Learning Linkage Rules using Genetic Programming

Robert Isele and Christian Bizer

Freie Universität Berlin, Web-based Systems Group
Garystr. 21, 14195 Berlin, Germany
mail@robertisele.com, chris@bizer.de

Abstract. An important problem in Linked Data is the discovery of links between entities which identify the same real world object. These links are often generated based on manually written linkage rules which specify the condition which must be fulfilled for two entities in order to be interlinked. In this paper, we present an approach to automatically generate linkage rules from a set of reference links. Our approach is based on genetic programming and has been implemented in the Silk Link Discovery Framework. It is capable of generating complex linkage rules which compare multiple properties of the entities and employ data transformations in order to normalize their values. Experimental results show that it outperforms a genetic programming approach for record deduplication recently presented by Carvalho et. al. In tests with linkage rules that have been created for our research projects our approach learned rules which achieve a similar accuracy than the original human-created linkage rule.

Keywords: Genetic Programming, Linked Data, Link Discovery, Duplicate Detection, Deduplication, Record Linkage

1 Introduction

In the decentralized Web of Data, many data sources use different URIs for the same real world object. Identifying these URI aliases, is a central problem in Linked Data. Two approaches are widely used for that purpose: The first category includes fully automatic tools which identify links using unsupervised learning [11]. The second category includes tools which improve the accuracy of the generated links using user-provided *linkage rules*. A linkage rule [30], specifies the conditions two entities must fulfill in order to be interlinked. For this purpose, a linkage rule typically uses one or more distance measures to compare the properties of the entities. If the data sources use different data types, the property values may be normalized by applying transformations prior to the comparison. Linkage rules aggregate multiple similarity measures into one compound similarity value. As these conditions are strongly domain dependent, a separate linkage rule is typically used for each type of entities.

In this paper, we present an approach to automatically learn linkage rules from a set of reference links. The approach is based on genetic programming

and generates linkage rules that can be understood and further improved by humans. Our approach has been implemented and evaluated in Silk [16], a link discovery framework which generates RDF links between data items based on linkage rules which are expressed using the Silk Link Specification Language (Silk-LSL). The current version of Silk which includes the presented learning method can be downloaded from the project homepage[1] under the terms of the Apache Software License.

The experimental evaluation shows that it produces better results than a recently developed genetic programming approach by Carvalho et. al. [8]. In tests with linkage rules that have been created for our research projects our approach learned rules which achieve a similar accuracy than the original human-created linkage rule.

This paper is organized as follows: The following Section gives an overview of related work. Section 3 describes the proposed approach in detail. Afterwards, Section 4 presents the results of the experimental evaluation of the learning algorithm. Finally, Section 5 concludes this paper.

2 Related Work

Supervised learning of linkage rules in the context of linked data can build on previous results in record linkage. In literature many approaches suitable for learning binary classifiers have been adapted for learning linkage rules [18]. This section gives an overview of the most widely used approaches.

Naive Bayes. Based on the original Fellegi-Sunter statistical model [13] of record linkage, methods from Bayesian statistics such as *Naive Bayes* classifiers [31] have been used to learn linkage rules. The main disadvantage of Naive Bayes classifiers from a practical point of view is that they represent a black box system to the user. This means that the user can not easily understand and improve the learned linkage rules.

Support Vector Machines. Another widely used approach is to learn parameters of the linkage rule using *Support Vector Machines* (SVM) [5]. A SVM is a binary linear classifier which maps the input variables into a high-dimensional space where the two classes are separated by a hyperplane via a kernel function [2]. In the context of learning linkage rules, SVMs are often employed to learn specific parameters of a linkage rule such as the weights of different similarity measures. One popular example is MARLIN (Multiply Adaptive Record Linkage with INduction) [1], which uses SVMs to learn how to combine multiple similarity measures.

Decision Trees. Linkage rules can also be modeled using *Decision Trees* which can be learned by a variety of algorithms including genetic algorithms. The main advantage of Decision Trees is that they provide explanations for each classification and thus can be understood and improved manually. Active Atlas [28, 29]

[1] `http://www4.wiwiss.fu-berlin.de/bizer/silk/`

learns mappings rules consisting of a combination of predefined transformations and similarity measures. TAILOR [10] is another tool which employs decision trees to learn linkage rules.

Genetic Programming. Another approach which is more expressive than decision trees and promising to learn complex linkage rules is *genetic programming* (GA). Genetic programming is an extension of the genetic algorithm [15] which has been first proposed by Cramer [6]. Similar to a genetic algorithm, it starts with a randomly created population of individuals. Each individual is represented by a tree which is a potential solution to the given problem. From that starting point the algorithm iteratively transforms the population into a population with better individuals by applying a number of genetic operators. These operations are applied to individuals which have been selected based on a fitness measure which determines how close a specific individual is to the desired solution. The three genetic operators typically used in genetic programming are [19]:

Reproduction: An individual is copied without modification.
Crossover: Two selected individuals are recombined into a new individual.
Mutation: A random modification is applied to the selected individual.

The algorithm stops as soon as either the configured maximum number of iterations or a user-defined stop condition is reached.

Genetic programming has been applied to many problems in a variety of domains [26]. In many of these areas genetic programming is capable of producing human-competitive results [23, 21, 22]. Examples include the synthesis of electrical circuits [20], the creation of quantum algorithms [27], and the development of controllers [22].

To the best of our knowledge, genetic programming for learning linkage rules has only been applied by Carvalho et. al. so far [7, 4, 8]. Their approach uses genetic programming to learn how to combine a set of presupplied pairs of the form `<attribute, similarity function>` (e.g. `<name, Jaro>`) into a linkage rule. These pairs can be combined by the genetic programming method arbitrarily by using mathematical functions (e.g. +, -, *, /, exp) and constants. Carvalho et. al. show that their method produces better results than the state-of-the-art SVM based approach by MARLIN [8]. Their approach is very expressive although it cannot express data transformations. On the downside, using mathematical functions to combine the similarity measures does not fit any commonly used linkage rule model [12] and leads to complex and difficult to understand linkage rules.

We are not aware of any previous application of genetic programming to learn linkage rules in the context of Linked Data.

3 Approach

This Section explains our approach of learning linkage rules using genetic programming. It is organized as follows: First of all, in order to learn a linkage

rule using genetic programming, a rule must be represented as a tree structure. Thus, Section 3.1 describes our approach of representing a linkage rule using 4 basic operators. For each candidate solution the fitness function described in Section 3.2 is used to determine the performance of a linkage rule. Section 3.3 describes how the initial population of candidate solutions is generated. After the initial population has been generated, the candidate solutions are iteratively transformed into better ones by breeding the population according to the rules described in Section 3.4. Finally, Section 3.5 describes our approach to avoid the occurrence of bloat in linkage rules.

3.1 Representation of a Linkage Rule

We represent a linkage rule as a tree which is built from 4 basic operators:

Property: Creates a set of values to be used for comparison by retrieving all values of a specific property of the entity.

Transformation: Transforms the input values according to a specific data transformation function.

Comparison: Evaluates the similarity between the values of two input operators according to a specific distance measure. A user-specified threshold specifies the maximum distance. If the underlying properties do not provide any values for a specific entity, no similarity value is returned.

Aggregation: Aggregates the similarity values from multiple operators into a single value according to a specific aggregation function. Aggregation functions such as the weighted average may take the weight of the operators into account. If an operator is marked as required, the aggregation will only yield a value if the operator itself provides a similarity value.

The resulting linkage rule forms a tree where the terminals are given by the properties and the nodes are represented by transformations, comparisons and aggregations. The linkage rule tree is strongly typed [25] i.e. it does not allow arbitrary combinations of its four basic operators. Figure 1 specifies the valid structure of a linkage rule. Figure 2 shows a simple example of a linkage rule.

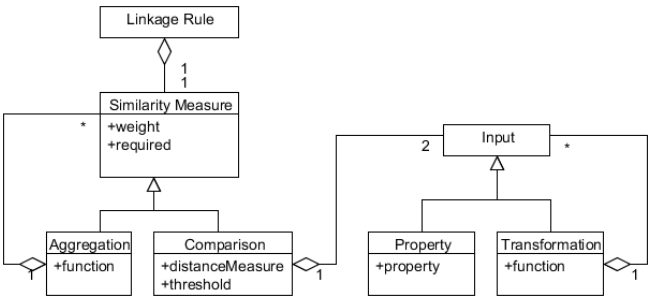

Fig. 1. Structure of a linkage rule

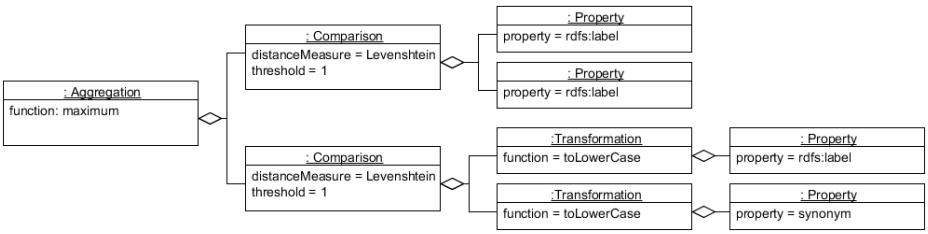

Fig. 2. Example linkage rule

3.2 Fitness Function

The quality of a linkage rule is assessed by the fitness function based on user-provided training data. The training data consists of a set of positive reference links (connecting entities which identify the same real world object) and a set of negative reference links (connecting entities which identify different objects). The prediction of the linkage rule is compared with the positive reference links while counting *true positives* (TP) and *false negatives* (FN) and the negative reference links while counting *false positives* (FP) and *true negatives* (TN). Based on these counts, a fitness value between -1 and 1 is assigned to the linkage rule by calculating *Matthews correlation coefficient* (MCC):

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

In contrast to many other popular fitness measures such as the F-measure (i.e. the harmonic mean of precision and recall), Matthews correlation coefficient yields good results even for heavily unbalanced training data.

3.3 Generating the Initial Population

This section explains our approach of generating the initial population: Before the population is generated, we build a list of property pairs which hold similar values as described below. Based on that, random linkage rules are built by selecting property pairs from the list and applying data transformations, comparisons and aggregations. Finally, we seed the population with common comparison patterns in order to increase the efficiency of the algorithm.

Finding Compatible Properties Prior to generating the population, we generate a list of pairs of properties which hold similar values. For this purpose, the datasets are preprocessed in order to find the 100 most frequent properties in the data set where the entity is in subject position and the 10 most frequent properties where the entity is in object position. The selection of the `owl:sameAs` property has been disallowed as it usually is the result of an existing run of a link discovery tool. For each possible property pair, the values of the entities referenced by the positive reference links as well as the negative reference links

are analyzed. This is done by tokenizing the values and counting the reference links for which there is a distance measure in the list of functions configured to be used for learning linkage rules according to which both values are similar (given a certain threshold). Finally, the list of compatible properties is generated by collecting all pairs of properties for which more positive than negative links are counted.

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

Note that this does not limit the algorithm to learn more complex linkage rules as it is the purpose of the genetic operators to generate more complex linkage rules from the ones in the initial population.

Seeding with Common Comparison Patterns Analyzing a set of linkage rules manualy developed for the LATC EU project (`http://latc-project.eu/`) revealed that certain patterns occur in many linkage rules. Most noticeable, 84 % of the analyzed linkage rules compare the labels and 66 % the geographic coordinates of the entities. For that reason, the population has been seeded not only with fully random linkage rules, but also with linkage rules which contain these two special cases. While these patterns can also be learned by the algorithm, previous work [26, 14] shows that seeding them in the initial population can improve the efficiency.

3.4 Breeding

In order to improve the population our approach employs all three common genetic operations: reproduction, crossover and mutation. At first, 1% of the individuals with the highest fitness are directly selected for reproduction following a elitist strategy [9]. After this, new individuals are generated using crossover and mutation until the population size is reached.

Instead of using subtree crossover, which is commonly used in genetic programming, our approach uses a set of specific crossover operators which are tailored to the domain. For each crossover operation an operator from this set is selected randomly and applied to two selected individuals. Each operator learns one aspect of the linkage rule. For our experiments, we used the following operators:

Function Crossover Used to find the best similarity, transformation or aggregation function. Selects one operator at random in each linkage rule and interchanges the functions. For example, it may select a comparison with the levensthein distance function in the first linakge rule and a comparison with the jaccard distance function in the second linkage rule and than interchange these two functions.

Operators Crossover As a linkage rule usually needs to combine multiple comparisons, this operator combines aggregations from both linkage rules. For this, it selects two aggregations, one from each linkage rule and combines theirs comparisons. The comparisons are combined by selecting all comparisons from both aggregations and removing each comparison with a propability of 50%. For example, it may select an aggregation of a label comparison and a date comparison in the first linkage rule and an aggregation of a label comparison and a comparison of the geographic coordinates in the second linkage rule. In this case the operator replaces the selected aggregations with a new aggregation which contains all 4 comparisons and then removes each comparison with a propability of 50%. Note that the comparisons are exchanged including the complete subtree i.e. the distance functions as well as existing transformations are retained.

Aggregation Crossover While most linkage rules are linear i.e. can be expressed using a single weighted average aggregation, some linkage rules need more complex aggregation hierarchies. In order to learn these hierachies, aggregation crossover selects a random aggregation or comparison operator in the first linkage rule and replaces it with a random aggregation or comparison operator from the second linkage rule. This way, the operator builds a hierachy as it may select operators from different levels in the tree. For example, it may select a comparison in the first linkage rule and replace it with a aggregation of multiple comparisons from the second linakge rule.

Transformation Crossover This operator is used to learn complex transformations by selecting a random path of transformations in both linkage rules. It then combines both paths by executing a two point crossover.

Threshold Crossover This operator is used to find the optimal thresholds. For this, one comparison operator is selected at random in each individual. The new threshold is then set to the average of both comparisons.

Weight Crossover Finds the optimal weights analog to the treshold crossover.

Mutation is implemented similarly by selecting a random crossover operator and executing a headless chicken crossover [17] i.e. crossing an individual from the population with a randomly generated individual.

3.5 Avoiding Bloat

One well-known problem in genetic programming is that over time the individuals may develop redundant parts which do not contribute to their overall fitness [3, 24]. One possibility to control this bloating is to penalize big trees in order to force the algorithm to favor smaller trees over bigger ones. In literature this method is known as parsimony pressure [32]. Another more sophisticated method is to automatically analyze the trees and remove redundant parts. For that purpose a simplification algorithm has been developed which detects redundant parts in the linkage rule and removes them. In order to avoid bloated linkage rules the simplification algorithm is executed every 5 generations.

4 Evaluation

The proposed learning approach has been evaluated in 3 different experiments. Because genetic algorithms are non-deterministic and may yield different results in each run, all experiments have been run 10 times. For each run the reference links have been randomly split into 2 folds for cross-validation. The results of all runs have been averaged and the standard deviation has been computed. For each experiment, we provide the evaluation results with respect to the training data set as well as the validation dataset. All experiments have been run on a 3GHz Intel(R) Core i7 CPU with 4 cores while the Java heap space has been restricted to 1GB.

4.1 Parameters

Table 1 lists the parameters which have been used in all experiments. As it is the purpose of the developed method to work on arbitrary datasets without the need to tailor its parameters to the specific datasets that should be interlinked, the same parameters have been used for all experiments.

Table 1. Learning Parameters

Parameter	Value
Population size	500
Maximum number of generations	50
Selection method	Tournament selection with a group size of 5
Probability of Crossover	75%
Probability of Mutation	25%
Stop Condition	MCC = 1.0

Our approach is independent of any specific aggregation functions, distance measures or data transformations. Thus, it can learn linkage rules with any functions provided to it. Table 2 shows the set of functions which has been used by us in all experiments. The details about the functions are provided in the Silk user manual on the website.

Table 2. Set of functions used in all experiments

Aggregation Functions	Distance Measures	Transformations
Average similarity	Levenshtein distance	Convert to lower case
Maximum similarity	Jaccard index	Tokenize the string
Minimum similarity	Numeric distance	Strip the URI prefix
	Geographic distance	

4.2 Experiment 1: Comparison with related work

At first, we evaluated how our approach compares to the genetic programming approach by Carvalho et. al. [8], which claims to produce better results than the state-of-the-art SVM based approach by MARLIN. One dataset commonly used for evaluating different record deduplication approaches is *Cora*. The Cora dataset contains citations to research papers from the Cora Computer Science research paper search engine. For the purpose of evaluating our approach, we converted the Cora dataset provided at [2] to RDF.

For evaluation we used 1617 randomly selected positive links and 1617 randomly selected negative reference links. Table 4.2 summarizes the cross validation results. On average, our approach achieved an F-measure of 96.9% against the training set and 93.6% against the validation set and needed less than 5 minutes to perform all 50 iterations on the test machine. The learned linkage rules compared by title, author and date. For the same dataset, Carvalho et. al. report an F-measure of 90.0% against the training set and 91.0% against the validation set [8].

Table 3. Average results of all learning runs. The last row contains the best results of Carvalho et. al. for comparison.

Iter.	Time in s (σ)	Train. F1 (σ)	Train. MCC (σ)	Val. F1 (σ)	Val. MCC (σ)
1	4.0 (0.3)	0.896 (0.022)	0.806 (0.042)	0.896 (0.021)	0.805 (0.041)
10	31.1 (3.9)	0.956 (0.013)	0.912 (0.026)	0.954 (0.015)	0.907 (0.029)
20	71.4 (18.3)	0.964 (0.008)	0.928 (0.017)	0.960 (0.010)	0.919 (0.020)
30	132.5 (48.5)	0.965 (0.007)	0.931 (0.013)	0.962 (0.007)	0.924 (0.015)
40	217.6 (106.7)	0.968 (0.004)	0.936 (0.008)	0.945 (0.036)	0.900 (0.053)
50	271.1 (140.1)	0.969 (0.003)	0.938 (0.007)	0.936 (0.056)	0.902 (0.057)
Ref.	-	0.900 (0.010)	-	0.910 (0.010)	-

4.3 Experiment 2: Learning linkage rules for geographic datasets

With 16 datasets in the LOD cloud[3], interlinking geographic datasets is a very common problem. For this reason we evaluated our approach by learning a linkage rule for interlinking cities in DBpedia and LinkedGeoData.

In order to evaluate the learned linkage rules we used a manually collected set of 100 positive and 100 negative reference links. Special care has been taken to include rare corner cases such as for example cities which share the same name but don't represent the same city and cities which are very closely located so that. Table 4.3 summarizes the cross validation results. In all runs, the stop condition (i.e. an MCC of 100%) has been reached before the 25th iteration.

[2] `http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/cora_dataset.html`

[3] `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`

Table 4. Average results of all learning runs.

Iter.	Time in s (σ)	Train. F1 (σ)	Train. MCC (σ)	Val. F1 (σ)	Val. MCC (σ)
1	2.6 (1.0)	0.984 (0.025)	0.970 (0.046)	0.932 (0.059)	0.883 (0.099)
10	3.8 (2.1)	0.996 (0.007)	0.993 (0.013)	0.932 (0.059)	0.883 (0.099)
20	3.9 (2.3)	0.998 (0.004)	0.996 (0.007)	0.964 (0.032)	0.945 (0.056)
25	4.0 (2.4)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)

4.4 Experiment 3: Learning complex linkage rules

While the vast majority of linkage rules commonly used are very simple, a few of them employ more complex structures. Interlinking drugs in DBpedia and Drugbank is an example where the original linkage rule which has been produced by humans is very complex. In order to match two drugs, it compares the drug names and their synonyms as well as a list of well-known and used identifiers (e.g. the CAS number[4]). In total, the manually written linkage rule uses 13 comparisons and 33 transformations. This includes complex transformations such as replacing specific parts of the strings.

All 1,403 Links which have been generated by executing the original linkage rule have been used as positive reference links. Negative reference links have been generated by shuffling the target URIs of the positive links.

Table 5 shows the averaged results of all runs. The learned linkage rules yield an F-Measure of 99.8% for the training data and 99.4% for the validation data. Figure 3 shows that from the 30th iteration the generated linkage rules on average only use 5.6 comparisons and 3.2 transformations and the simplification algorithm successfully avoids bloating in the subsequent iterations. Thus, the learned linkage rules use less than half of the comparisons and only one-tenth of the transformations of the manually written linkage rules.

Table 5. Average results of all learning runs

Iter.	Time in s (σ)	Train. F1 (σ)	Train. MCC (σ)	Val. F1 (σ)	Val. MCC (σ)
1	67.5 (2.2)	0.929 (0.026)	0.876 (0.042)	0.928 (0.029)	0.874 (0.045)
10	334.1 (157.4)	0.994 (0.002)	0.987 (0.003)	0.991 (0.003)	0.983 (0.006)
20	1014.1 (496.8)	0.996 (0.001)	0.992 (0.002)	0.988 (0.010)	0.977 (0.017)
30	1829.7 (919.3)	0.997 (0.001)	0.994 (0.002)	0.985 (0.016)	0.973 (0.027)
40	2685.4 (1318.9)	0.998 (0.001)	0.996 (0.002)	0.994 (0.002)	0.988 (0.004)
50	3222.2 (1577.7)	0.998 (0.001)	0.996 (0.001)	0.994 (0.002)	0.989 (0.004)

5 Conclusion and Outlook

We presented an approach for learning complex linkage rules which compare multiple properties of the entities and employ data transformations in order

[4] A unique numerical identifier assigned by the "Chemical Abstracts Service"

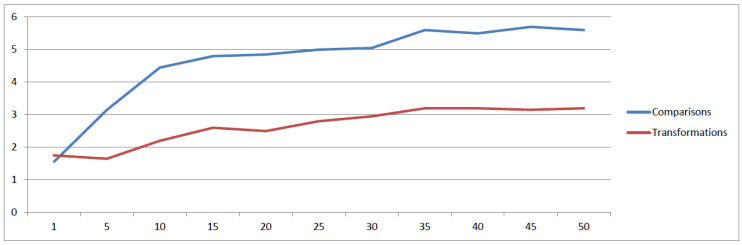

Fig. 3. Average number of comparisons and transformations

to normalize their values. As the current algorithm requires manually supplied reference links, future work will focus on the efficient generation of these. For this, we will investigate into semi-supervised learning and active learning in order to minimize the user effort to generate the reference links.

References

1. M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.
2. M. Bilenko and R. J. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. Technical report, 2002.
3. T. Blickle and L. Thiele. Genetic Programming and Redundancy. 1994.
4. M. Carvalho, A. Laender, M. Gonçalves, and A. da Silva. Replica identification using genetic programming. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1801–1806. ACM, 2008.
5. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.
6. N. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*, volume 183, page 187, 1985.
7. M. G. de Carvalho, M. A. Gonçalves, A. H. F. Laender, and A. 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.
8. M. G. de Carvalho, A. H. F. Laender, M. A. Goncalves, and A. S. da Silva. A genetic programming approach to record deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2010.
9. K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Ann Arbor, MI, USA, 1975.
10. M. Elfeky, V. Verykios, and A. Elmagarmid. Tailor: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 17–28. IEEE, 2002.
11. J. Euzenat, A. Ferrara, C. Meilicke, et al. First Results of the Ontology Alignment Evaluation Initiative 2010. *Ontology Matching*, page 85, 2010.
12. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

13. I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328), 1969.
14. C. Henrik Westerberg and J. Levine. Investigation of different seeding strategies in a genetic planner. *Applications of Evolutionary Computing*, pages 505–514, 2001.
15. J. Holland. Adaptation in natural and artificial systems. 1975.
16. R. Isele, A. Jentzsch, and C. Bizer. Silk server - adding missing links while consuming linked data. In *1st International Workshop on Consuming Linked Data (COLD 2010), Shanghai*, 2010.
17. T. Jones. Crossover, macromutation, and population-based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80, 1995.
18. H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197 – 210, 2010.
19. J. Koza. *Genetic programming: on the programming of computers by means of natural selection*.
20. J. Koza, F. Bennett III, F. Bennett, D. Andre, and M. Keane. Genetic Programming III: Automatic programming and automatic circuit synthesis, 1999.
21. J. Koza, M. Keane, and M. Streeter. What's AI done for me lately? Genetic programming's human-competitive results. *Intelligent Systems, IEEE*, 18(3):25–31, 2003.
22. J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*. Springer Verlag, 2005.
23. J. Koza, M. Keane, J. Yu, F. Bennett, and W. Mydlowec. Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1(1):121–164, 2000.
24. W. Langdon and R. Poli. Fitness causes bloat. *Soft Computing in Engineering Design and Manufacturing*, 1:13–22, 1997.
25. D. Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995.
26. R. Poli, W. Langdon, and N. McPhee. *A field guide to genetic programming*. Lulu Enterprises Uk Ltd, 2008.
27. L. Spector, H. Barnum, H. Bernstein, and N. Swamy. Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.
28. S. Tejada, C. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.
29. S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 350–359, New York, NY, USA, 2002. ACM.
30. W. E. Winkler. Matching and Record Linkage. In *Business Survey Methods*, pages 355–384, 1995.
31. W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002.
32. B. Zhang and H. M
"uhlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.