A Scalable Approach for Computing Semantic Relatedness using Semantic Web Data

Conference Paper · June 2016

DOI: 10.1145/2912845.2912864

4 authors, including:

Dennis Diefenbach
Université Jean Monnet
16 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Ricardo Usbeck
Universität Paderborn
44 PUBLICATIONS **237** CITATIONS
SEE PROFILE

Pierre Maret
Université Jean Monnet
112 PUBLICATIONS **372** CITATIONS
SEE PROFILE

A Scalable Approach for Computing Semantic Relatedness using Semantic Web Data

Dennis Diefenbach
Université de Lyon, CNRS
UMR 5516 Laboratoire Hubert Curien
Saint-Etienne, France
dennis.diefenbach@univ-st-etienne.fr

Ricardo Usbeck
Leipzig University
Leipzig, Germany
usbeck@informatik.uni-leipzig.de

Kamal Deep Singh
Université de Lyon, CNRS
UMR 5516 Laboratoire Hubert Curien
Saint-Etienne, France
kamal.singh@univ-st-etienne.fr

Pierre Maret
Université de Lyon, CNRS UMR 5516 Laboratoire Hubert Curien
Saint-Etienne, France
pierre.maret@univ-st-etienne.fr

ABSTRACT

Computing semantic relatedness is an essential operation for many natural language processing (NLP) tasks, such as Entity Linking (EL) and Question Answering (QA). It is still challenging to find a scalable approach to compute the semantic relatedness using Semantic Web data. Hence, we present for the first time an approach to pre-compute the semantic relatedness between the instances, relations, and classes of an ontology, such that they can be used in real-time applications.

CCS Concepts

•**Information systems** → *Information retrieval; Search engine indexing;*

Keywords

Semantic Relatedness, Scalability, Semantic Web

1. INTRODUCTION

Given the list { “Italy”, “Rome”, “Banana”, “Venice”} it is easy for humans to determine that between the four words "Banana" is the more unrelated word. This type of reasoning is very easy for humans. Such reasoning is also essential for computers to solve tasks like Entity Linking (EL) [4, 7] and Question Answering (QA) [12, 9].

To perform this task we have interpreted “Italy” as a country of Europe, “Rome” as the capital of Italy and “Venice” as a city in “Italy”. So we do not directly compare how related are the words, but the concepts they stand for. To make this clearer, consider the DBpedia instance “dbr:Psychedelic_Rock” referring to a music gender and the two instances “dbr:Apple_Inc.” referring to the company from California and “dbr:Apple_(band)” referring to a band called “Apple”. On one side since the band called Apple played psychedelic rock there is a high semantic relatedness among them, whereas there is a low semantic relatedness between the company and the music gender. On the other side, we use the same word “Apple” to refer to both the company and the band. Thus, the semantic relatedness is mainly encoded on the level of concepts behind words and not on the level of words. In this paper, we address the problem of computing the semantic relatedness not between words, but between concepts contained into ontologies such as DBpedia or any other ontologies.

One of the application of semantic relatedness is Entity Linking (EL), i.e. mapping entities in a sentence to an ontology. As an example take the sentence: “Yesterday Apple played great Psychedelic Rock.”. Using the semantic relatedness explained above it is clear that “Apple” here refers to the band and not to the company. Another application domain of semantic relatedness is ontology-based Question Answering (QA), i.e. searching for the answer to a question in an ontology. Let’s consider the question: “Who wrote The Lord of the Rings?”. In this case “The Lord of the Rings” could correspond to the book “dbr:The_Lord_of_the_Rings” or to the Films “dbr:The_Lord_of_the _Rings_(film_series)”. On the other hand “wrote” could correspond to the property “dbp:author” or to the property “dbo:composer”. Using semantic relatedness we can deduce that “dbo:composer” is the wrong choice since there is nothing in the context about music so that “wrote” must be mapped to “dbp:author”. This then allows to conclude that “dbr:The_Lord_of _the_Rings_(film_series)” is the wrong choice since a film has no author (but rather a director: “dbo:director”). In contrast to EL, QA is interested in considering the semantic relatedness between instances, properties and classes of the considered ontology.

To capture the semantic relatedness between instances, relations and classes in an ontology, a common approach is to measure the distance in the graph structure of the ontology. In this work we consider the same distance measure proposed in Sina [9] but we propose a technique to efficiently pre-compute it in a scalable way.

The main advantages of our approach are that: it is scalable, it works for instances, relations and classes, and it can be used for any ontology.

This paper is organized as follows. In section 2 we present works about the computation of semantic relatedness using Semantic Web Data. Moreover, we point out scalability issues in existing ap-

WIMS ’16, June 13-15, 2016, Nîmes, France

DOI: http://dx.doi.org/10.1145/2912845.2912864

proaches. We also discuss the current relatedness measures used in EL and QA. In section 3, we describe the main problem of computing the semantic relatedness numbers and the basic idea behind our algorithm. In section 3.1 we present the theory behind our approach and the reasons why it works. In section 3.2 we describe our implementation. In section 4 we present the performance results of our approach. Finally in section 5 we conclude and give an outlook on future works.

2. RELATED WORKS

There are different works that explore how to compute the semantic relatedness of instances in an ontology [5, 8]. These works generally focus on inventing new similarity measures and showing experimentally that in some tasks they perform better then previous ones. However, not much attention is paid on how fast these measures can be computed. For example in [8] the information used to compute the semantic relatedness between two instances is retrieved using SPARQL queries. This clearly leads to scalability issues.

The scalability is particularly problematic in the case of EL and QA. The reason is that the number of semantic relatedness computations can easily explode. Consider for example a question like "What are the names of the children of Obama and Michelle?". In DBpedia there are 979 instances which contain in their label "Obama" and 1204 instances which contain in their label "Michelle" (i.e. at least all people in DBpedia whose name is Michelle). In the case of a brute force approach, one will be interested in the computation of the semantic relatedness between 979·1204 pairs of instances!

Now, we present some of the current semantic relatedness measures used by EL and QA systems. The EL tool AIDA [4] and the QA system DEANNA [12], e.g., use the interlinked structure of Wikipedia. To compute the semantic relatedness between two instances they look at the corresponding Wikipedia articles and see how many links connect one article with the other. The higher the number of links, the higher is the semantic relatedness between the two instances. The main disadvantage of this measure is that it applies only to ontologies where there is a one-to-one correspondence between the Wikipedia articles and the instances of the ontology like in DBpedia [1] or YAGO2 [3]. Moreover, this strategy cannot be extended to compute the semantic relatedness of the relations and classes of an ontology.

The EL systems AGDISTIS [10] and Babelfy [7] compute the semantic relatedness in a similar way. Both start with a set of instances S. AGDISTIS constructs a graph G' which contains all nodes and edges of the ontology that can be reached from the instances in S with a breadth-search of a fixed depth. Babelfy also constructs a graph G'' by exploring the ontology around the instances in S using random walks. The set of nodes of G'' is S and two nodes s and s' are considered connected if, when exploring the ontology around the node s, the node s' was found (so if the distance between s and s' is not too big).

Both systems assume that the semantic relatedness of the instances is encoded in the graphs G' or G'', i.e. the instances that are more tightly connected in the graphs G' or G'' are more semantically related. The main advantage is that this information can be computed using only the ontology making it applicable for any ontology (differently for example from the EL system AIDA). The main problem is that computing the connection between the instances is time consuming. In AGDISTIS this is done on the fly, whereas Babelfy pre-computes them once. Babelfy is the work that is most near to what we are doing. Unfortunately the code is not open source and there are no numbers indicating the time needed for the pre-computations. Thus, we cannot compare our method with the one of Babelfy. One of the main differences with Babelfy is that we also include relation and classes in our pre-computation.

The QA system Sina [9] uses the shortest distance of maximal length n between instances and relations in the ontology to determine their semantic relatedness. The main disadvantage here is the time complexity. The numbers are computed using parallel execution of SPARQL queries on the fly which is computationally very expensive. This is one of the main reasons which leads to a runtime of 9.3 to 20.5 seconds per question.

To sum up, the EL tool AIDA and the QA system DEANNA do not directly use Semantic Web Data to compute the semantic relatedness. On the other side AGDISTIS, Babelfy and Sina do. All have in common that given two instances in an ontology they look if there is a path between them of maximal length n, where n is generally 2. This shows that the computation of these connections is of high interest for EL and QA. The main difference is that AGDISTIS and Babelfy consider only instances whereas Sina also considers relations. In this paper, we propose a scalable method to pre-compute these connections. In addition, we consider connections not only between instances, but we extend the computation to relations and classes, which improves the power of expression of the semantic relatedness. This has the big advantage that semantic relatedness computations become much faster. On the other hand when the ontology changes the numbers have to be recomputed.

3. PROBLEM AND IDEA

The problem that we want to address is to pre-compute the distance (i.e. the length of the shortest path) up to length n, between every pair of nodes in a ontology. Computing the shortest-path between two nodes in a graph is a standard problem in graph theory. A standard algorithms for computing the shortest distances between every pair of nodes in a graph is the Floyd-Warshall algorithm. But using such an approach for a big ontology is next to impossible for space reasons. Imagine that one wants to store the distance between every node of an ontology like DBpedia. The number of instances in DBpedia 2015 is approximately 10 million. If we store the distance between every two nodes using the data type short occupying 16 bits then we need $(10 \cdot 10^6) \cdot (10 \cdot 10^6) \cdot 16 = 16 \cdot 10^{14}$ bits $= 2 \cdot 10^{14}$ byte $= 200$ TB of space!

Therefore, the idea is to find only the shortest distance of some maximal length n. This is for example done in AGDISTIS and Sina using a breadth-search and in Babelfy using random walks. Note that both approaches explore the neighborhood of nodes to a given depth. If one assumes that each node has an average of 50 edges, to explore the nodes up to a depth of 3 one has to traverse $50 \times 50 \times 50 = 125.000$ edges per node. This is the reason why generally such approaches show poor performance in terms of computation time.

Instead we propose to compute these distances using a particular property of adjacency matrices. Consider the graph in figure 1.

Then the corresponding adjacency matrix is:

$$A_G = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

It can be shown that the entry i, j of A_G^k indicates how many walks of length k exist that are connecting i and j. This means that computing A_G^k is equivalent to exploring the graph around each node up

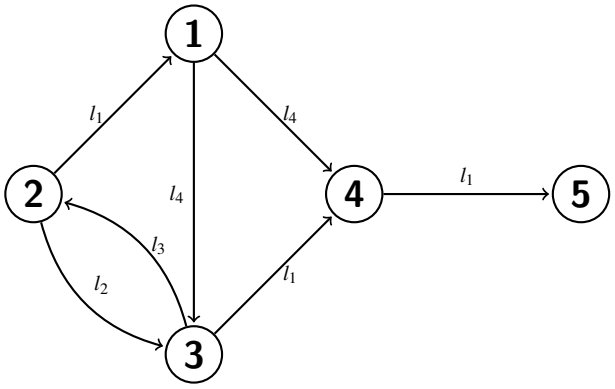


Figure 1: Example of a directed graph

to a depth of k. For example the $2,4$-entry of:

$$A_G^2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{2} & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

indicates that there are exactly two walks of length 2 connecting node 2 and 4 namely the walks $(2,1,4)$ and $(2,3,4)$. Since the $2,4$-entry of A_G is zero we know that there is no walk from 2 to 4 of length 1. Combining these two information we also get that the shortest distance between node 2 and node 4 is exactly 2.
Using this property the computation of the shortest paths of maximal length k can be reduced to some matrix addition/multiplication operations over the adjacency matrix of the graph, as we show in the next section. For the graph above the distances up to length 3 can be stored in a matrix in the following way (the first row and the first column indicate the start and end vertex):

$$D_1 = \left(\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 3 & 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 & 1 & 2 \\ 4 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

How exactly the computations have to be performed is explained in lemma 1 and lemma 2 in section 3.1.
Moreover, we extend this idea such that we can also compute the distance between vertices and edges having the same label (which correspond to properties with the same relation). This is important in QA. For example, one wants to know how distant is the instance "European Union" from the property "dbo:birthPlace". Note that we want the distance of "European Union" from the nearest edge with label "dbo:birthPlace" even if there are multiple! When taking into account not only the vertices, but also the labels we count the distance differently. Consider again the graph in figure 1. The distance between node 1 and label l_1 is for example 3 because of the sequence $(1,l_4,3,l_1)$ which has 4 elements. The distance between label l_2 and label l_4 is 6 because the sequence $(l_2,3,l_3,2,l_1,1,l_4)$ has 7 elements. Thus, our counting is similar to the case of walks, but we also consider labels and not just vertices in the count. For the graph above the distances up to length 6 can be stored in the following matrix:

$$D_2 = \left(\begin{array}{c|ccccc|cccc} & 1 & 2 & 3 & 4 & 5 & l_1 & l_2 & l_3 & l_4 \\ \hline 1 & 6 & 4 & 2 & 2 & 4 & 3 & 5 & 3 & 1 \\ 2 & 2 & 4 & 2 & 4 & 6 & 1 & 1 & 3 & 3 \\ 3 & 4 & 2 & 4 & 2 & 4 & 1 & 3 & 1 & 5 \\ 4 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline l_1 & 1 & 5 & 3 & 1 & 1 & 2 & 6 & 4 & 2 \\ l_2 & 5 & 3 & 1 & 3 & 5 & 2 & 4 & 2 & 6 \\ l_3 & 3 & 1 & 3 & 5 & 0 & 2 & 2 & 4 & 4 \\ l_4 & 5 & 3 & 1 & 1 & 3 & 2 & 4 & 2 & 6 \end{array}\right)$$

How exactly the computations have to be performed is explained in lemma 4, section 3.1.
The adjacency matrices of ontologies can be very big. In the case of DBpedia 2015 it is of the order of 10 million rows times 10 million columns. But on the other hand they are highly sparse. Imagine for example the row in the adjacency graph of DBpedia 2015 corresponding to "European Union". The whole row has roughly 300 non-zero entries (corresponding to the roughly 300 triples that have "European Union" as a subject). This means that for the shortest distance computation we can use the highly optimized libraries for sparse matrix operations. Our experiments show that this approach scales to big ontologies like DBpedia.
Having this information, we can compute the semantic relatedness by checking the distance between instances and relations. One can even modify it by taking into account the number of walks between nodes since we also compute them.
We now provide the theoretical explanation on how to use matrix addition and multiplication of the adjacency matrix to compute the distance between two nodes. Examples given in the theoretical justification help the readers to understand the formalization.

3.1 Theoretical Justification

We denote with $Mat(m \times n, \mathbb{R})$ the set of all real matrices with m rows and n columns.

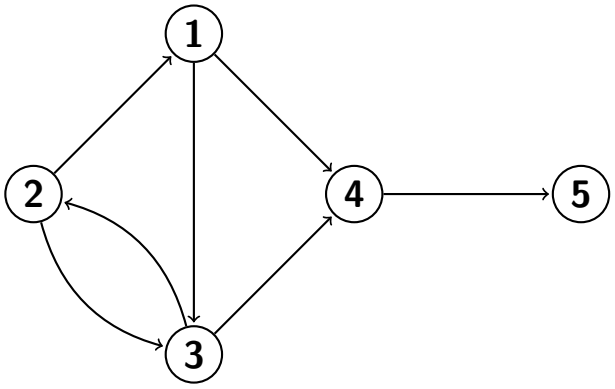
Definition 1. *(Graph) A graph is an ordered pair $G = (V,E)$ such that:*

- *V is a non-empty set, called the vertex set;*
- *E is a set, called edge set, such that either*
  - *$E \subset \{(v,w) : v,w \in V\}$, i.e. a subset of the pairs of V, or*
  - *$E \subset \{\{v,w\} : v,w \in V\}$, i.e. a subset of the 2-element subsets of V.*

*In the first case the graph is called **directed**, in the second **undirected**.*

Remark 1. In the following, we assume that if $|V| = n$ then $V = \{1,...,n\}$. This property can be achieved by renaming the vertices of G and obtaining a graph G'. G and G' are isomorphic.

Example 1. This is a directed graph $G = (V,E)$ with $V = \{1,2,3,4,5\}$ and $E = \{(1,3),(1,4),(2,1),(2,3),(3,2),(3,4),(4,5)\}$:



Definition 2. *(Adjacency Matrix) Let $G=(V,E)$ be a directed (or undirected) graph. If $n=|V|$ then the adjacency matrix of G is defined as $A_G=(a_{i,j})_{i,j}\in Mat(n\times n,\mathbb{N})$ such that*

$$a_{i,j}=\begin{cases}1 & \text{if } (i,j)\in E \text{ (or if } \{i,j\}\in E)\\ 0 & \text{otherwise}\end{cases}$$

Example 2. Let G be the graph in example 1. Then the corresponding adjacency matrix is:

$$A_G=\begin{pmatrix}0&0&1&1&0\\1&0&1&0&0\\0&1&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{pmatrix}$$

Definition 3. *Let $G=(V,E)$ be a directed (or undirected) graph.*

1. *A sequence $W=(v_0,...,v_n)$ is called a walk if:*
   (a) *$v_k\in V$ for $k=0,..,n$;*
   (b) *$e_i=\{v_i,v_{i+1}\}\in E$ (or $e_i=(v_i,v_{i+1})\in E$) for $k=0,...,n-1$.*

   The number n is called the length of the walk.
2. *A sequence $P=(v_0,...,v_n)$ is called a path if:*
   (a) *P is a walk;*
   (b) *all v_k for $k=0,...,n$ are distinct.*

   The number n is called the length of the path.

Remark 2. (a) If there is a walk between two vertices then there is always also a path, i.e. for each walk there is a corresponding path.

(b) A shortest walk W between two vertices is also a shortest path between them.

Example 3. Let G be the graph in example 1. The sequence $W=(2,3,2,1)$ is a walk of length 3 between 2 and 1, but not a path.

Lemma 1. *Let $G=(V,E)$ be a directed (or undirected) graph and A the corresponding adjacency matrix. If $A^k=(a_{i,j}^{(k)})_{i,j}$ then $a_{i,j}^k$ is equal to the number of walks of length k connecting the vertex i with the vertex j.*

PROOF. We provide the proof using induction. If $k=1$ then by definition of the Adjacency Matrix $a_{i,j}^{(1)}$ contains a 1 if there is a walk of length 1 connecting i and j and 0 if there is no walk.
Now we assume that the statement is true for $k-1$, i.e. if $A^{k-1}=(a_{i,j}^{(k-1)})_{i,j}$ then $a_{i,j}^{(k-1)}$ is equal to the number of walks of length $k-1$ from i to j. By definition of the matrix multiplication:

$$a_{i,j}^{(k)}=(A^k)_{i,j}=(A^{k-1}\cdot A)_{i,j}=\sum_{h=1}^{|V|}a_{i,h}^{(k-1)}\cdot a_{h,j}^{(1)}.$$

The expression $a_{i,h}^{(k-1)}\cdot a_{h,j}^{(1)}$ returns the number of walks of length $k-1$ form node i to node h times the number of walks of length 1 from node h to j. But these are exactly the number of walks of lengths k from i to j passing through h. Since the sum goes over all nodes in the graph G the entry $a_{i,j}^{(k)}$ corresponds exactly to the number of walks from i to j of length k. □

Example 4. Let A_G be the adjacency matrix build in 2. Then:

$$A_G=\begin{pmatrix}0&0&1&1&0\\1&0&1&0&0\\0&1&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{pmatrix}\quad A_G^2=\begin{pmatrix}0&1&0&1&1\\0&1&1&\mathbf{2}&0\\1&0&1&0&1\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}$$

$$A_G^3=\begin{pmatrix}1&0&1&0&1\\1&1&1&1&2\\0&1&1&2&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}\quad A_G^4=\begin{pmatrix}0&1&1&2&0\\1&1&2&2&1\\1&1&1&1&2\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}$$

The entry at the second row and third column of A_G^2 says for example that there are two walks of length 2 from node 2 to node 4.

Definition 4. *Let $A\in Mat(n\times m,\mathbb{R})$ with $A=(a_{i,j})_{i,j}$. The matrix $pos(A)=(p_{i,j})_{i,j}$ is defined as:*

$$p_{i,j}=\begin{cases}1 & \text{if } a_{i,j}>0\\ 0 & \text{otherwise}\end{cases}$$

Remark 3. Let $G=(V,E)$ be a directed or undirected graph. Then $pos(A_G^k)$ contains only ones and zeros. By lemma 1 the entries at row i and column j is 1 if and only if there is a walk of length k connecting node i with node j.

Definition 5. *Let $G=(V,E)$ be a directed or undirected graph and $v,w\in V$ two vertices. Then*

$$d(v,w)=min\{n:(v=v_0,...,v_n=w) \text{ is a path}\}$$

is called the distance between v,w. A path P with $P=(v=v_1,...,v_n=w)$ and $n=d(v,w)$ is called a shortest path between v and w (note that in general there is no unique shortest path).

Lemma 2. *Let $G=(V,E)$ be a directed or undirected graph.*

(a) *For $k>0$ the matrix*

$$B^{(k)}=pos(pos(A_G^k)-pos(A_G^{k-1})-...-pos(A_G^1))$$

contains in the i-th row and j-th column a 1 if the distance between node i and node j is k and zero otherwise.

(b) The matrix

$$D^{(l)} = \sum_{k=1}^{l} k \cdot B^{(k)}$$

contains in the i-th row and j-th column a zero if the distance between node i and j is bigger than l and k if it is exactly k.

PROOF.

(a) Let $B^{(k)} = (b_{i,j}^{(k)})_{i,j}$ and $pos(A_G^k) = (a_{i,j}^{(k)})_{i,j}$. Then the i,j entry of

$$pos(A_G^k) - pos(A_G^{k-1}) - ... - pos(A_G^1)$$

is

$$a_{i,j}^{(k)} - a_{i,j}^{(k-1)} - .... - a_{i,j}^{(1)}.$$

This is positive if and only if $a_{i,j}^{(k)}$ is one and $a_{i,j}^{(k-1)}, ..., a_{i,j}^{(1)}$ are zero. By remark 3 this is the case if and only if there exists a walk of length k connecting i with j and no walk of length less then k. But this means that $b_{i,j}^{(k)}$ is 1 if and only the shortest walks (and so also the shortest paths) connecting i and j are of length k, i.e. the distance between i and j is k.

(b) This follows directly from a).

□

Example 5. Let G be the graph of example 1. Then by example 2 and 4 we obtain:

$$B^{(1)} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad B^{(2)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 3 & 2 & 1 & 1 & 2 \\ 1 & 2 & 1 & 2 & 3 \\ 2 & 1 & 2 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

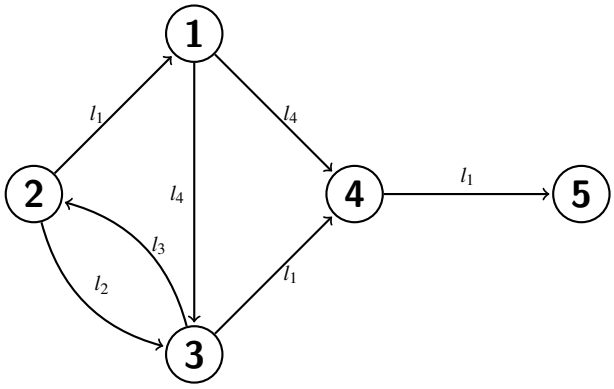
Up to now the labels of the edges where not considered. We want to consider them now.

Definition 6. *Let $G = (V,E)$ be a directed or undirected graph. G is called labeled with labels L if there is a function*

$$f : E \to L$$

, i.e. a function that assigns to each edge a label $l \in L$. We indicate a labeled graph with labels f as $G = (V,E,f)$.

Example 6. This is a labeled graph $G = (V,E,f)$ with $V = \{1,2,3,4,5\}$, $E = \{(1,3),\ (1,4),(2,1),(2,3),(3,2),(3,4),(4,5)\}$, $L = \{l_1,l_2,l_3,l_4\}$ and $f((1,3)) = l_4, f((1,4)) = l_4, f((2,1)) = l_1, f((2,3)) = l_2, f((3,2)) = l_3, f((3,4)) = l_1, f((4,5)) = l_1$:



Remark 4. In the following we assume that if $|L| = n$ then $L = \{1,...,n\}$. By renaming the labels of a graph G it is possible to construct a graph G' with this property. G and G' are isomorphic.

Definition 7. *Let $G = (V,E,f)$ be a directed or undirected labeled graph with labels L. We define an extended walk as a sequence $W = (s_0,...,s_n)$ such that:*

- *$s_i \in V \cup L$;*
- *for $k = 0,...,n-1$ $(s_i,s_{i+1}) \in V \times L$ or $(s_i,s_{i+1}) \in L \times V$, i.e. W is a sequence of alternating vertices and labels;*
- *if $(s_i,s_{i+1}) \in V \times L$ then there is an edge with label s_{i+1} starting from s_i, if $(s_i,s_{i+1}) \in L \times V$ then there is an edge with label s_i ending in s_{i+1}.*

The number n is called the length of the extended walk.

Example 7. Let $G = (V,E,f)$ be the labeled graph in example 6 then the sequence $W = (2,l_1,1,l_4,3,l_1)$ is an extended walk. The sequence $W' = (2,l_4,1)$ is not an extended sequence since there is no edge from node 2 to node 1 with label l_4.

Definition 8. *Let $G = (V,E,f)$ be a directed or undirected labeled graph with labels L. Let $v,w \in V \cup L$ then*

$$d_E(v,w) = min\{n : (v = v_0,...,v_n = w) \text{ is an extended walk}\}$$

An extended walk with $W = (v = v_0,...,v_n = w)$ and $n = d_E(v,w)$ is called a shortest extended walk between v and w.

Example 8. Let $G = (V,E,f)$ be the labeled graph in example 6 then $d_E(2,1) = 2$, $d_E(2,4) = 4$, $d_E(2,l_1) = 1$, $d_E(2,l_3) = 3$ and $d_E(l_1,l_4) = 2$.

Definition 9. *Let $G = (V,E,f)$ be a directed graph with labels L. If $n = |V|$ and $r = |L|$ then we define the matrix $R_{G,L,1} = (r_{i,j})_{i,j} \in Mat(n \times r, \mathbb{N})$ such that*

$$r_{i,l} = \begin{cases} 1 & \text{if there exists an } e \in E \text{ with } e = (i,j) \in E \text{ and } f(e) = l \\ 0 & \text{otherwise} \end{cases}$$

and the matrix $R_{G,L,2} = (r_{i,j})_{i,j} \in Mat(r \times n, \mathbb{N})$ such that

$$r_{l,j} = \begin{cases} 1 & \text{if there exists an } e \in E \text{ with } e = (i,j) \in E \text{ and } f(e) = l \\ 0 & \text{otherwise} \end{cases}.$$

Example 9. Let $G = (V,E,f)$ be the labeled graph in example 6 then:

$$R_{G,L,1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad R_{G,L,2} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Lemma 3. *Let $G = (V,E,f)$ be a directed graph with labels L. Then for $k \geq 0$:*

(a) $A_G^k \cdot R_{G,L,1}$ contains in the i-th row and l-th column the number of extended walks of length $2 \cdot k+1$ starting with vertex i and ending with label l.

(b) $R_{G,L,2} \cdot A_G^k$ contains in the l-th row and j-th column the number of extended walks of length $2 \cdot k+1$ starting with label l and ending with node j

(c) $R_{G,L,2} \cdot A_G^k \cdot R_{G,L,1}$ contains in the l-th row and m-th column the number of extended walks of length $2 \cdot k+2$ starting with label l and ending with label m.

PROOF.

(a) Let $A_G^k = (a_{i,j})_{i,j}$ and $R_{G,L,1} = (r_{i,r})_{i,r}$. By lemma 1 $a_{i,j}$ is equal to the number of path of length k from i to j. If $A_G^k \cdot R_{G,L,1} = (c_{i,r})_{i,r}$ then:

$$c_{i,r} = \sum_t a_{i,t} \cdot r_{t,r}$$

Since by definition $r_{t,r}$ is one if there is an edge that starts from t and has label l and zero otherwise, one sums up the number of all walks of length k starting from i and ending on a vertex where an edge with label l starts. But this is the number of extended walks of length $2 \cdot k+1$ starting with vertex i and ending with label l.

(b) Analogous to a) by exchanging $R_{G,L,1}$ with $R_{G,L,2}$ and changing the order in the matrix multiplication.

(c) Let $R_{G,L,2} \cdot A_G^k = (a_{s,j})_{s,j}$ and $R_{G,L,1} = (r_{j,r})_{j,r}$. By b) $a_{s,j}$ is equal to the number of extended walks of length k from label s to j. If $R_{G,L,2} \cdot A_G^k \cdot R_{G,L,1} = (c_{s,r})_{s,r}$ then:

$$c_{s,r} = \sum_j a_{s,j} \cdot r_{j,r}$$

Since by definition $r_{j,r}$ is one if there is an edge that starts from j and has label l and zero otherwise, one sums up the number of all extended walks of length $2 \cdot k+1$, from label s to vertex j where an edge with label r starts. But this is exactly the number of extended walks of length $2 \cdot k+2$ from s to r.

□

Example 10. Let $G = (V,E,f)$ be the labeled graph in example 6 then

$$A_G^2 \cdot R_{G,L,1} = \begin{pmatrix} 2 & 1 & 0 & 0 \\ \mathbf{4} & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

For example the entry equal to 4 corresponds to the number of extended walks from 2 to l_1 of length 5. The walks are $(2,l_1,1,l_4,3,l_1)$, $(2,l_2,3,l_3,2,l_1)$, $(2,l_1,1,l_4,4,l_1)$ and $(2,l_2,3,l_1,4,l_1)$.

Lemma 4. *For $h \geq 0$ let*

$$N_{2h+1} = \left(\begin{array}{c|c} 0 & A_G^h \cdot R_{G,L,1} \\ \hline R_{G,L,1} \cdot A_G^h & 0 \end{array} \right) \in Mat((n+l) \times (n+l), \mathbb{N}).$$

For $h > 0$ let:

$$N_{2h} = \left(\begin{array}{c|c} A_G^h & 0 \\ \hline 0 & R_{G,L,2} \cdot A_G^{h-1} \cdot R_{G,L,1} \end{array} \right) \in Mat((n+l) \times (n+l), \mathbb{N}).$$

Then:

Figure 2: Specification of the used hardware

Server	CPU	cores	RAM
laptop	2,4 GHz Intel Core 2 Duo	2	4 GB
lahc-2	Intel Xeon E5-2680 @ 2.70GHz	16	32 GB
lahc-6	Intel Xeon E5-2643 v3 @ 3.40GHz	12	64 GB

(a) For $k > 0$ the matrix

$$B^{(k)} = pos(pos(N_k) - pos(N_{k-1}) - ... - pos(N_1))$$

contains in the i-th row and j-th column a 1 if the shortest extended walk between i and j is of length k and a zero otherwise.

(b) The matrix

$$D^{(l)} = \sum_{k=1}^{l} k \cdot B^{(k)}$$

contains in the i-th row and j-th column a zero if the distance between i and j is longer then l and k if it is exactly k.

PROOF.

(a) Is analogous to the proof of lemma 2 a).

(b) Follows from a).

□

Remark 5. (a) We did not consider graph with multi-edges just to keep the notation simpler. Multi-edges appear in ontologies like DBpedia. All results carry over.

(b) Note that one can initialize the adjacency matrix A_G also by giving different weights to the edges. In this case the entries of A_G^k will be equal to the sum over the weights of all walks of length k connecting two nodes. This can also be used to express how similar two instances are.

3.2 Implementation

In this section we briefly describe the implementation details. The implementation is completely done in JAVA and follows a client-server model. A client sends a list of URIs to the server. The server receives the list, computes the pairwise distances between the URIs in the ontology and sends the result back to the client.
An initialization process is performed before the server can accept requests from the client. It takes as input a file containing the dump of an ontology. The file is parsed for the first time to create a dictionary that contains a one-to-one correspondence between URIs and integers. In fact two dictionaries are created: one for the subjects and the objects and the other for the relations. The dictionaries are implemented as Hash Maps to map an URI to an integer and as an Array List to map an integer to the corresponding URI.
Once the dictionaries are created the file is scanned a second time. Using the dictionaries, the URIs contained in the triples are converted to integers (literals are ignored). Each triple is used to construct the matrices A_G, $R_{G,L,1}$ and $R_{G,L,2}$. If a triple has a URI as the object then the entries are inserted in all three matrices, otherwise only an entry into $R_{G,L,1}$ is inserted. The sparse matrices are stored in the Dictionary of keys (DOK) format, i.e. triples indicating the row, the column and the value of the entry.
The above three matrices are imported in Octave [2] so that one can take advantage of the highly optimized libraries for sparse matrix multiplication. To compute the shortest distances of maximal

Figure 3: List of the DBpedia files used to construct the dumps

Name	Files	Size
dump-it	instance-types-en-uris_it.nt, labels-en-uris_it.nt, mappingbased-properties-en-uris_it.nt, infobox-properties-en-uris_it.nt	2.5 GB
dump-en	instance-types_en.nt, labels_en.nt, mappingbased-properties_en.nt	6.3 GB
dump-en-P	instance-types_en.nt, labels_en.nt, mappingbased-properties_en.nt, infobox-properties_en.nt	16 GB

Figure 4: Experimental results for different dumps on different hardware

N	dump	Server	Relations	Depth	Dictionary	Matrix	Total
1	dump-it	laptop	no	3	9.7 min 1.6 GB	1.5 min 481 MB	11.2 min 2.1 GB
2	dump-it	laptop	yes	6	9.7 min 1.6 GB	6.6 min 1.9 GB	16.3 min 3.5 GB
3	dump-en	lahc-2	no	3	7.3 min 4.0 GB	5.6 min 5.3 GB	12.9 min 9.3 GB
4	dump-en	lahc-2	yes	4	9.9 min 4.0 GB	7.2 min 0.5 GB	17.1 min 4.5 GB
5	dump-en	lahc-6	yes	6	5.0 min 8.0 GB	3.6 min 20.5 GB	8.6 min 28.5 GB
5	dump-en	lahc-6	yes	6	5.0 min 8.0 GB	3.6 min 20.5 GB	8.6 min 28.5 GB
6	dump-en-P	lahc-6	no	3	11.4 min 8.0 GB	24.8 min 21.2 GB	36.2 min 29.2 GB

length l, either the operations described in lemma 2 or 4 are performed depending on whether one wants to include the relations or not. When the computations are completed the server is ready to accept requests from the client.
When the client sends a list of URIs, first the server checks if all URIs are in the dictionary. If they are not then an error is returned. If they are in the dictionary then the rows and columns corresponding to the URIs are extracted from the matrix. This means that all the computations are done in the initialization phase and that a client request can be processed by reading the data.
The initialization step can be changed depending on the application. For some applications the number of walks can be important, in others, one is may be interested to assign different weights to edges with the same labels.
The code can be downloaded at:

https://github.com/WDAqua/ Indexing-server

4. EXPERIMENTS

In the following we present two experiments. The first shows that our approach scales and can be used to compute the desired numbers. In the second experiment, we present the results of a modified version of AGDISTIS [10] where the connections are not computed on the fly, but using the pre-computed numbers.
For the first experiment, we created a dump using some parts of the Italian and English DBpedia. The name of the files used and the total sizes of the corresponding dumps can be found in Table 3. In Table 2, we indicate the specifications of the hardware we used. Note that we use different hardware configurations mainly because the libraries of octave require that the whole data and the intermediate results fit in memory. The results of our experiments can be found in Table 4. Row 2 of Table 4 for example says that we computed the distances between nodes and relations of the Italian DBpedia on a laptop up to a depth of 6. The computation of the dictionary took 9.7 minutes and occupied 1.6 GB. The computation of the distances took 6.6 minutes and the resulting matrix occupied 1.9 GB. The total time for the computation is 16.3 minutes and we needed 3.5 GB to store the data. Remember that the depth is calculated differently depending whether relations are considered or not.

Comparatively note that AGDISTIS [10] by default explores the graph up to only a depth of 2 and does not consider the relations contained in the file infobox-properties_en.nt. Also Babelfy [7] explores the graph only up to a depth of 2. In the corresponding publication it is described that for each vertex a random walk is started one million times with a restart probability of 0.85. That means that the random walk reaches a depth of only 3 with a probability of $0.15 \times 0.15 \times 0.15 = 3.375 \cdot 10^{-3}$. So the random walks reach a depth of 3 on an average of only $(1 \cdot 10^6) \times (3.375 \cdot 10^{-3}) = 3375$ times. But since only nodes which were reached 100 times or more are considered and since generally there are many nodes at the depth of 3 (for a graph with 10 edges pro vertex on an average there are $10 \times 10 \times 10 = 1000$ nodes), the nodes at the depth of 3 are generally not considered. Finally note that the amount of data needed to store all the connections increases exponentially with the depth. Assume that the adjacency matrix of the graph has a size of 1 GB. If a node on an average is connected to 10 other nodes then one needs 10 GB to store all connections up to a depth of 2 and 100 GB to store all connections up to a depth of 3.
Sina is the most comparable approach to ours. However they don't consider classes of the ontology, and they have rather long processing time for a depth of 2. All this shows that our results scale. We can achieve a depth of 3 where others achieve a depth of 2. A depth of 4 is difficult to achieve due to processing and even more due to space constraints.
For the second experiment we created a modified version of AGDIS-

Figure 5: Benchmark of AGDISTIS against AGDISTIS-mod using GERBIL

Annotator	Dataset	Micro-F1	Micro-Precision	Micro-Recall	avg time ms/documents
AGDISTIS	ACE2004	0.5987	0.6020	0.5948	311.1964
AGDISTIS-mod	ACE2004	0.6209	0.6209	0.6209	238.3684
AGDISTIS	AIDA/CoNLL-Complete	0.4849	0.4851	0.4846	660.9863
AGDISTIS-mod	AIDA/CoNLL-Complete	0.4883	0.4883	0.4883	492.8392
AGDISTIS	AIDA/CoNLL-Test A	0.4884	0.4884	0.4884	736.7731
AGDISTIS-mod	AIDA/CoNLL-Test A	0.4913	0.4913	0.4913	559.7454
AGDISTIS	AIDA/CoNLL-Test B	0.4525	0.4525	0.4525	715.8788
AGDISTIS-mod	AIDA/CoNLL-Test B	0.4583	0.4583	0.4583	482.7229
AGDISTIS	AIDA/CoNLL-Training	0.492	0.4925	0.4915	625.1439
AGDISTIS-mod	AIDA/CoNLL-Training	0.4583	0.4583	0.4583	487.1205
AGDISTIS	AQUAINT	0.4542	0.4586	0.4498	611.3061
AGDISTIS-mod	AQUAINT	0.4704	0.4704	0.4704	535.8
AGDISTIS	DBpediaSpotlight	0.2485	0.2516	0.2455	194.807
AGDISTIS-mod	DBpediaSpotlight	0.2606	0.2606	0.2606	188.4655
AGDISTIS	IITB	0.4648	0.4665	0.4632	4,147.2621
AGDISTIS-mod	IITB	0.4664	0.4664	0.4664	3,602.75
AGDISTIS	KORE50	0.3228	0.3262	0.3194	391.5714
AGDISTIS-mod	KORE50	0.2986	0.2986	0.2986	201.84
AGDISTIS	MSNBC	0.5277	0.5452	0.5113	1,176.8947
AGDISTIS-mod	MSNBC	0.5523	0.5523	0.5523	928.2
AGDISTIS	Microposts2014-Test	0.3177	0.3177	0.3177	82.24
AGDISTIS-mod	Microposts2014-Test	0.3209	0.3209	0.3209	76.1573
AGDISTIS	Microposts2014-Train	0.4186	0.4186	0.4186	108.6434
AGDISTIS-mod	Microposts2014-Train	0.4199	0.4199	0.4199	98.1222
AGDISTIS	N3-Reuters-128	0.6348	0.6366	0.633	482.7229
AGDISTIS-mod	N3-Reuters-128	0.6261	0.6261	0.6261	289.7656
AGDISTIS	N3-RSS-500	0.6016	0.6022	0.601	168.6733
AGDISTIS-mod	N3-RSS-500	0.614	0.614	0.614	126.31
AGDISTIS	OKE 2015 Task 1 evaluation dataset	0.5811	0.5825	0.5798	375.67
AGDISTIS-mod	OKE 2015 Task 1 evaluation dataset	0.5723	0.5723	0.5723	308.2475
AGDISTIS	OKE 2015 Task 1 example set	1	1	1	120.6667
AGDISTIS-mod	OKE 2015 Task 1 example set	1	1	1	155.3333
AGDISTIS	OKE 2015 Task 1 gold standard sample	0.6114	0.6227	0.6006	329.6489
AGDISTIS-mod	OKE 2015 Task 1 gold standard sample	0.6568	0.6568	0.6568	251.7708

TIS [10]. The original version of AGDISTIS works as follows. The annotated entities in a text are expanded to generate possible candidates. This creates a candidate set S. Then a graph G' is constructed which contains all nodes and edges of the ontology that can be reached from the instances in S with a breadth-search of depth 2. This is done using a lucene index. Finally the HITS-algorithm is used to find the authority and hub values of all nodes in G'. The nodes in S with highest authority are then chosen as the correct candidates.

We constructed one modified version of AGDISTIS using our approach. We will call it AGDISTIS-mod. We keep all parts unchanged except the one that constructs the graph G'. In the modified part, a graph G'' is constructed using the pre-computed numbers in the following way. G'' has as nodes the instances in S. We put an edge between the instances in S if, in the ontology, there exists a connection of maximal length 2 between these instances.

We compared the two versions using Gerberil [11], a framework to compare Named-Entity-Recognition (NER) and Entity-Linking (EL) tools. We ran both versions against all available data sets. The results can be found in table 5. We indicate the micro F-measure, micro-precision, micro-recall and the average time per document. One can note two things. First the F-measure of the modified version is almost always slightly better than the original one. This means that it is not problematic to use the graph G'' instead of the graph G'. Moreover one can see an average time improvement of more than 18%. Note that the construction of the graph G'', on an average, takes between 0-10 ms. This means that the global time is mainly spend on the other two parts. Also note that AGDISTIS was already using an optimized solution.

This second experiment shows that, using our approach, it is possible to improve the performance of existing EL and QA systems.

5. CONCLUSION

We proposed a novel approach to efficiently pre-compute the semantic relatedness in a scalable way. Different from existing approaches our technique also allows to include relations and classes in the semantic relatedness computations. We have shown that our approach scales to big ontologies like DBpedia. Moreover, we have shown that the computed numbers can be used to improve existing systems.

In the future, we want to use these numbers for question answering. To reduce the footprint of the vocabulary, we aim to reuse vocabulary compression like it is done in HDT [6]. Also the storage of the matrix itself should be optimized. We hope that our work can bring the computation of semantic relatedness, using Semantic Web data, to new levels of scalability.

6. ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 642795. It was also funded by the Eurostars projects DIESEL (E!9367) and QAMEL (E!9725).

7. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[2] J. W. Eaton, D. Bateman, and S. Hauberg. *GNU Octave version 3.0. 1 manual: a high-level interactive language for numerical computations*. SoHo Books, 2007.

[3] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 3161–3165. AAAI Press, 2013.

[4] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics, 2011.

[5] I. Hulpuş, N. Prangnawarat, and C. Hayes. Path-based semantic relatedness on linked data and its use to word and entity disambiguation. In *The Semantic Web-ISWC 2015*, pages 442–457. Springer, 2015.

[6] M. A. Martínez-Prieto, J. D. Fernández, and R. Cánovas. Compression of rdf dictionaries. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 340–347. ACM, 2012.

[7] A. Moro, A. Raganato, and R. Navigli. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014.

[8] A. Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *AAAI spring symposium: linked data meets artificial intelligence*, volume 77, page 123, 2010.

[9] S. Shekarpour, E. Marx, A.-C. N. Ngomo, and S. Auer. Sina: Semantic interpretation of user queries for question answering on interlinked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 30:39–51, 2015.

[10] R. Usbeck, A.-C. N. Ngomo, M. Röder, D. Gerber, S. A. Coelho, S. Auer, and A. Both. Agdistis-graph-based disambiguation of named entities using linked data. In *The Semantic Web–ISWC 2014*, pages 457–471. Springer, 2014.

[11] R. Usbeck, M. Röder, A. Ngonga Ngomo, C. Baron, A. Both, M. Brümmer, D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann, et al. Gerbil-general entity annotation benchmark framework. In *24th WWW conference*, 2015.

[12] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390. Association for Computational Linguistics, 2012.