

3.2. Discussion

Our representation of a linkage rule differs from other commonly used representations [8] in a number of ways:

Matching Between Different Schemata: We allow the matching between data sets which use different schemata. This is enabled by two additions: Firstly, by allowing two property operators for each comparison and secondly by introducing data transformations. For example, a data source which uses the FOAF vocabulary [6] may represent person names using the `foaf:firstName` and `foaf:lastName` properties while a data source using the DBpedia ontology may represent the same names using just the `dbpedia:name` property. In order to compare entities expressed in different schemata or data formats, their values have to be normalized prior to comparing them for similarity. In this example we could achieve this in two ways: We could concatenate `foaf:firstName` and `foaf:lastName` into a single name before comparing them to `dbpedia:name` by using a character-based distance measure such as the Levenshtein distance. Alternatively, we could split the values of `dbpedia:name` using a tokenizer and compare them to the values of `foaf:firstName` and `foaf:lastName` by using a token-based distance measure such as the Jaccard coefficient.

Handling Noisy Data Sets: Another motivation for transformation operators is the matching of noisy data sets. A common example is data sources which contain values using an inconsistent letter case (e.g. “iPod” vs. “IPOD”). A way to address case inconsistency is to normalize all values to lower case prior to comparing them.

Representing Non-linear Classifiers: Arasu et al. [1] categorize widely used approaches for representing linkage rules as threshold-based boolean classifiers and linear classifiers. In [22], we show that the performance of entity matching can be improved with more expressive representations in that we allow aggregation operators to be nested in order to represent non-linear classifiers beyond pure boolean classifiers.

4. Active Learning Workflow

The main idea of ActiveGenLink is to evolve a population of candidate solutions iteratively while building a
set of reference links. The algorithm starts with a random population of linkage rules and an initially empty set of reference links. In each iteration, it selects a link candidate for which the current population of linkage rules is uncertain from a pool of unlabeled links. After the link has been labeled by an human expert, it evolves the population of linkage rules based on the extended set of reference links. Figure 4 summarizes the three steps which are involved in each iteration:

1. The query strategy selects link candidates to be labeled by the user from an unlabeled pool of entity pairs (i.e. pool-based sampling [34]). Entity pairs are selected according to a query-by-committee [35] strategy i.e. the selected link candidate is determined from the voting of all members of a committee, which consists of the current linkage rules in the population. As the linkage rules from the population are all trained on the current reference links they represent competing hypotheses. Thus, they are part of the version space which is the space of all linkage rules which are consistent with the known reference links. The aim of the query strategy is to select link candidates which reduce the version space as much as possible.

2. A human expert labels the selected link as correct or incorrect. Confirmed links are added to the positive reference link set and declined links are added to the negative reference link set.

3. The genetic programming algorithm evolves the population of linkage rules. The goal is to find linkage rules which cover the current set of reference links. The population is evolved until either a maximum number of iterations is reached or when one linkage rule in the population reaches an F-measure of 100%.

The pseudocode of ActiveGenLink for evolving a population of linkage rules based on a set of reference links is given in Algorithm 2.

Algorithm 2 Pseudocode of the GenLink algorithm. The specific parameter values used in our experiments are listed in Section 8.2.1.

function learnLinkageRule($P$, $R$) {
  while($|P| < \text{population size}$) {
    Compute fitness of all rules in $P$ based on $R$
    $r_1, r_2 \leftarrow$ select two linkage rules from $P$ using tournament selection
    $r \leftarrow$ random number from interval $[0,1]\$
    if($r < \text{mutation probability}$) {
      $r \leftarrow$ generate random linkage rule
      $P' \leftarrow P' \cup \text{crossover}(r_1, r_2)$
    } else {
      $P' \leftarrow P' \cup \text{crossover}(r_1, r_2)$
    }
  }
  return $P$
}

section gives an overview of the GenLink algorithm. A more detailed description of GenLink is found in [22].

On starting the execution of the active learning workflow, GenLink generates an initial population of candidate linkage rules according to the method described in Section 5.1.

After a user labeled a new link, GenLink iteratively evolves the current population based on the current reference links including the newly labeled link. In each iteration, a new population is generated by creating new linkage rules from the existing population until the population size is reached. The algorithm stops when either a predefined number of iterations is reached or when one linkage rule in the population reaches an F-measure of 100%.

The pseudocode of GenLink for evolving a population of linkage rules based on a set of reference links is given in Algorithm 2.

Algorithm 2 Pseudocode of the GenLink algorithm. The specific parameter values used in our experiments are listed in Section 8.2.1.
5.1. Generating the Initial Population

In genetic programming the initial population is usually generated randomly. Previous work has shown that starting with a fully random population works well on some record linkage data sets [7]. Two circumstances increase the search space (i.e. the set of all possible linkage rules) considerably: Firstly, data sets with a high number of properties. Secondly, if data sets which are represented using different schemata are to be matched the search space includes all possible property pairs from the source and target data set. In order to reduce the size of the search space, we employ a simple algorithm which pre-selects property pairs which hold similar values: Before the population is generated, we build a list of property pairs which hold similar values as described below. Based on that, random linkage rules are built by selecting property pairs from the list and building a tree by combining random data transformations, comparisons and aggregations.

Finding Compatible Properties. The purpose of this step is to generate a list of pairs of properties that share at least one token on any of their values. For each possible property pair, the values of the entities referenced by a positive reference link are analyzed. This is done by tokenizing and lowercasing the values and generating a new property pair of the form \((p_1, p_2)\) if there is a distance measure in a provided list of functions according to which 2 tokens are similar given a certain threshold \(\theta_d\). In our experiments, we used the Levenshtein distance with a threshold of 1. The pseudocode is given in Algorithm 3.

Algorithm 3 Find compatible properties given a set of reference links \(R^+\) and a distance threshold \(\theta\)

```plaintext
pairs ← ∅
for all \((e_a, e_b)\) ∈ \(R^+\) {
  for all properties \(e_a.p_i\) and \(e_b.p_j\) {
    for all distance measures \(f^d\) {
      \(v_a \leftarrow \text{tokenize} \text{lowerCase}(e_a.p_i)\)
      \(v_b \leftarrow \text{tokenize} \text{lowerCase}(e_b.p_j)\)
      if \((f^d(v_a, v_b) < \theta_d)\) add \((pi, pj)\) to pairs
    }
  }
}
return pairs
```

Figure 5 illustrates a simple example with two entities. In this example, the following two property pairs are generated: \((\text{label}, \text{label})\) and \((\text{director}, \text{directorName})\).

Generating a Random Linkage Rule. A random linkage rule is generated according to the following rules:

First of all, a linkage rule is built consisting of a random aggregation and up to two comparisons. For each comparison a random pair from the pre-generated list of compatible properties is selected. In addition, with a possibility of 50\%, a random transformation is appended to each property.

Note that although the initial linkage rule trees are very small, this does not limit the algorithm from growing bigger trees by using the genetic operators.

5.2. Breeding

Starting with the initial population, the genetic algorithm breeds a new population by evolving selected linkage rules using the genetic operators. The linkage rules are selected from the population based on two functions: The fitness function and the selection method.

The purpose of the fitness function is to assign a value to each linkage rule which indicates how close the given linkage rule is to the desired solution. A disadvantage of using the F-measure as fitness function is that it may yield skewed results if the number of positive and negative reference links is unbalanced as it only takes the true negative rate into account. We use Matthews correlation coefficient (MCC) as fitness measure. Matthews correlation coefficient [29] is defined as the degree of the correlation between the actual and predicted classes:

\[
\text{MCC} = \frac{n_{tp} \times n_{tn} - n_{fp} \times n_{fn}}{\sqrt{(n_{tp} + n_{fp})(n_{tp} + n_{fn})(n_{tn} + n_{fp})(n_{tn} + n_{fn})}}
\]

\(n_{tp}, n_{tn}, n_{fp}\) and \(n_{fn}\) denote the number of true positives, true negatives, false positives and false negatives which are computed based on the provided reference links (ignoring the remaining part of the data set). In order to prevent linkage rules from growing indefinitely, we extend the fitness function to penalize linkage rules based on their number of operators:

\[
\text{fitness} = (1 - pf) \times \text{mcc} - pf \times \text{operatorcount}
\]

The particular value of the penalty factor \(pf\) determines the extend to which large linkage rules are penalized. Choosing an appropriate value of the penalty factor is important as setting it to big decreases the learning performance as it prevents linkage rules from growing to their optimal size. On the other hand, small values may not punish linkage rules with redundant parts sufficiently. For our experiments we empirically determined the largest penalty factor that does not decrease the learning performance and fixed it to 0.05.

Based on the fitness of each linkage rule, the selection method selects the linkage rules to be evolved. As selection method we chose tournament selection as it has been shown to produce strong results in a variety of genetic programming systems [28] and is easy to parallelize.

In order to evolve the population we employ all three common genetic operators: reproduction, crossover and
mutation. At first, 1% of the individuals with the highest fitness are directly selected for reproduction following a elitist strategy [27]. After this, new individuals are generated using crossover and mutation until the population size is reached. Instead of using subtree crossover, GenLink is using a set of specialized crossover operators. Each of these operators only operates on one aspect of the linkage rule e.g. one crossover operator builds chains of transformations while another operator recombines different comparisons. In the case of mutation a headless chicken crossover [26] i.e. the selected linkage rule is combined with a randomly generated linkage rule by using the crossover operator.

The algorithm iteratively evolves the population until either a linkage rule has been found which covers all reference links or a configured maximum number of 50 iterations is reached.

6. Query Strategy

The goal of the query strategy is to reduce the number of links that need to be labeled. While many query strategies, such as uncertainty sampling, assume that a single model is being trained by the learning method, genetic algorithms train a population of alternative models at the same time. In order to take advantage of this set of competing models, we introduce version space reduction strategies known as query-by-committee. Query-by-committee methods select the query based on the voting of a committee of candidate solutions. Ideally, the committee is a subset of the version space, which is the set of candidate solutions that is consistent with the current reference links. In our case, the committee is built by the linkage rules in the current population. Query-by-committee strategies usually aim to reduce the version space by selecting the unlabeled candidate that reduces the version space the most.

In this section we introduce the most common query-by-committee method known as query-by-vote-entropy. After that, we propose an improved query strategy for the use case of learning linkage rules. Section 8.5 will evaluate the increase in performance on several real-world data sets.

6.1. Query-By-Vote-Entropy

One of the most commonly used query strategy is known as the query-by-vote-entropy [11] strategy. The query-by-vote-entropy strategy selects the candidate for which the members in the committee disagree the most:

Definition 4 (query-by-vote-entropy). Given an unlabeled pool \( U \), the query-by-vote-entropy selects the candidate with the maximum vote entropy:

\[
x^*_V = \arg \max_{u \in U} H(P_C(u))
\]

where \( H \) denotes the entropy, which is defined for the binary case as:

\[
H(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)
\]

and \( P_C \) represents the committee voting which is defined for an unlabeled link candidate \( u \in U \) and a committee \( C \) as:

\[
P_C(u) = \frac{\sum_{l \in C} l(u)}{|C|}
\]

where \( l(u) \) denotes the result of the evaluation of a linkage rule \( l \) from the committee on a link candidate \( u \) from the unlabeled pool.

The idea of the vote entropy is that unlabeled link candidates for which either most committee members confirm the candidate, or most committee members decline the candidate, receive a low score. On the other side, link candidates for which about half of the committee members confirm the candidate while the other half declines it, receive the highest disagreement score. In the ideal case 50% of the committee members confirm the link candidate while the other 50% decline it. In that case, following the goal of reducing the version space, the version space is cut in half.

6.2. Proposed Query Strategy

Given the query-by-vote-entropy strategy as a baseline, we now present an improved strategy which is based on two observations:

- **Link Distribution:** The unlabeled links are not distributed uniformly across the similarity space but build clusters of links which convey similar information concerning the characteristics according to which both interlinked entities are similar.

- **Suboptimal Committee:** The voting committee, which is built from the population of linkage rules, may contain suboptimal linkage rules that do not cover all reference links. As a result even linkage rules that only cover a small part of the set of reference links are allowed to vote when choosing a new candidate for labeling.

The next paragraphs will take a more detailed look at both observations and show how we account for them in the proposed query strategy.

6.2.1. Accounting for Link Distribution

Usually multiple link candidates convey similar information concerning the characteristics according to which both interlinked entities are similar. For this reason, labeling a single link candidate can be representative of a large number of related link candidates. Thus, it is not necessary to label the entire pool of possible link candidates, but only a subset thereof and it is the goal of the query
strategy to minimize the size of this subset of candidates which need to be labeled.

In order to get a better understanding of this observation, we look at a simple example using a data set about movies. For illustration we only consider two dimensions: The similarity of the movie titles as well as the similarity of the release date. Figure 6 shows the distribution of a set of link candidates between movies from the LinkedMDB data set used in Section 8 according to these two characteristics. For instance, the cluster in the top-right corner represents link candidates between movies that share a similar title as well as a similar release date, while the cluster in the bottom-right corner represents link candidates between movies that share a similar title, but have been released at different dates. We can easily see that a query strategy needs to label link candidates from all 4 clusters in order to include all necessary information to learn an appropriate linkage rule.

The idea of our proposed query strategy is to distribute the links onto different clusters by only selecting links for labeling that are different from already labeled links. For measuring the extend to which a specific unlabeled link differs from an existing reference link we use the Jensen-Shannon divergence.

The Jensen-Shannon divergence is a measure of the divergence of two probability distributions. As we are dealing with binary distributions, Jensen-Shannon divergence is defined as:

\[ D_{JS}(p \parallel q) = H\left(\frac{p + q}{2}\right) - \frac{H(p) + H(q)}{2} \]

Intuitively, we can interpret the Jensen-Shannon divergence between an unlabeled link candidate and a reference link as the amount of information that is needed to encode the label of the unlabeled link, given that the label of the reference link is known. If the divergence is zero, we expect every linkage rule to return the same label for the unlabeled link. If the divergence is non-zero, we expect the information gained by labeling the link candidate is not yet contained in the set of reference links.

Based on the Jensen-Shannon divergence, we propose the query-by-divergence strategy:

\[ x_1^* = \arg\max_{u \in U} \arg\min_{r \in R} D_{JS}(P_C(u) || P_C(r)) \]

We now take a closer look at how the result of the query-by-divergence strategy is computed: At first, the Jensen-Shannon divergence is used to determine the distance of each unlabeled link to every reference link. For each unlabeled link, the inner \( \arg\min \) expression selects the divergence from the closest reference link. By that we get a list of all unlabeled links together with a divergence value for each link. From this list, the outer \( \arg\max \) expression selects the unlabeled link for which the highest divergence has been found.

6.2.2. Accounting for Suboptimal Committee

Both the query-by-vote-entropy as well as the proposed query-by-divergence are based on the committee voting of the entire committee. When active learning is used together with genetic algorithms, the committee can be built from the members of the evolved population. The population typically contains suboptimal linkage rules that do not cover all existing reference links. For instance, if a data set about movies is deduplicated, the population may contain linkage rules that solely compare by the movie titles. While these linkage rules may cover most reference links, they do not fulfill reference links which relate movies that have the same title, but have been released in different years. By allowing linkage rules to vote that only cover a subset of the reference links, they distort the final voting result.

We account for the suboptimal committee by introducing a modified version of the committee voting which is based on 3 factors:

- An unlabeled pool \( U \).
- A reference link set \( R \), which is divided into positive reference links \( R^+ \) and negative reference links \( R^- \).
- A committee of linkage rules \( C \). The committee is formed by all linkage rules in the current population.

We now define the subset of the committee which is fulfilled by a specific reference link:

**Definition 6 (Restricted Committee).** Given a reference link \( r^+ \in R^+ \), we define the subset of the committee which fulfills \( r^+ \) as:

\[ C(r^+) = \{ l \in C | l(r^+) > 0.5 \} \]

Similarly, given a reference link \( r^- \in R^- \), we define the subset of the committee which fulfills \( r^- \) as:

\[ C(r^-) = \{ l \in C | l(r^-) < 0.5 \} \]
We define the restricted committee voting similarly to the committee voting $P_C(l)$, which we already defined for the query-by-vote-entropy strategy:

**Definition 7 (Restricted Committee Voting).** Given a link $u$ and a reference link $r$, the restricted committee voting is defined as:

$$
\bar{P}_C(u, r) = \frac{\sum_{c \in \mathcal{C}(r)} P_c(u)}{|\mathcal{C}(r)|}
$$

The idea of the restricted committee voting is that only linkage rules which fulfill a specific reference link are allowed to vote.

### 6.2.3. Combined Strategy

Based on the query-by-divergence strategy and the restricted committee voting, we can now define the query strategy used by the ActiveGenLink algorithm.

**Definition 8 (Proposed Query Strategy).** The proposed query strategy selects the unlabeled link candidate that has the maximum divergence from any existing reference link:

$$
x^*_I = \arg\max_{u \in \mathcal{U}} \arg\min_{r \in \mathcal{R}} D_{JS}(\bar{P}_C(u, r) || \bar{P}_C(r, r))
$$

When assessing the divergence between a unlabeled link candidate and a reference link, using the previously defined restricted committee voting guarantees that only linkage rules that fulfill the given reference link are allowed to vote. Thus, if a linkage rule from the population does not fulfill a specific reference link, it does not distort the computation of the divergence from that particular reference link.

### 7. Building the Unlabeled Pool

The overall goal of the active learning algorithm is to create a linkage rule which is able to label all possible entity pairs as matches or non-matches with high confidence. The number of possible entity pairs can be very high for large data sets and usually far exceeds the number of actual matches. For this reason, we use an indexing approach to build a sample which does not include definitive non-matches.

Given two data sets $A$ and $B$, the initial unlabeled pool $\mathcal{U} \subset A \times B$ is built according to the following sampling process: The sampling starts by querying for all entities in both data sets. Instead of retrieving all entities at once, a stream of entities is generated for each data set. For each property in the streamed entities, all values are indexed according to the following scheme:

1. All values are normalized by removing all punctuation and converting all characters to lower case.
2. The normalized values are tokenized.
3. A set of indices is assigned to each token. The indices are generated so that tokens within an edit distance of 1 share at least one index. The MultiBlock blocking algorithm is used to generate the index [24].
4. The indices of all tokens of a value are merged. If in total more than 5 indices have been assigned to a value, 5 indices are randomly selected while discarding the remaining indices.

After the index has been generated, all pairs of entities which have been assigned the same index are added to the unlabeled pool until a configured maximum size is reached. Figure 7 illustrates the sampling of a single property.

![Figure 7: Sampling of entities by label](image)

We illustrate the indexing by looking at the index that has been generated from the geographical data set that is used in Section 8.4 for evaluating the scalability of our approach. Figure 8 shows a simplified example of the index by only including the label and the geographic coordinates for each entity. Note that each similarity measure may create multiple indexes for a single entity, which for simplicity is not considered in the Figure. In this example, a link candidate would be added to the unlabeled pool for each pair of entities that is close in the visualized index.

![Figure 8: Indexed geographical entities](image)

### 8. Evaluation

In this section, we evaluate ActiveGenLink experimentally: At first, Section 8.1 introduces the data sets that we used for the evaluation. Section 8.2 describes our experimental setup. Section 8.3 evaluates if by labeling a small number of links, the proposed active learning algorithm is capable of learning linkage rules with a similar accuracy than the supervised learning algorithm GenLink [22] on
a larger set of reference links. We evaluated the scalability of the active learning algorithm in Section 8.4. Finally in Section 8.5, we evaluate the contribution of the proposed query strategy compared to the query-by-vote-entropy strategy.

8.1. Data Sets

For evaluation, we used six data sets from three areas for which reference links are available that we could use as gold standard within our experiments:

1. We evaluate our approach with two data sets from the Ontology Alignment Evaluation Initiative and compare our results to the participating systems.
2. We evaluate the learning performance on two well-known record linkage data sets and compare the performance with an existing state-of-the-art genetic programming approach.
3. We compare the learned linkage rules with linkage rules created by a human expert for two data sets.

While the record linkage data sets are already adhering to a consistent schema, the RDF data sets are split into a source and a target data set which adhere to different schemata.

Table 4 lists the used data sets together with the number of entities as well as the number of reference links in each data set. As only positive reference links have been provided by the data set providers, we generated the negative reference links. For two positive links \((a, b) \in R^+\) and \((c, d) \in R^+\) we generated two negative links \((a, d) \in R^-\) and \((c, b) \in R^-\). For the Cora and Restaurant data set this is sound as the provided positive links are complete. Since the remaining data sources are split into source and target data sets, generating negative reference links is possible as entities in the source and target data sets are internally unique.

| Data Set         | Entities \(|A|\) | \(|B|\) | \(|R^+|\) | \(|R^-|\) |
|------------------|----------------|-------|---------|---------|
| Cora             | 1879           | 1617  | 1617    |         |
| Restaurant       | 864            | 112   | 112     |         |
| SiderDrugBank    | 924            | 4772  | 859     | 859     |
| NewYorkTimes     | 5620           | 1819  | 1920    | 1920    |
| LinkedMDB        | 199            | 174   | 100     | 100     |
| DBpediaDrugBank  | 4854           | 1403  | 1403    |         |

Table 4: The number of entities in each data set as well as the number of reference links.

Table 5 shows the number of properties in the source and target data sets and their coverage i.e. the percentage of properties which are actually set on an entity on average.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Properties ([A.P])</th>
<th>([B.P])</th>
<th>Coverage (C_A)</th>
<th>(C_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>4</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restaurant</td>
<td>5</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SiderDrugBank</td>
<td>8</td>
<td>79</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>NewYorkTimes</td>
<td>38</td>
<td>110</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>LinkedMDB</td>
<td>100</td>
<td>46</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>DBpediaDrugBank</td>
<td>110</td>
<td>79</td>
<td>0.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 5: The total number of properties in each data set as well as the percentage of properties which are actually set on an entity.

8.2. Experimental Setup

The ActiveGenLink algorithm has been implemented in the Silk Link Discovery Framework, which can be downloaded from the project homepage\(^1\). The Silk Link Discov-

\(^1\)http://silk.wbsg.de/

ery Framework supports users in discovering relationships between data items within different Linked Data sources.

Each experiment has been executed by loading all entities in the corresponding data set, but no reference links, and running ActiveGenLink on the loaded entities. Instead of using a human annotator, the link candidates that have been selected by the query strategy have been automatically labeled as correct if the link candidate has been found in the positive reference links and as incorrect otherwise. We ensured for each data set that the positive reference links are complete i.e. for each pair of matching entities there is a positive reference link. Each time a link has been labeled and after the approach updated the learned linkage rule, we evaluated the performance of the learned linkage rule using the complete set of reference links. We executed all experiments until either the learned linkage rule fully covered all reference links or 50 iterations have been reached. For each experiment we also compared the final performance to the performance of the supervised learning approach when being trained on all reference links.

Each experiment has been run 10 times, averaging the final results. All experiments have been run on a 3GHz Intel(R) Core i7 CPU with 4 cores while the Java heap space has been restricted to 1GB.

8.2.1. Parameters

Table 6 lists the parameters which have been used in all experiments for the active learning algorithm. It also lists the parameters which have been used for the genetic programming algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlabeled Pool size (</td>
<td>U</td>
</tr>
<tr>
<td>Maximum links to be labeled</td>
<td>50</td>
</tr>
<tr>
<td>Population size</td>
<td>500</td>
</tr>
<tr>
<td>Max. iterations per labeled link</td>
<td>25</td>
</tr>
<tr>
<td>Selection method</td>
<td>Tournament selection</td>
</tr>
<tr>
<td>Tournament size</td>
<td>5</td>
</tr>
<tr>
<td>Probability of Crossover</td>
<td>75%</td>
</tr>
<tr>
<td>Probability of Mutation</td>
<td>25%</td>
</tr>
<tr>
<td>Stop Condition</td>
<td>F-measure = 1.0</td>
</tr>
<tr>
<td>Penalty Factor</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 6: Active Learning Parameters
8.3. Comparison with Supervised Learning

In this section, we evaluate the performance of the ActiveGenLink approach on the same data sets as have been used to evaluate the supervised GenLink algorithm presented in [22]. We show that by labeling a small number of links, ActiveGenLink achieves a comparable performance as GenLink on the complete reference link set.

Note that the interpretation of the training F-measure is different for the supervised and the active learning evaluation: In the supervised evaluation in each run all available reference links have been split into a training set and an equal-sized validation set for the cross validation. Thus, the training F-measure denotes the performance on the training set on which the algorithm has been trained on while the validation F-measure denotes the performance on the validation set. In the active learning evaluation, the training F-measure denotes the performance on the links which have been labeled so far. Here, the validation F-measure denotes the performance on the complete reference link set.

8.3.1. Ontology Alignment Evaluation Initiative

The Ontology Alignment Evaluation Initiative (OAEI)\(^2\) is an international initiative aimed at organizing the evaluation of different ontology matching systems. In addition to schema matching, OAEI also includes an instance matching track since 2009 which regularly evaluates the ability to identify similar entities among different Linked Data sources.

We chose a data set from each the OAEI 2010 challenge as well as the OAEI 2011 challenge. For both data sets, we compared our results with the results of the participating systems in the instance matching track. In the OAEI, the systems where asked to identify similar entities in a data set without being allowed to employ existing reference links for matching. Note that, as the OAEI only compares unsupervised systems and does not consider systems which are supplied with existing reference links, our approach has an advantage over the participating systems. For that reason, we used the official OAEI results merely as a baseline for our approach and evaluated the number of links which had to be labeled in order to outperform the participating unsupervised systems.

The SiderDrugBank data set was selected from the OAEI 2010 data interlinking track\(^3\). We chose this data set amongst the other drug related data sets because it was the one for which the participating systems ObjectCoref [19] and RiMOM [38] performed the worst. This data set contains drugs from Sider, a data set of marketed drugs and their side effects, and DrugBank, containing drugs approved by the US Federal Drugs Agency. Positive reference links are provided by the OAEI.

Table 7 summarizes the active learning results for the SiderDrugBank data set. After labeling 2 links validation F-measure outperforms the unsupervised ObjectCoref System and after labeling a third link the F-measure already outperforms the RiMOM system. About 30 links had to be labeled until a linkage rule could be learned which achieves a similar F-measure than the ones learned by GenLink by using all 1718 reference links.

In order to get a better idea on how the learned linkage rule evolves during the iterations, we illustrate one active learning run by showing 3 linkage rules which have been learned after different number of iterations: Figure 9 shows a learned linkage rule after 10 links have been labeled, Figure 10 after labeling 20 links and finally Figure 11 after labeling 30 links. The first linkage rule is very simple and only consists of one comparison of the drug names in both data sets. By looking at the two subsequent linkage rules, we can see that this specific comparison, with minor modifications, was carried through all learned linkage rules. The second linkage rule introduces a second comparison with the brand name of the drug in DrugBank. This comparison is also carried over to the third linkage rule which adds a third comparison which com-

---

\(^2\)http://oaei.ontologymatching.org

\(^3\)http://oaei.ontologymatching.org/2010/im/index.html
Figure 10: Example of a learned linkage rule after labeling 20 links.

Figure 11: Example of a learned linkage rule after labeling 30 links.

parses the PubChem Compound Identifiers which identify unique chemical structures.

The NewYorkTimes data set was selected from the OAEI 2011 data interlinking track\(^4\). Amongst the 7 data sets from this track, we chose the data set for which the participating systems performed the worst on average: Interlinking locations in the New York Times data set with their equivalent in DBpedia. Besides other types of entities, the New York Times data set contains 5620 manually curated locations. In addition, it contains 1920 manually verified links between locations in the New York Times data set itself and the same location in DBpedia.

The NewYorkTimes evaluation data set has been build by extracting all 5620 locations from the official New York Times data set\(^5\). Locations in DBpedia have been retrieved by requesting each interlinked DBpedia location using the official SPARQL endpoint\(^6\).

Table 8 summarizes the active learning results for the NewYorkTimes data set. AgreementMaker and SEREMI have been outperformed after labeling 4 links. 30 links had to be labeled in order to outperform the Zhishi.links system after an F-measure of 90% has been reached by labeling 25 links. The NewYorkTimes data set is the only data set in which the maximum number of 50 links was not sufficient to achieve the same F-measure as GenLink achieved on the full reference link set.

8.3.2. Frequently Used Record Linkage Data sets

A number of data sets have been used frequently to evaluate the performance of different record linkage approaches. We evaluated our approach using the Cora data set and the Restaurant data set. The Cora data set contains citations to research papers from the Cora Computer Science research paper search engine. For each citation it contains the title, the author, the venue as well as the date of publication. The Restaurant data set \(^3\) contains a set of records from the Fodor’s and Zagat’s restaurant guides. For each restaurant it contains the name, address, phone number as well as the type of restaurant. For both data sets, we used the XML version provided at \(^7\).

Table 9 summarizes the active learning results for the Cora data set. The results show that after labeling 5 links, the learned linkage rules already achieves a F-measure of

---

\(^4\)http://oaei.ontologymatching.org/2011/instance/

\(^5\)http://data.nytimes.com/

\(^6\)http://dbpedia.org/sparql

\(^7\)http://www.hpi.uni-potsdam.de/naumann/projekte/dude_duplicate_detection.html
was sufficient to achieve the same performance as GenLink on all reference links.

8.3.3. Comparison With Manually Created Linkage Rules

In addition, we evaluated how the learned linkage rules compare to linkage rules which have been manually created by a human expert for the same data set. For this we employed 2 data sets: A data set about movies and a complex life science data set.

LinkedMDB: The LinkedMDB data set is an easy to understand data set about movies which is non-trivial as the linkage rule cannot just compare by label (different movies may have the same name), but also needs to include other properties such as the date or the director. The manually written linkage rule compares movies by their label as well as their release date. For the evaluation we used a manually created set of 100 positive and 100 negative reference links. Special care was taken to include relevant corner cases such as movies which share the same title but over 90% on the complete reference link set. After labeling 40 links, the active learning algorithm achieves similar results as GenLink on all 3234 reference links.

Table 9 summarizes the active learning results for the Restaurant data set. The last row contains the results of the supervised algorithm.

Table 9: Results for the Cora data set. The last row contains the results of the supervised algorithm.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time in s (σ)</th>
<th>Train. F1 (σ)</th>
<th>Val. F1 (σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.2 (0.4)</td>
<td>1.000 (0.000)</td>
<td>0.489 (0.037)</td>
</tr>
<tr>
<td>2</td>
<td>1.5 (0.5)</td>
<td>1.000 (0.000)</td>
<td>0.637 (0.111)</td>
</tr>
<tr>
<td>3</td>
<td>2.2 (0.1)</td>
<td>1.000 (0.000)</td>
<td>0.776 (0.075)</td>
</tr>
<tr>
<td>4</td>
<td>2.6 (0.1)</td>
<td>1.000 (0.000)</td>
<td>0.778 (0.029)</td>
</tr>
<tr>
<td>5</td>
<td>3.4 (0.1)</td>
<td>1.000 (0.000)</td>
<td>0.761 (0.035)</td>
</tr>
<tr>
<td>6</td>
<td>4.2 (0.5)</td>
<td>1.000 (0.000)</td>
<td>0.870 (0.075)</td>
</tr>
<tr>
<td>7</td>
<td>5.0 (0.2)</td>
<td>1.000 (0.000)</td>
<td>0.932 (0.059)</td>
</tr>
<tr>
<td>8</td>
<td>5.6 (0.2)</td>
<td>1.000 (0.000)</td>
<td>0.935 (0.061)</td>
</tr>
<tr>
<td>9</td>
<td>6.2 (0.0)</td>
<td>1.000 (0.000)</td>
<td>0.993 (0.002)</td>
</tr>
<tr>
<td>10</td>
<td>6.8 (0.0)</td>
<td>1.000 (0.000)</td>
<td>0.993 (0.003)</td>
</tr>
<tr>
<td>GL</td>
<td>6.3 (5.3)</td>
<td>0.996 (0.004)</td>
<td>0.993 (0.006)</td>
</tr>
</tbody>
</table>

Table 10: Results for the Restaurant data set. The last row contains the results of the supervised algorithm.

Table 10 summarizes the active learning results for the LinkedMDB data set. The results show that the active learning algorithm achieve an F-measure of over 90% by labeling 5 links. After labeling 15 links the learned linkage rules almost reached an F-measure of 100%.

DBpediaDrugBank: While the vast majority of linkage rules commonly used in the Linked Data context are very simple, a few of them employ more complex structures. Interlinking drugs in DBpedia and DrugBank is an example where the original linkage rule which has been produced by a human expert is very complex. In order to match two drugs, it compares the drug names and their synonyms as well as a list of well-known and used identifiers (e.g. the CAS number8) which are provided by both data sets but are missing for many entities. In total, the manually written linkage rule uses 13 comparisons and 33 transformations. This includes complex transformations such as replacing specific parts of the strings. All 1,403 links which have been generated by executing the original linkage rule have been used as positive reference links.

Table 11 summarizes the active learning results for the DBpediaDrugBank data set. The results show that the active learning algorithm achieve an F-measure of over 90% by labeling 10 links. After labeling 50 links the learned linkage rules achieve an F-measure of 99.4%. The generated linkage rules on average only use 5.6 comparisons

Table 11: Results for the LinkedMDB data set.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time in s (σ)</th>
<th>Train. F1 (σ)</th>
<th>Val. F1 (σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.3 (0.9)</td>
<td>1.000 (0.000)</td>
<td>0.643 (0.015)</td>
</tr>
<tr>
<td>2</td>
<td>3.3 (1.0)</td>
<td>1.000 (0.000)</td>
<td>0.706 (0.012)</td>
</tr>
<tr>
<td>3</td>
<td>4.3 (1.0)</td>
<td>1.000 (0.000)</td>
<td>0.815 (0.010)</td>
</tr>
<tr>
<td>4</td>
<td>6.7 (2.4)</td>
<td>1.000 (0.000)</td>
<td>0.818 (0.005)</td>
</tr>
<tr>
<td>5</td>
<td>7.9 (2.5)</td>
<td>1.000 (0.000)</td>
<td>0.911 (0.020)</td>
</tr>
<tr>
<td>10</td>
<td>26.9 (12.1)</td>
<td>0.955 (0.045)</td>
<td>0.987 (0.013)</td>
</tr>
<tr>
<td>15</td>
<td>40.1 (19.0)</td>
<td>1.000 (0.000)</td>
<td>0.999 (0.002)</td>
</tr>
</tbody>
</table>

Table 11: Results for the LinkedMDB data set.

Table 12: Results for the DBpediaDrugBank data set.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time in s (σ)</th>
<th>Train. F1 (σ)</th>
<th>Val. F1 (σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23.4 (3.3)</td>
<td>1.000 (0.000)</td>
<td>0.740 (0.124)</td>
</tr>
<tr>
<td>2</td>
<td>38.0 (8.4)</td>
<td>1.000 (0.000)</td>
<td>0.748 (0.118)</td>
</tr>
<tr>
<td>3</td>
<td>52.0 (12.6)</td>
<td>1.000 (0.000)</td>
<td>0.646 (0.017)</td>
</tr>
<tr>
<td>4</td>
<td>85.3 (29.7)</td>
<td>1.000 (0.000)</td>
<td>0.797 (0.134)</td>
</tr>
<tr>
<td>5</td>
<td>101.3 (32.7)</td>
<td>1.000 (0.000)</td>
<td>0.813 (0.150)</td>
</tr>
<tr>
<td>10</td>
<td>250.9 (56.1)</td>
<td>1.000 (0.000)</td>
<td>0.945 (0.030)</td>
</tr>
<tr>
<td>15</td>
<td>522.8 (259.9)</td>
<td>1.000 (0.000)</td>
<td>0.983 (0.007)</td>
</tr>
<tr>
<td>20</td>
<td>700.7 (354.7)</td>
<td>1.000 (0.000)</td>
<td>0.983 (0.007)</td>
</tr>
<tr>
<td>25</td>
<td>2558.8 (2107.6)</td>
<td>1.000 (0.000)</td>
<td>0.984 (0.009)</td>
</tr>
<tr>
<td>30</td>
<td>4461.2 (3916.9)</td>
<td>1.000 (0.000)</td>
<td>0.984 (0.009)</td>
</tr>
<tr>
<td>35</td>
<td>6832.3 (6200.4)</td>
<td>1.000 (0.000)</td>
<td>0.986 (0.010)</td>
</tr>
<tr>
<td>40</td>
<td>9885.8 (9104.3)</td>
<td>0.993 (0.007)</td>
<td>0.925 (0.072)</td>
</tr>
<tr>
<td>45</td>
<td>14951.6 (13845.9)</td>
<td>0.994 (0.006)</td>
<td>0.989 (0.008)</td>
</tr>
<tr>
<td>50</td>
<td>21387.5 (19937.3)</td>
<td>1.000 (0.000)</td>
<td>0.993 (0.008)</td>
</tr>
</tbody>
</table>

Table 12: Results for the DBpediaDrugBank data set.

8 A unique numerical identifier assigned by the “Chemical Abstracts Service”
and 3.2 transformations. Thus, the learned linkage rules use less than half of the comparisons and only one-tenth of the transformations of the human written linkage rules.

8.4. Scalability

In this experiment we show that ActiveGenLink is able to scale to large data sets. For evaluation we use the example of learning a linkage rule for interlinking settlements in DBpedia and LinkedGeoData. At the time of writing, DBpedia contains 323,257 settlements while LinkedGeoData contains 560,123 settlements. The execution of the learned linkage rules generates over 70,000 links. While, in the case of passive learning the learning algorithm only needs to take the provided reference links into account, active learning also needs to take all unlabeled data into account in order to generate the queries. As the unlabeled data consists of the complete Cartesian product, which in this example amounts to over 180 billion pairs, the active learning algorithm clearly cannot work on the whole unlabeled set. For this reason, this experiment evaluates if the sampling algorithm, which has been introduce in Section 6, to two other strategies:

- **Random**: Selects a random link from the unlabeled pool for labeling (baseline).
- **Entropy**: Selects a link according to the query-by-vote-entropy strategy.

As all query strategies would eventually label the complete Cartesian product of entities, they would also eventually achieve the maximum f-Measure, which can be reached by the supervised learning algorithm. The purpose of this section is to show that our proposed query strategy achieves a higher f-Measure when labeling a small number of links.

Table 15 compares the validation F-measure after labeling 10 links. In all cases, the query-by-vote-entropy strategy as well as our proposed query strategy outperformed the random baseline. Our approach outperforms the query-by-vote-entropy strategy on all data sets. For the restaurant data set, the query-by-vote-entropy strategy achieved the same F-measure as achieved by GenLink on all reference links.

### 8.4.1. Passive Learning

Table 13 summarizes the cross validation results when running GenLink on the set of reference link. In all runs, an F-measure of 100% has been reached before the 25th iteration.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time (σ)</th>
<th>Train. F1 (σ)</th>
<th>Val. F1 (σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.6s (1.0)</td>
<td>0.984 (0.025)</td>
<td>0.932 (0.039)</td>
</tr>
<tr>
<td>10</td>
<td>3.8s (2.1)</td>
<td>0.996 (0.007)</td>
<td>0.932 (0.039)</td>
</tr>
<tr>
<td>20</td>
<td>3.8s (2.3)</td>
<td>0.998 (0.004)</td>
<td>0.964 (0.032)</td>
</tr>
<tr>
<td>25</td>
<td>4.0s (2.4)</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
</tr>
</tbody>
</table>

Table 13: Passive learning

### 8.4.2. Active Learning

Next we evaluated if ActiveGenLink is able to build a reference link set interactively. We employed the same setup as used in the previous experiment. Table 14 shows results for each iteration.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time</th>
<th>Train. F1 (σ)</th>
<th>Val. F1 (σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7.3s</td>
<td>1.000 (0.000)</td>
<td>0.982 (0.023)</td>
</tr>
<tr>
<td>10</td>
<td>15.6s</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
</tr>
</tbody>
</table>

Table 14: Active learning

For each iteration it shows the runtime, the F-measure based on the manually confirmed links (Training F1) and the F-measure based on the full reference link set (Validation F1). The runtime only include the time needed by the algorithm itself and not the time needed by the human to label the link candidates. It does further not include the time needed to build the initial unlabeled pool. As the public endpoints of DBpedia and LinkedGeoData have been used, which offer very restricted query performance, loading the initial unlabeled pool required about 2 hours.

In all three runs, the algorithm managed to learn a linkage rule with an F-measure of 100% after the second iteration. In the first iteration it missed the case that two cities with the same name may in fact relate to different cities. In the second iteration it managed to include this rare case in the proposed link candidates.

On average, ActiveGenLink needed about 1.5 seconds for each iteration which includes learning a linkage rule from the existing reference links and selecting a link from the unlabeled pool.

### 8.5. Comparison of Different Query Strategies

In this section, we compare the performance of the proposed query strategy, which has been presented in Section 6, to two other strategies:

- **Random**: Selects a random link from the unlabeled pool for labeling (baseline).
- **Entropy**: Selects a link according to the query-by-vote-entropy strategy.

Table 15 compares the validation F-measure after labeling 10 links. In all cases, the query-by-vote-entropy strategy achieved a higher f-Measure when labeling a small number of links.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Random</th>
<th>Entropy</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>0.604 (0.222)</td>
<td>0.841 (0.041)</td>
<td>0.917 (0.035)</td>
</tr>
<tr>
<td>Restaurant</td>
<td>0.568 (0.195)</td>
<td>0.888 (0.029)</td>
<td>0.993 (0.092)</td>
</tr>
<tr>
<td>SiderDrug</td>
<td>0.309 (0.189)</td>
<td>0.666 (0.007)</td>
<td>0.795 (0.044)</td>
</tr>
<tr>
<td>NYT</td>
<td>0.467 (0.174)</td>
<td>0.756 (0.080)</td>
<td>0.809 (0.039)</td>
</tr>
<tr>
<td>LinkedMDB</td>
<td>0.774 (0.235)</td>
<td>0.948 (0.035)</td>
<td>0.988 (0.005)</td>
</tr>
<tr>
<td>DBpediaDrug</td>
<td>0.654 (0.146)</td>
<td>0.902 (0.076)</td>
<td>0.953 (0.011)</td>
</tr>
</tbody>
</table>

Table 15: Query Strategy: F-measure after 10 iterations
vote-entropy also reaches the same F-measure as GenLink on all reference links. For the remaining data sets, our approach still outperforms query-by-vote-entropy strategy.

9. Related Work

There is a large body of work on unsupervised entity matching as well as on supervised learning of linkage rules.

Unsupervised Learning. In the recent years, a number of approaches have been proposed for unsupervised entity matching. ObjectCoref [19, 20] is a self-training interlinking approach, which starts with a kernel that consists of known equivalences and iteratively extends this kernel with discriminative property-value pairs. RiMOM [38] and AgreementMaker [10] are approaches for ontology matching which have been extended with matchers for instance matching. SERIMI [2] is another unsupervised interlinking approach, which matches entities based on their labels as well as their structural similarity. Zhishi.links [33] employs an indexing technique on the labels of the entities as well as on discovered homonyms, which allows it to scale to larger data sets than similar approaches. SLINT is a recent system, which also employs an indexing technique to improve its performance.

The Ontology Alignment Evaluation Initiative (OAEI) aims at evaluating different approach for unsupervised vocabulary as well as instance matching. In general, while there are systems that yield good results on some data sets used by the OAEI, they don’t achieve satisfactory results on all data sets [16].

Supervised Learning. In order to improve the linking accuracy, a number of supervised approaches for learning linkage rules from existing reference links have been proposed.

Many learning approaches for entity matching can be categorized in algorithms that learn linear classifiers as well as algorithms that learn threshold-based boolean classifiers [1]. One popular application of support vector machines (SVMs) [9] to entity matching is MARLIN (Multiply Adaptive Record Linkage with INduction) [3], which uses SVMs to learn linear classifiers. Threshold-based boolean classifiers are usually represented with decision trees. Active Atlas [36, 37] learns linkage rules that are based on decision trees that combine a set of predefined transformations and similarity measures. TAILOR [15] is another tool which employs decision trees to learn linkage rules.

Another promising approach for supervised entity matching is to use genetic programming to learn linkage rules that combine operators into an operator tree. To the best of our knowledge, genetic programming for supervised learning of linkage rules has only been applied by Carvalho et al. [12, 7, 13] as well as in our work presented in [21, 22].

Active Learning. The field of related work that applies active learning to entity matching is significantly smaller.

Arasu et al. [1] propose a scalable active learning approach for entity matching by introducing the assumption of monotonicity of precision. While they show that their approach can scale to large data sets, it is only able to learn simple linear or boolean classifiers, while our approach is capable of learning expressive linkage rules which include non-linear aggregation hierarchies and data transformations.

RAVEN [31] is an approach for active learning of linear or boolean linkage rules which is specifically targeted at RDF-based data sets. RAVEN suffers from the same limitation as the approach by Arasu et al. as it only covers linear and boolean classifiers.

Freitas et al. [14] present an approach which combines genetic programming and active learning to learn rules for record deduplication. Their algorithm is based on a supervised genetic programming algorithm proposed by Carvalho et al. [13]. Their approach uses genetic programming to learn how to combine a set of predefined pairs of the form $\langle$attribute, similarity function$\rangle$ (e.g. $<$name, Jaro$>$) into a linkage rule. These pairs can be combined by the genetic programming method to a linkage rule tree by using mathematical functions (e.g. $+$, $-$, $\ast$, $/$, $\exp$) and constants. Carvalho et al. show that their method produces better results as the state-of-the-art SVM based approach by MARLIN [13]. Their approach is very expressive although it cannot express data transformations. On the downside, using mathematical functions to combine the similarity measures does not fit any commonly used linkage rule model [17] and leads to complex and difficult to understand linkage rules.

EAGLE [32] is another approach which applies genetic programming and active learning to the problem of learning linkage rules interactively. EAGLE learns linkage rules which are represented as a tree, which allows it to learn rules with a similar complexity as ours, but does not support transformations.

EAGLE as well as the genetic programming approach by Freitas et al., both use query strategies that are based on the disagreement of the committee members. Similar to the query-by-vote-entropy strategy discussed earlier, their query strategies select the link candidate for which the committee is the most uncertain. ActiveGenLink uses an refined query strategy which accounts for suboptimal committee members as well as aims for a uniform distribution of reference links in the similarity space.

To the best of our knowledge, none of the existing active learning approaches for learning linkage rules support

<table>
<thead>
<tr>
<th></th>
<th>Random</th>
<th>Entropy</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>0.762</td>
<td>0.938</td>
<td>0.945</td>
</tr>
<tr>
<td>Restaurant</td>
<td>0.707</td>
<td>0.994</td>
<td>0.993</td>
</tr>
<tr>
<td>SiderDrug</td>
<td>0.615</td>
<td>0.926</td>
<td>0.954</td>
</tr>
<tr>
<td>NYT</td>
<td>0.543</td>
<td>0.741</td>
<td>0.859</td>
</tr>
<tr>
<td>LinkedMDB</td>
<td>0.885</td>
<td>0.973</td>
<td>0.989</td>
</tr>
<tr>
<td>DBpediaDrug</td>
<td>0.788</td>
<td>0.973</td>
<td>0.989</td>
</tr>
</tbody>
</table>

Table 16: Query Strategy: F-measure after 20 iterations
of data transformations to normalize values prior to comparison.

10. Implementation

This section gives an overview of the implementation of ActiveGenLink in the Silk Workbench which is part of the Silk Link Discovery Framework. The Silk Workbench is a web application which guides the user through the process of interlinking different data sources. The Silk Link Discovery Framework is available for download\(^9\) under the terms of the Apache License and all experiments that are presented in this paper can thus be repeated by the interested reader.

The Silk Workbench supports learning linkage rules using the active learning approach presented in this article: In each iteration, it shows the 5 most uncertain links to the user for confirmation. Figure 12 shows an example of a set of links which are to be verified by the user. After the user confirmed or declined a set of links, the Workbench evolves the current population of linkage rules. Figure 13 shows an excerpt of an evolved population.

Learned linkage rules can be viewed and edited in a graphical editor depicted in Figure 14. The editor enables the user to experiment and see how changes of the linkage rule affect the accuracy of the generated links. The editor is divided in two parts: The right part shows the linkage rule and enables the user to modify it. The left pane contains the most frequent used property paths for the given data sets as well as a list of linkage rule operators, which can be used to modify to the current linkage rule. In the top right corner of the editor precision, recall and F-measure based on the given reference links are shown.

11. Conclusion and Future Work

In this article, we presented the ActiveGenLink algorithm, which combines genetic programming and active learning in order to learn expressive linkage rules which include data transformations and combine different similarity measures non-linearly. We reduce the manual effort of interlinking data sources by automating the generation of linkage rules. The user is only required to perform the much simpler task of confirming or declining a set of link candidates that yield a high information gain. We proposed a novel query strategy, which requires the user to label fewer links than the query-by-vote-entropy strategy. Our experiments show that our algorithm is capable of learning accurate linkage rules after asking the user a relatively small number of questions (between 15 and 50 within our experiments).

Future work will focus on further improving the distribution of the link candidates, which are chosen by the query strategy for manual labeling. For this we will explore into clustering algorithms in order to select links from the unlabeled pool which are in the center of diverse cluster of similar unlabeled link candidates.

\(^9\)http://silk.wbsg.de/
Acknowledgment

This work was supported in part by the EU FP7 project LOD2 - Creating Knowledge out of Interlinked Data10 (Ref. No. 257943).

References


10http://lod2.eu/