Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

EAGLE
Learning of Link Specifications using Genetic Programming

Bachelor thesis

Leipzig, August 2012

Klaus Lyko
Bachelor of Science Informatik
Matrikel-Nummer 9512094

Betreuer Prof. Dr. Ing. habil. Klaus-Peter Fähnrich
Betriebliche Informationssysteme, Institut für Informatik
Universität Leipzig

Contents

Abstract

On the way to the Linked Data Web, efficient and semi-automatic approaches for generating links between several data sources are needed. Many common Link Discovery frameworks require a user to specify a link specification, before starting the linking process. While time-efficient approaches for executing those link specification have been developed over the last years, the discovery of accurate link specifications remains a non-trivial problem. In this thesis, we present EAGLE, a machine-learning approach for link specifications. The overall goal behind EAGLE is to limit the labeling effort for the user, while generating highly accurate link specifications. To achieve this goal, we rely on the algorithms implemented in the LIMES framework and enhance it with both batch and active learning mechanisms based on genetic programming techniques. We compare both batch and active learning and evaluate our approach on several real world datasets from different domains. We show that we can discover link specifications with f-measures comparable to other approaches while relying on a smaller number of labeled instances and requiring significantly less execution time.

1 Introduction

The Web is arguably the largest digital source of knowledge on the planet. Yet, for intelligent software agents the usage of the Web is limited. Most of the information is only available in an unstructured fashion as HTML documents. To harness the big amount of data available for more sophisticated question answering and search methods the Web of documents has to be transformed into a Web of Data. Following the Semantic Web Initiative of the World Wide Web Consortium (W3C)[1] a large amount of data is now also available in a machine readable fashion, mostly in the RDF[2] format. As information can be interpreted by machines it is possible to answer questions not by retrieving a list of documents based on keyword-based search techniques, but by interlinking several relevant data sources to form a satisfactory and complete answer as possible. To do that we not only need data sources which provide structured data, information and knowledge but also links between the facts of several knowledge bases.

1 `http://www.w3.org/2001/sw/`
2 `http://www.w3.org/RDF/`

1.1 Motivation

Those methods are part the Linked Data paradigm. The ultimate goal of its practices is the realization of the vision of the Semantic Web[3] through the transition of the document-orientated web into a Web of interlinked Data [3].The Web of Data is steadily growing, yet the amount of statements which are links between knowledge bases remains low. While there are more than 30 billion number of triples in the Web of Linked Data, less than 4% of them are statements interlinking datasets[4]. Determining additional links manually has several drawbacks. The size of some knowledge bases (DBpedia[5] for instance) makes this a long and impractical process. Users may have to have expert knowledge about the domain at hand. Furthermore, links have to be maintained as knowledge bases underlie changes, new instances are added, others are changed, or deleted, and also the underlying ontologies are subject to modifications. So, a machine supported linking process is needed to face these drawbacks. Trying to discover links between datasets in the Web of Data remains a challenge for Computer Science, because two main problems have to be addressed:
1. The Complexity of the link discovery process per se: a priori the complexity of linking two datasets with n respectively m instances is $O(nm)$. For large real-world matching tasks this quadratic complexity becomes unpractical soon.
2. The selection of an appropriate similarity space and link specification. The configuration of Link discovery frameworks is usually done manually, often by simply guessing which properties to compare using which measurement. Yet, given the large amount of properties of instances and the amount of available measures to compute the similarity of two properties

3 The term and its vision was advocated by Tim Berners-Lee [4]
4 `http://www4.wiwiss.fu-berlin.de/lodcloud/`
5 `http://dbpedia.org`

at hand, this manual process also becomes unpractical fast. Furthermore, the link specification requires a domain specific knowledge of the given knowledge bases.

1.2 Approach

To address the first task, the foundation of this work is the universal link discovery framework LIMES [29, 30]. The original LIMES approach uses the triangle inequality to calculate approximations of the distances between instances to address the quadratic complexity issue of link discovery and thereby presupposed that the datasets are in a metric space [29]. The newer LIMES version efficiently deals with numeric values and complex configurations, thereby can handle the diversity of data types in the Web of Data and outperforms the state-of-the-art SILK framework [28]. To address the second great challenge in Link Discovery - finding appropriate and precise Link Specifications - we enhance the instance matching framework LIMES with machine-learning mechanisms utilizing Genetic Programming techniques to semi-automatically discover link specification for LIMES. From the user's perspective, a semi-automatic approach to generate link specifications has to limit the burden for users by

1. reducing the time to detect a link specification (time efficiency),
2. generating precise link specifications, that only generate a small amount of false positive links while discovering as many links as possible (precision and recall) and
3. providing the user with a readable and modifiable specification that utilizes the whole space of available measures and operates to generate such link specifications of arbitrary complexity.

In this thesis we present EAGLE, a supervised machine learning approach for link specifications, that abides the three criteria above. One of the main drawbacks of supervised machine learning approaches is that they usually require a large amount of training data in oder to generate results of high accuracy. Yet the generation of labeled training data can be a difficult and expensive process with respect to both time and knowledge required to do so. EAGLE surmounts this problem by implementing an active learning approach [39]. Therewith, we are able to minimize the amount of training data necessary to compute precise link specifications, by allowing the interactive annotation of highly informative training data.
By combining this active learning mechanisms with genetic programming we are able to use the full spectrum of available similarity measures (i.e., Levenshtein, Jaccard for strings) for comparing property values and support manifold means for combining these measures. Our algorithm is able to compute link specifications of arbitrary complexity.
We evaluate our approach on six datasets (both synthetic and real-world datasets) and show that we can calculate precise link specifications while minimizing the work for users by applying an active learning mechanism and still ensure a comparable small time complexity.
Note that some of the results presented herein were already published and presented at the 9th Extended Semantic Web Conference 2012 [6], this thesis will contain further insights and will evaluate the approach on additional datasets [32].

6 http://2012.eswc-conferences.org/

1.3 Structure

The rest of this thesis is organized as follows: First we discuss state of the art approaches in related research areas, such as Machine Learning, Genetic Programming and Link Discovery, in section 2. Thereafter, section 3 introduces the formal groundwork used, while section 4 gives insight into the implementation of EAGLE. The evaluation of our approach is presented and discussed in section 5. Ultimately we conclude and give an outline, and present future research in section 6.

2 State of the art

2.1 Link Discovery

Linking is the process of establishing links between resources which are somehow related. In the domain of Linked Data it means the process of generating typed links between entities (e.g. classes, properties or instances) of knowledge bases. The most common goal is to mark instances (e.g. via owl:sameAs links) of the two knowledge bases that represent the same real world entity. 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) [15] focuses on computing links between universities and conferences while GNAT [37] discovers links between music data sets. Further simple or domain-specific approaches can be found in [10, 35, 17, 40, 36]. In addition, domain-independent approaches have been developed, that aim to facilitate link discovery all across the Web. For example, RDF-AI [38] implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and postprocessing of data sets. SILK [18] 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 [28], 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 [42, 14, 6, 20].

2.2 Machine Learning and Genetic Programming

Because of the variety and complexity of learning systems and the broad concept of learning itself any attempt to give a precise and unambiguous definition of learning is doomed to failure. Instead we will give a practical definition for Machine Learning:
Machine Learning is a sub field of Artificial Intelligence concerned with the study of algorithms that automatically improve their performance with experience. [24, 27]
Machine learning algorithms have been successfully applied to a variety of problems. Most notably in the area of Data Mining and natural language processing.
Link Discovery can be viewed as a classification problem with only two predefined classes: matches and non-matches. Given a set of pairs of entities from two knowledge bases the goal is to decide whether or not they represent the same real-world entity and therefore should be classified as a match. Thus this problem can be tackled as a binary classification problem using well established machine learning approaches for classification problems, such as support vector machines, artificial neural networks [19, 9, 31].
The topic of machine learning can be classified by many dimensions. Instead of giving an complete overview we will only address those topics related to our approach. A review and historical overview of what Machine Learning includes, can be found in [2, 26, 27]. Most Machine Learning ap-

proaches for the efficient computation of link specification use supervised techniques[12]. That means, they require labeled training data. Based on how this training data is generated we further discern active and batch learning approaches. For batch learning approaches the training data is generated only once previously to the start of the actual learning process. The selection of instances for this training data is the result of an arbitrary process, and not part of the learning algorithm. Whereas, active learning is an interactive process divided into multiple steps. Each step the learning algorithm itself decides upon which set of instances he discerns as most informative. These data is then presented to an oracle (e.g. a human being) for labeling, thus the learner increases its set of training data for the next learning step.
Most recently also unsupervised learning approaches for Link Specifications have been developed [34, 33]. These approaches have the benefit that they do not require any user feedback, as they only use inherent features under certain assumptions of the problem to measure the quality of link specifications. Although initial results appear promising, most recent studies suggest that applying these techniques on real world datasets is more difficult. Results show that there is yet no consistent parametrization of these algorithms and future research is required [33].
Many branches of Machine Learning, Artificial Intelligence or in general problem solving are based on, and motivated by the observation of biological processes and natural phenomena. Evolutionary Computing (EC) is a research area within computer science, that draws its inspiration from the process of biological evolution. Its origins dates back before the breakthrough of computers. The basic principles where developed and introduced in the 1960s separately in the USA by Fogel, Owens, and Wash under the name *evolutionary programming*, while J.H Holland called the approach *genetic algorithms* and Rechenberg and Schwefel developed *evolution strategies* in Germany. These approaches where unified under the concept of EC

in the 1990s[13]. Koza introduced with his work about **genetic programming** the fourth sub area of EC[22].

Despite its long history all algorithms in the field of EC follow the same underlying idea. For the specific problem a number of example solutions are randomly created, forming a population of individuals within the same environment of limited resources. As in nature a competition over surviving between the individuals is applied, so as to only the fittest individuals survive as a seed for the next generation. So any EC algorithm has its specific fitness measure denoting the performance of an individual with respect to the specific problem at hand. Despite survival of the fittest a set of new individuals is created each generation by applying reproduction and mutation to the population. Reproduction means recombining the genetic code of two or more so-called parent solutions to form an offspring of new candidates. Those candidates also compete based on their fitness value with the rest of the population to form the next generation.

The difference between evolutionary algorithms (EA) and genetic programming (GP) is that the chromosomes(solutions) of individuals of GP can be of arbitrary length, as individuals are represented as program trees, whereas EA represent the solutions as fixed-length Strings over a specific alphabet (most often $0, 1$).

2.3 Learning Link Specifications

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 [18] now implements a batch

learning approach to discovery link specifications based on genetic programming which is similar to the approach presented in [7]. 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 [31] 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 [41]. 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 [8] and MARLIN [5] rely on models such as Support Vector Machines [19, 9], decision trees [43] and rule mining [1] to detect classifiers for record linkage. Our approach, EAGLE, goes beyond previous work in three main ways. First, we implemented 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 [31]. Thus, it does not need to be seeded to converge efficiently like previous approaches [18].
To the best of our knowledge the deduplication method presented in [12] which is a further development of previously batch learnings approaches[7, 11] is most closely related to our active learning mechanism. It also combines active learning with means of genetic programming and suggests, that by using these mechanisms less human labeling effort is required, while ensuring good results nonetheless. In contrast to our approach Fre-

itas et al. use a reinforcement learning technique, where only if a majority voting of a committee of the evolved individuals is not sufficient to predict the classification of a pair of instances, they are presented to an oracle for manual labeling [12]. In contrast we not only support a wider range of measures to compute the similarity of certain data fields (properties), but we also can compute combinations of these atomic measures of arbitrary complexity using both boolean and linear operators to combine atomic measures.

3 Genetic Programming for Link Specification

In the following section we introduce the formalization and notation used for EAGLE. After defining the Link Discovery Problem, we introduce the grammar used by LIMES for link specifications and show how it can be represented by trees. Thus, enabling us to model the discovery of links as a GP problem. We will specify how the various genetic operators effect these link specification trees and finally, we present the active learning model that underlies our work.

3.1 Link Discovery

The link discovery problem is similar to the record linkage problem and is also known as record linkage, object matching, duplicate detection[21]. In general the goal is to discover pairs $(s, t) \in T \times S$ related via a relation R given two sets of entities S, T. Such a relation could be the `owl:sameAs`[1] predicate for instance.

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 $R(s, t)$.*

1 `http://www.w3.org/TR/owl-ref/#sameAs-def`

In most cases the sets S and T are subsets of the instances contained in two knowledge bases K_T and K_S, which may or may not be disjoint. We try to discover links between instances $s \in S$ and $t \in T$ using a complex similarity measure σ which is based on a comparison of their properties via certain (atomic) similarity measures. The two entities s and t are then considered to by linked via R if $\sigma(s,t) \geq \theta$.[30]. Typically both a specification of the two sets S and T and of this similarity condition is defined by 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 in general a logical predicate. Each $s \in S$ must comply with each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, and each $r \in R$ with each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_m^S$ respectively. A typical restriction states the `rdf:type` of the elements of the set they describe, i. e. that the elements must be instances of a specific class: $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass`. Furthermore they can describe features the elements must have, e.g. they have a certain property: $\mathcal{R}(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each similarity σ_i is used to compare pairs of property values of instances s and t. We will not consider all possible pairs of properties of s and t for the atomic similarity measures σ_i, but only those pairs of a property mapping done previously. As presented in [28] we can compute the class and property mappings of two knowledge bases efficiently without burden the user with additional work or expert knowledge about the linking task.

3.2 Designing Link Specifications as Trees

As stated before our definition of a link specification is based on a specification of so called *atomic similarity measures*. A similarity measure is a function m that specifies the similarity between two entities $s \in S$ and $t \in T$: $m : S \times T \rightarrow [0, 1]$, whereas values closer to 0 indicate lesser similarities. We call such a measure atomic if it relies on just one similarity measure σ to compute the similarity of a pair $(s, t) \in S \times T$. For examples a trigrams measure for strings, or a euclidean distance for numeric values. A similarity measure m is either a atomic similarity measure or the combination of two similarity measures m_1 and m_2 via a metric operator such as OR, AND or a linear combination ADD. We call a link specification atomic if it compares the pairs $(s, t) \in S \times T$ by using a atomic similarity measure and threshold θ, thus returning only those pairs that satisfy the condition $\sigma(s, t) \geq \theta$. A link specification is either an atomic links specification or a combination of two link specifications via operators, such as AND (the intersections of the results of two specifications), OR (the union of the results), XOR (symmetric difference).

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 3.1, 3.2, 3.3 and 3.4.

3.3 Genetic Structure of Link Specifications

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 [23, 22] is to generate

Figure 3.1: Atomic measure

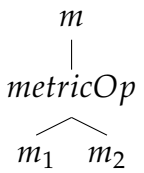

Figure 3.2: Complex measure

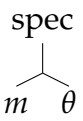

Figure 3.3: Atomic specification

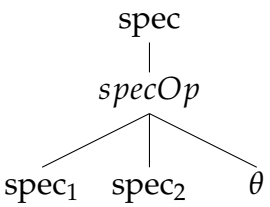

Figure 3.4: Complex specification

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. We propose the combination of GP and active learning [39]. 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.

Algorithm 1 depicts the basic work flow of our approach. Based on the specification of both knowledge bases *KS* and *KT* we will get both sets on entities, with all defined properties and according to the preprocessing op-

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

erations. Note, that we require the specification of a property mapping (KS, KT), as this will significantly reduce the computation complexity, ad we will only compare property pairs defined by it. Thus, preventing that the evolutionary process will produce programs comparing properties, who will most likely not benefit the linking process, but requiring computation time nonetheless. We can semi-automatically compute property mappings using efficient stable marriage algorithms [28]. By default we convert every string property to lowercase values.

3.4 Evolution of a 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. The difference between functions and terminals is, that functions are inner nodes of the program trees, nodes with a specified number of children.

Whereas, terminals represent leaves of the program trees. That are functions without parameters (child nodes in the program tree). 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 property pairs. 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 as defined by a property mapping defined beforehand.

Algorithm 2 Evolves a population

```
if population is empty then
  create size random individuals
end if
Compute fitness of population
Build new generation by applying genetic operators to population
return population
```

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 3.5). 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 3.5). 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 3.6 exemplifies the functionality of the crossover operator.

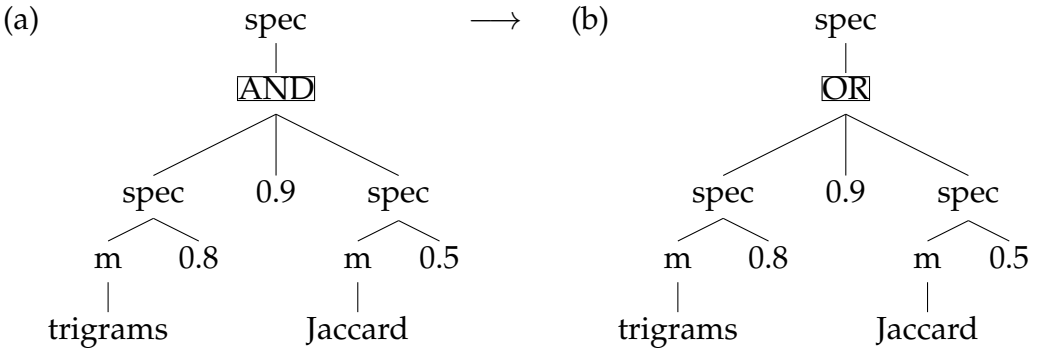

Figure 3.5: Mutation example. Mutation changes boolean operator.

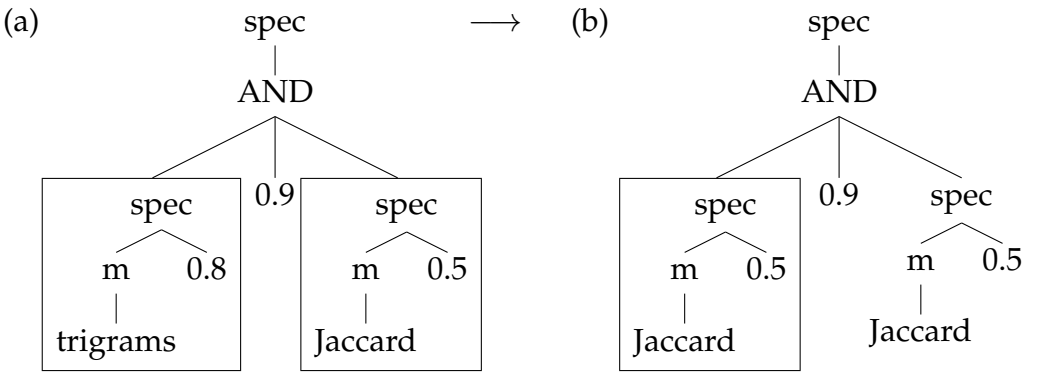

Figure 3.6: Crossover example. Consider we have two individuals with a program tree like in (a). A crossover operation can replace subtrees to produce an offspring like (b).

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.

3.5 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

$$\frac{2PR}{P+R} \text{ where } P = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{M}|} \text{ and } R = \frac{|\mathcal{M} \cap \mathcal{O}|}{|\mathcal{O}|}. \tag{3.1}$$

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

3.6 Computation of most informative link candidates

The main idea behind applying a active learning method is the reduced amount of labeling effort required. As we ask the user to label only a specific number of highly informative training data, rather then a randomly selected. Thus we hope that our active learning solution will converge faster (and thereby presenting the user lesser data to label) to find the best possible solution for the Link Discovery process. 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 [25]. 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 $\delta((s, t))$:

$$\delta((s,t)) = (n - |\{\mathcal{M}_i^t : (s,t) \in \mathcal{M}_i\}|)(n - |\{\mathcal{M}_i^t : (s,t) \notin \mathcal{M}_i\}|), \quad (3.2)$$

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.

3.7 Genetic Operators

Our randomly created population of link specifications is transformed into a population of fitter individuals by applying a number of genetic operators to individuals selected according to their fitness value. There are three basic genetic operators:
Reproduction: An individual is copied to the next generation without any changes.
Mutation: An individual is changed by random modification.
Crossover: The program trees of two individuals are recombined into a new individual.

3.7.1 Reproduction

To preserve good individuals throughout the evolutionary process, the genetic operator reproduction is assigned to individuals based on there fitness value. Here we use a tournament setting between two randomly selected individuals of the population g^t of generation t. The individual with the better (lower) fitness value will be copied unchanged to the next generation g^{t+1}. Individuals are selected for the tournament randomly according to a reproduction probability $p_{reproduction}$. Furthermore, note that we always preserve the single most fittest individual for the next generation.

3.7.2 Mutation

Almost all nodes of the tree representing the link specification can be subject to random changes. Those changes are called mutation. We have to ensure that after mutating a specific node in the program tree, the resulting tree with the mutated node is still a valid program tree. Therefore, each specific node can only mutate into a limited set of other nodes, and each terminal within a specified range of values respectively. The probability of an individual to be selected for mutation is specified with the $p_{mutation}$ parameter. For instance, a node representing a similarity measure between string properties can only mutate into another similarity measure used for string values. Figure 3.5 gives an example.

3.7.3 Crossover

As described above, crossover operates on two parent individuals and builds a new offspring by swapping subtrees of the parent genotypes.

Individuals of the new population g^{t+1} of generation $t + 1$ are selected randomly according to the probability $p_crossover$ to be subject to crossover.

4 Implementation

4.1 LIMES

We use the lossless and time-efficient LIMES framework for link discovery. LIMES (Link Discovery for metric spaces) is utilizing the triangle inequality in metric spaces to filter out large numbers of instance pairs, which cannot suffice a given link specification [30]. Thereby minimizing the time complexity of a link discovery task.

4.1.1 LIMES Link Specification language

Furthermore, LIMES provides an easy-to-use and highly flexible language for link specifications. Besides several so called atomic similarity measures m to compare the values of two property fields, such as the Levenshtein distance, Cosine similarity or Trigrams measure for the comparison of strings, and the Euclidean distance of numeric values, it is also possible to combine multiple atomic measures with so-called operators. E.g. set-constraint-operators to merge two given mappings, such as AND, MINUS, XOR and OR. It is also possible to linear combine atomic measures with the metric operator ADD. Furthermore all results are filtered according to a similarity threshold θ.
Given two sets S (source) and T (target) of entities, let denote $A \subseteq S \times T$

and $B \subseteq S \times T$ two sets of pairs $(s,t) \in S \times T$ called mappings. The set-constraint operators works on the mappings produced by atomic measures or operators, and filters them according to the threshold θ: $AND(A,B)$ returns all pairs of instances (s,t) in the intersection $A \cap B$ of the two source mappings. While $OR(A,B)$ is the union $A \cup B$, $MINUS(A,B)$ the difference $A \backslash B$ and $XOR(A,B)$ denotes the difference $A \cup B \backslash A \cap B$. Note all pairs in the resulting merged mappings have to apply to $\sigma(s,t) \geq \theta$. The LIMES link specification language also supports other operators, such as MAX, MIN and MULT. But those are not generated by EAGLE.

4.2 Realization using GP

To define a good link specification a central task for us, is to specify such a complex link specification, expressing how to combine properties of the two knowledge bases. We will also call this a metric expression. Second it is also necessary to specify a global threshold θ beyond which instance pairs are considered to represent the same object, and therefore will be linked. To enhance LIMES with features of genetic programming we use the Java Genetic Algorithms Package JGAP.[1] JGAP can be used both for genetic algorithms and genetic programming.

4.2.1 Preliminaries

We assume all other necessary steps for link specification, such as defining the two knowledge bases to be done in advance. Especially any preprocessing step is done outside of our learning process. And furthermore,

1 http://jgap.sourceforge.net/

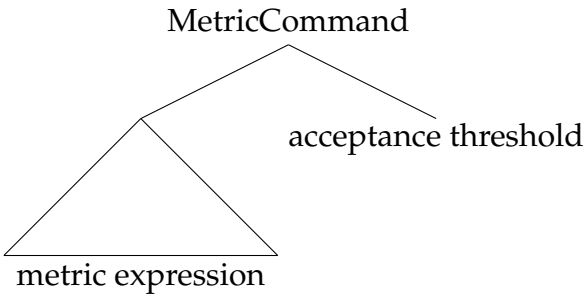

Figure 4.1: Basic structure of chromosomes

we assume that there already exists a property mapping between the two knowledge bases. Specifying which source property can be linked to which target property, and of which type (string vs. number) both properties are. This minimizes the computation complexity, as only *reasonable* combinations of source and target properties can be part of any individual representing link specification, avoiding the comparison of properties which most likely won't benefit the finding of links.

4.2.2 Genotype

The genotype of each of our individuals consists of only one chromosome, representing all the information on how to link instances $s \in S$ and $t \in T$. Figure 4.1 illustrates the basic setup of the GP programs[2] we evolve. The root of each program is a function[3] called *MetricCommand*, which takes two parameters, whereupon the first is a function defining the metric expression - a complex link specification, and the second one a terminal[4] representing the global acceptance threshold.

2 We also call them individuals or example solutions.
3 All nodes in a GP tree are called *commands* or *command nodes*, which either represent a function or a terminal.
4 A command without children and as such a leaf in the program tree.

The left subtree of a *MetricCommand* builds the metric expression. The inner nodes of this tree represent functions for similarity measures and compositions of those, called operators. The leaves of this subtree are terminals representing the properties of the two knowledge bases and thresholds or coefficients for the specific similarity measure[5]. Instead of representing the properties of a knowledge base individually by there names and evolving them separately, we take the property mapping done beforehand into consideration. We model properties as property pairs. We assign each pair of properties in the property mapping a specific number and represent it as a number terminal in the program tree. Thereby, we can support mutation of the properties by randomly selecting a new value within the bounds of the allowed property pairs. Furthermore, we distinguish between number and string properties, by a similar node only selecting pairs of number properties out of the property mapping. This is done, because certain atomic similarity measure only operate on numbers, such as the euclidean metric.
For example, consider the following matching task: We want to match two knowledge bases dubbed *source* and *target* with a Jaccard measure taking the properties called *name* as input, and consider all pairs of instances with a similarity above 0.8 as a match. The corresponding configuration file for LIMES would consist of the following statements:

```
<METRIC>jaccard(source.name, target.name)</METRIC>
<ACCEPTANCE>
   <THRESHOLD>0.8</THRESHOLD>
</ACCEPTANCE>
```

5 The thresholds of similarity measures are only considered if the command is part of an operator, e.g. child of of a boolean AND operator. Otherwise, the global threshold - the right child of the *MetricCommand* - is used for filtering pairs of instances.

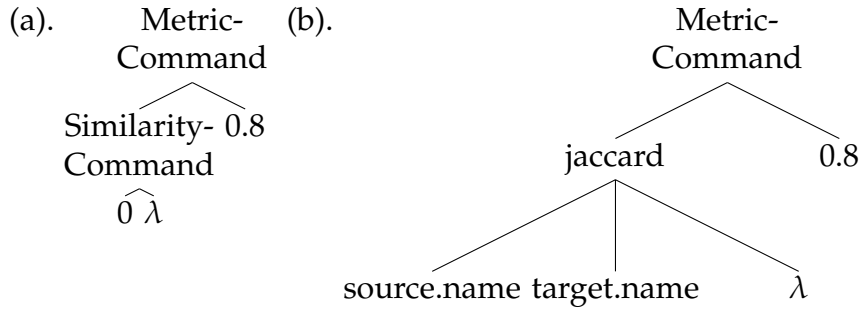

Figure 4.2: A GP program tree representing the expression *jaccard(source.name, target.name)* using 0.8 acceptance threshold. Suppose 0 is the index of the list of property matches, representing the pair *source.name* and *target.name*; λ is a random double value. This parameter is part of the SimilarityCommand because it is possible that the same node will later on be the child of an boolean operator, which combines two mappings, and therefore needs local acceptance thresholds for its children, in case, they are atomic similarity measures. The value of λ will only be part of an execution of the SimilarityCommand in the latter case. Figure a illustrates how both the similarity measure and properties are represented, while Figure b shows the result if all functions are executed.

A genetic program representing such an execution plan is given in Figure 4.2. For each function and terminal we created own implementations of the JGAP *GPCommand* interface. Thus, we can support custom mutations on the several parts of a link specification. Table 4.1 gives a overview of the basic commands we defined to model a link specification.

4.2.3 Evolution

The evolutionary process is modeled using built in JGAP functionalities. Both the creation of a initial population of individuals, as the steps to evolve to further generations (selecting individuals for the genetic op-

Nr.	GP command	children	types	values
1	*MetricCommand*	2	$(4 \dots 8, 9)$	-
2	*StringPropertyPair*	0	-	$1 \dots n$
3	*NumberPropertyPair*	0	-	$1 \dots m$
4	*StringMeasure*	2	$(2, 8)$	trigrams, jaccard, cosine, levensthein, overlap
5	*NumberMeasure*	2	$(3, 8)$	euclidean
6	*AddMetric*	4	$(\{4, 5\}, \{4, 5\}, 8, 8)$	ADD
7	*BooleanCommand*	3	$(4 \dots 7, 4 \dots 7, 8)$	AND, OR, XOR, MINUS
8	*Terminal*	0	-	$[0 \dots 1)$
9	*Terminal*	0	-	$[0 \dots 1)$

Table 4.1: Custom GP commands with number and types of their children nodes. Values mean of all possible mutations. Where m and n are the boundaries defined by a property mapping. Types is a list with the possible types the specific child. E.g. A *StringMeasure* command has two child nodes. The first is a *StringPropertyPair* and the second a threshold modeled via the terminal 8.

erators reproduction, mutation and crossover) use predefined functionalities of the JGAP library. But as a central step we defined our own FitnessFunction, both for the Batch Learning and Active Learning process.

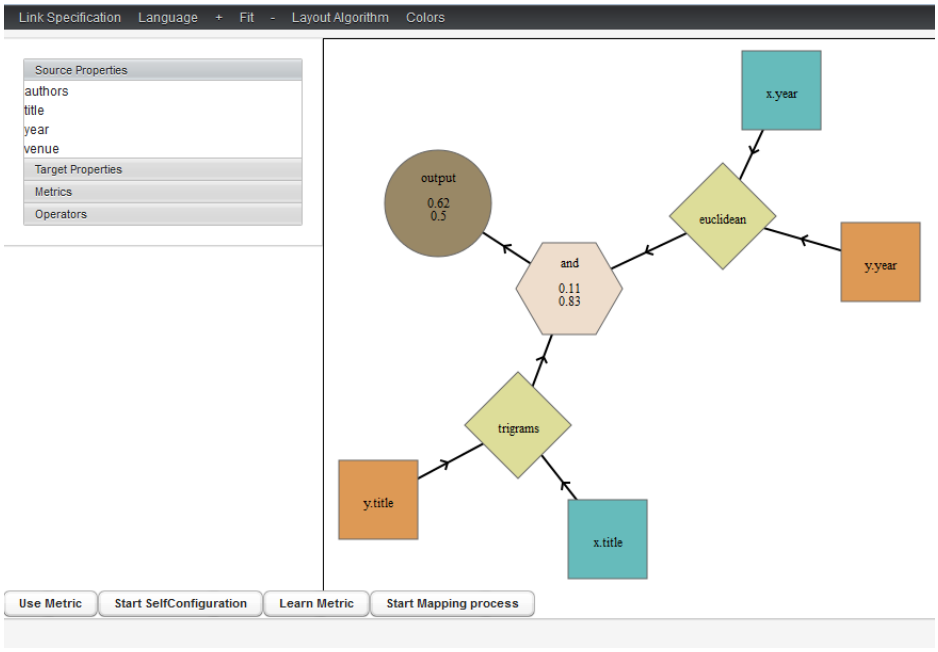

Figure 4.3: Screenshot of the SAIM graphical user interface for the LIMES framework.

4.3 SAIM - Graphical user interface

A Web application for the LIMES framework and all learning algorithms presented here is currently under development, we already published a pre-alpha demo [6]. Figure 4.3 shows the main user interface of SAIM .

6 http://aksw.org/Projects/SAIM

5 Evaluation

We evaluate EAGLE on six datasets of different complexity and from different domains. In this section we first introduce the experimental setup and the datasets we used, then we present and discuss the results of our evaluation.

5.1 Experimental Setup

We evaluated our approach in six experiments. In our experiments, our main goal was 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.

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 I	ACM	DBLP	6.01×10^6	2224
Publications II	DBLP	Google Scholar	168.11×10^6	5347
Products I	Abt	Buy	1.18×10^6	1097
Products II	Amazon	GoogleProducts	4.39×10^6	1300

Table 5.1: Characteristics of the datasets used for evaluation. S stands for source, T for target.

Label	String attributes	Number attributes
Drugs	name	-
Movies	title, director	-
Publications I	title, authors, venue	year
Publications II	title, authors, venue	year
Products I	title, description, manufacturer	price
Products II	title, description, manufacturer	price

Table 5.2: Characteristics of the datasets used for evaluation. Listing of all attribute pairs. String attributes are preprocessed by transforming them to lowercase strings. Number attributes as numbers

5.1.1 Datasets

For this purpose, we devised experiments whose characteristics with respect to their size and the size of the reference mappings are shown in Table 5.1 and 5.2. 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[1] link specification. Here, we fetched the links generated by a link specification that linked movies in DBpedia to movies in LinkedMDB [16], 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 remaining experiments (Publications I, Publication II, Prod-

1 `http://lact-project.ec`

ucts I and Products II), we used the datasets described in [20]. For all datasets we manually generated a property mapping. As showed in [28] we can compute a property mapping of two knowledge bases efficiently. Thereby avoiding additional work or expert knowledge about the linking task for the user. For all string properties we automatically assigned them to the lowercase preprocessing function. Additional in the Movies experiment we remove the braces in both the name and director properties on both the DBPedia and LinkedMDB datasets Finally in case of the Publications I experiment another aim was to compare our approach with other approaches with respect to both runtime and F-measure.

5.1.2 Parameters

All experiments were carried out on one kernel of an AMD Opteron Quad-Core processor (2GHz) with the followings settings: The mutation $p_{mutation}$ and crossover $p_{crossover}$ rates were set to 0.6. While $p_{reproduction}$ was set 0.5, with preservation of the fittest individual each generation. As initial experiments suggested that these values were sufficient to produce both stable results and populations able to adapt to the given problem. Our initial experiments also suggested that larger populations produce better results with a higher likelihood, we want to verify this be setting the population size 20 or 100. In all active learning experiments, we carried out 10 inquiries per iteration cycle. In addition, we had the population evolve for 50 generations between all inquiries. For the batch learners, we set the number of generations to the according number of generations the active learner performed to reach the same number of inquiries as the batch learner at that stage. To generate the training data for the batch learner we randomly select an according number of matches from the reference mapping. Note that this setup is of a slightly disadvantage for active

learning as the batch learners then have most likely more positive examples as training data and more iterations on the same amount of 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.

5.1.3 Quality Measures

To evaluate the solutions produced be the link specification learner we compute precision, recall and the F1 metric of the resulting mappings in respect to the reference mappings. The *precision* P of a mapping m as result of a link specification is the proportion of correctly identified links (true positives) of all identified links (true positives plus false positives). The *recall* R of a mapping produced by a link specification is the proportion of correctly identified links of all the links between the knowledge bases as defined in the reference mapping. The standard f-Measure F is the harmonic mean of P and R: $F = \frac{2 \times P \times R}{P+R}$.

5.2 Experimental Results

In this section we present and discuss the results of our experiments. Three main items will be of interest:

1. How many inquiries are sufficient to produce accurate and stable mappings.
2. What influence has the population size to the experiments.

3. Compare the active and batch learning approach with respect to the quality of the results and the size of training data needed to produce them.

5.2.1 Baseline Experiments

To get a baseline for our experiments we run batch learning experiments using both complete sets of instances S,T and using the complete reference mapping O for each dataset as labeled training data. We run the experiments with a population of 20 individuals over 100 generations using the same parameters as for the other experiments. The best solution is then selected to get the baseline for each dataset. As we use the full background knowledge we can fairly say, that those F-measures are the best the learning algorithms can theoretically achieve. Table 5.3 presents the results of our baseline experiments for all datasets, the generation we first computed the maximum and a link specification producing these results. Example link specifications for all experiments computing mappings of these quality can be found in the appendix.

Dataset	F-measure	Precision	Recall	Gen	Figure
Drugs	99.9 %	100 %	99.8 %	100	A.1
Movies	97.5 %	97.9 %	97.2%	9	A.2
Publications I	97.0 %	96.2 %	97.9 %	15	A.3
Publications II	77.2 %	77.5 %	77.0 %	4	A.4
Products I	35.1 %	39.3 %	31.6 %	55	A.5
Products II	37.6 %	38.4 %	36.9 %	2	A.6

Table 5.3: Baseline experiments. Listing best f-measure, precision and recall reached running a population of 20 individuals for 100 generations using the full labeled reference data. Column *Gen* depicts the minimal generation the best result was first reached. The last column *Figure* names the figure in the Appendix showing one possible links specification visualized using the SAIM prototype.

As table 5.3 suggests we are able to compute high quality link specification for the comparably more simpler datasets Drugs, Movies Publications I. The reference mapping of these datasets is an almost perfect 1:1 mapping between instance $s \in S$ and $t \in T$. Therefore, the negative influence of maximizing the precision to the recall is bounded. The rest of datasets (Publication II and both product datasets) are of more complex nature. Here it is more likely that one $s \in S$ is mapped to a number of instances $t \in T$. Hence, the interaction of precision and recall tends to be greater.

5.2.2 Accuracy batch vs. active learning

All comparable results of the batch and active learners are presented in the following figures. In all figures, Batch stands for the batch learners while

AL stands for the active learners. The numbers in brackets are the sizes of the populations used.
Will also included the results in more details in the A.2 section of the appendix.
The results of the Drugs experiment - figure 5.1 - show, that EAGLE can easily detect simple link specifications. In this experiment 10 labeled instance were already sufficient to generate link specifications equivalent to the baseline of 99.9% F-measure. This holds for all runs. The standard derivation lied around 0.1% for both active and batch learners.

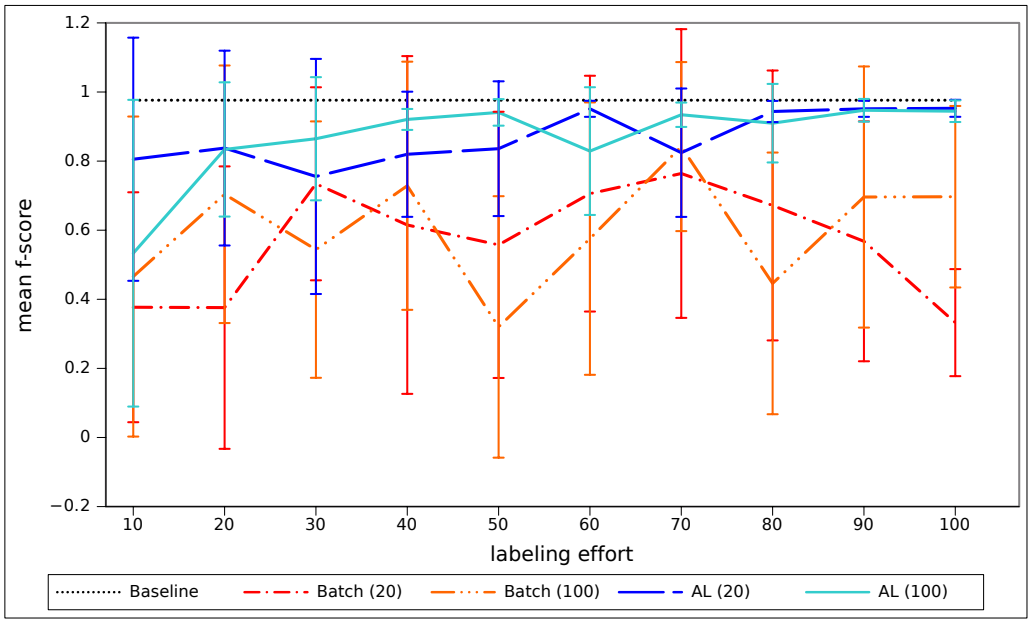

Figure 5.1: Results of the Movies experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 97.5% F-measure.

For the more complex Movies experiment 50 inquiries were required to detect the best link specification with 94.1% F-measure. Here the advantage of the active learners is obvious. The batch learner tend to diverge

significantly as shown by their standard derivation bars, while the active learner not only perform better but they are also more stable.
The easiest real-world dataset at hand Publications I, which links bibliographic items of ACM to those of DBLP, shows similar results. Again, the batch learner tend to more unstable results. The results for the real-

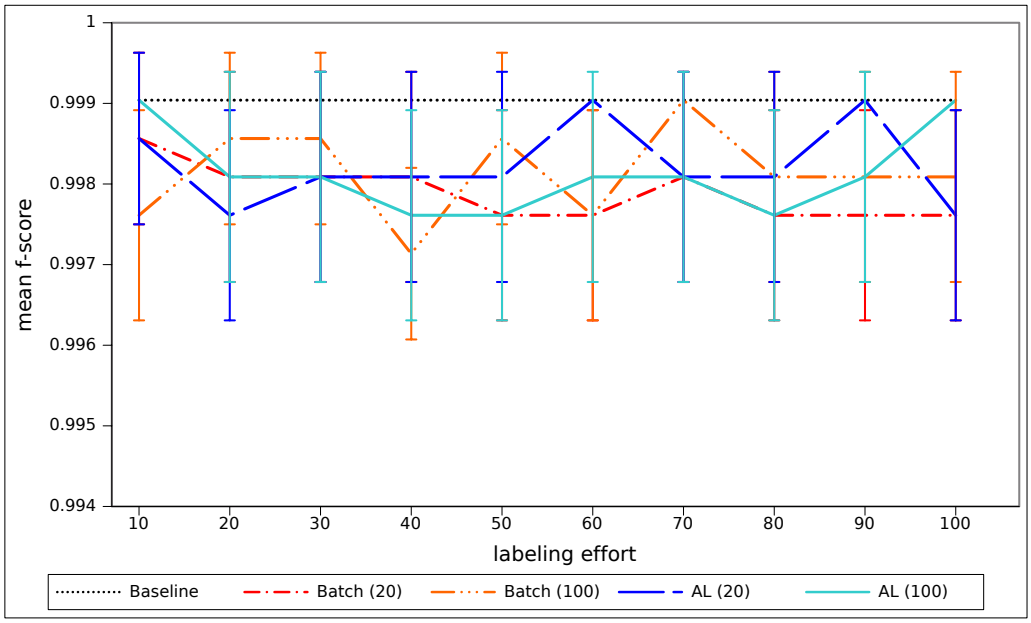

Figure 5.2: Results of the Drugs experiment. Mean F-Measure of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 99.9% F-measure.

world dataset Publications II and Product I and II also suggest the main advantage of the Active Learners. They tend to converge faster and with higher probability to the best F-Measures. As shown in figure 5.5 already 30 questions to the oracle were sufficient for the active learners in the Product I experiment to achieve F-Measures around 70% of the baseline. Even better results shows the Products II experiment. After 50 inquiries to the oracle both active learner achieve F-measure between 30% and 36%, where the baseline is 37.6 %. In the Products II experiments 50 inquiries

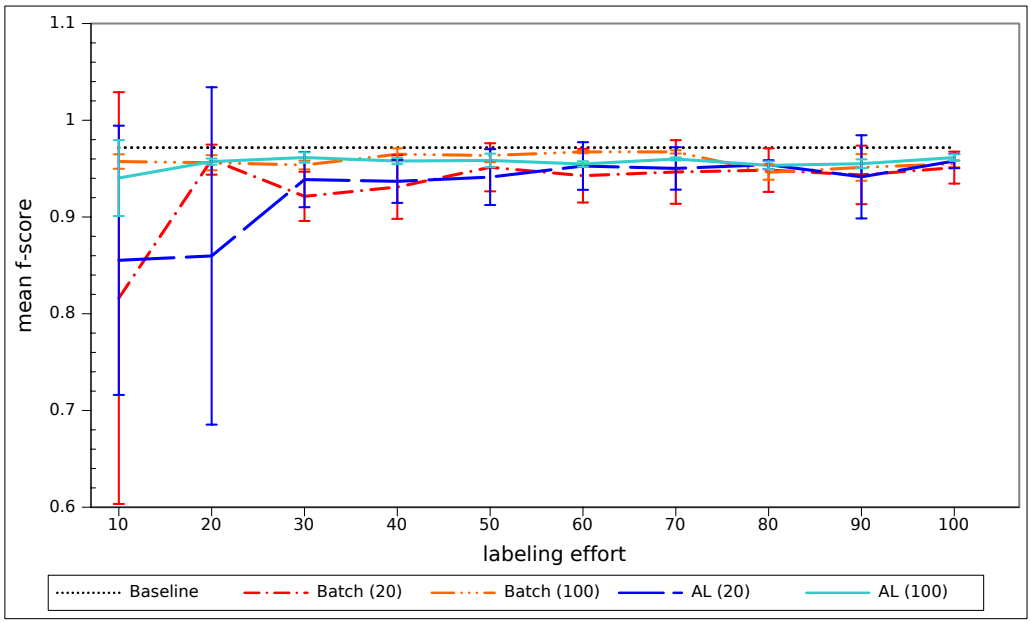

Figure 5.3: Results of the Publications I experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 97.0% F-measure.

were sufficient for the active learners to compute link specifications results better then 30% F1-Measure which is only 6% beneath the baseline. The observation, that the batch learner diverge more, holds not for all these experiments. As the error bars in figures 5.4 and 5.5 suggest. Only the Products II experiment shown in figure 5.6 supports this hypothesis. It is obvious that our approach is easily able to compute easy link specification as in the synthetic Drugs experiment or the real-world experiment Publications I. But as was to be expected we have to sacrifice quality of the link specification to more complex link problems, respectively the labeling burden for users grows with the complexity at hand. Using the active learning approach we are able to ease these effects.
Using different populations sizes does not appear to have an consistent impact on the results. Note that using greater population sizes results in

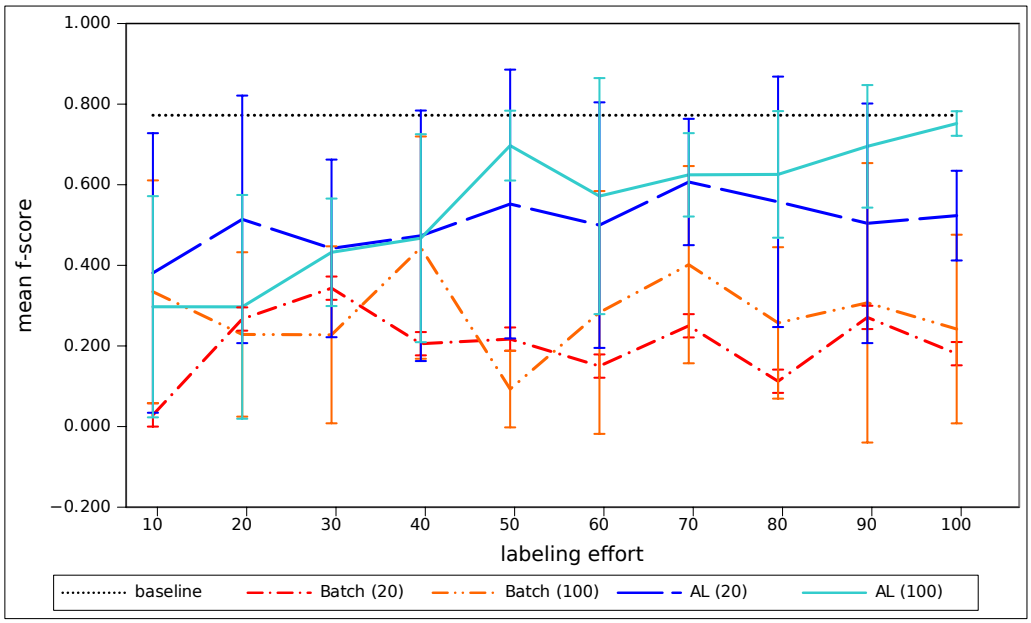

Figure 5.4: Results of the Publications II experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 77.2% F-measure.

greater run-times, as every generation the fitness value of more individuals has to be computed. Using the moderate population size of 20 individuals seams to sufficient in most cases as it offers a heterogeneous enough gene pool for an adaptive evolutionary process.

5.2.3 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 5.4), we reach an accuracy comparable to that of the other approaches. One of the main advantages of our approach is that it is considerably

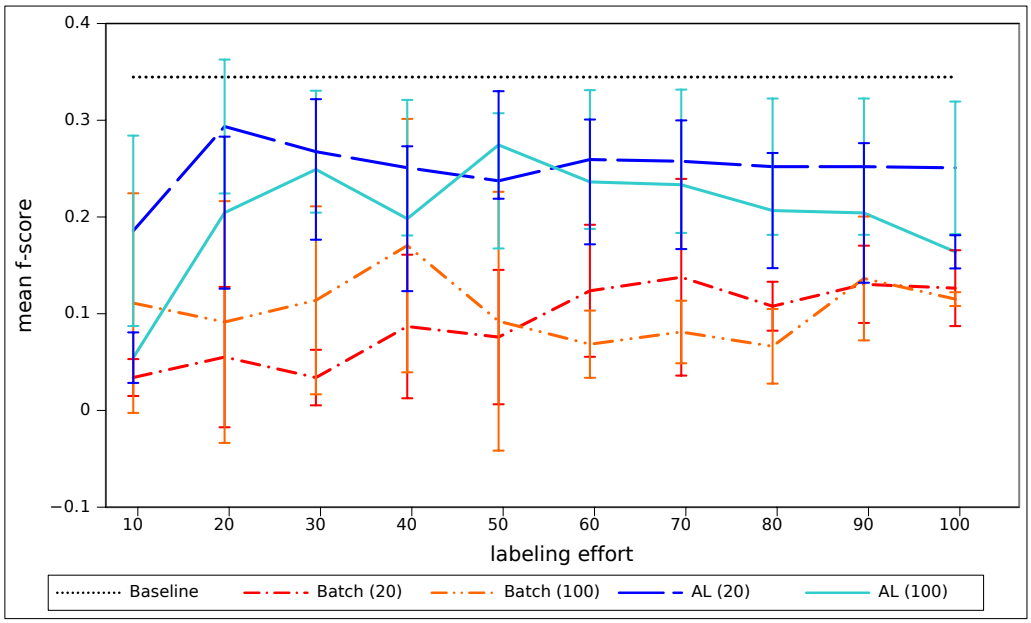

Figure 5.5: Results of the Products I experiment. Mean F1-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 34.5% F1-measure.

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.

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

Table 5.4: Comparison of best performances of different machine learning approaches on Publications I (ACM-DBLP)

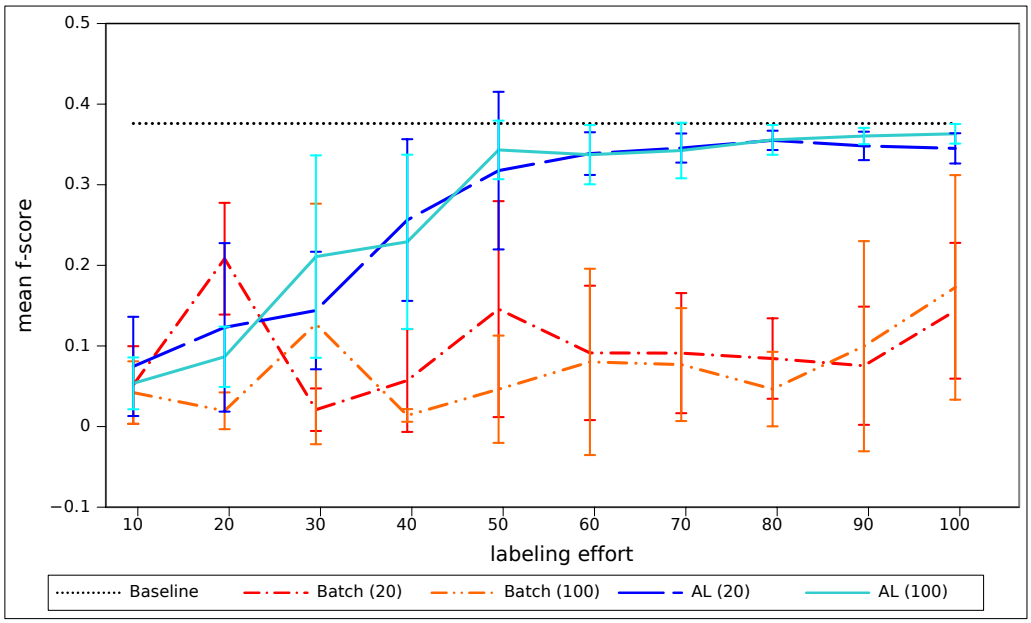

Figure 5.6: Results of the Products II experiment. Mean F-measures of five runs of batch and active learner, both using population sizes of 20 and 100 individuals. Baseline is at 37.6% F-measure.

5.3 Discussion

We showed that we can effectively compute accurate link specification using genetic programming techniques. Using an active learning approach we're able to alleviate the labeling effort for an oracle. Relying on the efficient LIMES algorithms we support considerable fast run-times. Therefore, it is possible to use the learning techniques of EAGLE for link specifications in an interactive environment.

We only studied the effects of different population sizes. So far with no conclusive results, it seems that populations of 20 individuals are sufficient in most cases. Maybe the effects of different population sizes is lessen, due to the rather larger number of generations the evolutionary process is applied each iteration. Also studying impact of different values for the probabilities

of the genetic operators mutation, crossover and reproductions should be subject to future work and may change this observation.
Overall, note that genetic programming is a non-deterministic search method. So, we cannot guarantee high quality results every time. For rather simple linking problems we can easily compute accurate link specification using only a small number of training data, as the Drugs and Publication I experiment suggest. If the problem at hand is of a more complex nature, as for experiment Publication II and Products I and II, where the reference data is not a 1:1 mapping and the data is more noisier, we can compute link specification close to 70% of the baselines. But, EAGLE either requires more training data or the quality of the computed link specifications are subject to higher fluctuations. We only analyzed one fitness measures, namely the F1 measure. For linking problems of different complexity emphasizing either precision or recall using different β values may lead to more satisfiable results.
Nonetheless, we showed that we can support the process of finding accurate link specification using genetic programming without being dependent of users with expert knowledge of the linking problem at hand. Furthermore, we are able to compete with other approaches, both with respect to the quality of the link specifications as to run-times.

6 Conclusion and Future Work

In this work 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 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.
For future work we will further explore how certain parameters like population size, number of generations and the probabilities of the genetic operators interact and influence the computation of efficient link specifications.
During implementing the presented approach certain limits of the JGAP library get obvious. In the current version we only support setting global parameters for both mutation and crossover probabilities. Once an individual got selected for one of these genetic operators, the individual is subject to a random change. Especially the change of thresholds to an arbitrary value is not efficient in every case. Individuals with worser fitness values may benefit more from drastic changes then those with better fitness values. Moreover, adapting more specific mutation principles as used in [34] may

further improve the performance of our approach.
Genetic programming is a nondeterministic approach and thus able to efficiently adapt to complex learning tasks. But there are several drawbacks to this. For instance, during evolution of a link specification it is possible that we evolve redundant metric expressions, For example combining the same measures over the same property pairs with different thresholds. Such a link specification would be equivalent to an specification comparing the properties only once using the stricter measure with respect to its threshold. See figure A.7 for an example. Implementing tools to avoid building redundant link specifications in the first place, or punish those individuals while evolution, would save runtime as we avoid unnecessary computation, while ensuring the same linking results.
Our active learning approach in particular could be improved. The most influential part is the selection of the most informative link candidates presented to an oracle for labeling. We just used all individuals of a population to compute them with no respect to their fitness. As described in [12] enhancing mechanisms with reinforcement learning techniques could result in a more proper selection.
Furthermore, we intend to make the discovery of most suitable preprocessing steps subject to an unsupervised learning process. Thereby, we will be able to further automating the Link Discovery process. All these features will then also be added to the SAIM web application.

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22:207–216, June 1993.

[2] J.R. Anderson, R.S. Michalski, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*. Number Vol. 1 in Machine Learning: An Artificial Intelligence Approach. M. Kaufmann, 1985.

[3] Sören Auer, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. Introduction to linked data and its lifecycle on the web. In Axel Polleres, Claudia dŠAmato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science*, pages 1–75. Springer Berlin / Heidelberg, 2011.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[5] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[6] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2008.

[7] Moisés G. Carvalho, Albero H. F. Laender, Marcos André Gonçalves, and Altigran S. da Silva. Replica identification using genetic programming. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1801–1806, New York, NY, USA, 2008. ACM.

[8] Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD '08*, pages 1065–1068, 2008.

[9] Nello Cristianini and Elisa Ricci. Support vector machines. In *Encyclopedia of Algorithms*. Cambridge University Press, 2008.

[10] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann de Meer. idmesh: graph-based disambiguation of linked data. In *WWW*, pages 591–600, 2009.

[11] Moisés G. de Carvalho, Marcos André Gonçalves, Alberto H. F. Laender, and Altigran S. da Silva. Learning to deduplicate. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '06, pages 41–50, New York, NY, USA, 2006. ACM.

[12] J. de Freitas, G.L. Pappa, A.S. da Silva, M.A. Gonç andalves, E. Moura, A. Veloso, A.H.F. Laender, and M.G. de Carvalho. Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, july 2010.

[13] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2007.

[14] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19:1–16, 2007.

[15] Hugh Glaser, Ian C. Millard, Won-Kyung Sung, Seungwoo Lee, Pyung Kim, and Beom-Jong You. Research on linked data and co-reference resolution. Technical report, University of Southampton, 2009.

[16] Oktie Hassanzadeh and Mariano Consens. Linked movie data base. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Kingsley Idehen, editors, *Proceedings of the WWW2009 Worshop on Linked Data on the Web (LDOW 2009)*, 2009.

[17] Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann. 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 (NeFoRS2010)*, 2010.

[18] R. Isele, A. Jentzsch, and C. Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.

[19] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Comput.*, 15:1667–1689, July 2003.

[20] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow.*, 2(2):1574–1577, 2009.

[21] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3:484–493, September 2010.

[22] John R. Koza. *Genetic Programming*. MIT Press, Cambridge MA, 1992.

[23] John R. Koza. Genetic programming, 1992.

[24] P. Langley. *Elements of Machine Learning*. M. Kaufmann, 1996.

[25] Ray Liere and Prasad Tadepalli. Active learning with committees for text categorization. In *In proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 591–596, 1997.

[26] R.S. Michalski, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*. Number Vol. 2 in Machine Learning: An Artificial Intelligence Approach. M. Kaufmann, 1986.

[27] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 2 edition, 1997.

[28] Axel-Cyrille Ngonga Ngomo. A Time-Efficient Hybrid Approach to Link Discovery. In *Sixth International Ontology Matching Workshop*, 2011.

[29] Axel-Cyrille Ngonga Ngomo. A time-efficient hybrid approach for link discovery. In *ISWC'11 Workshop on Ontology Matching(OM-2011)*, Bonn, Germany, 2011.

[30] Axel-Cyrille Ngonga Ngomo and Sören Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *Proceedings of IJCAI*, 2011.

[31] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. RAVEN – Active Learning of Link Specifications. In *Proceedings of OM@ISWC*, 2011.

[32] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *Proceedings of ESWC*, 2012.

[33] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Unsupervised learning of link specifications: Deterministic vs. non-deterministic. *Unpublished manuscript*, 2012.

[34] A. Nikolov, A D'Aquin, and E. Motta. Unsupervised learning of data linking configurations. In *Proceedings of ESWC*, 2012.

[35] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *ASWC*, pages 332–346, 2009.

[36] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanasz, and Wolfgang Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, 2011.

[37] Yves Raimond, Christopher Sutton, and Mark Sandler. Automatic interlinking of music datasets on the semantic web. In *Proceedings of the 1st Workshop about Linked Data on the Web*, 2008.

[38] François Scharffe, Yanbin Liu, and Chuguang Zhou. 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.

[39] Burr Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2009.

[40] Jennifer Sleeman and Tim Finin. Computing foaf co-reference relations with rules and machine learning. In *Proceedings of the Third International Workshop on Social Data on the Web*, 2010.

[41] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *ISWC*, pages 649–664, 2011.

[42] William Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census - Research Report Series, 2006.

[43] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets Syst.*, 69:125–139, January 1995.

s

7 Kurzzusammenfassung

Um die Vision eines Linked Data Webs zu verwirklichen werden effiziente halbautomatische Verfahren benötigt, um Links zwischen verschiedenen Datenquellen zu generieren. Viele bekannte Link Discovery Frameworks verlangen von einem Benutzer eine Linkspezifikation manuell zu erstellen, bevor der eigentliche Vergleichsprozess zum Finden dieser Links gestartet werden kann.
Zwar wurden jüngst zeit- und ressourcenschonende Werkzeuge zur Ausführung von Linking-Operationen entwickelt, aber die Generierung möglichst präziser Linkspezifikationen ist weiterhin ein kompliziertes Unterfangen. Diese Arbeit präsentiert EAGLE - ein Werkzeug zum halbautomatischen Lernen solcher Linkspezifikationen. EAGLE erweitert das zeiteffiziente LIMES Framework um aktive Lernalgorithmen basierend auf Methoden der Genetischen Programmierung. Ziel ist es den manuellen Arbeitsaufwand während der Generierung präziser Linkspezifikationen für Benutzer zu minimieren. Das heißt insbesondere, dass die Menge an manuell annotierten Trainingsdaten minimiert werden soll. Dazu werden Batch- als auch aktive Lernalgorithmen verglichen. Zur Evaluation werden mehrere Datensätze unterschiedlichen Ursprungs und verschiedener Komplexität herangezogen. Es wird gezeigt, dass EAGLE zeiteffizient Linkspezifikationen vergleichbarer Genauigkeit bezüglich der F-Maße gernerieren kann, während ein geringerer Umfang an Trainingsdaten für die aktiven Lernalgorithmen benötigt wird.

A Appendix

A.1 Images

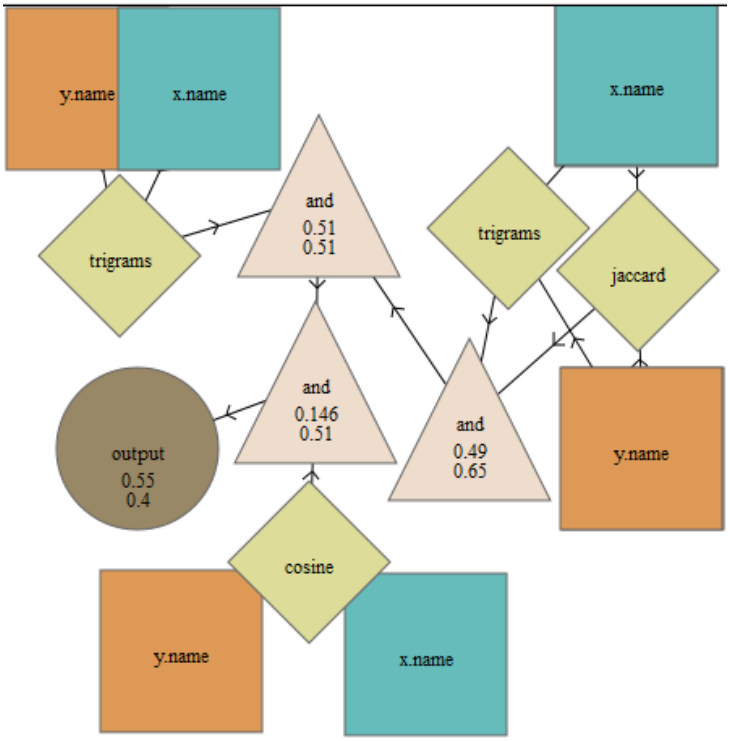

Figure A.1: Link Specification of the baseline of the Drugs experiment linking Dailymed with Drugbank.

A Appendix

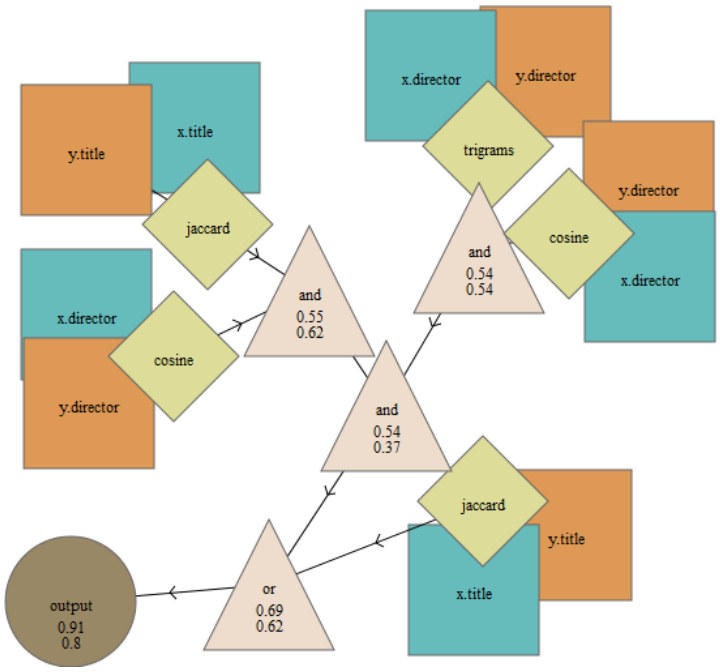

Figure A.2: Link Specification of the baseline of the Movies Experiment.

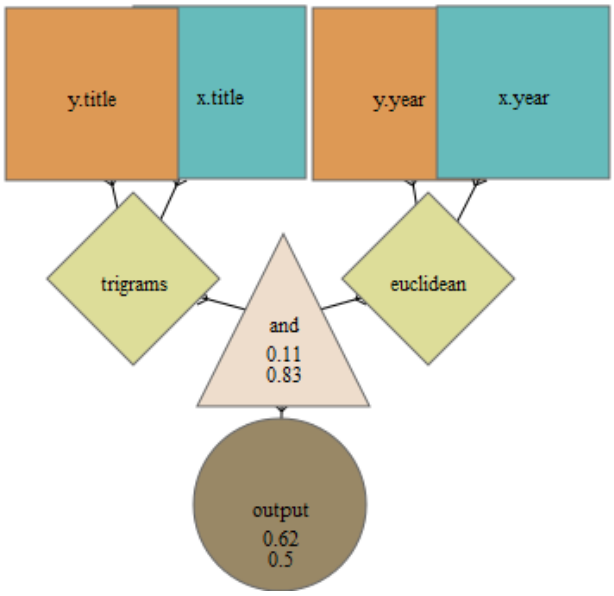

Figure A.3: Link Specification of the baseline of the Publications I experiment linking DBLP with ACM.

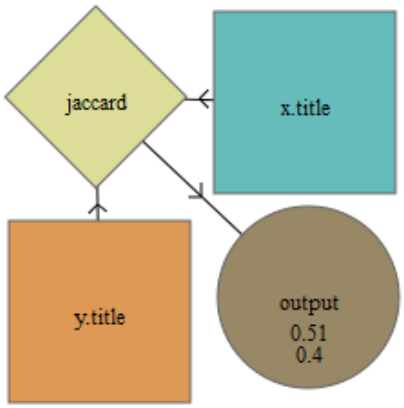

Figure A.4: Link Specification of the baseline of the Publications II experiment linking DBLP with Google Scholar.

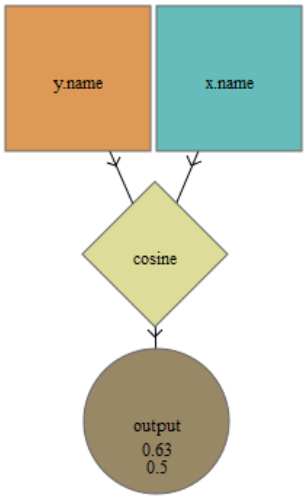

Figure A.5: Link Specification of the baseline of the Products I experiment.

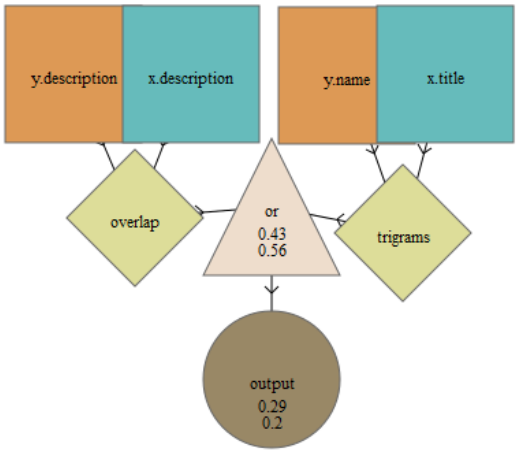

Figure A.6: Link Specification of the baseline of the Products II experiment.

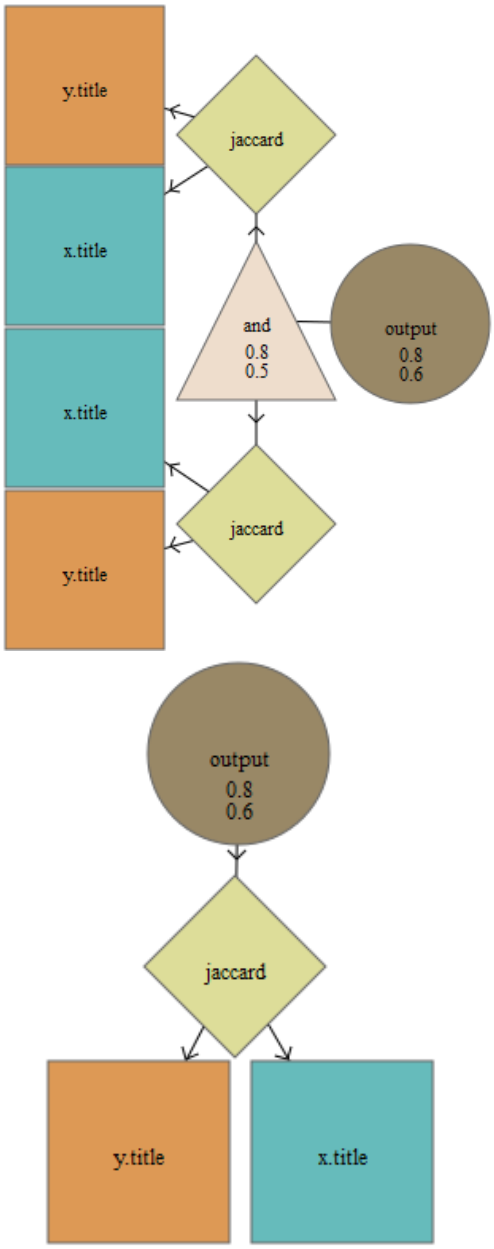

Figure A.7: Example of a redundant Link Specification and its equivalent minimalization. Visualization was done with the SAIM prototype.

A.2 Tables

The following tables show the detailed results of all experiments. In all table headers Batch stands for the batch learners while AL stands for the active learners. The numbers in brackets are the sizes of the populations used. The first row of each learner group contains the mean F-Measures achieved by the learner. The rows *std der* contain the standard derivations over all 5 runs. For the first 3 experiments (Drugs, Movies and Publication I) we also included the mean runtimes in milliseconds in the row *mean dur*.

Oracle	baseline	AL (20)			AL (100)			Batch (20)			Batch (100)		
			std der	mean dur		std der	mean dur		std der	mean dur		std der	mean dur
10	0.999	0.999	0.001	578	0.999	0.000	2206	0.999	0.001	632	0.998	0.001	2374
20	0.999	0.998	0.001	1044	0.998	0.001	4390	0.998	0.001	1069	0.999	0.001	4337
30	0.999	0.998	0.001	1715	0.998	0.001	6440	0.998	0.001	1408	0.999	0.001	6248
40	0.999	0.998	0.001	2595	0.998	0.001	8730	0.998	0.001	1797	0.997	0.001	8444
50	0.999	0.998	0.001	2852	0.998	0.001	11727	0.998	0.001	2169	0.999	0.001	10530
60	0.999	0.999	0.000	3502	0.998	0.001	12960	0.998	0.001	2523	0.998	0.001	11966
70	0.999	0.998	0.001	4722	0.998	0.001	16759	0.998	0.001	2758	0.999	0.000	15000
80	0.999	0.998	0.001	4885	0.998	0.001	19111	0.998	0.001	3383	0.998	0.001	17000
90	0.999	0.999	0.000	5899	0.998	0.001	22075	0.998	0.001	3614	0.998	0.001	18995
100	0.999	0.998	0.001	7090	0.999	0.000	25971	0.998	0.001	4323	0.998	0.001	21685

Table A.1: Results Drug experiment.

Oracle	baseline	AL (20)			AL (100)			Batch (20)			Batch (100)		
			std der	mean dur		std der	mean dur		std der	mean dur		std der	mean dur
10	0.976	0.805	0.352	182	0.533	0.444	257	0.377	0.333	1846	0.466	0.463	1910
20	0.976	0.838	0.282	1729	0.834	0.194	2455	0.376	0.409	1658	0.704	0.373	2658
30	0.976	0.756	0.340	3146	0.865	0.178	5199	0.734	0.279	1666	0.544	0.371	2476
40	0.976	0.820	0.181	5237	0.921	0.030	8576	0.615	0.489	1835	0.729	0.359	3066
50	0.976	0.836	0.195	7971	0.941	0.038	13508	0.558	0.385	2423	0.320	0.378	3745
60	0.976	0.951	0.023	10467	0.829	0.185	19217	0.706	0.341	1845	0.576	0.394	4511
70	0.976	0.824	0.186	12453	0.934	0.035	26862	0.764	0.418	1962	0.842	0.245	4632
80	0.976	0.944	0.031	15850	0.910	0.114	34929	0.672	0.391	2085	0.446	0.379	4844
90	0.976	0.951	0.023	19411	0.947	0.033	44719	0.568	0.347	2120	0.696	0.378	5757
100	0.976	0.953	0.025	24443	0.944	0.031	55642	0.332	0.155	2428	0.697	0.263	6280

Table A.2: Results Movies experiment.

Oracle	baseline	AL (20)			AL (100)			Batch (20)			Batch (100)		
			std der	mean dur		std der	mean dur		std der	mean dur		std der	mean dur
10	0.972	0.855	0.139	4085	0.940	0.039	22675	0.816	0.213	3123	0.957	0.007	18452
20	0.972	0.860	0.174	12476	0.957	0.003	49108	0.959	0.016	10547	0.956	0.008	42828
30	0.972	0.939	0.029	29548	0.961	0.005	122050	0.921	0.025	8932	0.954	0.005	71798
40	0.972	0.937	0.022	33936	0.958	0.003	139713	0.931	0.033	17437	0.965	0.006	164930
50	0.972	0.941	0.029	66705	0.958	0.007	316386	0.951	0.025	21529	0.964	0.007	274164
60	0.972	0.953	0.025	86453	0.955	0.003	451691	0.943	0.028	23011	0.967	0.001	285193
70	0.972	0.950	0.022	211435	0.960	0.002	447795	0.947	0.033	52306	0.967	0.002	348586
80	0.972	0.954	0.005	261320	0.953	0.004	994446	0.948	0.023	58422	0.946	0.008	507449
90	0.972	0.942	0.043	456647	0.955	0.005	1053398	0.944	0.030	39483	0.951	0.014	374205
100	0.972	0.958	0.007	374837	0.961	0.003	1583122	0.951	0.017	194215	0.955	0.003	656968

Table A.3: Results Publications I experiment.

Oracle	baseline	AL (20)	std der	AL (100)	std der	Batch (20)	std der	Batch (100)	std der
10	0.772	0.381	0.347	0.297	0.274	0.029	0.029	0.335	0.276
20	0.772	0.514	0.307	0.297	0.278	0.267	0.029	0.229	0.204
30	0.772	0.442	0.220	0.432	0.133	0.343	0.029	0.228	0.220
40	0.772	0.473	0.311	0.467	0.258	0.206	0.029	0.444	0.276
50	0.772	0.552	0.333	0.697	0.087	0.217	0.029	0.093	0.095
60	0.772	0.500	0.305	0.572	0.293	0.150	0.029	0.283	0.301
70	0.772	0.607	0.157	0.625	0.103	0.250	0.029	0.402	0.245
80	0.772	0.558	0.311	0.626	0.157	0.112	0.029	0.257	0.188
90	0.772	0.504	0.297	0.695	0.152	0.271	0.029	0.307	0.347
100	0.772	0.523	0.111	0.752	0.030	0.181	0.029	0.242	0.234

Table A.4: Results Publications II experiment.

Oracle	base-line	AL (20)		AL (100)		Batch (20)		Batch (100)	
			std der		std der		std der		std der
10	0.345	0.186	0.098	0.055	0.026	0.034	0.019	0.111	0.114
20	0.345	0.294	0.069	0.204	0.079	0.055	0.073	0.091	0.125
30	0.345	0.268	0.063	0.249	0.073	0.034	0.029	0.114	0.097
40	0.345	0.251	0.070	0.198	0.075	0.087	0.074	0.170	0.131
50	0.345	0.237	0.070	0.274	0.056	0.076	0.069	0.092	0.134
60	0.345	0.259	0.072	0.236	0.065	0.124	0.068	0.068	0.035
70	0.345	0.258	0.074	0.233	0.067	0.138	0.102	0.081	0.032
80	0.345	0.252	0.070	0.207	0.060	0.108	0.025	0.066	0.039
90	0.345	0.252	0.070	0.204	0.072	0.130	0.040	0.136	0.064
100	0.345	0.251	0.069	0.164	0.017	0.126	0.039	0.115	0.007

Table A.5: Results Products I experiment.

Oracle	base-line	AL (20)		AL (100)		Batch (20)		Batch (100)	
			std der		std der		std der		std der
10	0.376	0.075	0.062	0.054	0.032	0.052	0.048	0.042	0.039
20	0.376	0.123	0.105	0.087	0.037	0.208	0.069	0.020	0.023
30	0.376	0.144	0.073	0.211	0.126	0.021	0.026	0.127	0.149
40	0.376	0.256	0.100	0.229	0.108	0.057	0.064	0.014	0.008
50	0.376	0.318	0.098	0.343	0.036	0.146	0.134	0.046	0.067
60	0.376	0.339	0.026	0.337	0.037	0.091	0.083	0.080	0.116
70	0.376	0.346	0.018	0.342	0.034	0.091	0.075	0.077	0.070
80	0.376	0.355	0.012	0.356	0.018	0.084	0.050	0.047	0.046
90	0.376	0.348	0.018	0.360	0.010	0.075	0.073	0.100	0.130
100	0.376	0.345	0.019	0.363	0.012	0.144	0.084	0.173	0.139

Table A.6: Results Products II experiment.

List of Figures

List of Tables

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet.
Mir ist bekannt, dass eine Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Ort Datum Unterschrift