Applying edge-counting semantic similarities to Link Discovery: Scalability and Accuracy

Kleanthi Georgala[1,2], Mohamed Ahmed Sherif[2], Michael Röder[2], and Axel-Cyrille Ngonga Ngomo[1,2]

[1] Department of Computer Science, Paderborn University, Germany,
[2] Department of Computer Science, University of Leipzig, Germany
`georgala@informatik.uni-leipzig.de`
{`mohamed.sherif,michael.roeder,axel.ngonga`}`@upb.de`

Abstract. With the growth in number and variety of RDF datasets comes an increasing need for both scalable and accurate solutions to support link discovery at instance level within and across these datasets. In contrast to ontology matching, most linking frameworks rely solely on string similarities to this end. The limited use of semantic similarities when linking instances is partly due to the current literature stating that they (1) do not improve the F-measure of instance linking approaches and (2) are impractical to use because they lack time efficiency. We revisit the combination of string and semantic similarities for linking instances. Contrary to the literature, our results suggest that this combination can improve the F-measure achieved by instance linking systems when the combination of the measures is performed by a machine learning approach. To achieve this insight, we had to address the scalability of semantic similarities. We hence present a framework for the rapid computation of semantic similarities based on edge counting. This runtime improvement allowed us to run an evaluation of 5 benchmark datasets. Our results suggest that combining string and semantic similarities can improve the F-measure by up to 6% absolute.

1 Introduction

RDF knowledge graphs (KGs) are used in a plethora of applications [10], especially when published using the Linked Data paradigm. The provision of links[3] between such KGs is of central importance for numerous tasks such as federated queries [22] and question answering [25]. Popular solutions to linking instances (often called link discovery, short LD in the literature, see [12] for a survey) often implement specialized measures for particular datatypes (e.g., geospatial or temporal data). In all other cases, state-of-the-art LD frameworks such as SILK [1] and LIMES [14] rely on string similarities and machine learning to compute links between instances in RDF KGs. While the use of string similarities has been shown to work well in a large number of papers (see, e.g., [12,4]), string similarities have the major drawback of not considering the semantics of the sequences of tokens they aim to compare. Hence, most string similarity measures return low scores for pairs of strings such as (`lift`, `elevator`), (`holiday`,

[3] The fourth principal of Linked Data, see `http://www.w3.org/DesignIssues/LinkedData`

`vacation`), (`headmaster`, `principal`) and (`aubergine`, `eggplant`), although they often stand for the same real-world concepts. Edge-counting semantic similarities (e.g., [26,9,20]) alleviate this problem by using a dictionary to compute a semantic distance between sequence of tokens within the need for an overlap. The synonymy between `aubergine` and `eggplant` would hence lead semantic similarity to assign the pair (`aubergine`, `eggplant`) a similarity score close to 1.

The use of semantic similarities has been paid little attention to in LD for at least two reasons: First, *semantic similarities scale poorly* and are thus impractical when used on large knowledge graphs.[4] Moreover, current works (e.g., [11]) suggest that they lead to no improvement in F-measure. The goal of this paper is hence twofold: (1) we present means to accelerate the computation of four popular bounded edge-counting semantic similarities. (2) We then combine string and semantic similarities using two state-of-the-art machine learning approaches for LD. Our results refute the current state of the art and suggest that semantic similarities can help achieve better results in LD.

2 Preliminaries

The formal framework underlying our preliminaries is derived from [23]. A KG K is a set of triples $(s, p, o) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{I}$ is the set of all IRIs, $\mathcal{B}$ is the set of all RDF blank nodes and $\mathcal{L}$ is the set of all literals. LD frameworks aim to compute the set $M = \{(s,t) \in S \times T : R(s,t)\}$ where S and T are sets of RDF resources and R is a binary relation. Note that this setting generalizes what is often known as *entity matching* or *deduplication* [12], where the relation R must be `owl:sameAs`. Given that M is generally difficult to compute directly, declarative LD frameworks compute an approximation $M' \subseteq S \times T$ of M by executing a *link specification* (LS), which we define formally in the following. Let $\mathbb{M}$ be the set of all similarity functions. We define a similarity function $m \in \mathbb{M}$ as a function $m : S \times T \times \mathcal{P}^2 \rightarrow [0,1]$, where $\mathcal{P}$ is the set of all properties, where $p_s, p_t \in \mathcal{P}$. We write $m(s, t, p_s, p_t)$ to signify the similarity of s and t w.r.t. their properties p_s resp. p_t. An *atomic LS* L is a pair $L = ((m(p_s, p_t), \theta)$, where $\theta \in [0,1]$ is a similarity threshold. A *complex LS* L is a tuple $L = op(L_1, L_2)$ where two subspecification L_1 and L_2 are combined using the specification operator *op*. Here, we consider the binary operators union ($\sqcup$), intersection ($\sqcap$) and difference ($\backslash$).

The edge-counting semantic similarities are based on a lexical vocabulary. We define a *lexical vocabulary* as a directed acyclic graph (DAG) $G = (V, E)$, where:

- The set of vertices V is a set of concepts c_i, were each c_i stands for a set of synonyms. We denote $|V|$ with n_V.
- $E \subseteq V \times V$ is a set of directed edges $e_{jk} = (c_j, c_k)$. We denote $|E|$ with n_E.
- The *edge* e_{jk} stands for the hypernymy relation from a parent concept c_j to a child concept c_k. We write $c_j \rightarrow c_k$ and we say that c_j is a hypernym of c_k. We also define the hyponymy relation as a directed relation from a child concept c_k to a parent concept. We write $c_j \leftarrow c_k$ and we say that c_j is a hyponym of c_k. Hypernymy and hyponymy are transitive.

[4] This general finding is supported by our evaluation results presented in Section 4.

- The *root* r is the unique node of the dictionary that has no parent concept.
- A *leaf concept* c_i is a concept node without any children concepts.
- A concept is a *common subsumer* of c_1 and c_2 (denoted $cs(c_1, c_2)$) iff that concept is a hypernym of both c_1 and c_2.
- The *least common subsumer* (LSO) of c_1 and c_2 (denoted $lso(c_1, c_2)$) is "the most specific concept which is an ancestor of both c_1 and c_2" [26].
- We define the *directed path* from c_1 to c_2 via a common subsumer $cs(c_1, c_2)$ as: $path(c_1, c_2) = \{c_1 \leftarrow c_i \leftarrow \ldots \leftarrow cs(c_1, c_2) \rightarrow c_j \rightarrow \ldots \rightarrow c_2 : i, j, k \in \mathbb{N}, i, j, k \leq n_v\}$. Note that there can be multiple $path(c_1, c_2)$ between two concepts.
- $len(c_1, c_2)$ is *the length of the shortest* $path(c_1, c_2)$ between two concepts c_1 and c_2. Note that len defines a metric. Hence, it is symmetric and abides by the triangle inequality, i.e., $len(c_1, c_2) \leq len(c_1, c_3) + len(c_2, c_3)$ for any $(c_1, c_2, c_3) \in V^3$.
- We define $depth_m(c_i)$ as the length of the shortest path between r and c_i. Analogously, $depth_M(c_i)$ as the maximum $depth(c_i)$. We set $D = \max\limits_{c \in V} depth_M(c)$.

Note that the following holds:

- $depth_m(r, c_i) = len(r, c_i)$
- $depth_m(lso(c_1, c_2)) \leq min(depth_m(c_1), depth_m(c_2))$
- $depth_M(lso(c_1, c_2)) \leq min(depth_M(c_1), depth_M(c_2))$
- (triangle inequality) $|len(r, c_1) - len(r, c_2)| \leq len(c_1, c_2) \Leftrightarrow |depth_m(c_1) - depth_m(c_2)| \leq len(c_1, c_2)$

The Shortest Path (SP) similarity [20] of two concepts c_1 and c_2 is defined as the length of their shortest path in comparison to the maximum distance ($2D$). We use the normalized formulation of SP, i.e.,

$$\text{SP}(c_1, c_2) = \frac{2D - len(c_1, c_2)}{2D}. \tag{1}$$

The Leacock and Chodorow metric (LCH) takes both the path between two concepts and the depth of the hierarchy into consideration [8]. We use the normalized formulation of LCH:

$$\text{LCH}_N(c_1, c_2) = \begin{cases} 1 & \text{if } c_1 = c_2 \\ \frac{-\log\left(\frac{len(c_1, c_2)}{2D}\right)}{\log(2D)} & \text{else.} \end{cases} \tag{2}$$

The normalized Wu Palmer (WP) similarity takes the path between two concepts and the depth of their LSO into consideration [26]:

$$\text{WP}(c_1, c_2) = \frac{2 \times depth_M(lso(c_1, c_2))}{2 \times depth_M(lso(c_1, c_2)) + N_1 + N_2} \tag{3}$$

where $N_1 = len(lso(c_1, c_2), c_1)$ and $N_2 = len(lso(c_1, c_2), c_2)$. The Li et al. metric (LI) is another take on using the path between two concepts and their LSO to define a similarity [9]:

$$\text{LI}(c_1, c_2) = e^{-\alpha len(c_1, c_2)} \frac{e^{\beta depth(lso(c_1, c_2))} - e^{-\beta depth(lso(c_1, c_2))}}{e^{\beta depth(lso(c_1, c_2))} + e^{-\beta depth(lso(c_1, c_2))}} \tag{4}$$

where $\text{LI}(c_1, c_2) \in (0, 1)$. We set $depth(lso(c_1, c_2)) = depth_M(lso(c_1, c_2))$, since the original specification does not state which $depth(lso(c_1, c_2))$ to use.

3 Approach

Fundamentally, hECATE aims to compute the set $M' = \{(s,t) \in S \times T : m(s,t,p_s,p_t) \geq \theta\}$, where m is an edge-counting similarity. To achieve this goal, the approach makes use of upper bounds which can be derived from the formulation of this family of measures. Take the SP similarity for example: For any two concepts c_1 and c_2, SP$(c_1, c_2) \geq \theta$ implies $len(c_1, c_2) \leq 2D(1-\theta)$. Formally, this means that we can discard all comparisons of pairs (c_1, c_2) with $len(c_1, c_2) > 2D(1-\theta)$ without compromising the computation of M'. Note that the computation of $len(c_1, c_2)$ can be carried out online or offline, which affects the total runtime of our approach as discussed in Section 4. As similar bounds can be derived for the other edge-counting measures, hECATE generalizes the computation of M' for edge-counting semantic similarities by using the following algorithm. Our approach takes (1) two sets of resources, S and T, (2) an atomic LS $L = ((m(p_s, p_t), \theta)$, where m is one of the four semantic similarities described in Section 2, and (3) a lexical vocabulary structured as DAG (VDAG) as input. Our goal is to compute the mapping $M' = [[L]]$ For each pair (s,t), hECATE retrieves and pre-processes the property values for p_s resp. p_t. The pre-processing consists of tokenizing and extracting all stop-words from the objects of the triples (s, p_s, o_s) and (t, p_t, o_t). In order to include a pair (s,t) in M', the algorithm compares each set of source tokens from o_s ($sTokens$) to each set of target tokens of o_t ($tTokens$). The pair of objects (o_s, o_t) with the highest similarity which abides by the bounds we derive for each measure is finally used to compute the similarity between s and t, and decides whether or not this pair should be added to M'. To do so, for each token $sToken \in sTokens$, we find the $tToken \in tTokens$ that is most similar. First, the algorithm checks if $sToken$ and $tToken$ have been compared before If the tokens are being compared for the first time the algorithm checks if the tokens are equal and assigns the value of 1 to $TTSim$. Otherwise, it calls the function `compare(sToken, tToken, VDAG)` that compares the corresponding sets of concepts obtained from the input `VDAG`. [5] Then, $TTSim$ is compared to the maximum token-to-token similarity and $maxTTSim$ is updated. The procedure continues until the highest similarity between the current $sToken$ and a $tToken$ is found or $maxTTSim$ is equal to 1. The algorithm aggregates the highest similarities $maxTTSim$ of all $sToken \in sTokens$ and calculates an average similarity. This is done for all pairs of ($sTokens$, $tTokens$) searching for the pair with the maximum similarity. If this $maxSimilarity > \theta$ the pair (s,t) can be added to the final mapping M'. The key behind hECATE lies in the token comparison algorithm `compare(sToken, tToken, VDAG)` (Algorithms 1 and 3). For a pair of tokens `(sToken, tToken)`, we retrieve the set of concepts they belong to in the `VDAG`. If both sets of concepts are not empty, we compare each source `sCon` with each target concept `tCon` and define the maximum similarity of two tokens as the highest similarity of the corresponding concept pairs. To do so, we first retrieve the set of all hypernym paths of each concept to the root of the `VDAG` using the `getPaths(concept, VDAG)` algorithm. This algorithm traverses the `VDAG` by utilizing the hypernym relation. It starts from the *concept* node and explores all paths to

[5] Note that our algorithm handles homonyms by considering that a token can be included in more than one concept.

the root node.For SP and LCH, we additionally retrieve the maximum depth D found in the `VDAG` and the `len(sCon, tCon)` before calculating the corresponding similarity as described in Equations 1 and 2 resp. For calculating `len(sCon, tCon)` our algorithm relies on the set of hypernym paths of the concepts (Algorithm 2). For each pair of hypernym paths hp_1 and hp_2 the two concepts have, the algorithm iterates over both paths simultaneously, from top to bottom, until they do not share a common node. Then, it proceeds in calculating the length of the newly found path, as the number of concepts that the two paths do not have in common. Finally, the minimum length that has been found is returned. For WP and LI, the comparison algorithm retrieves the depth of the LSO between `sCon` and `tCon` (`depth( lso(sCon, tCon))`), and N_1 and N_2 by calling the function `getLSO` (hps_1, hps_2) (Algorithm 4). This function utilizes the set of hypernym paths in a similar manner as the min length algorithm. For each combination of hypernym paths hp_1 and hp_2 of the concepts, the algorithm traverses them simultaneously searching for the last node they have in common. If this node is deeper than any other common node found so far or it has the same depth but the remaining paths are shorter, it is taken as new LSO. Accordingly, the remaining path lengths N_1 and N_2 are updated. Based on the deepest LSO and the derived values for `depth`$_M$ `(lso(sCon, tCon)`, N_1 and N_2, we proceed in calculating the corresponding similarity as described in Equations 3 and 4 resp.

Our first extension of hECATE is based on the idea of pre-computing and storing a set of values that are used often in our algorithm. For edge-counting similarities, these are the hypernym paths. Consequently, the extension hECATE-I of hECATE precomputes all hypernym paths for all concepts included in the `VDAG`, using the `getPaths( concept, VDAG)` function. Therefore, every time the `getPaths(concept, VDAG)` is invoked at runtime, hECATE-I retrieves the paths from an index. Our second extension of hECATE, hECATE-IF, combines hECATE-I with the idea of minimizing unnecessary comparison between concepts by filtering out pairs of source and target concepts that do not satisfy a condition for each semantic similarity. The filtering is performed inside `compare(sToken, tToken, VDAG)` for each pair of concepts `sCon` and `tCon`. Given a semantic similarity, if a pair of concepts satisfies the corresponding filtering condition, then the algorithm proceeds normally as described before. If the condition is not met the algorithm does not compute the similarity between the two concepts. For the SP similarity, two concepts will be considered for comparison, if the following holds:

$$\begin{aligned} \text{SP}(c_1, c_2) \geq \theta &\Leftrightarrow \frac{2D - len(c_1, c_2)}{2D} \geq \theta \\ &\Rightarrow |depth_m(c_1) - depth_m(c_2)| \leq 2D(1-\theta) \end{aligned} \tag{5}$$

For the WP similarity, the following must hold:

$$\begin{aligned} WU(c_1, c_2) \geq \theta &\Leftrightarrow \frac{2depth_M(lso(c_1, c_2))}{2depth_M(lso(c_1, c_2)) + N_1 + N_2} \geq \theta \\ &\Leftrightarrow 2depth_M(lso(c_1, c_2)) \geq \theta(N_1 + N_2) + 2\theta depth_M(lso(c_1, c_2)) \\ &\Leftrightarrow N_1 + N_2 \leq \frac{2depth_M(lso(c_1, c_2))(1-\theta)}{\theta} \\ &\Rightarrow N_1 + N_2 \leq \frac{2\min(depth_M(c_1), depth_M(c_2))(1-\theta)}{\theta} \end{aligned} \tag{6}$$

Based on the triangle inequality and Section 2, Equation 6 can be written as:

Algorithm 1: $compare(sCon, tCon, VDAG)$ for SP or LCH

```
Input: source concept sCon, target concept tCon, and a vocabulary DAG VDAG
Output: a $similarity$ value
1 $D \leftarrow VDAG.getMaxDepth(sCon)$
2 $hps_1 \leftarrow getPaths(sCon, VDAG)$
3 $hps_2 \leftarrow getPaths(tCon, VDAG)$
4 $minLength \leftarrow getMinLength(hps_1, hps_2)$
5 Return $computeSimilarity(D, minLength)$
```

Algorithm 2: $getMinLength(hps_1, hps_2)$

```
Input: two sets of hypernym paths, $hps_1$ and $hps_2$
Output: $len(sCon, tCon)$
1 $size \leftarrow MAX\_VALUE$
2 foreach $hp_1 \in hps_1$ do
3     foreach $hp_2 \in hps_2$ do
4         $l_1 \leftarrow 0$, $l_2 \leftarrow 0$
5         while $l_1 < hp_1.size() \wedge l_2 < hp_2.size() \wedge hp_1.get(l_1) == hp_2.get(l_2)$ do
6             $l_1 \leftarrow l_1 + 1$, $l_2 \leftarrow l_2 + 1$
7         $newSize \leftarrow hp_1.size() + hp_2.size() - 2l_1$
8         if $newSize < size$ then
              $size \leftarrow newSize$ ;
9 Return $size$
```

Algorithm 3: $compare(sCon, tCon, VDAG)$ for WP or LI

```
Input: source concept sCon, target concept tCon, and a vocabulary DAG VDAG
Output: a $similarity$ value
1 $hps_1 \leftarrow getPaths(sCon, VDAG)$
2 $hps_2 \leftarrow getPaths(tCon, VDAG)$
3 $depth, N_1, N_2 \leftarrow getLSO(hps_1, hps_2)$
4 Return $computeSimilarity(N_1, N_2, depth)$
```

Algorithm 4: $getLSO(hps_1, hps_2)$

```
Input: two sets of hypernym paths, $hps_1$ and $hps_2$
Output: $depth_M(lso(sCon, tCon))$, $N_1$ and $N_2$
1 $dLSO \leftarrow 0$, $N_1 \leftarrow 0$, $N_2 \leftarrow 0$
2 foreach $hp_1 \in hps_1$ do
3     foreach $hp_2 \in hps_2$ do
4         $l_1 \leftarrow 0$, $l_2 \leftarrow 0$
5         while $l_1 < hp_1.size() \wedge l_2 < hp_2.size() \wedge hp_1.get(l_1) == hp_2.get(l_2)$ do
6             $l_1 \leftarrow l_1 + 1$, $l_2 \leftarrow l_2 + 1$
7         $newSize \leftarrow hp_1.size() + hp_2.size() - 2l_1$
8         $oldSize \leftarrow N_1 + N_2$
9         if condition is met then
10            $dLSO \leftarrow l_1$, $N_1 \leftarrow hp_1.size() - l_1$
11            $N_2 \leftarrow hp_2.size() - l_2$
12 Return $dLSO, N_1, N_2$
```

$$\begin{aligned} len(c_1, c_2) &\leq \frac{2\min(depth_M(c_1), depth_M(c_2))(1-\theta)}{\theta} \Rightarrow \\ |depth_m(c_1) - depth_m(c_2)| &\leq \frac{2min(depth_M(c_1), depth_M(c_2))(1-\theta)}{\theta} \end{aligned} \tag{7}$$

For the LCH similarity, two concepts will be considered for comparison, iff:

$$\begin{aligned} \text{LCH}(c_1, c_2) \geq \theta \Leftrightarrow \frac{-log\frac{len(c_1,c_2)}{2D}}{log(2D)} \geq \theta \Leftrightarrow \frac{log(2D) - log(len(c_1, c_2))}{log(2D)} \geq \theta \Leftrightarrow \\ 1 - \frac{log(len(c_1, c_2))}{log(2D)} \geq \theta \Leftrightarrow log(len(c_1, c_2)) \leq log(2D)(1-\theta) \Leftrightarrow \\ len(c_1, c_2) \leq 2^{log(2D)(1-\theta)} \Rightarrow |depth_m(c_1) - depth_m(c_2)| \leq 2^{log(2D)(1-\theta)} \end{aligned} \tag{8}$$

When considering the LI similarity, we make the following variable replacements for the sake of legibility: $x = depth_M(lso(c_1, c_2))$, $y = min(depth_M(c_1), depth_M(c_2))$ and $z = len(c_1, c_2)$. Then, two concepts will be considered for comparison, iff:

$$\textsc{Li}(c_1,c_2) \geq \theta \Leftrightarrow e^{-\alpha z}\frac{e^{\beta x}-e^{-\beta x}}{e^{\beta x}+e^{-\beta x}} \geq \theta \Leftrightarrow e^{\alpha z} \leq \frac{e^{\beta x}-e^{-\beta x}}{(e^{\beta x}+e^{-\beta x})\theta} \Leftrightarrow e^{\alpha z} \leq \frac{\frac{(e^{2\beta x}-1)}{e^{\beta x}}}{\frac{(e^{2\beta x}+1)}{e^{\beta x}}\theta} \Leftrightarrow$$

$$e^{\alpha z} \leq \frac{(e^{2\beta x}-1)}{(e^{2\beta x}+1)\theta} \Rightarrow e^{\alpha z} \leq \frac{(e^{2\beta y}-1)}{(e^{2\beta y}+1)\theta} \Leftrightarrow \alpha z \leq ln(e^{2\beta y}-1) - ln\theta - ln(e^{2\beta y}+1) \Leftrightarrow$$

$$|depth_m(c_1) - depth_m(c_2)| \leq \frac{ln(e^{2\beta y}-1) - ln\theta - ln(e^{2\beta y}+1)}{\alpha} \tag{9}$$

Based on Equations 5, 7, 8 and 9, each filtering condition requires the knowledge of $depth_m(sCon)$, $depth_M(sCon)$, $depth_m(tCon)$ and $depth_M(tCon)$. Hence, we further extend the index hECATE-IF relies on by precomputing $depth_m(c_i)$ and $depth_M(c_i)$ for every concept c_i.

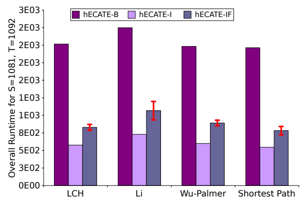
4 Evaluation

Our evaluation addresses the following three research questions: Q_1. How do our strategies for improving the runtime of semantic similarities compare to each other w.r.t. runtime?, Q_2. How do the different edge-counting semantic similarities compare w.r.t. runtime?, and Q_3. Can semantic similarities improve the F-measure of LD systems?

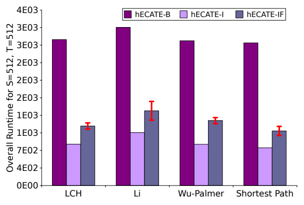
We evaluate our approach against five benchmark data sets: Abt-Buy, Amazon-GP and DBLP-ACM described in [7], DailyMed-Drugbank (dubbed DM-DB) and Movies described in [16]. We use WordNet[6] as a DAG. To address Q_1 and Q_2, we conduct a set of experiments using the basic hECATE algorithm (dubbed hECATE-B) as a baseline as well as hECATE-I and hECATE-IF. For an easier comparison, all methods are implemented in the LD framework LIMES [14]. For hECATE-B and hECATE-I, we create one atomic LS for each semantic similarity, where m iss the name of the edge-counting similarity, $\theta = 0.1$. We use the `'description'` as the source and target properties for *Abt-Buy* and *Amazon-GP* datasets, `'title'` for the *DBLP-Scholar* and *Movies* datasets and `'name'` for the *DM-DB* dataset. For hECATE-IF, we use the same values for m, p_s and p_t as before, but θ is derived from the interval $[0.1, 1]$ with an increment step of 0.1, since the θ is given as a parameter to the filtering functions. For each dataset, we perform the aforementioned LSs against 2^v instances from the source and target datasets. We start with $v = 2$ and increment v until all instances are covered (e.g., the maximal value of v is 9 for the Amazon-Google dataset). We define a maximum runtime for each LS of 2 hrs. Each experiment is executed 3 times and we present the average values.

As explained in Section 1, the second goal of this work is to evaluate edge-counting semantic similarities in LD in terms of accuracy. Consequently, for Q_3, we use the hECATE extension with the best runtime performance based on the results of Q_1 and executed a set of experiments using 2 machine learning (ML) algorithms: WOMBAT [23] and DRAGON [19]. We choose these two approaches because (1) they achieve state-of-the-art performance while being deterministic, (2) they are open-source, meaning our experiments can be easily reproduced and (3) they are able to generate complex link specifications with any arbitrary number of measures. We perform a 10-fold cross

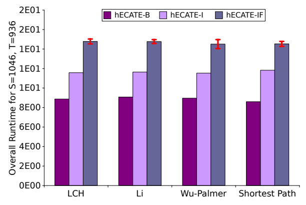
[6] `https://wordnet.princeton.edu/`



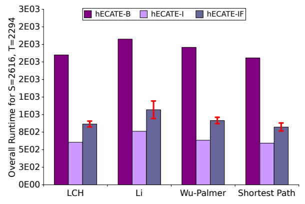
(a) Abt-Buy



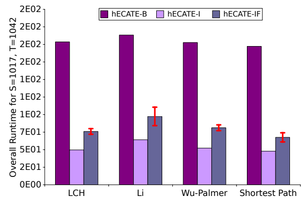
(b) Amazon-GP



(c) DM-DB



(d) DBLP-ACM



(e) Movies

Fig. 1: Average runtime in seconds of hECATE-B, hECATE-I and hECATE-IF on all datasets. For hECATE-IF, the standard deviation among different θ values is added.

validation by allowing WOMBAT and DRAGON to use only string similarities (StrSim), only semantic similarities (SmtSim) and a combination of both (StrSmtSim) as input. We use the `levenshtein`, `cosine` and `qgrams` similarity measures for strings implemented in LIMES [14]. For each dataset, we use all properties apart from those that corresponded to numeric values. WOMBAT is configured as presented in [23] and DRAGON is configured as presented in [19]. We use two termination criteria for WOMBAT: Either a LS with F-measure of 1 is found or a maximal depth of refinement of 10 is reached. For the string similarities, WOMBAT produced LSs with a minimum θ value of 0.4 and for the semantic similarities, the minimum θ value is set to 0.7. DRAGON terminates either when no new nodes are found or when the height of the decision tree reached the maximum of 3. Additionally, we compare the achieved F1 scores with scores for EAGLE [15], EUCLID [13], J48 [5] reported by [19], a Multilayer Perceptron classifier reported by [24] and the *Pessimistic* as well as *Re-weighted* versions of the work presented at [6].

As expected, Figure 1 shows that hECATE-B has the highest runtimes compared to hECATE-I and hECATE-IF in all datasets, except DM-DB. This supports the claim that semantic similarities typically scale poorly. The results show that both extensions improve the runtime of all semantic similarities, making them more amenable for LD and scalable for larger datasets. Precisely, LCH's, WP's and SP's runtimes improve by 71% and 57% on average when hECATE-I and hECATE-IF strategies are used resp. LI has the least improvement by 65% and 50%. Comparing the two extensions, in all datasets and for all semantic similarities, hECATE-I outperforms hECATE-IF by 30% on average. A detailed analysis of the runtimes shows that even though hECATE-IF reduces the number of comparisons between semantically different concepts and thus the comparison time, the additional runtime cost of filtering creates an overhead that re-

Table 1: Number of concept comparisons performed by hECATE-I and hECATE-IF for the Movies dataset. The numbers for hECATE-B are the same as for hECATE-I.

Threshold		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
SP	hECATE-I	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M
	hECATE-IF	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	**61.0M**	**51.4M**	**10.3M**
WP	hECATE-I	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M
	hECATE-IF	61.8M	61.8M	61.8M	**61.7M**	**61.5M**	**60.3M**	**56.7M**	**44.7M**	**27.8M**	**10.3M**
LCH	hECATE-I	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M	61.8M
	hECATE-IF	61.8M	61.8M	61.8M	**61.4M**	**60.4M**	**56.5M**	**42.4M**	**42.4M**	**28.5M**	**28.5M**
LI	hECATE-I	61.9M	61.9M	61.9M	61.9M	61.9M	61.9M	61.9M	61.9M	61.9M	61.9M
	hECATE-IF	61.9M	**61.4M**	**59.9M**	**56.6M**	**51.5M**	**42.1M**	**28.3M**	**28.2M**	**10.0M**	**00.0M**

Table 2: Average F-measure achieved by WOMBAT, DRAGON, EUCLID, EAGLE, J48 and Multilayer Perception within a 10-fold cross validation. The semantic similarities use the hECATE-I strategy.

Algorithm	WOMBAT			DRAGON			EUCLID	EAGLE	J48	Perceptron
Similarities	StrSim	SmtSim	StrSmtSim	StrSim	SmtSim	StrSmtSim	StrSim	StrSim	StrSim	StrSim
Abt-Buy	0.65	0.65	**0.66**	0.51	0.02	0.10	0.00	0.56	0.43	0.43
Amazon-GP	0.71	0.60	**0.77**	0.64	0.06	0.05	0.71	0.73	0.41	0.36
DBLP-ACM	0.97	0.74	0.97	0.93	0.81	0.93	**0.98**	**0.98**	0.77	0.97
DM-DB	0.94	0.71	0.97	0.89	0.65	0.89	**1.00**	**1.00**	0.94	-
Movies	1.00	0.73	**1.00**	0.93	0.80	0.93	0.98	0.99	0.84	-

sults in a worse total execution time than hECATE-I (Table 1). Regarding the DM-DB dataset, the only property for both source and target datasets, *name*, consists of only one value, which corresponds to the official name of a drug. That value can only be associated with one concept. As a result, introducing an indexing and/or filtering technique produces an unnecessary overhead. Overall, Q_1 can be answered with hECATE-I being the most efficient approach.

To answer Q_2 we compare the runtimes of the single semantic similarities revealing that LI has the worst runtime (see Figure 1). For the Movies dataset, we notice that hECATE-I requires 100K more token comparisons for LI compared to the other similarities (Table 1). The better runtime of the other similarities is caused by a condition inside our algorithm which stops as soon as two tokens/concepts have a similarity of 1. In contrast to the other similarities, $\text{LI}(c_1, c_2) \in (0, 1)$, i.e., it can never be 1. However, based on Table 1, LI's runtime shows a great improvement as the values of θ increase in relation to the other metrics. This justifies the fact that the runtimes for LI have the highest standard deviation, whereas SP, LCH and WP are less influenced by the different values of θ. The answer for Q_2 is that for all hECATE strategies, SP is the fastest similarity, whereas LI is the slowest.

To answer Q_3, we add the 4 edge-counting measures LI, WP, SP, and LCH to the state-of-the-art algorithms WOMBAT [23] and DRAGON [19]. We evaluate their performance with and without string similarities using a ten-fold cross validation. Table 2

Table 3: Maximum F-measure achieved by WOMBAT, DRAGON, Pessimistic and Re-weighted using 2% of the data for training over 7 iterations [6]. The semantic similarities use the hECATE-I strategy.

Algorithm	WOMBAT			DRAGON			Pessimistic	Re-weighted
Similarities	StrSim	SmtSim	StrSmtSim	StrSim	SmtSim	StrSmtSim	StrSim	StrSim
Abt-Buy	0.35	**0.39**	0.34	0.24	0.10	0.24	0.36	0.37
Amazon-GP	**0.53**	0.33	0.43	0.45	0.13	0.35	0.39	0.43
DBLP-ACM	0.91	0.55	0.91	0.90	0.66	0.90	0.93	**0.95**
DM-DB	0.94	0.71	**0.97**	0.94	0.71	0.96	-	-
Movies	0.97	0.33	**0.97**	0.96	0.33	0.96	-	-

shows the results of our experiments with these machine-learning algorithms. In the 6 right most columns of Table 2, we report the F1 score of the string-based LD algorithms. While the performance of DRAGON remained the same or even worsened for 3 of the 5 datasets, adding semantic similarities to the WOMBAT algorithm improved its overall performance for 3 datasets by up to 6% F-measure absolute. As expected, this effect is most pronounced in datasets which rely on long textual descriptions such as Amazon-GP. A look into the specifications learned by WOMBAT suggests that this effect is due to the approach combining semantic and string similarities using operators such as $\sqcup$ and learning the correct threshold for each of these measures. The improvement on the DM-DB datasets is achieved using the $\setminus$ operator, not allowing semantically similar concepts to be matched together. This refutes current results (see [11] where the same similarities have been used) and suggests that the refinement operators can combine semantic and string similarities in a way that improves the F-measure. For enabling a comparison with [6], we used the same configuration setting and report the maximum F-measure in Table 3. It can be seen that WOMBAT outperforms the Pessimistic and Re-weighted methods on the majority of the datasets.

5 Related Work

We give a brief overview of linking approaches which use semantic similarities. An exhaustive list of frameworks can be found in [12]. Over the past few years, semantic similarities were used in ontology matching (OM) [21]. In this context, concepts in two ontologies O_1 and O_2 are often matched based on a third ontology, e.g., WordNet. This ontology can be viewed as a background knowledge source or a mediating ontology [2]. Frameworks such as *AgreementMaker* [3], *Zhishi.links* [18] and *RuleMiner* [17] utilize semantic similarities in this way to improve structural matching on the ontology level. While these enhancements have a positive effect on their instance level matching, to the best of our knowledge no instance linking tool has used semantic similarities directly and shown an improvement of the overall linking results. [11] compare the effect of a predefined set of combinations of string and semantic similarities for label comparison and suggest that semantic similarities do not improve the F-measure of the instance matching task. Our results suggest the contrary by showing that dataset-specific combinations of measures actually can achieve a better performance.

6 Conclusions and Future Work

To study the effect of semantic similarities on LD, we presented hECATE, a generic framework for improving the runtime of edge-counting semantic similarities. Our evaluation of the framework shows that there is still a lot of potential in improving the runtime of semantic similarities for LD. We used hECATE to evaluate the performance of string similarities in LD on five datasets. Our evaluation shows that combining semantic similarities with string similarities can indeed increase the F-measure achieved by LD algorithms. This result is of central importance as it goes against current assumptions. The reason why we are indeed able to use semantic similarities for improving the F-measure of LD in some cases lies in the refinement operator employed by WOMBAT. In future works, we will investigate means that will allow improving the runtimes of semantic similarities, extend our works beyond edge-counting similarities and aim to classify datasets w.r.t. how suitable they are for semantic similarities.

Acknowledgments

This work has been supported by the EU H2020 project KnowGraphs (GA no. 860801) as well as the BMVI projects LIMBO (GA no. 19F2029C) and OPAL (GA no. 19F2028A).

References

1. Bizer, C., Volz, J., Kobilarov, G., Gaedke, M.: Silk - A Link Discovery Framework for the Web of Data. In: 18th International World Wide Web Conference (April 2009)
2. Cross, V., Silwal, P., Morell, D.: Using a reference ontology with semantic similarity in ontology alignment. In: Proceedings of the 3rd ICBO (2012)
3. Cruz, I.F., Antonelli, F.P., Stroe, C.: Agreementmaker: Efficient matching for large real-world schemas and ontologies. PVLDB **2**, 1586–1589 (2009)
4. Euzenat, J., Ferrara, A., Meilicke, C., Nikolov, A., Pane, J., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Šváb-Zazamal, O., Svátek, V., et al.: Results of the ontology alignment evaluation initiative 2010. Tech. rep., University of Trento (2011)
5. Holmes, G., Donkin, A., Witten, I.H.: Weka: a machine learning workbench. In: Proceedings of ANZIIS '94. pp. 357–361 (Nov 1994)
6. Kejriwal, M., Miranker, D.P.: Semi-supervised instance matching using boosted classifiers. In: The Semantic Web. Latest Advances and New Domains. pp. 388–402. Springer International Publishing (2015)
7. Köpcke, H., Thor, A., Rahm, E.: Evaluation of Entity Resolution Approaches on Real-world Match Problems. Proc. VLDB Endow. **3**(1-2), 484–493 (Sep 2010)
8. Leacock, C., Chodorow, M.: Combining local context and wordnet similarity for word sense identification (01 1998)
9. Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans. on Knowl. and Data Eng. **15**(4), 871–882 (Jul 2003)
10. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In: International Semantic Web Conference. pp. 376–394 (2018)

11. McCrae, J.P., Buitelaar, P.: Linking Datasets Using Semantic Textual Similarity. CYBERNETICS AND INFORMATION TECHNOLOGIES **18**(1), 109–123 (2018)
12. Nentwig, M., Hartung, M., Ngonga Ngomo, A.C., Rahm, E.: A survey of current link discovery frameworks. Semantic Web pp. 1–18 (2015)
13. Ngomo, A.C.N., Lyko, K.: Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: OM (2013)
14. Ngonga Ngomo, A.C.: On Link Discovery using a Hybrid Approach. Journal on Data Semantics **1**(4), 203–217 (2012)
15. Ngonga Ngomo, A.C., Lyko, K.: Eagle: Efficient active learning of link specifications using genetic programming. In: The Semantic Web: Research and Applications. pp. 149–163. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
16. Ngonga Ngomo, A.C., Lyko, K.: Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: Proceedings of the Ontology Matching Workshop (2013)
17. Niu, X., Rong, S., Wang, H., Yu, Y.: An effective rule miner for instance matching in a web of data. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management. pp. 1085–1094. CIKM '12, ACM, New York, NY, USA (2012)
18. Niu, X., Rong, S., Zhang, Y., Wang, H.: Zhishi.links results for OAEI 2011. Ontology Matching p. 220 (2011)
19. Obraczka, D., Ngomo, A.C.N.: Dragon: Decision tree learning for link discovery. In: 19TH International Conference On Web Engineering. Springer (2019)
20. Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE Trans. Systems, Man, and Cybernetics **19**, 17–30 (1989)
21. Rodríguez, M.A., Egenhofer, M.J.: Determining semantic similarity among entity classes from different ontologies. IEEE Trans. on Knowl. and Data Eng. **15**(2), 442–456 (2003)
22. Saleem, M., Ali, M.I., Verborgh, R., Ngonga Ngomo, A.C.: Federated query processing over linked data. In: Tutorial at ISWC (2015)
23. Sherif, M., Ngonga Ngomo, A.C., Lehmann, J.: WOMBAT - A Generalization Approach for Automatic Link Discovery. In: 14th Extended Semantic Web Conference, Portorož, Slovenia, 28th May - 1st June 2017 (2017)
24. Soru, T., Ngomo, A.C.N.: A comparison of supervised learning classifiers for link discovery. In: Proceedings of the 10th Intern. Conf. on Semantic Systems. pp. 41–44. ACM (2014)
25. Usbeck, R., Ngonga Ngomo, A.C., Haarmann, B., Krithara, A., Röder, M., Napolitano, G.: 7th open challenge on question answering over linked data (QALD-7). In: Semantic Web Evaluation Challenge. pp. 59–69. Springer International Publishing (2017)
26. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics. pp. 133–138. ACL '94, Association for Computational Linguistics, Stroudsburg, PA, USA (1994)