EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming

Axel-Cyrille Ngonga Ngomo and Klaus Lyko

Department of Computer Science
University of Leipzig Johannisgasse 26, 04103 Leipzig
ngonga@informatik.uni-leipzig.de, lyko.klaus@gmail.com
http://limes.sf.net

Abstract. With the growth of the Linked Data Web, time-efficient approaches for computing links between data sources have become indispensable. Most Link Discovery frameworks implement approaches that require two main computational steps. First, a link specification has to be explicated by the user. Then, this specification must be executed. While several approaches for the time-efficient execution of link specifications have been developed over the last few years, the discovery of accurate link specifications remains a tedious problem. In this paper, we present EAGLE, an active learning approach based on genetic programming. EAGLE generates highly accurate link specifications while reducing the annotation burden for the user. We evaluate EAGLE against batch learning on three different data sets and show that our algorithm can detect specifications with an F-measure superior to 90% while requiring a small number of questions.

1 Introduction

The growth of the Linked Data Web over the last years has led to a compendium of currently more than 30 billion triples [3]. Yet, it still contains a relatively low number of links between knowledge bases (less than 2% at the moment). Devising approaches that address this problem still remains a very demanding task. This is mainly due to the difficulty behind Link Discovery being twofold: First, the quadratic complexity of Link Discovery requires time-efficient approaches that can efficiently compute links when given a specification of the conditions under which a link is to be built [14,21] (i.e., when given a so-called *link specification*). Such specifications can be of arbitrary complexity, ranging from a simple comparison of labels (e.g., for finding links between countries) to the comparison of a large set of features of different types (e.g., using population, elevation and labels to link villages across the globe). In previous work, we have addressed this task by developing the LIMES[1] framework. LIMES provides time-efficient approaches for Link Discovery and has been shown to outperform other frameworks significantly [20].

[1] http://limes.sf.net

E. Simperl et al. (Eds.): ESWC 2012, LNCS 7295, pp. 149–163, 2012.

The second difficulty behind Link Discovery lies in the detection of accurate link specifications. Most state-of-the-art Link Discovery frameworks such as LIMES and SILK [14] adopt a property-based computation of links between entities. To ensure that links can be computed with a high accuracy, these frameworks provide (a) a large number of similarity measures (i.e., Levenshtein, Jaccard for strings) for comparing property values and (b) manifold means for combining the results of these measures to an overall similarity value for a given pair of entities. When faced with this overwhelming space of possible combinations, users often adapt a time-demanding trial-and-error approach to detect an accurate link specification for the task at hand. There is consequently a blatant need for approaches that support the user in the endeavor of finding accurate link specifications. From a user's perspective, approaches for the semi-automatic generation of link specification must support the user by

1. reducing the time frame needed to detect a link specification (time efficiency),
2. generating link specifications that generate a small number of false positives and negatives (accuracy) and
3. providing the user with easily readable and modifiable specifications (readability).

In this paper, we present the EAGLE algorithm, a supervised machine-learning algorithm for the detection of link specifications that abides by the three criteria presented above. One of the main drawbacks of machine-learning approaches is that they usually require a large amount of training data to achieve a high accuracy. Yet, the generation of training data can be a very tedious process. EAGLE surmounts this problem by implementing an active learning approach [27]. Active learning allows the interactive annotation of highly informative training data. Therewith, active learning approaches can minimize the amount of training data necessary to compute highly accurate link specifications.

Overall, the contributions of this paper are as follows:

- We present a novel active learning approach to learning link specifications based on genetic programming.
- We evaluate our approach on three different data sets and show that we reach F-measures of above 90% by asking between 10 and 20 questions even on difficult data sets.
- We compare our approach with state-of-the-art approaches on the DBLP-ACM dataset and show that we outperform them with respect to runtime while reaching a comparable accuracy.

The advantages of our approach are manifold. In addition to its high accuracy, it generates readable link specifications which can be altered by the user at will. Furthermore, given the superior runtime of LIMES on string and numeric properties, our approach fulfills the requirements for use in an interactive setting. Finally, our approach only requires very small human effort to discover link specifications of high accuracy as shown by our evaluation.

The rest of this paper is organized as follows: First, we give a brief overview of the state of the art. Thereafter, we present the formal framework within which

EAGLE is defined. This framework is the basis for the subsequent specification of our approach. We then evaluate our approach with several parameters on three different data sets. We demonstrate the accuracy of our approach by computing its F-measure. Moreover, we show that EAGLE is time-efficient by comparing its runtime with that of other approaches on the ACM-DBLP dataset. We also compare our approach with its non-active counterpart and study when the use of active learning leads to better results.

2 Related Work

Over the last years, several approaches have been developed to address the time complexity of link discovery. Some of these approaches focus on particular domains of applications. For example, the approach implemented in RKB knowledge base (RKB-CRS) [11] focuses on computing links between universities and conferences while GNAT [25] discovers links between music data sets. Further simple or domain-specific approaches can be found in [9,23,13,28,24]. In addition, domain-independent approaches have been developed, that aim to facilitate link discovery all across the Web. For example, RDF-AI [26] implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of data sets. SILK [14] is a time-optimized tool for link discovery. It implements a multi-dimensional blocking approach that is guaranteed to be lossless thanks to the overlapping blocks it generates. Another lossless Link Discovery framework is LIMES [20], which addresses the scalability problem by implementing time-efficient similarity computation approaches for different data types and combining those using set theory. Note that the task of discovering links between knowledge bases is closely related with record linkage [30,10,5,17]. To the best of our knowledge, the problem of discovering accurate link specifications has only been addressed in very recent literature by a small number of approaches: The SILK framework [14] now implements a batch learning approach to discovery link specifications based on genetic programming which is similar to the approach presented in [6]. The algorithm implemented by SILK also treats link specifications as trees but relies on a large amount of annotated data to discover high-accuracy link specifications. The RAVEN algorithm [22] on the other hand is an active learning approach that treats the discovery of specifications as a classification problem. It discovers link specifications by first finding class and property mappings between knowledge bases automatically. RAVEN then uses these mappings to compute linear and boolean classifiers that can be used as link specifications. A related approach that aims to detect discriminative properties for linking is that presented by [29]. In addition to these approaches, several machine-learning approaches have been developed to learn classifiers for record linkage. For example, machine-learning frameworks such as FEBRL [7] and MARLIN [4] rely on models such as Support Vector Machines [16,8], decision trees [31] and rule mining [1] to detect classifiers for record linkage. Our approach, EAGLE, goes beyond previous work in three main ways. First, it is an active learning approach. Thus, it does not require the large amount of training data required by batch learning approaches such as FEBRL, MARLIN and

SILK. Furthermore, it allows to use the full spectrum of operations implemented in LIMES. Thus, it is not limited to linear and boolean classifiers such as those generated by FeBRL and RAVEN. Finally, it can detect property and class mappings automatically. Thus, it does not need to be seeded to converge efficiently like previous approaches [14].

3 Preliminaries

In the following, we present the core of the formalization and notation necessary to implement EAGLE. We first formalize the Link Discovery problem. Then, we give an overview of the grammar that underlies links specifications in LIMES and show how the resulting specifications can be represented as trees. We show how the discovery of link specifications can consequently be modeled as a genetic programming problem. Subsequently, we give some insight in active learning and then present the active learning model that underlies our work.

3.1 Link Discovery

The link discovery problem, which is similar to the record linkage problem, is an ill-defined problem and is consequently difficult to model formally [2]. In general, link discovery aims to discover pairs $(s, t) \in S \times T$ related via a relation R.

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

The sets S resp. T are usually (not necessarily disjoint) subsets of the instances contained in two (not necessarily disjoint) knowledge bases $\mathcal{K}_S$ resp. $\mathcal{K}_T$. One way to automate this discovery is to compare the $s \in S$ and $t \in T$ based on their properties using a (complex) similarity measure σ. Two entities $s \in S$ and $t \in T$ are then considered to be linked via R if $\sigma(s, t) \geq \theta$ [21]. The specification of the sets S and T and of this similarity condition is usually carried out within a *link specification.*

Definition 2 (Link Specification). *A link specification consists of three parts: (1) two sets of restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$ resp. $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$ that specify the sets S resp. T, (2) a specification of a complex similarity metric σ via the combination of several atomic similarity measures σ_1, ..., σ_n and (3) a set of thresholds θ_1, ..., θ_n such that θ_i is the threshold for σ_i.*

A restriction $\mathcal{R}$ is generally a logical predicate. Typically, restrictions in link specifications state the `rdf:type` of the elements of the set they describe, i.e., $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass` or the features that the elements of the set must have, e.g., $\mathcal{R}(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each $s \in S$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, while each $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. Each similarity σ_i is used to compare pairs of property values of instances s and t. EAGLE relies on the approach presented in [20] to detect the class and property mappings. Consequently, in the remainder of this paper, the term *link specification* will be used to denote the complex similarity condition used to determine whether two entities should be linked.

3.2 Link Specifications as Trees

Our definition of a link specification relies on the definition of *atomic similarity measures* and *similarity measures*. Generally, a similarity measure m is a function such that $m : S \times T \rightarrow [0,1]$. We call a measure atomic (dubbed $atomicMeasure$) when it relies on exactly one similarity measure σ (e.g., trigrams similarity for strings) to compute the similarity of a pair $(s,t) \in S \times T$. A similarity measure m is either an atomic similarity measure $atomicMeasure$ or the combination of two similarity measures via a metric operator *metricOp* such as MAX, MIN and linear combinations ADD.

1. $m \rightsquigarrow atomicMeasure$
2. $m \rightsquigarrow \mathit{metricOp}(m_1, m_2)$

We call a link specification atomic (*atomicSpec*) if it compares the value of a measure m with a threshold θ, thus returning the pairs (s,t) that satisfy the condition $\sigma(s,t) \geq \theta$. A link specification $\mathit{spec}(m,\theta)$ is either an atomic link specification or the combination of two link specifications via specification operators *specOp* such as AND (intersection of the set of results of two specs), OR (union of the result sets), XOR (symmetric set difference), or $DIFF$ (set difference). Thus, the following grammar for specifications holds :

1. $\mathit{spec}(m,\theta) \rightsquigarrow \mathit{atomicSpec}(m,\theta)$
2. $\mathit{spec}(m,\theta) \rightsquigarrow \mathit{specOp}(\mathit{spec}(m_1,\theta_1), \mathit{spec}(m_2,\theta_2))$

Each link specification that abides by the grammar specified above can be consistently transformed into a tree that contains the central constructs shown in Figures 1, 2, 3 and 4.

4 Approach

As we have formalized link specifications as trees, we can use Genetic Programming (GP) to solve the problem of finding the most appropriate complex link specification for a given pair of knowledge bases. Given a problem, the basic idea behind genetic programming [18] is to generate increasingly better solutions of the given problem by applying a number of genetic operators to the current population. In the following, we will denote the population at time t by g^t. Genetic operators simulate natural selection mechanisms such as mutation and reproduction to enable the creation of individuals that best abide by a given fitness function. One of the key problems of genetic programming is that it is a non-deterministic procedure. In addition, it usually requires a large training data set to detect accurate solutions. In this paper, we propose the combination of GP and active learning [27]. Our intuition is that by merging these approaches, we can infuse some determinism in the GP procedure by allowing it to select the most informative data for the population. Thus, we can improve the convergence of GP approaches while reducing the labeling effort necessary to use them. In the following, we present our implementation of the different GP operators on link specifications and how we combine GP and active learning.

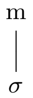

Fig. 1. Atomic measure

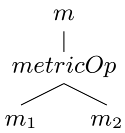

Fig. 2. Complex measure

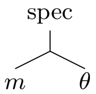

Fig. 3. Atomic specification

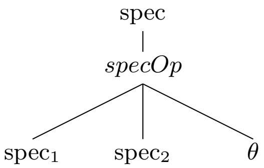

Fig. 4. Complex specification

4.1 Overview

Algorithm 1 gives an overview of the approach implemented by EAGLE. After the detection of matching classes and properties using the approach explicated in [22], we begin by generating a random population of individual link specifications. To evolve a population to the next one a number of steps is required: First, all existing individuals must be assigned a fitness score. This score reflects how well a link specification performs on the training data at hand. Subsequently, the genetic operators reproduction, mutation and crossover are applied to the individuals of the current population in order to let the individuals adapt to the problem. The *reproduction* determines which individual is carried into the next generation. Note that throughout this paper, we use a tournament setting of two randomly chosen individuals. The *mutation* operator can applied to a single individual. It alters this individual by randomly changing parts of its tree (i.e., of his genome) with the aim of creating a new individual. The *crossover* operator also aims at generating new individuals. It achieves this goal by crossing over branches of the program trees of two parent individuals.

Algorithm 1. EAGLE

```
Require: Specification of the two knowledge bases $KS$ and $KT$
  Get set $S$ and set $T$ of instances as specified in $KS$ respectively $KT$.
  Get property mapping ($KS$, $KT$)
  Get reference mapping by asking user to label $n$ random pairs $(s, t) \in S \times T$
  repeat
    Evolve population(population,size) generations times.
    Compute $n$ most informative link candidates and ask user to label them.
  until stop condition reached
```

Algorithm 2. Evolves a population

```
if population is empty then
    create size random individuals
end if
Compute fitness of population
Apply genetic operators to population
return population
```

In the following, we will explicate each of the steps of our algorithm in more detail. Each of these steps will be exemplified by using the link specification shown in Figure 5.

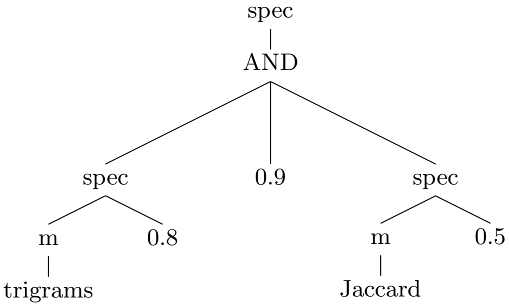

Fig. 5. Examplary link specification

4.2 Evolution of Population

Evolution is the primary process which enables GP to solve problems and drives the development of efficient solutions for a given problem. At the beginning of our computation the population is empty and must be built by individuals generated randomly. This is carried out by generating random trees whose nodes are filled with functions or terminals as required. For this paper, we defined the operators (functions and terminals) in the genotype for the problem to generate link specifications as follows: all *metricOp* and *specOp* were set to be functions. Terminal symbols were thresholds and measures. Note that these operators can be extended at will. In addition, all operators were mapped to certain constraints so as to ensure that EAGLE only generates valid program trees. For example, the operator that compares numeric properties only accepts terminals representing numeric properties from the knowledge bases.

Let g^t be the population at the iteration t. To evolve a population to the generation g^{t+1} we first determine the fitness of all individuals of generation g^t (see Section 4.3). These fitness values build the basis for selecting individuals for the genetic operator reproduction. We use a tournament setting between two selected individuals to decide which one is copied to the next generation g^{t+1}.

On randomly selected individuals the operator mutation is applied according with a probability called the *mutation rate*. The mutation operator changes single nodes in the program tree of the individuals. A mutation can affect an individual in three different ways: First, it can alter the thresholds used by the individual. Second, a mutation can alter the properties contained in the individual's genome. Finally, mutations can modify the measures included in the individuals (see Figure 6). The third genetic operator, *crossover*, operates on two parent individuals and builds a new offspring by swapping two random subtrees of the parent genotypes. Figure 7 exemplifies the functionality of the crossover operator.

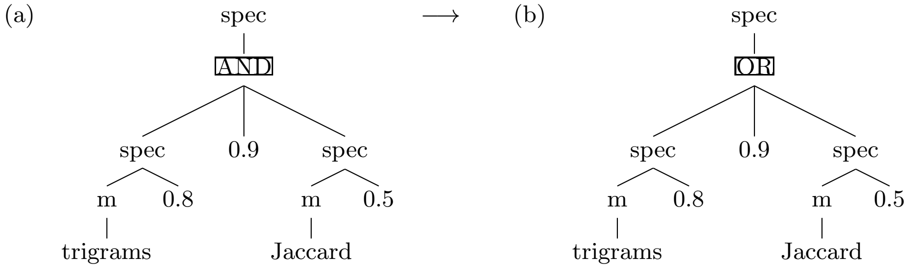

Fig. 6. Mutation example. Mutation changes boolean operator.

The individuals selected to build the population of g^{t+1} are the n fittest from the union of the set of newly created individuals and g^t. Note that we iteratively generate new populations of potential fitter individuals.

4.3 Fitness

The aim of the fitness function is to approximate how well a solution (i.e., a link specification) solves the problem at hand. In the supervised machine learning setting, this is equivalent to computing how well a link specification maps the training data at hand. To determine the fitness of an individual we first build the link specification that is described by the tree at hand. Given the set of available training data $\mathcal{O} = \{(x_i, y_i) \in S \times T\}$, we then run the specification by using the sets $S(\mathcal{O}) = \{s \in S : \exists t \in T : (s,t) \in \mathcal{O}\}$ and $T(\mathcal{O}) = \{t \in T : \exists s \in S : (s,t) \in \mathcal{O}\}$. The result of this process is a mapping $\mathcal{M}$ that is then evaluated against $\mathcal{O}$ by the means of the standard F-measure defined as

$$\frac{2PR}{P+R} \text{ where } P = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{M}|} \text{ and } R = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{O}|}. \tag{1}$$

Note that by running the linking on $S(\mathcal{O})$ and $T(\mathcal{O})$, we can significantly reduce EAGLE's runtime.

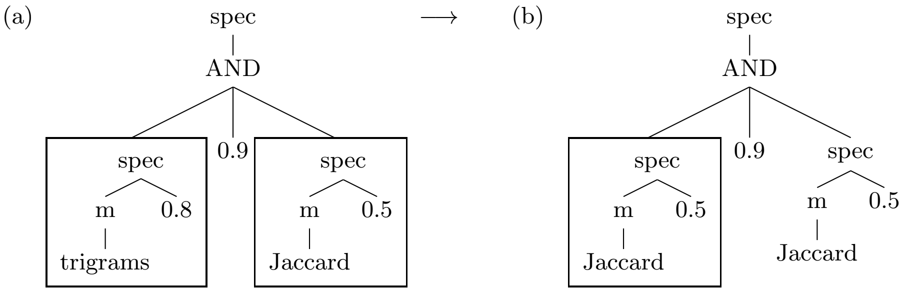

Fig. 7. Crossover example. Consider we have two individuals with a program tree like in (a). A crossover operation can replace subtrees to produce an offspring like (b).

4.4 Computation of Most Informative Link Candidates

The main idea behind the reduced of the amount of labeling effort required by active learning approaches is that they only required highly informative training data from the user. Finding these most informative pieces of information is usually carried out by measuring the amount of information that the labeling of a training data item would bear. Given the setting of EAGLE in which several possible solutions co-exist, we opted for applying the idea of active learning by committees as explicated in [19]. The idea here is to consistently entertain a finite and incomplete set of solutions to the problem at hand. The most informative link candidates are then considered to be the pairs $(s, t) \in S \times T$ upon which the different solutions disagree the most. In our case, these are the link candidates that maximize the disagreement function $\delta((s, t))$:

$$\delta((s,t)) = (n - |\{\mathcal{M}_i^t : (s,t) \in \mathcal{M}_i\}|)(n - |\{\mathcal{M}_i^t : (s,t) \notin \mathcal{M}_i\}|), \qquad (2)$$

where $\mathcal{M}_i$ are the mappings generated by the population g^t. The pairs (s, t) that lead to the highest disagreement score are presented to the user, who provides the system with the correct labels. This training set is finally updated and used to compute the next generations of solutions.

5 Evaluation

5.1 Experimental Setup

We evaluated our approach in three experiments. In our experiments, our main goal not only to show that we can discover link specifications of different complexity with high accuracy. In addition, we also aimed to study the effect of the population size and of active learning on the quality of link specifications. For this purpose, we devised three experiments whose characteristics are shown in Table 1.

The goal of the first experiment, called Drugs, was to measure how well we can detect a manually created LIMES specification. For this purpose, we generated `owl:sameAs` link candidates between Drugs in DailyMed and Drugbank by using their `rdfs:label`. The second experiment, Movies, was carried out by using the results of a LATC[2] link specification. Here, we fetched the links generated by a link specification that linked movies in DBpedia to movies in LinkedMDB [12], gathered the `rdfs:label` of the movies as well as the `rdfs:label` of their directors in the source and target knowledge bases and computed a specification that aimed to reproduce the set of links at hand as exactly as possible. Note that this specification is hard to reproduce as the experts who created this link specification applied several transformations to the property values before carrying out the similarity computation that led to the results at hand. Finally, in our third experiment (Publications), we used the ACM-DBLP dataset described in [17]. Our aim here was to compare our approach with other approaches with respect to both runtime and F-measure.

Table 1. Characteristics of the datasets used for the evaluation of EAGLE. S stands for source, T for target.

Label	S	T	$\|S\| \times \|T\|$	Oracle size
Drugs	Dailymed	Drugbank	1.09×10^6	1046
Movies	DBpedia	LinkedMDB	1.12×10^6	1056
Publications	ACM	DBLP	6.01×10^6	2224

All experiments were carried out on one kernel of an AMD Opteron Quad-Core processor (2GHz) with the followings settings: the population size was set to 20 or 100. The maximal number of generations was set to 50. In all active learning experiments, we carried out 10 inquiries per iteration cycle. In addition, we had the population evolve for 10 generations between all inquiries. The mutation and crossover rates were set to 0.6. For the batch learners, we set the number of generations to the size of the training data. Note that this setup is of disadvantage for active learning as the batch learners then have more data and more iterations on the data to learn the best possible specification. We used this setting as complementary for the questions that can be asked by the active learning approach. During our experiments, the Java Virtual Machine was allocated 1GB RAM. All experiments were repeated 5 times.

5.2 Results

The results of our experiments are shown in the Figures below[3]. In all figures, Batch stands for the batch learners while AL stands for the active learners. The

[2] `http://lact-project.eu`

[3] Extensive results are available at the LIMES project website at `http://limes.sf.net`

numbers in brackets are the sizes of the populations used. The results of the Drugs experiments clearly show that our approach can easily detect simple link specifications. In this experiment, 10 questions were sufficient for the batch and active learning versions of EAGLE to generate link specifications with an F-measure equivalent to the baseline of 99.9% F-measure. The standard deviation lied around 0.1% for all experiments with both batch and active learner.

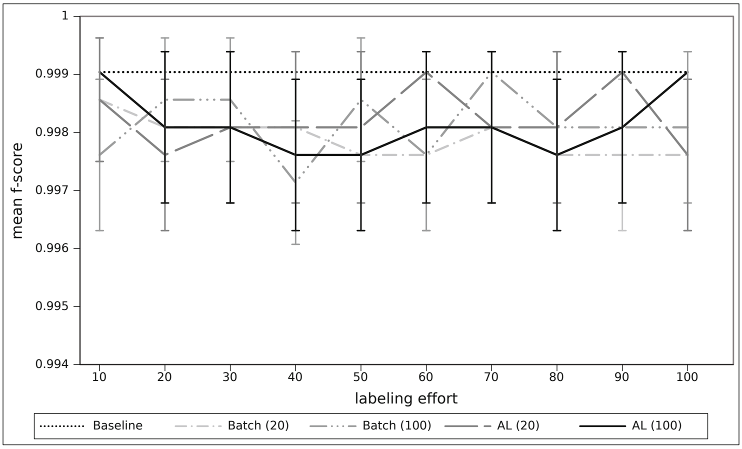

Fig. 8. Results of the Drugs experiment. Mean F-Measure of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 99.9% F-measure.

On the more complex Movies experiments, we required 50 inquiries to discover the best link specification that led to an F-measure of 94.1%. This experiment clearly shows the effect of active learning on genetic programming algorithms. While the batch learners fed with any data size tend to diverge significantly across the different experiments as shown by their standard deviation bars, the active learning approaches do not only perform better, they are also more stable as shown by the smaller standard deviation values. Similar results are shown on the most complex and largest data set at hand, ACM-DBLP.

In addition to being accurate, our approach is very time-efficient. For example, it only required approximately 250ms to run a specification on the first and second data sets when all the data was in memory. On average, it requires less than 700ms on the last data set. It is important to notice that the features of this dataset include real numbers, which considerably worsen the runtime of link specifications.

Our results suggest that the use of small populations affects the outcome of the learning process significantly, especially when the specification to be learned is complex. For example, on the Publications data set, the learners that rely on solely 20 individuals per generation are up to 9.8% worse than the learners which use populations of 100 individuals. Setting the population to 100 seems to

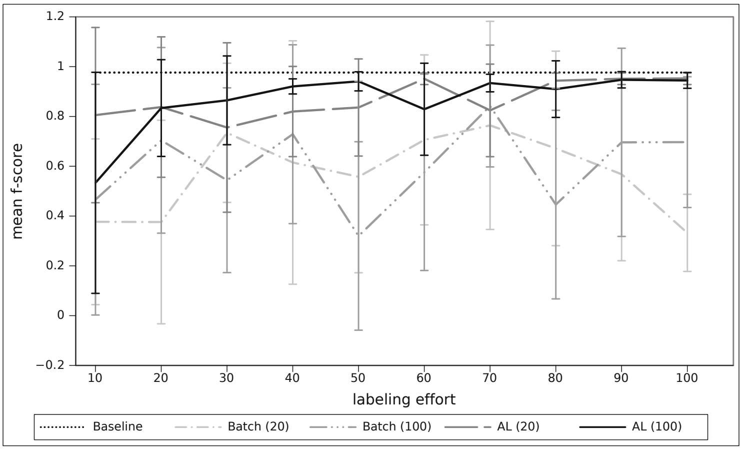

Fig. 9. Results of the Movies experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 97.6% F-measure.

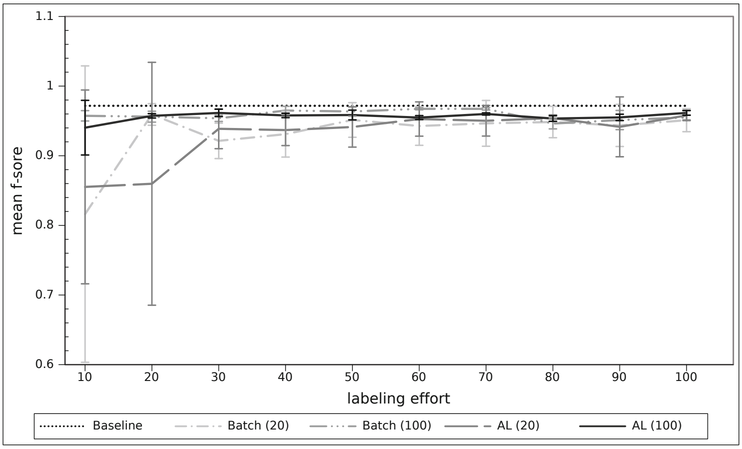

Fig. 10. Results of the Publications experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. The baseline is at 97.2% F-measure.

generate sufficiently good results without requiring a large amount of memory. Yet, when trying to link very complex datasets, an even larger setting would be advisable.

5.3 Comparison with other Approaches

As stated above, we chose the ACM-DBLP data set because it has been used in previous work to compare the accuracy and learning curve of different machine learning approaches for deduplication. As our results show (see Table 2), we reach an accuracy comparable to that of the other approaches. One of the main advantages of our approach is that it is considerably more time-efficient that all other approaches. Especially, while we are approximately 3 to 7 times faster than MARLIN, we are more than 14 times faster than FeBRL on this data set.

Table 2. Comparison of best performances of different machine learning approaches on ACM-DBLP

	EAGLE	FEBRL (SVM)	MARLIN (SVM)	MARLIN (AD-Tree)
F-Measure	97.2%	97.5%	97.6%	96.9%
Runtime	337s	4320s	2196s	1553s

So far, only a few other approaches have been developed for learning link specifications from data. RAVEN [22] is an active learning approach that view the learning of link specifications as a classification task. While it bears the advantage of being deterministic, it is limited to learning certain types of classifiers (boolean or linear). Thus, it is only able to learn a subset of the specifications that can be generated by EAGLE. Another genetic programming-based approach to link discovery is implemented in the SILK framework [15]. This approach is yet a batch learning approach and it consequently suffers of drawbacks of all batch learning approaches as it requires a very large number of human annotations to learn link specifications of a quality comparable to that of EAGLE.

6 Conclusion and Future Work

In this paper we presented EAGLE, an active learning approach for genetic programming that can learn highly accurate link specifications. We compared EAGLE with its batch learning counterpart. We showed that by using active learning, we can tackle complex datasets with more ease and generate solutions that are more stable (i.e., that display a smaller standard deviation over different runs). We also compared EAGLE with other approaches such as FeBRL and MARLIN on the ACM-DBLP dataset. We showed that for we achieve a similar F-measure while requiring a significantly smaller runtime. We also demonstrated that the runtime of our approach makes it suitable for interactive scenarios. In future work, we will study the effect of different parameterizations in more details. Especially, we will utilize different fitness functions and study the correlation of fitness functions with the overall F-score. Furthermore, we will aim at devising automatic configuration approaches for EAGLE.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. 22, 207–216 (1993)
2. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: SIGMOD Conference, pp. 783–794 (2010)
3. Auer, S., Lehmann, J., Ngonga Ngomo, A.-C.: Introduction to Linked Data and Its Lifecycle on the Web. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 1–75. Springer, Heidelberg (2011)
4. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD, pp. 39–48 (2003)
5. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1), 1–41 (2008)
6. Carvalho, M.G., Laender, A.H.F., Gonçalves, M.A., da Silva, A.S.: Replica identification using genetic programming. In: Proceedings of the 2008 ACM Symposium on Applied Computing, SAC 2008, pp. 1801–1806. ACM, New York (2008)
7. Christen, P.: Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In: KDD 2008, pp. 1065–1068 (2008)
8. Cristianini, N., Ricci, E.: Support vector machines. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms. Springer (2008)
9. Cudré-Mauroux, P., Haghani, P., Jost, M., Aberer, K., de Meer, H.: idmesh: graph-based disambiguation of linked data. In: WWW, pp. 591–600 (2009)
10. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19, 1–16 (2007)
11. Glaser, H., Millard, I.C., Sung, W.-K., Lee, S., Kim, P., You, B.-J.: Research on linked data and co-reference resolution. Technical report, University of Southampton (2009)
12. Hassanzadeh, O., Consens, M.: Linked movie data base. In: Bizer, C., Heath, T., Berners-Lee, T., Idehen, K. (eds.) Proceedings of the WWW 2009 Worshop on Linked Data on the Web, LDOW 2009 (2009)
13. Hogan, A., Polleres, A., Umbrich, J., Zimmermann, A.: Some entities are more equal than others: statistical methods to consolidate linked data. In: Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS 2010) (2010)
14. Isele, R., Jentzsch, A., Bizer, C.: Efficient Multidimensional Blocking for Link Discovery without losing Recall. In: WebDB (2011)
15. Isele, R., Bizer, C.: Learning Linkage Rules using Genetic Programming. In: Sixth International Ontology Matching Workshop (2011)
16. Sathiya Keerthi, S., Lin, C.-J.: Asymptotic behaviors of support vector machines with gaussian kernel. Neural Comput. 15, 1667–1689 (2003)
17. Köpcke, H., Thor, A., Rahm, E.: Comparative evaluation of entity resolution approaches with fever. Proc. VLDB Endow. 2(2), 1574–1577 (2009)
18. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). The MIT Press (1992)
19. Liere, R., Tadepalli, P.: Active learning with committees for text categorization. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, pp. 591–596 (1997)
20. Ngonga Ngomo, A.-C.: A Time-Efficient Hybrid Approach to Link Discovery. In: Sixth International Ontology Matching Workshop (2011)

21. Ngonga Ngomo, A.-C., Auer, S.: LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In: Proceedings of IJCAI (2011)
22. Ngonga Ngomo, A.-C., Lehmann, J., Auer, S., Höffner, K.: RAVEN – Active Learning of Link Specifications. In: Proceedings of OM@ISWC (2011)
23. Nikolov, A., Uren, V., Motta, E., de Roeck, A.: Overcoming Schema Heterogeneity between Linked Semantic Repositories to Improve Coreference Resolution. In: Gómez-Pérez, A., Yu, Y., Ding, Y. (eds.) ASWC 2009. LNCS, vol. 5926, pp. 332–346. Springer, Heidelberg (2009)
24. Papadakis, G., Ioannou, E., Niedere, C., Palpanasz, T., Nejdl, W.: Eliminating the redundancy in blocking-based entity resolution methods. In: JCDL (2011)
25. Raimond, Y., Sutton, C., Sandler, M.: Automatic interlinking of music datasets on the semantic web. In: Proceedings of the 1st Workshop about Linked Data on the Web (2008)
26. Scharffe, F., Liu, Y., Zhou, C.: RDF-AI: an architecture for RDF datasets matching, fusion and interlink. In: Proc. IJCAI 2009 Workshop on Identity, Reference, and Knowledge Representation (IR-KR), Pasadena, CA, US (2009)
27. Settles, B.: Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison (2009)
28. Sleeman, J., Finin, T.: Computing foaf co-reference relations with rules and machine learning. In: Proceedings of the Third International Workshop on Social Data on the Web (2010)
29. Song, D., Heflin, J.: Automatically Generating Data Linkages Using a Domain-Independent Candidate Selection Approach. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 649–664. Springer, Heidelberg (2011)
30. Winkler, W.: Overview of record linkage and current research directions. Technical report, Bureau of the Census - Research Report Series (2006)
31. Yuan, Y., Shaw, M.J.: Induction of fuzzy decision trees. Fuzzy Sets Syst. 69, 125–139 (1995)