pgfid1886505417940015 pgfid43526777417952303 pgfid53820051017728389 pgfid63005688338530525 pgfid93526777438542813 pgfid103820051038318899

RAVEN: Towards Zero-Configuration Link Discovery

Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, Konrad Höffner

Department of Computer Science, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
{ngonga|lehmann|auer}@informatik.uni-leipzig.de,
konrad.hoeffner@uni-leipzig.de
WWW home page: http://aksw.org

Abstract. With the growth of the Linked Data Web, time-efficient approaches for computing links between data sources have become indispensable. Yet, in many cases, determining the right specification for a link discovery problem is a tedious task that must still be carried out manually. In this article we present RAVEN, an approach for the semi-automatic determination of link specifications. Our approach is based on the combination of stable solutions of matching problems and active learning leveraging the time-efficient link discovery framework LIMES. RAVEN is designed to require a small number of interactions with the user in order to generate classifiers of high accuracy. We focus with RAVEN on the computation and configuration of Boolean and weighted classifiers, which we evaluate in three experiments against link specifications created manually. Our evaluation shows that we can compute linking configurations that achieve more than 90% F-score by asking the user to verify at most twelve potential links.

Keywords: Linked Data, Link Discovery, Algorithms, Constraints

1 Introduction

The rationale of the Linked Data paradigm is to facilitate the transition from the document-oriented to the Semantic Web by extending the current Web with a commons of interlinked data sources [6]. Two key challenges arise when trying to discover links between data sources: the computational complexity of the matching task *per se* and the selection of an appropriate configuration for maximizing precision and recall.

The first challenge lies in the a-priori *complexity* of a matching task being proportional to the product of the number of instances in the source and target data source, an unpractical proposition as soon as the source and target knowledge bases become large. With the *LIMES framework*[1] [30, 29], we addressed this challenge by providing a lossless approach for time-efficient link discovery that is significantly faster than the state-of-the-art.

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

The second challenge of the link discovery process lies in the specification of an appropriate *configuration* for the tool of choice. Such a specification usually consists of (1) a set of restrictions on the source and target knowledge base, (2) a list of properties of the source and target knowledge base to use for similarity detection, (3) a combination of suitable similarity measures (e.g., Levenshtein [25]) and (4) similarity thresholds. Until now, such link discovery specifications are usually defined *manually*, in most cases via a time-consuming trial-and-error approach. Yet, the choice of a suitable configuration decides upon whether satisfactory links can be discovered. Specifying complex link configurations is a tedious process, as the user does not necessary know which combinations of properties lead to an accurate linking configuration. The difficulty of devising suitable link discovery specifications is amplified on the Web of Data by the *sheer size* of the knowledge bases (which often contain millions of instances) and their *heterogeneity* (i.e., by the complexity of the underlying ontologies, which can contain thousands of different types of instances and properties) [6].

In this paper, we present the RApid actiVE liNking (RAVEN) approach. To the best of our knowledge, RAVEN is the first approach to apply active learning techniques for the semi-automatic generation of specifications for link discovery. Our approach is based on a combination of stable matching problems (as known from machine learning) and a novel active learning algorithm derived from perceptron learning. RAVEN allows to determine:

- A sorted *matching of classes to interlink*; this matching represents the set of restrictions on the source and target knowledge bases.
- A stable *matching of properties* based on the selected restrictions that specifies the similarity space within which the linking is to be carried out.
- A highly accurate *link specification* including similarity measures and thresholds obtained via active learning.

Our evaluation with three series of experiments shows that we can compute linking configurations that achieve more than 90% F-score by asking the user to verify at most twelve potential links. The RAVEN approach is *generic* enough to be implemented within any link discovery framework that supports complex link specifications. The results presented herein were obtained by implementing RAVEN within the LIMES framework. We chose LIMES because it implements lossless approaches[2] and is very time-efficient. A graphical user interface for the approach is available with SAIM[3].

This article is an extension of the corresponding workshop article presenting the RAVEN approach in [31]. Changes include an extended discussion of the evaluation results, e.g. the inclusion of property and class matching. Furthermore, the related work part was extended as well as further illustrations and explanations were added throughout the paper.

After discussing related work in Section 2, we explain preliminary notions for link discovery and stable matching in Section 3. The approach itself is discussed

[2] That is, approaches that are guaranteed to retrieve all links that abide by a given a link specification. Note that some blocking approaches do not fulfill this requirement.

[3] http://saim.sf.net

in Section 4. We continue by analysing RAVEN with three different linking tasks in Section 6 and finally conclude in Section 7 with an outlook on future work.

2 Related Work

Previous work related to this article can be divided in two main areas: the computation of links called *link discovery* and the *learning of link heuristics*.

2.1 Link Discovery

Current approaches for link discovery on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* approaches.

Domain-specific link discovery frameworks aim at discovering links between knowledge bases from a particular domain. For example, the approach implemented in RKB knowledge base (*RKB-CRS*) [15] focuses on computing links between universities and conferences while *GNAT* [35] discovers links between music data sets. Further simple or domain-specific approaches can be found in [10, 32, 45, 18, 41, 33].

Universal link discovery frameworks are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases. For example, *RDF-AI* [39] implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of data sets. SILK [45] is a time-optimized tool for link discovery. It implements a multi-dimensional blocking approach that projects the instances to match in a multi-dimensional metric space. Subsequently, this space is subdivided into to overlapping blocks that are used to retrieve matching instances without loosing links. Another lossless link discovery framework is *LIMES* [30], which addresses the scalability problem by utilizing the *triangle inequality* in metric spaces to compute pessimistic estimates of instance similarities. Based on these approximations, LIMES can filter out a large number of instance pairs that cannot suffice the matching condition set by the user. The experiments presented herein were carried out employing LIMES for performing the link discovery computations.

The task of discovering links between knowledge bases is closely related with *record linkage* [46, 12] and *deduplication* [8]. The database community has produced a vast amount of literature on efficient algorithms for solving these problems. Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking (see, e.g., [22]) have been developed to address the problem of the quadratic time complexity of brute force comparison methods. In addition, automatic techniques that aim at easing schema matching have also been developed. For example, [23] generates synthetic data out real data to create a ground truth that allow for tuning schema matching systems. [28] employ ensemble-learning techniques (especially boosting) to combine schema matchers while [11] implements a library a supervised classifiers to achieve the same goal.

2.2 Learning Link Heuristics

The second relevant research area for this paper is related to learning link specifications, which is usually carried out using a combination of shallow natural language processing (NLP) and machine learning methods. The existing methods aim at either one or both of the following goals: On the one hand, link creation should be made *more reliable* than purely manual approaches by using manual samples (supervised learning) for estimating precision and recall, user feedback (active learning) or analyzing network and other characteristics. On the other hand, those methods should also *simplify* the link creation process. As explained above, finding good interlinking heuristics can be burdensome and both non-experts and experts in the considered domain may struggle to find corresponding classes, properties, metrics and weights for their combination.

There has been a significant body of research work dedicated to matching ontologies [34, 24, 44, 16], including benchmarks in the ontology alignment evaluation initiative (OAEI). A recent comprehensive survey can be found in [4], which covers many aspects of the research field. Finding links on instance level, which is the primary concern of this paper, has received less attention, although OAEI has been extended in 2009 with benchmarks in this area [13].

One approach in this area is *RiMOM* [26], which combines several techniques to compute matchings. When matching instances, it takes the schema of the knowledge bases into account. RiMOM combines several strategies and similarity functions and works unsupervised, i.e. without training. Another approach is *ObjectCoRef* [19]. It is based on learning the most distinctive features, i.e. property-value pairs, of entities in knowledge bases. In contrast to other approaches, it is not aimed at computing all links between two knowledge bases, but considers the task of linking entities in a whole cloud of knowledge bases (typically, the LOD cloud[4]) in a semi-supervised approach. Another recent approach is *SERIMI* [2]. It proceeds in two phases: a selection and a disambiguation phase. In the selection phase, SERIMI computes a mapping which interlinks entities in two input knowledge bases with low precision and high recall. In this phase, it relies on string similarity of the labels of entities. The disambiguation phase filters the output of the first phase by deciding amongst candidates with equal or similar labels.

In both cases, one of the problems is to obtain appropriate data for utilizing machine learning approaches without overburdening the user [5, 21]. For this reason, active learning has been employed by the database community [37, 38, 1]. Active learning approaches usually present only few match candidates to the user for manual verification. The technique is particularly efficient in terms of required user input [40], because the user is only confronted with those match candidates which provide a high benefit for the underlying learning algorithm.

The RAVEN approach presented in this article goes beyond the state-of-the-art in several ways: It is the first RDF-based approach to use active learning for obtaining interlinking heuristics. In addition, it is the first active learning

[4] http://lod-cloud.net

algorithm in this area. Moreover, it is the first approach to detect corresponding classes and properties automatically for the purpose of link discovery. Note that this challenge is very specific to and particularly relevant for the Data Web. In similar approaches developed for databases, the mapping of columns is often assumed to be known [1]. Yet, this assumption cannot be made when trying to link knowledge bases from the Web of Data because of the possible size of the underlying ontology. By supporting the automatic detection of links, we are able to handle heterogeneous knowledge bases with extremely large schemata.

3 Preliminaries

Our approach to the active learning of linkage specifications extends ideas from several research areas, especially classification and stable matching problems. In the following, we present the notation that we use throughout this article and explain the theoretical framework underlying our work.

3.1 Problem Definition

The link discovery problem, which is similar to the record linkage problem, is an ill-defined problem and is consequently difficult to model formally [1]. In general, link discovery aims to discover pairs $(s, t) \in S \times T$ related via a relation R.

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

The sets S resp. T are usually (not necessarily disjoint) subsets of the instances contained in two (not necessarily disjoint) knowledge bases $\mathcal{K}_S$ resp. $\mathcal{K}_T$. In most cases, the computation of whether $R(s, t)$ holds for two elements is carried out by projecting the elements of S and T based on their properties in a similarity space $\mathfrak{S}$ and setting $R(s, t)$ iff some similarity condition is satisfied. The specification of the sets S and T and of this similarity condition is usually performed within a *link specification* which is the input for a link discovery framework such as LIMES or SILK.

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 τ_1, ..., τ_n such that τ_i is the threshold for σ_i.*

A restriction $\mathcal{R}$ is generally a logical predicate. Typically, restrictions in link specifications state (a) the `rdf:type` of the elements of the set they describe, i.e., $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass` or (b) the features the elements of the set must have, e.g., $\mathcal{R}(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each $s \in S$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, while each $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. Note that the atomic similarity functions $\sigma_1, ..., \sigma_n$

can be combined to σ by different means. In this paper, we will focus on using Boolean operators and real weights combined as conjunctions. Also note that we are aware that several other categories of approaches can be used to determine pairs (s, t) such that $R(s, t)$, including approaches based on ontology matching, semantic similarity and formal inference. In this paper, we will be concerned exclusively with link discovery problems that can be specified via link specifications as defined above.

According to the formalizations of link discovery and link specifications above, finding matching pairs of entities can be defined as a classification task, where the classifier $\mathcal{C}$ maps each pair $(s, t) \in S \times T$ to one of the classes $\{-1, +1\}$.

Definition 3 (Link Discovery as Classification). *Given the set $S \times T$ of possible matches, the goal of link discovery is to find a classifier $\mathcal{C} : S \times T \rightarrow \{-1, +1\}$ such that $\mathcal{C}$ maps non-matches to the class -1 and matches to $+1$. $\mathcal{M}$ is then the set $\{(s, t) : \mathcal{C}(s, t) = +1\}$.*

In general, we assume classifiers that operate in an n-dimensional similarity space $\mathfrak{S}$. Each of the dimensions of $\mathfrak{S}$ is defined by a similarity function σ_i that operates on a certain pair of attributes of s and t. Each classifier $\mathcal{C}$ on $\mathfrak{S}$ can be modeled via a specific function $\mathcal{F}_{\mathcal{C}}$. $\mathcal{C}$ then returns $+1$ iff the logical statement $\mathcal{P}_{\mathcal{C}}(\mathcal{F}_{\mathcal{C}}(s, t))$ holds and -1 otherwise, where $\mathcal{P}_{\mathcal{C}}$ is what we call the specific predicate of $\mathcal{C}$. In this work, we consider two families of classifiers: *linear weighted* classifiers $\mathcal{L}$ and *Boolean conjunctive* classifiers $\mathcal{B}$. The specific function of linear weighted classifiers is of the form

$$\mathcal{F}_{\mathcal{L}}(s, t) = \sum_{i=1}^{n} \omega_i \sigma_i(s, t), \tag{1}$$

where $\omega_i \in \mathbb{R}$. The predicate $\mathcal{P}_{\mathcal{L}}$ for a linear classifier is of the form $\mathcal{P}_{\mathcal{L}}(X) \leftrightarrow (X \geq \tau)$, where $\tau = \tau_1 = ... = \tau_n \in [0, 1]$ is the similarity threshold. A Boolean classifier $\mathcal{B}$ is a conjunction of n atomic linear classifiers $\mathcal{C}_1$, ... ,$\mathcal{C}_n$, i.e., a conjunction of classifiers that each operate on exactly one of the n dimensions of the similarity space $\mathfrak{S}$. Thus, the specific function $\mathcal{F}_{\mathcal{B}}$ is a Boolean function of the form

$$\mathcal{F}_{\mathcal{B}}(s, t) = \bigwedge_{i=0}^{n} (\sigma_i(s, t) \geq \tau_i) \tag{2}$$

and the specific predicate is simply $\mathcal{P}_{\mathcal{B}}(X) = X$. Note, that given that classifiers are usually learned by using iterative approaches, we will denote classifiers, weights and thresholds at the t^{th} iteration by using superscripts, i.e., $\mathcal{C}^t$, ω_i^t and τ_i^t.

Current approaches to learning in record matching assume that the similarity space $\mathfrak{S}$ is given. While this is a sensible premise for mapping problems which rely on simple schemas, the large schemas (i.e., the ontologies) that underlie many data sets in the Web of Data do not allow such an assumption. DBpedia [7, 43] (version 3.6) for example contains 289,016 classes which are partially mapped to 275 classes from the main DBpedia ontology. Moreover, it contains

42,016 properties, which are partially mapped to 1,335 properties from the main DBpedia ontology. Thus, it would be extremely challenging and tedious at best for a user to specify the properties to map when carrying out a simple deduplication analysis, let alone more complex tasks using the DBpedia data set. Other data sets in the LOD cloud, such as LinkedGeoData [42, 3] are even larger or have a similar size. Thus, being able to scale to those datasets is of crucial importance. In the following, we give a brief overview of stable matching problems, which we use to solve the problem of suggesting appropriate sets of restrictions on data and matching properties to generate a similarity space $\mathfrak{S}$ in which the link discovery problem can be carried out.

3.2 Stable Matching Problems

The best known stable matching problem is the stable marriage problem $\mathcal{SM}$ as formulated by [14]. The basic problem here is as follows: given two sets of males and females of equal magnitude, compute male-female pairings that are such that none of the partners p_1 in the pairings can cheat on his/her partner with another person p_2 that he/she prefers. Cheating is considered to be possible iff this other person, i.e., p_2, also considers the partner willing to cheat (p_1) more suitable than his/her current partner.

Formally, we assume two sets M (males) and F (females) such that $|M| = |F|$ and two functions $\mu : M \times F \to \{1, ..., |F|\}$ resp. $\gamma : M \times F \to \{1, ..., |M|\}$, that give the degree to which a male likes a female and vice versa. $\mu(m, f) > \mu(m, f')$ means that m prefers f to f'. Note, that for all f and f' where $f \neq f'$ holds, $\mu(m, f) \neq \mu(m, f')$ must also hold. Analogously, $m \neq m'$ implies $\gamma(m, f) \neq \gamma(m', f)$. A bijective function $s : M \to F$ is called a stable matching iff for all m, m', f, f' the following holds:

$$(s(m) = f) \wedge (s(m') = f') \wedge (\mu(m, f') > \mu(m, f)) \to (\gamma(m', f') > \gamma(m, f')) \quad (3)$$

In [14] an algorithm for solving this problem is presented and it is shown how it can be used to solve a generalization of the stable marriage problem, i.e., the well-known Hospital/Residents ($\mathcal{HR}$) problem. Formally, $\mathcal{HR}$ assumes a set R of residents $r \in R$ that each have a sorted preference list of $p(r)$ of hospitals they would like admission to. The list $p(r)$ can be derived from the preference function μ as defined for the stable marriage problem. We write $p(x, y) = n$ to state that y is at position n in the preference list of x, where x can be a hospital or a resident. Each hospital h from the set H of hospitals also has a preference list $p(h)$ of residents and a maximal capacity $c(h)$. Similarly to $p(r)$, the list $p(h)$ can be derived from the preference function γ as defined for the stable marriage problem. A stable solution of the Hospital/Residents problem is a mapping of residents to hospitals such that:

- Each hospital accepts maximally $c(h)$ residents;
- No resident r is assigned to a hospital h such that a hospital h' which had a higher preference in $p(r)$ would be willing to admit r and vice-versa.

Algorithm 1 RAVEN's stable matching algorithm

Require: Set of residents R
Require: Set of hospitals H
Require: Preference function p
  $\mathcal{M} = \emptyset$ // Mapping of hospitals to residents
  for $r \in R$ **do**
    $i(r) = 0$ //index for preference function
  end for
  for $h \in H$ **do**
    $c(h) = \left\lceil \frac{|R|}{|H|} \right\rceil$ //capacity setting
  end for
  while $R \neq \emptyset$ **do**
    for $r \in R$ **do**
      $h = p(r)[i(r)]$
      if $|\mathcal{M}(h)| < c(h)$ **then**
        $\mathcal{M}(h) := \mathcal{M}(h) \cup \{r\}$
        $R = R \backslash \{r\}$
      else
        if $\exists r' \in \mathcal{M}(h)\ p(h, r) < p(h, r')$ **then**
          $r'' = \arg\min_{r' \in \mathcal{M}(h)} p(h, r')$
          $R = R \cup \{r''\}$
          $\mathcal{M}(h) := \mathcal{M}(h) \backslash \{r''\}$
          $\mathcal{M}(h) := \mathcal{M}(h) \cup \{r\}$
          $R = R \backslash \{r\}$
        end if
      end if
      $i(r) + +$
    end for
  end while
  return $\mathcal{M}$

Note that we assume there are no ties, i.e., that the functions μ and γ are injective. Given these premises, [14] shows that a stable matching always exists. Consequently, Algorithm 1 is guaranteed to return a solution. Note that we set the capacity of the hospitals to the smallest whole number that ensures that each resident finds a hospital. Also note that $p(h, r) < p(h, r')$ means that h prefers r over r'. More details on stable matching can be found in [27].

4 The RAVEN Approach

Our approach, dubbed RAVEN (RApid actiVE liNking), addresses the task of linking two knowledge bases S and T by using the active learning paradigm within the pool-based sampling setting [40]. Overall, the goal of RAVEN is to find the best classifier $\mathcal{C}$ that achieves the highest possible precision, recall or F_1 score as desired by the user. The algorithm also aims to minimize the burden on the user by limiting the number of link candidates that must be labeled by the user to a minimum.

Algorithm 2 The RApid actiVE liNking (RAVEN) algorithm

Require: Source knowledge base $\mathcal{K}_S$
Require: Target knowledge base $\mathcal{K}_T$
  Find stable class matching between classes of $\mathcal{K}_S$ and $\mathcal{K}_T$
  Find stable property matching for the selected classes
  Compute sets S and T; Create initial classifier $\mathcal{C}^0$; $t := 0$
  while termination condition not satisfied **do**
    Ask the user to classify 2α examples; Update $\mathcal{C}^t$ to $\mathcal{C}^{t+1}$; t := t+1
  end while
  Compute set $\mathcal{M}$ of links between S and T based on $\mathcal{C}^t$
  return $\mathcal{M}$

An overview of our approach is given in Algorithm 2. In a first step, RAVEN aims to detect the restrictions that will define the sets S and T. To achieve this goal, it tries to find a stable matching of pairs of classes, whose instances are to be linked. This is done by applying a two-layered approach and generating a sorted list of class mappings that are presented to the user, who can choose the pair that is to be matched. The second step of our approach consists of finding a stable matching between the properties that describe the instances of the classes specified in the first step. The user is also allowed to alter the suggested matching at will. Based on the property mapping, we compute S and T and generate an initial classifier $\mathcal{C} = \mathcal{C}^0$ in the third step. We then refine $\mathcal{C}$ iteratively by asking the user to classify pairs of instances that are most informative for our algorithm. $\mathcal{C}$ is updated until a termination condition is reached, for example $\mathcal{C}^t = \mathcal{C}^{t+1}$. The final classifier is used to compute the links between S and T, which are returned by RAVEN. In the following, we expand upon each of these three steps.

4.1 Stable Matching of Classes

The first component of a link specification is a set of restrictions that must be fulfilled by the instances that are to be matched. We present herein how such a matching can be carried out for restrictions that are of the form $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass`, as they are the most commonly used restriction. We use a two-layered approach for matching classes in knowledge bases. Our *default approach* begins by selecting a user-specified number of `sameAs` links between the source and the target knowledge base randomly. Then, it computes μ and γ on the classes C_S of $\mathcal{K}_S$ and C_T of $\mathcal{K}_T$ as follows[5]:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ sameAs } t \wedge t \texttt{ type } C_T\}|. \quad (4)$$

Although several million `sameAs` links exist on the Web of Data, some knowledge bases that refer to the same entities do not share any links. Consequently, our default approach fails when trying to process such pairs of knowledge bases. In this case, we run our *fallback* approach. It computes μ and γ on the classes of S and T as follows:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}|, \quad (5)$$

where `p` and `q` can be any property. Hence, the similarity of two classes is computed as the number of property values shared by triples such that their subjects are instances of C_S resp. C_T. Note that this similarity is independent from the properties through which the instances are connected to the property values.

We can draw on stable matchings, as introduced in Section 3.2 in order to find the most suitable pair of classes as follows: Let $c(S)$ be the number of classes C_S of S such that $\mu(C_S, C_T) > 0$ for any C_T. Furthermore, let $c(T)$ be the number of classes C_T of T such that $\gamma(C_S, C_T) > 0$ for any C_S. The capacity of each C_T is set to $\lceil c(S)/c(T) \rceil$, thus ensuring that the hospitals provide enough capacity to map all the possible residents. Once μ, γ and the capacity of each hospital has been set, we solve the equivalent $\mathcal{HR}$ problem. It is important to note that although the functions μ and γ are equivalent in both our default and our fallback approaches, the resulting problem is not symmetric, i.e., $p(C_S, C_T) = \zeta$ does not imply that $p(C_T, C_S) = \zeta$.

After the $\mathcal{HR}$ problem has been solved, we present the user with a stable matching sorted in descending order relatively to $\mu(C_S, C_T)$, thereby selecting the pair with the highest $\mu(C_S, C_T)$ as default match. Note that the fallback approach is approximately 30% slower than our default approach. Also, if the fallback approach fails, then we require the user to enter the class mapping manually.

4.2 Stable Matching of Properties

The detection of the best matching pairs of properties is very similar to the computation of the best matching classes. For datatype properties `p` and `q`, we

[5] Note that we used `type` to denote `rdf:type` and `owl:sameAs` to denote `sameAs`.

set:

$$\mu(p,q) = \gamma(p,q) = \{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}. \quad (6)$$

Note that while this equation might appear to be the same as Equation 4, they are actually different as p and q are bound in this equation. Thus, the similarity value that is computed is the number of common values that p and q link to and not (as in Equation 4) the number of common objects to triples whose subjects are instances of C_S resp. C_T. The initial mapping of properties defines the similarity space in which the link discovery task will be carried out. Note that none of the prior approaches to active learning for record linkage or link discovery automatized this step. We associate each of the basis vectors σ_i of the similarity space to exactly one of the pairs (p, q) of mapping properties detected by RAVEN. Once the restrictions and the property mapping have been specified, we can fetch the elements of the sets S and T. Given S and T, we can now compute an initial classifier that will be iteratively updated by RAVEN.

4.3 Initial Classifier

The specific formula for the initial linear weighted classifier $\mathcal{L}^0$ results from the formal model presented in Section 3 and is given by

$$\mathcal{F_L}^0(s,t) = \sum_{i=1}^{n} \omega_i^0 \sigma_i(s,t). \quad (7)$$

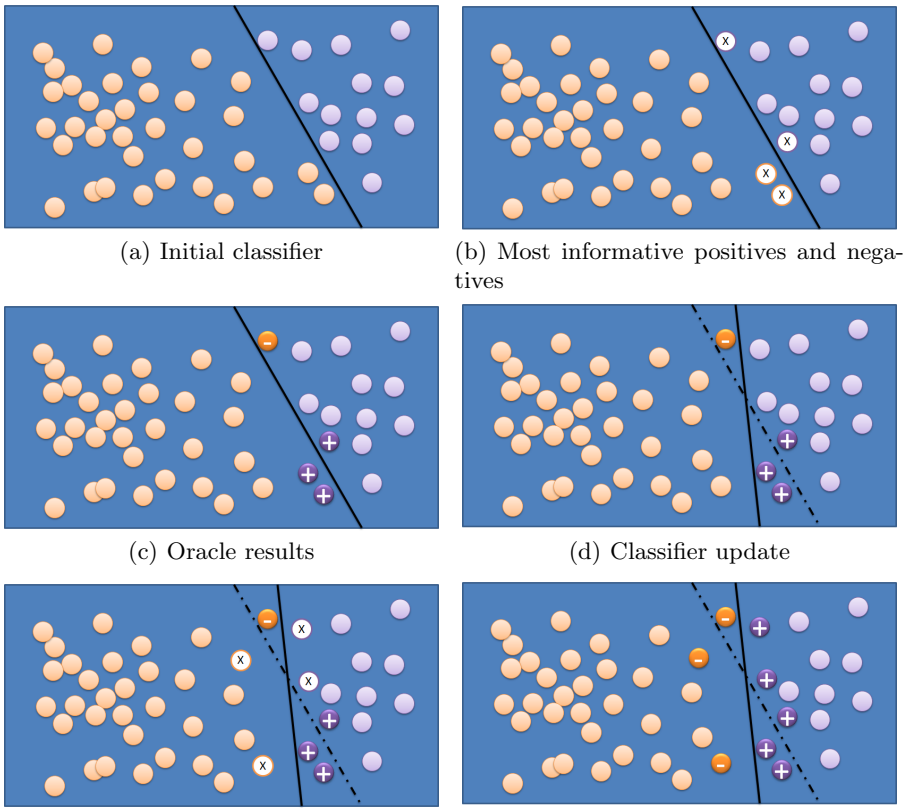
Several initialization methods can be used for ω_i^0 and the initial threshold τ^0 of $\mathcal{P_L}$. Here we chose the use the simplest possible approach by setting $\omega_i^0 := 1$ and $\tau^0 := \kappa n$, where $\kappa \in [0,1]$ is a user-specified *threshold factor*. Note that setting the overall threshold to κn is equivalent to stating that the arithmetic mean of the $\sigma_i(s,t)$ must be equal to κ.

The equivalent initial Boolean classifier $\mathcal{B}^0$ is given by

$$\mathcal{F_B}^0(s,t) = \bigwedge_{i=0}^{n} (\sigma_i^0(s,t) \geq \tau_i^0) \text{ where } \tau_i^0 := \kappa. \quad (8)$$

4.4 Updating Classifiers

RAVEN follows an iterative update strategy, which consists of asking the user to classify 2α elements of $S \times T$ (α is explained below) in each iteration step t and using these to update the values of ω_i^{t-1} and τ_i^{t-1} computed at step $t-1$. The main requirements to the update approach is that it computes those elements of $S \times T$ whose classification allow to maximize the convergence of $\mathcal{C}^t$ to a good classifier and therewith to minimize the burden on the user. The update strategy of RAVEN varies slightly depending on the family of classifiers. In the following, we present how RAVEN updates linear and Boolean classifiers.



(a) Initial classifier

(b) Most informative positives and negatives

(c) Oracle results

(d) Classifier update

(e) Most informative positives and negatives

(f) Termination

Fig. 1. Active learning as implemented by RAVEN. The most informative negative and positive examples are marked with an "X". The classified examples are marked with "+" for positive and "-" for negative.

Updating linear classifiers. The basic intuition behind our update approach is that given an initial classified $\mathcal{L}^0$, we aim to iteratively present the user with those elements from $S \times T$ whose classification is most unsure until we reach a certain termination condition. An example of such an initial classifier is shown in Figure 1(a). We call the elements presented to the user *examples*. We call an example *positive* when it is assumed by the classifier to belong to $+$. Else we call it *negative*. In Figure 1, the elements that the classifier assigns to the class $+$ are drawn in violet, while the elements of $-$ are drawn in orange. Once the user has provided us with the correct classification for the examples presented to him, the classifier can be updated effectively so as to better approximate the target classifier. In the following, we will define the notion of *most informative example* for linear classifiers before presenting our update approach.

When picturing a classifier as a boundary in the similarity space $\mathfrak{S}$ that separates the classes + and −, the examples whose classification is most uncertain are obviously those elements from $S \times T$ who are closest to the boundary specified by the classifier at hand. Note that we must exclude examples that have been classified previously, as presenting them to the user would not improve the classification accuracy of RAVEN while generating extra burden on the user, who would have to classify the same link candidate twice. Figure 1(b) depicts the idea behind most informative examples for linear classifiers. The elements of + and − that were not previously classified and that are closest to the boundary of $\mathcal{L}$ are selected as being most informative. These are the elements that are presented to the oracle (i.e., the user) for classification. An example of an oracle-given classification is given in Figure 1(c). The nodes marked with a + were marked by the user as being correct links, while those marked with a − were labeled as incorrect. Here, our classifier only classified one example correctly. Given this information, we aim to generate a classifier that minimize the overall error of RAVEN. To achieve this goal, we update the classifier by using an approach derived from perceptron learning as shown in Figure 1(d). The basic intuition behind choosing perceptron learning over other approaches is that we assume that our classifier should not be altered too drastically after each iteration, a goal which can be achieved with this approach. Recomputing a completely new classifier after each iteration using algorithms such as SVM [9] could lead to large deviations between the classifiers. In our evaluation, we analyse this aspect in more detail by testing different perceptron learning rates.

This classifier update strategy is rather conservative and is based on the assumption that the previous classifier was not completely random and should not be altered too drastically. In our example, using another update approach such as computing a Support Vector Machine based on the user-given classification would have led to a new classifier that would have been almost orthogonal to the initial one. We then iterate the computation of the most informative positive and negative examples (see Figure 1(e)) until a termination condition is reached, e.g., until the classifier can predict the user classification correctly (see Figure 1(e)).

Formally, let $\mathcal{M}^t$ be the set of $(s,t) \in S \times T$ classified by $\mathcal{L}^t$ as belonging to +1. Furthermore, let $\mathcal{P}^{t-1}$ (resp. $\mathcal{N}^{t-1}$) be the set of examples that have already been classified by the user as being positive examples, i.e, links (resp. negative examples, i.e., wrong links) in the first $t-1$ iterations. We define a set Λ as being a set of most informative examples λ for $\mathcal{L}^{t+1}$ when the following two conditions hold:

$$\forall \lambda \in S \times T \ (\lambda \in \Lambda \rightarrow \lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1}) \tag{9}$$

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow |{\mathcal{F}_{\mathcal{L}}}^t(\lambda') - \tau^t| \geq |{\mathcal{F}_{\mathcal{L}}}^t(\lambda) - \tau^t|. \tag{10}$$

Note that there are several sets of most informative examples of a given magnitude. We denote a set of most informative examples of magnitude α by Λ_α. A set of most informative positive examples, Λ^+, is a set of pairs such that

$$\forall \lambda \notin \Lambda^+ \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : ({\mathcal{F}_{\mathcal{L}}}^t(\lambda) < \tau^t) \vee (\forall \lambda^+ \in \Lambda^+ : {\mathcal{F}_{\mathcal{L}}}^t(\lambda) > {\mathcal{F}_{\mathcal{L}}}^t(\lambda^+)). \tag{11}$$

In words, Λ^+ is the set of examples that belong to class + according to $\mathcal{C}$ and are closest to $\mathcal{C}$'s boundary. Similarly, the set of most informative negative examples, Λ^-, is the set of examples such that

$$\forall \lambda \notin \Lambda^- \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : (\mathcal{F}_{\mathcal{L}}{}^t(\lambda) \geq \tau^t) \vee (\forall \lambda^- \in \Lambda^- : \mathcal{F}_{\mathcal{L}}{}^t(\lambda) < \mathcal{F}_{\mathcal{L}}{}^t(\lambda^-)). \quad (12)$$

We denote a set of most informative (resp. negative) examples of magnitude α as Λ^+_α (resp. Λ^-_α). The 2α examples presented to the user consist of the union $\Lambda^+_\alpha \cup \Lambda^-_\alpha$, where Λ^+_α and Λ^-_α are chosen randomly amongst the possible sets of most informative positive resp. negative examples .

The update rule for the weights of $\mathcal{L}^t$ is derived from the well-known Perceptron algorithm (see e.g., [36]), i.e.,

$$\omega_i^{t+1} = \omega_i^t + \eta^+ \sum_{\lambda \in \Lambda^+} \rho(\lambda)\sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-} \rho(\lambda)\sigma_i(\lambda), \quad (13)$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{L}^t$ are the same and 1 when they differ.

The threshold is updated similarly, i.e,

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda^+_\alpha} \rho(\lambda)\mathcal{F}_{\mathcal{L}}{}^t(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-_\alpha} \rho(\lambda)\mathcal{F}_{\mathcal{L}}{}^t(\lambda). \quad (14)$$

Note that the weights are updated by using the dimension which they describe while the threshold is updated by using the whole specific function. Finally, the sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated to

$$\mathcal{P}^t := \mathcal{P}^{t-1} \cup \Lambda^+_\alpha \quad (15)$$

and

$$\mathcal{N}^t := \mathcal{N}^{t-1} \cup \Lambda^-_\alpha. \quad (16)$$

Updating Boolean classifiers The notion of *most informative example* differs slightly for Boolean classifiers. λ is considered a most informative example for $\mathcal{B}$ when the conditions

$$\lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} \quad (17)$$

and

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow \sum_{i=1}^{n} |\sigma_i^t(\lambda') - \tau_i^t| \geq \sum_{i=1}^{n} |\sigma_i^t(\lambda) - \tau_i^t| \quad (18)$$

hold. The update rule for the thresholds τ_i^t of $\mathcal{B}$ is then given by

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda^+_\alpha} \rho(\lambda)\sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-_\alpha} \rho(\lambda)\sigma_i(\lambda), \quad (19)$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{C}_{t-1}$ are the same and 1 when they differ. The sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated as given in Equations 15 and 16.

5 Implementation

As stated above, RAVEN was implemented based on LIMES but the core ideas presented herein can be implemented in any link discovery framework. Figure 2 gives an overview of the workflow behind RAVEN. Once the classes and properties have been matched, RAVEN begins by generating an initial classifier. This classifier is converted into a link specification object that is forwarded to the LIMES kernel. The kernel then runs the link specification and generates a set of potential links with are sent back to RAVEN. RAVEN then computes the most informative positive and negative links and sends this set of inquiries to the oracle (i.e., the user). The oracle then classifies the links and sends the correct classification back to RAVEN. If the classification has not been altered, RAVEN terminates. Else the current classifier is updated and then transformed into a link specification, therewith starting the cycle anew. Note that the RAVEN algorithm is deterministic.

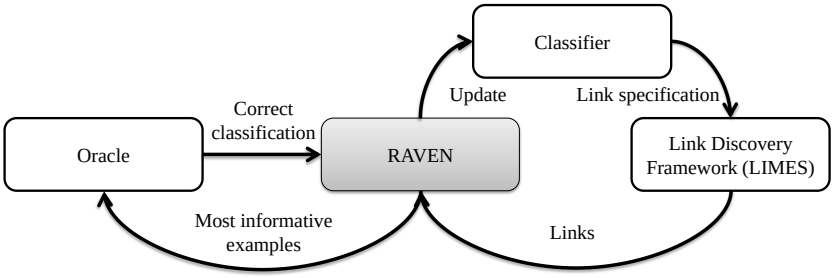


Fig. 2. Workflow of the RAVEN implementation.

6 Experiments and Results

6.1 Experimental Setup

We carried out three series of experiments to evaluate our approach. In our first experiment, dubbed *Diseases*, we aimed to map diseases from DBpedia with diseases from Diseasome. In the *Drugs* experiments, we linked drugs from Sider with drugs from Drugbank. Here, only 43 links were to be detected. Finally, in the *Side-Effects* experiments, we aimed to link side-effects of drugs and diseases in Sider and Diseasome. The reference data consisted of 454 links. The size of the reference data is given in Table 1

In all experiments, we aimed to compute how well linear and Boolean classifiers learned by RAVEN could approximate a manually specified configuration for mapping two knowledge bases. We used the following setup: The learning rates η^+ and η^- as explained in Section 4.4 were set to the same value η, which

Experiment	S	T	$\|S\|$	$\|T\|$	Number of correct links
Diseases	DBpedia	Diseasome	4647	4213	178
Drugs	Sider	Drugbank	914	4772	43
Side-Effects	Sider	Diseasome	1737	4213	454

Table 1. Overview of experimental data.

we varied between 0.01 and 0.1. We set the number of inquiries, i.e. the number of questions asked to the user, per iteration to 4. The threshold factor κ, explained in Section 4.3 was set to 0.8. In addition, the number of instances used during the automatic detection of class resp. property matches was set to 100 resp. 500. If no class mapping was found via `sameAs` links, then the fallback solution was called and compared the property values of 1000 instances chosen randomly from the source and target knowledge bases. We used the trigrams metric as default similarity measure for strings and the Euclidean similarity as default similarity measure for numeric values. As reference data, we used the set of instances that mapped perfectly according to a configuration created manually, which is similar to the approach in [20]. We can then compute precision, recall and F-score against those reference links. We also measured the total number of inquiries, i.e. questions to the oracle, that RAVEN needed to reach its maximal F-score. All experiments were carried out on an Intel Core2 Duo computer with 2.53GHz and 3072MB RAM.

6.2 Results

Experiment	Class Mapping
Diseases	`ds:diseases`→`dbp:Disease`*
	`ds:diseases`→`yago:Disease114070360`
	`ds:diseases`→`yago:NeurologicalDisorders`
	`ds:diseases`→`yago:TypesOfCancer`
	`ds:diseases`→`yago:GeneticDisorders`
	`ds:diseases`→`yago:BloodDisorders`
	`ds:diseases`→`yago:TransmissibleSpongiformEncephalopathies`
	`ds:diseases`→`yago:Syndromes`
	`ds:diseases`→`yago:ChromosomeInstabilitySyndromes`
	`ds:diseases`→`yago:PigmentDisorders`
	`ds:diseases`→`yago:CongenitalDisorders`
Drugs	`dbk:drugs`→`sd:drugs`*
	`dbk:drugs`→`sd:Offer`
Side-Effects	`sd:sideEffects`→`ds:diseases`*

Table 2. Initial property and class mappings computed by RAVEN in our experiments. The class mappings marked with an asterisk were returned by the stable matching algorithm and used in the classifier.

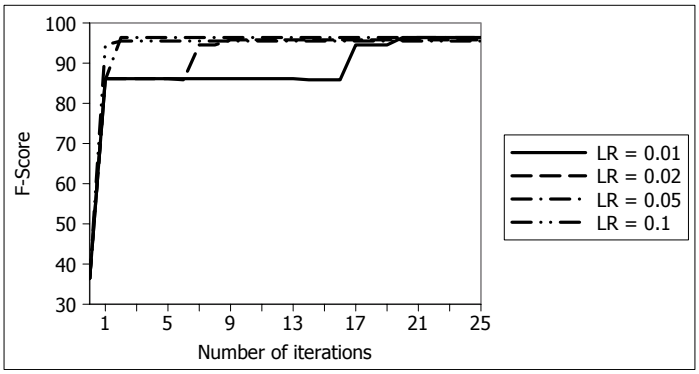
Experiment	Property Mapping
Diseases	rdfs:label → dbp:name*
	rdfs:label → foaf:name
	rdfs:label → rdfs:label
	ds:name → dbp:name
	ds:name → foaf:name*
	ds:name → rdfs:label
Drugs	dbk:brandName → sd:drugName*
	dbk:genericName → sd:drugName
	rdfs:label → sd:drugName
	dbk:brandName → rdfs:label
Side-Effects	rdfs:label → rdfs:label*
	rdfs:label → ds:name
	sd:sideEffectName → rdfs:label
	sd:sideEffectName → ds:name*

Table 3. Initial property mappings computed by RAVEN in our experiments. The property mappings marked with an asterisk were returned by the stable matching algorithm and used in the classifier.

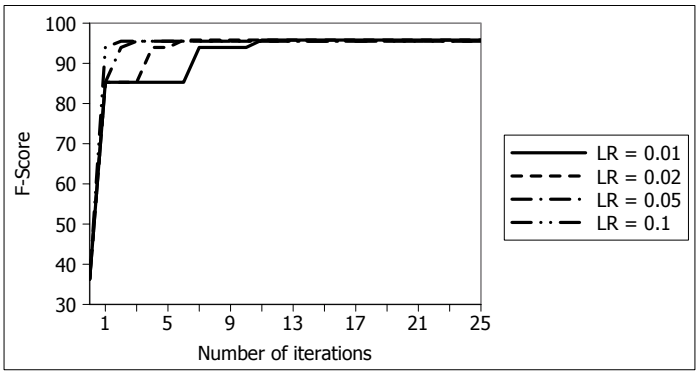
Tables 2 and 3 present an excerpt of the mappings computed automatically by RAVEN. The elements of the stable matching computed from these mapping were used as initial configuration. The results of our experiments are shown in Figures 3, 4 and 5. The first experiment, Diseases, proved to be the most difficult for RAVEN. Although the `sameAs` links between `Diseasome` and `DBpedia` allowed our experiment to run without making use of the fallback solution, we had to send 12 inquiries to the user when the learning rate was set to 0.1 to determine the best configuration that could be learned by linear and Boolean classifiers. Smaller learning rates led to the system having to send even up to 80 inquiries ($\eta = 0.01$) to determine the best configuration. In this experiment linear classifiers outperform Boolean classifiers in all setups by up to 0.8% F-score.

The second and the third experiment display the effectiveness of RAVEN. Although the fallback solution had to be used in both cases, our approach is able to detect the right configuration with an accuracy of even 100% in the Side-Effects experiment by asking the user no more than 4 questions. This is due to the linking configuration of the user leading to two well-separated sets of instances. In these cases, RAVEN converges rapidly and finds a good classifier rapidly. Note that in these two cases, all learning rates in combination with both linear and Boolean classifiers led to the same results (see Figures 4 and 5). The configuration learned for the Side-Effects experiment is shown in Figure 6.

Although we cannot directly compare our results to other approaches as it is the first active learning algorithm for learning link specifications, results reported in the database area suggest that RAVEN achieves state-of-the-art performance. The runtimes required for each iteration ensure that our approach can be used in real-world interactive scenarios. In the worst case, the user has to wait for 1.4 seconds between two iterations as shown in Figure 7. The runtime for the com-



(a) Linear classifier



(b) Boolean classifier

Fig. 3. Learning curves on Diseases experiments.

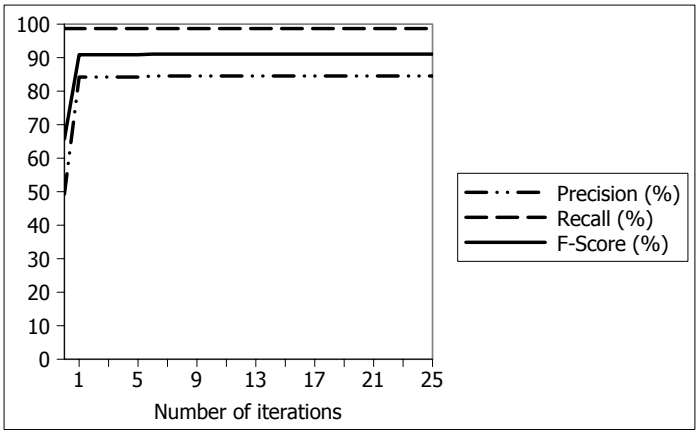


Fig. 4. Learning curve of the Drugs experiments.

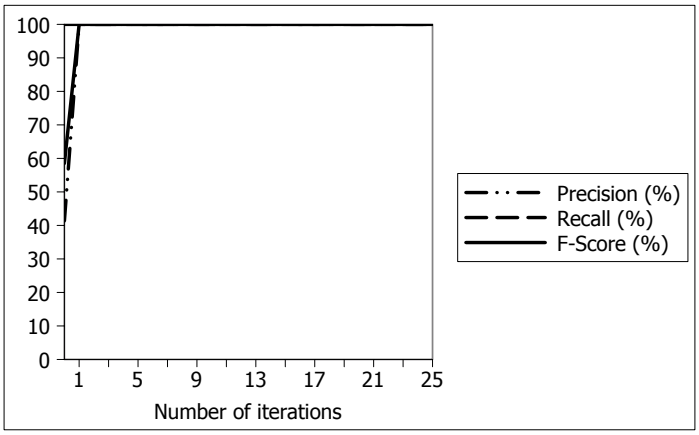


Fig. 5. Learning curve of the Side-Effect experiment.

putation of the initial configuration depends heavily on the connectivity to the SPARQL endpoints. In our experiments, the computation of the initial configuration demanded 60 seconds when `sameAs` links existed between the knowledge bases. When the fallback solution was used, the runtimes increased and reached 90 seconds.

7 Conclusion and Future Work

In this paper, we presented RAVEN, an active learning approach tailored towards semi-automatic link discovery on the Web of Data. We showed how RAVEN uses stable matching algorithms to detect initial link configurations. We opted to use the solution of the hospital residence problem ($\mathcal{HR}$) without ties because of the higher time complexity of the solution of $\mathcal{HR}$ with ties, i.e., L^4, where L is the size of the longest preference list, i.e., $max(|R|, |H|)$. Still, our work could be extended by measuring the effect of considering ties on the matching computed by RAVEN. Our experimental results showed that RAVEN can compute accurate link specifications (F-score between 90% and 100%) by asking the user to classify a very small number of positive and negative examples (between 4 and 12 for a learning rate of 0.1). Our results also showed that our approach can be used in an interactive scenario because of LIMES' time efficiency, which allowed to compute new links in less than 1.5 seconds in the evaluation tasks. The advantages of this interactive approach can increase the quality of generated links while reducing the effort to create them. The RAVEN algorithm as well as a graphical user interface will be made available as open source within the SAIM[6] framework.

In future work, we will explore how to detect optimal values for the threshold factor κ automatically, for example, by using clustering approaches. In addition,

[6] http://saim.sf.net

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
<PREFIX>
  <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
  <LABEL>rdf</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
  <LABEL>rdfs</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
  <LABEL>owl</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www4.wiwiss.fu-berlin.de/sider/resource/sider/</NAMESPACE>
  <LABEL>sd</LABEL></PREFIX>
<PREFIX>
  <NAMESPACE>http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/
        </NAMESPACE>
  <LABEL>ds:</LABEL></PREFIX>
<SOURCE>
  <ID>sider</ID>
  <ENDPOINT>http://www4.wiwiss.fu-berlin.de/sider/sparql</ENDPOINT>
  <VAR>?x</VAR>
  <PAGESIZE>-1</PAGESIZE>
  <RESTRICTION>?x rdf:type sd:side_effects</RESTRICTION>
  <PROPERTY>rdfs:label</PROPERTY>
  <PROPERTY>sd:sideEffectName</PROPERTY>
  </SOURCE>
<TARGET>
  <ID>diseasome</ID>
  <ENDPOINT>http://www4.wiwiss.fu-berlin.de/diseasome/sparql</ENDPOINT>
  <VAR>?y</VAR>
  <PAGESIZE>-1</PAGESIZE>
  <RESTRICTION>?y rdf:type ds:diseases</RESTRICTION>
  <PROPERTY>rdfs:label</PROPERTY>
  <PROPERTY>ds:name</PROPERTY>
</TARGET>
<METRIC>
ADD(1.0083130081300813*Trigram(x.rdfs:label,y.rdfs:label),
1.0083130081300813*Trigram(x.sd:sideEffectName,y.ds:name))
</METRIC>
<ACCEPTANCE>
  <THRESHOLD>1.632</THRESHOLD>
  <FILE>sideEffects.ttl</FILE>
  <RELATION>owl:sameAs</RELATION>
</ACCEPTANCE>
<REVIEW>
  <THRESHOLD>1.632</THRESHOLD>
  <FILE>sideEffectsReview.ttl</FILE>
  <RELATION>owl:sameAs</RELATION>
</REVIEW>
<EXECUTION>Simple</EXECUTION>
<GRANULARITY>4</GRANULARITY>
<OUTPUT>TURTLE</OUTPUT>
</LIMES>
```

Fig. 6. LIMES configuration learned for the Side-Effects experiment.

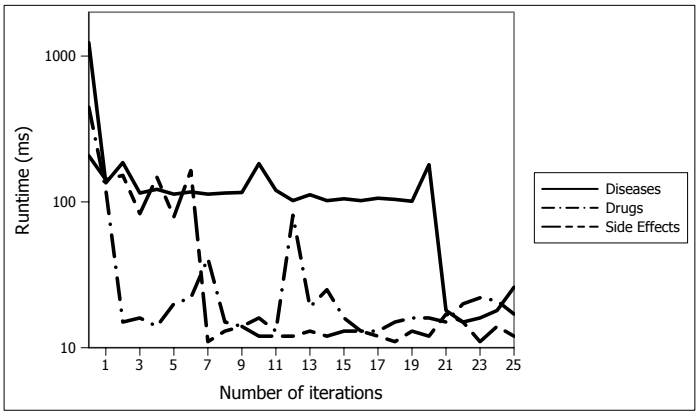


Fig. 7. Average runtimes for each iteration.

we will investigate the automatic detection of domain-specific metrics that can model the idiosyncrasies of the dataset at hand. Another promising extension to RAVEN is the automatic detection of the target knowledge base to even further simplify the linking process, since users often might not even be aware of appropriate linking targets (see [17] for research in this area). By these means, we aim to provide the first zero-configuration approach to link discovery.

References

1. A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD Conference*, pages 783–794, 2010.
2. S. Araujo, J. Hidders, D. Schwabe, de Vries, and A. P. SERIMI - resource description similarity, RDF instance matching and interlinking, July 06 2011.
3. S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData - adding a spatial dimension to the web of data. In *Proc. of 8th International Semantic Web Conference (ISWC)*, 2009.
4. Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.
5. M. Bilenko and R. J. Mooney. On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003.
6. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.
7. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 2009.
8. J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2008.
9. N. Cristianini and E. Ricci. Support vector machines. In *Encyclopedia of Algorithms*. 2008.

10. P. Cudré-Mauroux, P. Haghani, M. Jost, K. Aberer, and H. de Meer. idmesh: graph-based disambiguation of linked data. In *WWW*, pages 591–600, 2009.
11. F. Duchateau, R. Coletta, Z. Bellahsene, and R. J. Miller. (not) yet another matcher. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 1537–1540, New York, NY, USA, 2009. ACM.
12. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19:1–16, 2007.
13. J. Euzenat, A. Ferrara, C. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, V. Svátek, and C. Trojahn. Results of the ontology alignment evaluation initiative 2010. In *Fifth International Workshop on Ontology Matching*, pages 85–117, 2009.
14. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
15. H. Glaser, I. C. Millard, W.-K. Sung, S. Lee, P. Kim, and B.-J. You. Research on linked data and co-reference resolution. Technical report, University of Southampton, 2009.
16. M. Granitzer, V. Sabol, K. W. Onn, D. Lukose, and K. Tochtermann. Ontology alignment - A survey with focus on visually supported semi-automatic techniques. *Future Internet*, 2(3):238–258, 2010.
17. C. Guéret, P. Groth, F. van Harmelen, and S. Schlobach. Finding the achilles heel of the web of data: Using network analysis for link-recommendation. In *ISWC*, pages 289–304, 2010.
18. A. Hogan, A. Polleres, J. Umbrich, and A. Zimmermann. Some entities are more equal than others: statistical methods to consolidate linked data. In *NeFoRS*, 2010.
19. W. Hu, J. Chen, G. Cheng, and Y. Qu. Objectcoref & falcon-AO: results for OAEI 2010. In P. Shvaiko, J. Euzenat, F. Giunchiglia, H. Stuckenschmidt, M. Mao, and I. F. Cruz, editors, *OM*, volume 689 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
20. R. Isele and C. Bizer. Learning linkage rules using genetic programming. In P. Shvaiko, J. Euzenat, T. Heath, C. Quix, M. Mao, and I. F. Cruz, editors, *OM*, volume 814 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
21. H. Köpcke and E. Rahm. Training selection for tuning entity matching. In *QDB/MUD*, pages 3–12, 2008.
22. H. Köpcke, A. Thor, and E. Rahm. Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow.*, 2(2):1574–1577, 2009.
23. Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.
24. U. Leser and F. Naumann. *Informationsintegration - Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt.verlag, 2007.
25. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.
26. J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng*, 21(8):1218–1232, 2009.
27. D. Manlove, R. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261 – 279, 2002.
28. A. Marie and A. Gal. Boosting schema matchers. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, pages 283–300, Berlin, Heidelberg, 2008. Springer-Verlag.
29. A.-C. Ngonga Ngomo. A time-efficient hybrid approach to link discovery. In *Proceedings of OM@ISWC*, 2011.

30. A.-C. Ngonga Ngomo and S. Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of IJCAI*, 2011.
31. A.-C. Ngonga Ngomo, J. Lehmann, S. Auer, and K. Hffner. Raven – active learning of link specifications. In *Proceedings of OM@ISWC*, 2011.
32. A. Nikolov, V. S. Uren, E. Motta, and A. N. D. Roeck. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *ASWC*, pages 332–346, 2009.
33. G. Papadakis, E. Ioannou, C. Niedere, T. Palpanasz, and W. Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, 2011.
34. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
35. Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the semantic web. In *Proceedings of the 1st Workshop about Linked Data on the Web*, 2008.
36. R. Rojas. *Neural Networks: A Systematic Introduction*. Springer, 1 edition, July 1996.
37. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.
38. S. Sarawagi, A. Bhamidipaty, A. Kirpal, and C. Mouli. Alias: An active learning led interactive deduplication system. In *VLDB*, pages 1103–1106, 2002.
39. F. Scharffe, Y. Liu, and C. 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.
40. B. Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2009.
41. J. Sleeman and T. Finin. Computing foaf co-reference relations with rules and machine learning. In *Proceedings of SDoW*, 2010.
42. C. Stadler, J. Lehmann, K. Hffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, 2011.
43. C. Stadler, M. Martin, J. Lehmann, and S. Hellmann. Update Strategies for DBpedia Live. In *6th Workshop on Scripting and Development for the Semantic Web Colocated with ESWC 2010 30th or 31st May, 2010 Crete, Greece*, 2010.
44. W. R. van Hage. *Evaluating Ontology-Alignment Techniques*. PhD thesis, Vrije Universiteit Amsterdam, 2008.
45. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC*, pages 650–665, 2009.
46. W. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census - Research Report Series, 2006.