Approaches to Measure Class Importance in Knowledge Graphs

Daniel Fernández-Álvarez\textsuperscript{a,*}, Johannes Frey\textsuperscript{b}, José Emilio Labra Gayo\textsuperscript{a}, Daniel Gayo-Avello\textsuperscript{a}, and Sebastian Hellmann\textsuperscript{b}

\textsuperscript{a} Department of Computer Science, University of Oviedo, Oviedo, Spain
\textsuperscript{b} AKSW Group, University of Leipzig, Leipzig, Germany

E-mails: fernandezalvdaniel@uniovi.es, labra@uniovi.es, dani@uniovi.es
E-mails: frey@informatik.uni-leipzig.de, hellmann@informatik.uni-leipzig.de

Abstract.

The amount, size, complexity, and importance of Knowledge Graphs (KGs) have increased during the last decade. Many different communities have chosen to publish their datasets using Linked Data principles, which favors the integration of this information with many other sources published using the same principles and technologies. Such a scenario requires to develop techniques of Linked Data Summarization. The concept of a class is one of the core elements used to define the ontologies which sustain most of the existing KGs. Moreover, classes are an excellent tool to refer to an abstract idea which groups many individuals (or instances) in the context of a given KG, which is handy to use when producing summaries of its content. Rankings of class importance are a powerful summarization tool that can be used both to obtain a superficial view of the content of a given KG and to prioritize many different actions over the data (data quality checking, visualization, relevance for search engines...).

In this paper, we discuss the notion of class importance in the context of a KG, analyze existing techniques, and propose a novel approach called ClassRank. To do so, we measure the importance of classes of the DBpedia ontology using different approaches and compare the obtained results with the actual usage of these classes in the logs of the DBpedia SPARQL endpoint.

Keywords: Importance ranking, Class importance, PageRank, RDF, DBpedia, Knowledge Graph

1. Introduction

With the development of semantic web technologies, a huge volume of information has been published as Linked Data (LD) or Linked Open Data (LOD) in the form of Resource Description Framework (RDF) graphs. LOD is being used for a wide range of different applications, including search engines [1–3] or recommendation systems [4, 5]. Many different knowledge domains are covered by LOD datasets, and there are several projects whose main goal is to store and to offer as many general-purpose LOD content as possible. DBpedia [6], Wikidata [7], YAGO [8], and OpenCyc [9] are insightful examples.

In such a context, automatic summarization techniques are now more necessary than ever. Providing a simplified understanding of a graph’s content is desirable in two ways. On the one hand, it allows consumers to decide whether a graph can be suitable for their purposes. On the other hand, it can be a tool to discover which are the most important topics, entities, or types of relations within a given source. This is particularly relevant in cross-domain datasets maintained by different people, organisms, or tools. The bigger, more complex, or greater it is the number of different agents maintaining a LOD source, the harder it is to produce accurate handcrafted summaries.

There are many valid approaches to produce summaries of different natures of LOD content, especially for schema elements such as classes [10–12]. These summaries frequently consist of reduced graphs which are representative of the original structure. All these techniques need to identify which are the most important elements in the target graph. These importance
rankings can be used to produce any of the aforementioned summaries, but they can also act as a summary by themselves.

The concept of class has a key role in these processes. Classes are abstract ideas which group many individuals sharing a common set of features under the same label. In general, RDF graphs are explored or exploited is using SPARQL queries. Thus, knowing the set of properties associated to a given class allows for writing SPARQL queries involving their instances, since they all share a common structure w.r.t property usage.

Despite this, the problem of detecting class importance to elaborate class rankings has not received enough attention from the scientific community. The notion of importance has not been properly defined yet. In this paper, we discuss the notion of importance opposed to the notion of relevance against a given purpose, and evaluate different approaches to measure class importance. Our contribution is twofold:

- We make a compilation and comparison of existing unsupervised techniques to produce class importance rankings. The techniques are compared in terms of quality of their results, type of information analyzed and computational complexity.

- We propose a new technique called ClassRank. Our approach is based on PageRank [13] scores and assigns each class an importance score computed upon the importance of its instances. ClassRank can capture the importance of classes with few instances when those instances are important enough.

To evaluate these techniques, we applied them all over the English chapter of DBpedia. Then, the results produced are compared to the actual usage of each class in a sample of logs from the official DBpedia SPARQL endpoint. Class usage in SPARQL logs has already been used to build rankings of class importance [14].

In section 2, we provide and discuss some notions which are required for a good understanding of this paper’s content. In section 3, we briefly introduce all the techniques which are used to compute class importance in our experiments. In section 4, we describe our experiments and the obtained results in a detailed way. In section 5, we discuss the results obtained during the experimentation. In section 6, we analyze some works related to ours. We focus mainly on techniques to measure class importance/relevance not used in our experiments. Finally, in section 7, we summarize the conclusions of our work.

2. Preliminary notions

2.1. Assertion Box and Terminological Box considerations

In this paper, we will assume that the reader is familiar with basic concepts of RDF such as URIs ($U$), literals ($L$), or blank nodes ($B$). An RDF graph $G$ can be formally defined as a set of triples $(s, p, o) \in (U \cup B) \times (U \times (U \cup B \cup L))$. We can distinguish two different types of statements from a conceptual point of view, commonly referred to as Terminological Box (T-BOX) and Assertion Box (A-BOX) in the literature. T-BOX statements stand for abstract concepts, aka classes. They are used to describe schemata, and they are key elements to define ontologies. On the other hand, A-Box statements contain instance information. They are more associated with Knowledge Graphs (KGs) containing information about actual individuals and how these individuals are linked between themselves and their respective schema elements.

The techniques applied in this paper always aim to get scores and rankings for classes, not instances. However, to obtain such a result, some of them consider structures purely composed of T-BOX statements, while some others use knowledge related to the A-BOX part of the graph as well.

2.2. Importance vs relevance

Discovering the most important nodes in a graph has been proven as a key task in order to perform further actions, such as summarization or prioritization of contents. Even with that, the notion of importance itself remains ambiguous. Several metrics that compute different topological features have been purposed to determine the importance of elements. However, there is not an approach that outperforms the rest in terms of correctness. The suitability of the different available techniques may depend on the planned usage for the obtained rankings.

By contrast, the notion of relevance is linked to a purpose, therefore it is much easier to define it within a context. For instance, search engines provide rankings of elements w.r.t. its relevance to a given query. Recommendation systems produce lists with the most relevant items for their users. Different classification
systems may produce different groupings attending to different criteria, so the algorithms used check the relevance of each criterion against those criteria.

Relevance is not just easier to define, but also to evaluate. Both search engines and recommendation systems are designed to be used by some final users, who are legitimate judges to classify a set of results as relevant or not for their expectations and preferences. Therefore, those users can determine whether a given classification is correct. In the case of importance decoupled of a specific purpose, the idea of what are the most important elements in a dataset may be different for the owner or creator of the source and for each one of its consumers. And there is not a definitive argument proving which one of them is right.

For our experimentation, we have adopted the idea of class importance used in [14]. The authors compare several centrality measures against an artificial gold standard. They associate each class a score in light of how frequently that class is mentioned in a SPARQL query against an endpoint exposing the graph’s content. Further details about how to identify class mentions in the logs and how to measure class importance with them will be provided in section 4. The rankings obtained from the logs will be used as a reference to evaluate the approaches mentioned in section 3.

3. Metrics

3.1. Importance metrics applied over schema structures

In this section we will work with a formal definition of a graph $G = (V, E)$, where $V$ is a set of nodes or vertices and $E$ is a set of edges linking those nodes. In our experimentation, all the techniques presented in this section will be applied over graphs composed only by T-Box elements. In the interest of brevity, in following sections of this paper we will refer to them all as Only Terminological Techniques (OTT).

3.1.1. Degree

The degree is one of the simplest measures of graph centrality. The degree of a node $e$ is the number of edges incident to $e$. We will denote the degree of a node $e \in V$ as $D(e)$.

3.1.2. Betweenness

The Betweenness $B(e)$ of a node $e$ is the ratio of shortest paths between any pair of nodes $(u, v)/e \in V \land e \neq u \neq v$ that pass through $e$ compared to the total number of shortest paths. Let $\sigma(e)$ be the function which gives the number of shortest paths passing through $e$ and $\sigma()$ be the total number of shortest paths. Then, the Betweenness $B(e)$ can be defined as:

$$B(e) = \sum_{e \neq u \neq v} \frac{\sigma(e)}{\sigma()}$$

3.1.3. Bridging Centrality

A bridging path is an indirect connection between two aggregate nodes in a graph, i.e., a link of two densely connected components (e.g. a domain knowledge, an organization) via a third node known as bridging node. On the top of this concept, the Bridging Centrality $BC(e)$ of a node $e$ assigns to $e$ a score which aims to measure how much $e$ acts as a bridge mainly for the nodes in its neighborhood in $G$. For such a goal it combines both local and global metrics of centrality. It is based on Betweenness and the Bridging Coefficient $B_v(e)$ of a node $e$, which is defined as follows:

$$B_v(e) = \frac{D(e)^{-1}}{\sum_{i \in N(e)} D(i)^{-1}}$$

where $N(e)$ is the set of nodes in the immediate neighborhood of $e$. With this, $BC(e)$ is defined as:

$$BC(e) = B(e) \cdot B_v(e)$$

In the contexts of KGs, $BC(e)$ can be used to identify useful nodes linking different information topics or knowledge domains.

3.1.4. Closeness and Harmonic Centrality

The Closeness $C(e)$ of a node $e$ gives a hint about how close $e$ is to every other node in $G$. The Closeness score of $e$ consists of the average of the length of the shortest paths from $e$ to every $v \in V \land v \neq e$. The Harmonic Centrality $HC(e)$ consists of a slight modification of the Closeness, computing the harmonic mean of distances instead of the average. Let $d(u, v)$ be the function that gives the length of the shortest path between $u$ and $v$. With this, $HC(e)$ can be defined as follows:

$$HC(e) = \frac{1}{\sum_{u \neq v} d(e, u)}$$

Harmonic Centrality and Closeness produce an inverse sorting of elements, i.e., the reverse rank of Closeness would be the same rank of Harmonic Cen-
Radiality. The element with the highest score in Closeness is the less central one, i.e., the one whose paths to every other node happen to be the longest ones. By contrast, the element with the highest score in Harmonic centrality is the most central one. Since the rest of the techniques employed produce scores in which the higher is the value the more important is the element, we will use Harmonic Centrality instead of Closeness to obtain comparable rankings.

3.1.5. Radiality
The Radiality, as well as the Closeness or Harmonic Centrality, aims to quantify how close is a node to all the rest in a graph. Radiality is based on the concept of Diameter of a graph \( \Delta(G) \), which is the maximum distance between any pair of nodes in \( V \). With this, the Radiality \( R(e) \) of a node \( e \) can be defined as follows:

\[
R(e) = \frac{1}{\sum_{u \neq e} (\Delta(G) - \left( \frac{1}{\deg(u)} \right))}
\]

3.2. Importance metrics applied over the whole graph structure

The techniques introduced in this subsection have in common that they all use T-Box and A-Box knowledge of the target KG to produce a result. In the interest of brevity, we will refer to them as Also Assertion Techniques (AAT).

3.2.1. Instance counting
This importance metric is tightly linked to the RDF world and, specifically, to the class-instance relation. The more instances a class has, the more important the class is considered to be. Several public and widely-used data sources offer statistics about number of instances as a clue of class importance, such as Wikidata \(^1\), or offer separate files in their dumps to manage triples about instantiation, such as DBpedia \(^2\). Instance Counting (IC) is a simple but scalable importance metric.

Typically in RDF sources the relation of instance-class between two elements \( e_c \) and \( c \) is expressed using the property `rdf:type` \(^3\) in a triple \((e_c, \text{rdf:type}, c)\). However, properties with a similar semantic to `rdf:type` can be used, such as `wdt:instanceof` in Wikidata.

3.2.2. PageRank
PageRank is based in a notion of importance which can be informally explained with the next statement: an element gains importance if it receives more links from other elements, if those links come from important elements, and if those elements have few outgoing links. PageRank scores are values in \([0, 1]\) with a nice statistical interpretation. The PageRank score of a node \( e \) is the probability that a random surfer starting at a random node and jumping from node to node following links stops at node \( e \). PageRank was originally designed to rank the importance of pages in the World Wide Web. In order to model the actual behavior of an internet user that may follow links between pages, but may write as well some new URL in his browser to access to a page non-linked from the current one, PageRank uses a param \( \alpha \). The probability \( d \) that the random surfer has of getting bored of following links and jumps to a random page is \( d = 1 - \alpha \).

3.2.3. ClassRank
We propose a novel technique in order to compute class importance called ClassRank. The ClassRank score of a class \( c \) consist of the aggregation of the PageRank scores of its instances. Let \( \text{PR}(e, \alpha) \) be the PageRank of \( e \) with a damping factor of \( \alpha \). And let \( I(c) \) be a function which gives the set of all the instances of the class \( c \). Then the ClassRank score \( CR(c, \alpha) \) of a class \( c \) with a damping factor of \( \alpha \) can be simply defined as follows:

\[
CR(c, \alpha) = \sum_{e \in I(c)} \text{PR}(e, \alpha)
\]

As well as IC, ClassRank qualifies the importance of a class w.r.t. their instances. Nevertheless, while IC purely quantifies the number of instances, ClassRank is able to keep a balance between the quantity and quality (aka importance) of those instances.

ClassRank assigns scores in \([0, 1]\) which also has a nice statistical interpretation. \( CR(c, \alpha) \) is the probability that a random surfer such as the one described for PageRank has to land in an instance of \( c \). In order to decide which are the instances of a given class \( c \), ClassRank relies on the concept of class-pointer. We define a class-pointer as a property which links an instance with its class. Commonly, there is just one pure class-pointer in each RDF graph, being `rdf:type` the most common one. Nevertheless, it can happen that a given KG or a combination of several KGs of different nature could use more than one class-pointer. Also, a given

\(^{2}\)https://databus.dbpedia.org/dbpedia/mappings/instance-types/2019.09.01 Accessed in 2020/03/28
\(^{3}\)All the prefixes used in this paper are commonly used and can be solved using the online tool http://prefix.cc/ Accessed in 2020/03/28
user may desire to use as a class-pointer a certain property whose defined semantics does not express a strict relation of instance-class, but a similar yet useful notion.

An example of such an arguable property could be :occupation. Let’s consider a triple (:sarah, :occupation, :doctor) representing that someone called Sarah works as a doctor. It cannot be said that the essential type of :sarah is :doctor but, even with that, it can be said that Sarah is a doctor in informal speech. Let $G$ be a graph just describing many people’s job, where the rdf:type of all the individuals is :Person. Then, choosing rdf:type as class-pointer to compute $G$ with ClassRank will produce not really useful results. The only class with instances would be :Person. However, an execution of ClassRank using :occupation as class-pointer, or rdf:type and :occupation at a time, will give a distribution of importance among the different occupations (classes) stated. In order to support the described scenarios, ClassRank can be considered arbitrary class-pointer(s).

Even if ClassRank is inspired and built over PageRank scores, it is important to remark that $PR(c, \alpha) \neq CR(c, \alpha)$. While PageRank measures the importance of the URI of a class within a graph, ClassRank uses this URI as a pure label to represent the accumulated importance of a grouping of elements whose common feature is having the same class. Actually, as it is defined, $PR(c, \alpha)$ does not have any effect on $CR(c, \alpha)$ unless $c$ is its own instance. Formally stated, $PR(c, \alpha)$ does not have any effect on $CR(c, \alpha)$ unless it is true that $(c, p, c) \in G \land p \in CP(G)$, where $CP(G)$ is the set of class-pointers of $G$.

ClassRank’s pseudo-code has been formalized in Algorithm 1. However, some of the conventions used must be described. We define a graph $G$ as a set of triples $G = \{t_1, t_2, \ldots, t_n\}$. A triple $t$ is a group $t = (s_t, p_t, o_t)$ (subject, predicate and object). We use the macro $f_{PR}(G, \alpha)$ to refer to the standard PageRank function. $f_{PR}(G, \alpha)$ receives a graph $G$ and a damping factor $\alpha$ as input, and it returns a vector of size $n$, being $n$ the number of nodes contained in $G$. We use $E$ to denote the set of nodes contained in $G$, and $P$ to denote the set of properties used in any $t \in G$. We denote the set of classes to be classified with $E_C$, and the set of class-pointer properties with $P_C$. We use $f_{0}$ to denote an empty function $f_{0} : \emptyset \rightarrow \emptyset(\emptyset E_{C})$, i.e., a function whose domain is the empty set $\emptyset$ and whose co-domain consist of all possible subsets (powerset) of $E_{C}$.

In line 13, we initialize a vector of maps, and to represent each map we are using function notation.

**Algorithm 1** ClassRank pseudo-code

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input: $G$ = Target Graph</td>
</tr>
<tr>
<td>2</td>
<td>Input: $P_{r}$ = Set of properties identified as class-pointers</td>
</tr>
<tr>
<td>3</td>
<td>Input: $\alpha$ = Damping factor</td>
</tr>
<tr>
<td>4</td>
<td>Input: $\theta$ = Security threshold</td>
</tr>
<tr>
<td>5</td>
<td>Input: $E_{C}$ = Target classes (it can be an empty set)</td>
</tr>
<tr>
<td>6</td>
<td>$I_{r} = \emptyset$</td>
</tr>
<tr>
<td>7</td>
<td>$p_1 = \begin{pmatrix} 0 &amp; 0 &amp; \ldots &amp; 0 \end{pmatrix}$</td>
</tr>
<tr>
<td>8</td>
<td>$p_2 = \begin{pmatrix} 0 &amp; 0 &amp; \ldots &amp; 0 \end{pmatrix}$</td>
</tr>
<tr>
<td>9</td>
<td>$p_n = \begin{pmatrix} 0 &amp; 0 &amp; \ldots &amp; 0 \end{pmatrix}$</td>
</tr>
<tr>
<td>10</td>
<td>$Q_{r_{a_{0}},e_{0}} \leftarrow Q_{r_{a_{0}},e_{0}} + 1$</td>
</tr>
<tr>
<td>11</td>
<td>$i \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>12</td>
<td>$L_{r} \leftarrow f_{PR}(G, \alpha)$</td>
</tr>
<tr>
<td>13</td>
<td>$L' \leftarrow \begin{pmatrix} 0 &amp; 0 &amp; \ldots &amp; 0 \end{pmatrix}$</td>
</tr>
<tr>
<td>14</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>15</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>16</td>
<td>$L_{r_{a_{0}},e_{0}} \leftarrow L_{r_{a_{0}},e_{0}} + L_{r_{a_{0}},e_{0}}$</td>
</tr>
<tr>
<td>17</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>18</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>19</td>
<td>$L_{r_{a_{0}},e_{0}} \leftarrow L_{r_{a_{0}},e_{0}} + L_{r_{a_{0}},e_{0}}$</td>
</tr>
<tr>
<td>20</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>21</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>22</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>23</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>24</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>25</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>26</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>27</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>28</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>29</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>30</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>31</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>32</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>33</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>34</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>35</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>36</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>37</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>38</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>39</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>40</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>41</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>42</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>43</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>44</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>45</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>46</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>47</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>48</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>49</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
<tr>
<td>50</td>
<td>$S_{r} \leftarrow \begin{pmatrix} f_{0} &amp; f_{0} &amp; \ldots &amp; f_{0} \end{pmatrix}$</td>
</tr>
<tr>
<td>51</td>
<td>$I_{r} \leftarrow I_{r} \cup {o_{j}}$</td>
</tr>
</tbody>
</table>

Given a certain function $f$, we denote its domain with $D(f)$, and its graph with $G(f)$. We modify the definition of a function $f$ by adding or modifying elements in $G(f)$, i.e., in order to define $a(\alpha) = b$, we will use $G(f)[\alpha] \leftarrow b$. 

52
Algorithm 1 receives as input a target graph $G$, a set $PC$ of class-pointers, a damping factor $\alpha$ used for the PageRank execution, a security threshold $\theta$, and a set of target classes $E_C$, which can be empty if the target classes are not known a priori.

The threshold $\theta$ is used to ignore facts that occur rarely, which may mean that they are noise or they have a non-significant presence in $G$.

The algorithm returns three results: 1) The standard PageRank vector for every entity $\{e/e \in E\}$, denoted as $L$; 2) the ClassRank vector for every class $\{e_C/e_C \in E_C\}$, denoted as $L'$; and 3) a matrix containing information about which entities point to which classes using which class-pointer, denoted as $S$. $L'$ provides the importance of each class, while $S$ allows to analyze the source of that importance.

We have divided ClassRank in three stages preceded by a preliminary one to prepare some data structures.

**Preliminary stage: Initializations** At this stage, we initialize some data structures that will be used during the calculations of the ClassRank scores.

$L_e$ is a set which will contain identifiers of the detected classes. $Q$ is a matrix of $(m \cdot n)$, where $m = |E|$ and $n = |PC|$. In $Q$ we annotate how many times a given object $o_i$ is linked with a given class-pointer $p_i$.

**Stage 1: PageRank** At this stage, we calculate the internal relevance of each entity in $G$ and we store it in vector $L$.

We are using the PageRank algorithm as a base centrality metric. Probably the most common alternatives to PageRank used to rank entities regarding graph centrality are Hypertext Induced Topic Selection (HITS) [15], and Stochastic Approach for Link-Structure Analysis (SALSA) [16]. However, both approaches are used mainly in Information Retrieval contexts since they both rank each node w.r.t. a given query/topic.

**Stage 2: Class detection** This stage is executed in lines 4 to 12. It can be informally summarized with the next statement: just in case the set of target classes is not known a priori, if some node is pointed a significant number of times by a certain class-pointer, then this node represents a target class.

When $E_C \neq \emptyset$ it means that the set of classes to be ranked it is known a priori. Otherwise, this set should be discovered in the graph executing the code from lines 4 to 12. The security threshold $\theta$ used in Algorithm 1 has been introduced in order to filter wrong identifications of classes causing noise. This is specially handy in sources maintained by many agents making small editions, where human actions can cause marginal mistakes. This threshold should be used carefully, since it may also cause a certain number of false negatives for all those actual classes that are pointed less than $\theta$ times by a class-pointer.

**Stage 3: ClassRank scores** The ClassRank score of each class is calculated as the aggregation of the PageRank scores of its instances in lines 13 to 21. This stage can be informally summarized with the next statement: if there is a high enough number of triples that have the same class-pointer as predicate and the same class URI as object, then the PageRank scores of the subjects of those triples are added to the ClassRank score of the class URI.

In lines 15-16, for each triple $t_i = (s_i, p_i, o_i)$, we check if $o_i$ is a class and $p_i$ is a class-pointer linked to $o_i$ at least $\theta$ times. If this is true, we perform three other actions:

- In lines 17-18 we include $p_i$ as class-pointer of $o_i$ in the vector of maps $S$, just in case it had not been already included.
- In lines 19-20 we add the PageRank score of $s_i$ to the ClassRank score of $o_i$, just in case it had not been already added.
- In line 21 we specify in the vector of maps $S$ that $s_i$ is instance of $o_i$ due to the class-pointer $p_i$.

There is a public implementation of ClassRank available in a GitHub repository.

### 4. Experiments

In order to evaluate the quality of the rankings produced by the described techniques, we use a random sample of log files from the DBpedia SPARQL endpoint. When computing AAT approaches, the version of the KG used is the English Chapter of DBpedia published in 2017. The dump files used are public and can be downloaded.

As suggested in [14], we have considered mentions of classes in SPARQL queries as a reliable metric of how important a class is in a given source. A class is considered to be mentioned in a query when:

---


– The URI of the class is mentioned.
– The URI of an instance of the class is mentioned.
– The URI of an element is mentioned, in case it is used in a triple with a property whose domain/range forces the URI to be an instance of a class.

Let \( p \) be a property which links an instance to its class; let \( d(p) \) and \( r(p) \) be the domain and the range of the property \( p \); and let \( G \) be the KG under analysis. Then, formally, a class \( e \) is considered to be mentioned in a query if \( e \) is mentioned and it is true that \( (e, p, o) \in G \land c \in d(p) \lor (s, p, e) \in G \land c \in r(p) \), even if \( (e, p, o) \notin G \).

We elaborated a ranking that sorts the classes according to its number of mentions in the log (most mentioned at the top). Then, we computed the target KG with the rest of the techniques, made a ranking for each one (most important classes at the top), and compared the results with the reference rankings.

4.1. How to compare the rankings: Rank-Biased Overlap

The compared rankings have two peculiarities. First, it is feasible to have tied elements. Second, the significance of changes in the top of the ranking is higher than changes in the low spots. Search engine results or classification in sports are insightful examples of this. When comparing two search engines, the first results shown to the user are much more relevant than the ones in position 100th. Similarly, the event of a player climbing from the second seed to the top seed in a given sport use to receive more social attention than any other jump in deeper regions of the ranking. Importance rankings in RDF sources are used to prioritize some elements for different tasks or to get a general idea about the content of a given source. Then, the top-ranked elements are more relevant than the low-ranked ones.

For these reasons, it is desirable to use a metric that can compare the similarity of two rankings naturally handling these two features. We have found that Rank-Biased Overlap fits in our requirements. A discussion about the convenience of this technique in this kind of scenario, as opposed to classic approaches such as Spearman [17], where all the elements have the same impact over the final score, is provided in [18].

Essentially, RBO checks the overlap of two rankings at incrementally increasing depths. RBO is originally defined as a distance measure between two rankings, where 0 means minimum distance and 1 means maximum distance. However, it can be trivially transformed into a metric by calculating \( 1 - \text{RBO} \), where 1 means maximum similarity, and vice-versa. From this point, we will work with the definition of RBO as a metric.

The elements checked at each depth \( d \) are those in rank \([1, \ldots, d]\). This means that the first element is the most important of them all, since it will be checked looking for overlap at every iteration. The following element with more importance over the score will be the second one, and so on. At each iteration, RBO computes the ratio of overlapped elements. It produces a result adding all those ratios weighted using an infinite series of weights whose sum converges always to a fixed value. However, those weights can be tuned to give a certain amount of importance to a wider/shorter region of the top rank using a parameter \( p \). The \( p \) parameter has a nice statistical interpretation. It models the user’s persistence when performing a manual checking of the rankings. Low values of \( p \) arbitrarily decrease the probability that a user has to keep exploring ranks, and vice-versa. The extreme case \( p = 0 \) causes that the only position checked is the first one, so RBO gives a result of 0 (no overlap) in case the first element of both rankings is not the same, or 1 (perfect overlap) in case that first element matches, ignoring the rest of the ranking. The higher is the value of \( p \), the less probable it is that the user stops exploring the ranking.

Greater values of \( p \) arbitrarily increase the importance of wider prefixes of the rankings. This means that, even if each iteration \( k \) will still have a greater impact over the results than \( k + 1 \), greater values of \( p \) decrease that difference. \( p \) can be also interpreted as a parameter to configure the exact amount of importance over the final score that a given prefix length has. For instance, a value of \( p = 0.9 \) gives 86% of the importance to the top 10 elements. This is, the sum of weights of the first 10 iterations of RBO will be 0.86. Although there is not a function to obtain a value of \( p \) for a couple of chosen values of importance and prefix length, the authors in [18] provide the following useful equation:

\[
W_{\text{RBO}}(1 : d) = 1 - p^{d - 1} + \frac{\ln r}{r} \cdot d \cdot \left( \ln \frac{1}{p} - \sum_{i=1}^{d-1} \frac{1}{i} \right)
\]

In the equation \( 1, d \) is the depth or prefix length, and \( W_{\text{RBO}}(1 : d) \) is the accumulated weight of the elements of a ranking in positions 1 to \( d \) in the final RBO score.

We have developed a script \( f_{\theta}(d, w, \theta) \) which receives a length \( d \), a weight \( w \), and an error threshold \( \theta \),
and returns a value of $p$ which approximately solves the equation 1 for $d$ and $w = W_{RBO}(1 : d)$. The script computes the equation 1 for $d$ with different values of $p$, obtaining each time a result $w_p$. The script stops when it founds a $p_i / |w_p_i| - w < \theta_e$, and returns $p_i$. This script allows us to find accurate enough values of $p$ for any chosen pair of prefix length and accumulated weight.

The sum of the weights at each depth of RBO always converges to a fixed value. This makes RBO an adequate candidate to compare infinite rankings without having the infinite tail’s importance dominating the finite head’s importance. When computing RBO for a given depth, even if this depth is the size of the compared rankings, the algorithm produces two results: $rbo_{min}$ and $rbo_{ext}$. The value $rbo_{min}$ is the overlapped score obtained after having checked the target rankings until depth $d$. $rbo_{ext}$ is the residual score that would have been added to the result in case the explored rankings had infinite but equal and equally sorted elements beyond depth $d$. With this, we can have that the max possible score for infinite lists is $rbo_{max} = rbo_{min} + rbo_{ext}$.

Then, RBO can be defined as a function $f_{RBO}(R, L, p, d) \rightarrow rbo_{min}$ and $rbo_{ext}$. It compares two rankings $R$ and $L$ until depth $d$, with a user persistence modeled by $p$, and returns a score $rbo_{min}$ in the range $[0, 1]$, and a residual $rbo_{ext}$ based on the assumption that $R$ and $L$ can have infinite elements.

The authors in [18] provide a formula to express RBO as a single point $rbo_{ext}$ instead of a range. This formula extrapolates the tendency observed until depth $d$ and assumes that it will stay stable along the infinite tail and provide an score $rbo_{ext}$ where $rbo_{min} \leq rbo_{ext} \leq rbo_{max}$.

In our experimentation, we will use $rbo_{ext}$ to obtain a single score point of similarity between two rankings. The rankings compared will always have the same number of elements, but the way in which ties are handled may cause that they do not have the same number of ranks. When two elements have identical score within the same ranking, they are both assigned to rank $k$, and the element after them is placed at rank $k + 1$. This means that the total number of ranks could be smaller than the total number of elements. In our experimentation we will always execute RBO with the longest possible depth, i.e., the depth of the ranking with the greatest number of ranks.

### Table 1

<table>
<thead>
<tr>
<th></th>
<th>Statistics about the DBpedia SPARQL endpoint logs used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nº of entries in the log</td>
<td>74,281,130</td>
</tr>
<tr>
<td>Nº of correct queries</td>
<td>74,265,325</td>
</tr>
<tr>
<td>Nº of queries which mention a class</td>
<td>59,073,904</td>
</tr>
</tbody>
</table>

4.2. Exploring DBpedia logs

We have been able to mine access logs of the DBpedia SPARQL endpoint of 14 different random days during 2017. Fourteen different files contain the logs. Each file includes every SPARQL request for a whole day to the endpoint\(^6\). The total combined size of the archives is 58.771GB. Some extra statistics about the log files are provided in Table 1.

We have computed the number of times that a class has been mentioned in the logs using the criteria previously described in this section. The scripts used to perform such a mining task are publicly available in a GitHub repository\(^7\).

Each line in the logs contains data related to a single request to the endpoint. The version of the logs that we were able to compute was filtered and anonymized to preserve users’ privacy. We could use the following information for each entry:

- Hashed IP from where the request was performed.
- HTTP request. SPARQL queries are embedded in GET requests.
- Timestamp of the request truncated to hour precision.
- HTTP status code (200 OK, 404 Not found, 5XX server error...).

4.2.1. Human and machine requests

Distinguish between requests performed by humans and requests performed by bots or applications that trigger a single/few queries. SPARQL queries, or performing small tasks in some applications that trigger a single/few queries. It cannot be strictly said that human traffic is a more reliable notion of importance than machine or total traffic. Nevertheless, the action of very few automatic

\(^6\)The logs are described and available to be downloaded at https://github.com/DaniFdezAlvarez/classrank/blob/develop/experimentation/doc/dbpedia/README.md#user-content-logs Accessed in 2020/03/28

\(^7\)https://github.com/DaniFdezAlvarez/classrank/tree/develop/experimentation/query_mining Accessed in 2020/03/28
agents represents a really notorious percentage of the total traffic in the logs. Just to provide some revealing numbers about this fact, the sum of requests performed by the 10 IPs with more associated requests is 32,321,486, i.e., 44% of the total of requests. If we consider the top-100 IPs, the number grows to 57,453,010 (77% of the total). Thus, the notion of importance purely based on human actions seems to adopt a wider and more generalist point of view, not so polarized by automatic voracious consumers of the endpoint.

When mining logs, there is not a perfect technique to distinguish between human and machine search sessions, or even to properly perform the task of identifying an isolated search session. Also, the most accurate approaches use to rely on information that we cannot access, such as user agent, more precise timestamps or user identifiers. For instance, authors in [19] distinguish between queries performed by humans, which they call organic queries, and queries performed by automatics processes, which they call robot queries. The SPARQL logs that they can access do not contain IPs, but they do contain user-agent. They use this field to detect browser user-agents, which are usually connected to organic queries, and some other agents linked to known apps used by humans.

In our scenario, we combined the information of hashed IP and timestamp to detect hosts that seem to have a human-like amount and rate of requests. We picked an arbitrarily low amount of requests by hour within a single day and consider every IP showing a request rate under that threshold to belong to a device generating human traffic. The chosen threshold has been 2.

We are aware that there are several situations in which this heuristic and this arbitrary threshold may cause false positives and false negatives, such as the following ones:

- A true human agent produces too many requests within a single day, which discards not just the request of that day, but also every log entry related to the same IP.
- A machine agent produces a low enough number of requests every day.
- Several humans are performing a reasonable number of requests at the same time using different devices, but they are in the same network, so the endpoint servers see them all as belonging to the same IP.

<table>
<thead>
<tr>
<th>Nº of requests</th>
<th>Human</th>
<th>Not-human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio of requests</td>
<td>0.13%</td>
<td>99.87%</td>
</tr>
<tr>
<td>Nº of different IPs</td>
<td>9,234</td>
<td>12,808</td>
</tr>
<tr>
<td>Ratio of IPs</td>
<td>41.89%</td>
<td>58.11%</td>
</tr>
</tbody>
</table>

Table 2

Number and quantity of requests and IPs in the logs

- A certain IP is linked to different routers on different days due to, for instance, Dynamic Host Configuration Protocol (DHCP) changes.

The mentioned issues are hard to prevent without more precise information about each log entry. However, the threshold have been chosen to pick IPs with high chances of belonging to a human agent by sacrificing recall but yet having a big sample of entries that can be representative of human behavior.

In Table 2, we provide the number and ratio of requests and different IPs related to humans or bots, respectively. One can see that there are many more queries not related to human behavior, but we found more than 93K requests coming from more than 9k different IPs associated with human behavior.

In order to avoid verbosity, along this paper we will use the following abbreviations:

- Human Hosts (HH) to refer to log entries associated to IPs showing human behavior.
- Machines Hosts (MH) to refer to the not associated to human behavior.
- Every Host (EH) to refer to all entries in the log.

4.3. Producing the different rankings

4.3.1. Reference rankings

We have produced two different rankings using the logs of DBpedia. One of them is the ranking of class usage considering EH entries with a valid SPARQL query. The second one contains just HH entries. Both rankings will be used as reference and will be compared to the rankings produced by each importance metric.

In Table 3, we include the top-20 elements of each reference list. The whole list, as well as the complete rankings produced by the considered techniques, are publicly available.

In this section, we describe how we computed the rankings for Degree, Betweenness, Bridging Centrality, Harmonic Centrality, and Radiality.

Except for Degree, whose complexity is linear to the number of nodes, all the aforementioned techniques need the computation of the shortest paths between all the nodes in the graph, which requires at least a computation time of $O(V \cdot (V + E))$ \cite{14}, being $V$ the number of nodes and $E$ the number of edges. Thus, these algorithms are hard to compute in a source such as the analyzed section of the English chapter of DBpedia, with more than 110M triples.

The authors in \cite{14} use these techniques to measure class importance by applying them to graphs containing only T-BOX statements. In our paper, we used the same approach. We applied the OTT over the version of the DBpedia ontology corresponding to the time in which the logs were generated\footnote{https://github.com/DaniFdezAlvarez/classrank/blob/develop/experimentation/doc/dbpedia/dbo.ttlAccessedin2020/03/28} to rank the classes. No A-BOX statements are taken into account.

### 4.3.2. OTT metrics

The PageRank and ClassRank rankings have been produced considering all the relations between both T-Box and A-Box elements in the target KG. Triples with literals were discarded. There was no need to deal with blank nodes since the target source did not contain any.

To build the ranking of classes of PageRank, we just filtered all the A-BOX elements in the obtained PageRank vector and sorted the remaining T-Box terms in decreasing order w.r.t. to its score. The damping factor for the PageRank execution was set to $\alpha = 0.85$, which is the most usual configuration of PageRank \cite{20}.

The ClassRank scores are built on top of the PageRank ones described in the previous paragraph, so the setting $\alpha = 0.85$ was used. When executing ClassRank the A-BOX statements were discarded. No A-BOX statements are taken into account.

### 4.3.3. AAT metrics

In this section, we describe how we computed the rankings for Degree, Betweenness, Bridging Centrality, Harmonic Centrality, and Radiality.

The authors in \cite{14} use these techniques to measure class importance by applying them to graphs containing only T-BOX statements. In our paper, we used the same approach. We applied the OTT over the version of the DBpedia ontology corresponding to the time in which the logs were generated\footnote{https://github.com/DaniFdezAlvarez/classrank/blob/develop/experimentation/doc/dbpedia/dbo.ttlAccessedin2020/03/28} to rank the classes. No A-BOX statements are taken into account.

### 4.3.2. OTT metrics

In this section, we describe how we computed the rankings for Degree, Betweenness, Bridging Centrality, Harmonic Centrality, and Radiality.

The PageRank and ClassRank rankings have been produced considering all the relations between both T-Box and A-Box elements in the target KG. Triples with literals were discarded. There was no need to deal with blank nodes since the target source did not contain any.

To build the ranking of classes of PageRank, we just filtered all the A-BOX elements in the obtained PageRank vector and sorted the remaining T-Box terms in decreasing order w.r.t. to its score. The damping factor for the PageRank execution was set to $\alpha = 0.85$, which is the most usual configuration of PageRank \cite{20}.

The ClassRank scores are built on top of the PageRank ones described in the previous paragraph, so the setting $\alpha = 0.85$ was used. When executing ClassRank the A-BOX statements were discarded. No A-BOX statements are taken into account.

### 4.3.3. AAT metrics

The PageRank and ClassRank rankings have been produced considering all the relations between both T-Box and A-Box elements in the target KG. Triples with literals were discarded. There was no need to deal with blank nodes since the target source did not contain any.

To build the ranking of classes of PageRank, we just filtered all the A-BOX elements in the obtained PageRank vector and sorted the remaining T-Box terms in decreasing order w.r.t. to its score. The damping factor for the PageRank execution was set to $\alpha = 0.85$, which is the most usual configuration of PageRank \cite{20}.

The ClassRank scores are built on top of the PageRank ones described in the previous paragraph, so the setting $\alpha = 0.85$ was used. When executing ClassRank the A-BOX statements were discarded. No A-BOX statements are taken into account.
Rank, the set of target classes is known a priori: all classes contained in the DBpedia ontology. As a consequence, there was no need to perform class discovery in stage 2 of Algorithm 1. The property rdf:type was the only class-pointer considered. Since the only property linking an A-BOX term with any element in the DBpedia ontology is rdf:type, this is a straightforward decision in the context of our experiment. The same decision was taken in order to compute IC, i.e., the only property that we considered to link an instance to its class is rdf:type.

4.4. Results

To evaluate the similarity of two rankings \( R \) and \( L \) using different weights for their top positions, we tested different configurations of \( p \) in \( RBO(R, L, p) \). We used Equation 1 to obtain \( p \) values for different prefix sizes with a fixed importance of 0.9. In every case, the margin error to calculate \( p \) with our script was set to \( \alpha = 0.00001 \).

The \( p \) value for each prefix size appears in Table 4. Note that each \( p \) configuration solves Equation 1 for several pairs of \( W_{RBO}(1 : d) \) and \( d \). For instance, \( p \approx 0.876343 \) is an adequate value for \( d = 10 \) and \( W_{RBO}(1 : d) = 0.9 \), but it is also valid for \( d = 6 \) and \( W_{RBO}(1 : d) = 0.8 \). We fixed \( W_{RBO}(1 : d) = 0.9 \) just to provide understandable values of \( p \) instead of testing arbitrary increments.

In Figure 1 we show the comparison between all the considered class importance metrics and the EH ranking. In Figure 2 we show the same comparison against the HH ranking. In both cases, the x-axis indicates the prefix size used, and the y-axis indicates the similarity score with RBO.

Figures 1 and 2 present different values for the prefix size. The values have been selected w.r.t. the nature of the rankings produced by their respective content log.

When building the rankings of class mention in the logs, some elements receive the same number of mentions, especially in low spots of the ranking. These ties, as explained in section 4.1 can cause that the number of ranks in a ranking can be lower than the number of total elements to rank.

The total number of classes to rank is 827. When considering EH entries, tied elements produce a ranking with 544 spots. To evaluate the performance of the techniques at different depths, we compared them all starting at the minimum length of 10 and increased 10 spots until reaching the arbitrary depth of 200. This depth covers more than 1/3 of the 544 ranks.

When it comes to the HH entries, there are more ties. Several classes are not even mentioned in a single query. This produces a ranking composed of just 62 different positions. For this reason, we chose a different max depth to evaluate the different approaches against this ranking. We started at the minimum prefix length of 10 and increased 5 spots until the arbitrary depth of 60. This depth covers almost all the different ranks and includes almost every class with at least one mention (there are 583 elements tied at 62nd with no mention at all).

The facts observed in Figures 1 and 2 will be developed in the following subsections.

4.4.1. OTT approaches vs AAT approaches

AAT techniques clearly outperform the OTT ones. The only exception to this observation can be found with a prefix length of 10 in HH entries. In this scenario, Betweenness, with a score of 0.187, slightly outperforms PageRank, with a score of 0.172. However, PageRank performs better than Betweenness at every other computed prefix length. Besides, the differences between these two kinds of approaches seem to increase as long as we keep analyzing deeper positions of the rank. All the techniques seem to improve when exploring deeper prefixes. However, in both figures, OTT approaches seem to show a constant but slow linear improvement. By contrast, AAT ones start showing a faster improvement with the first increments of prefix size. At some point, AAT approaches stabilize and show the same linear progression that OTT ones. This happens around the top-100 in Figure 1 and around top-40 in Figure 2.

4.4.2. Performance of OTT approaches

Betweenness is the best performing technique among the OTT group. However, the performance of Be-
Figure 1. Line chart comparing the centrality metrics with the class mention in the whole log
Figure 2. Line chart comparing the centrality metrics with the class mention in the HH-filtered log
tiveness is still far from the performance of the AAT approaches, except for the mentioned measure taken at the top-10 of HH entries.

Despite being the simplest idea of them all and the cheapest from a computational point of view, Degree outperforms every other technique except Betweenness at every considered depth. As can be seen in Figures 1 and 2, this difference is more noticeable when considering the top spots of the ranking. When exploring deeper positions, the techniques yet outperformed by Degree reach scores closer to this approach.

The tendency observed in Figure 1 indicates that Radiality may even outperform Degree when exploring sections of the ranking deeper than the top-200. This situation seems different in Figure 2, where, despite the tendency is similar, the HH log has been nearly completely explored due to tied elements at the depths of 60, 61 and 62.

### 4.4.3. Performance of AAT approaches

As can be seen in Figures 1 and 2, ClassRank outperforms PageRank and IC in the two experiments at almost any considered depth. The only exception happens with a prefix length of 10 in the HH ranking, where IC gets a score of 0.24450 against the slightly lower score of 0.24446 reached by ClassRank.

The performance comparison between PageRank and IC is different for the HH and EH rankings. In the case of EH, IC slightly outperforms PageRank until depth 80, but PageRank works better from that depth to 200. Nevertheless, both techniques show very similar performance at any depth. The average relative difference between them is 1.27%.

When considering just the HH entries, IC significantly outperforms PageRank at every depth, with an average relative improvement of IC over PageRank of 10.71%. In this case, the performance of IC is closer to ClassRank. ClassRank has an average relative improvement of 5.88% over IC. This improvement is never lower than 3.65%, which happens at depth 60 (longest prefix). The greatest relative improvement is 11.42%, and it happens at depth 10 (shortest prefix). As can be seen in Figure 2, the absolute difference is quite similar at every depth. Notable changes in relative differences happen because both techniques improve when exploring longer prefixes with a similar curve, so the initial distance, which keeps similar for every depth, is less relevant when the scores are higher. The average score difference is 0.0209 (over a maximum of 1), with a standard deviation of 0.0036.

### 5. Discussion

#### 5.1. Reliability of the considered techniques

Three different groups of techniques have been analyzed regarding the amount of information used to measure class relevance.

The first group corresponds with techniques that make use of log information. In this group, we include the measurement of class mention as it has been described in this paper. For the case of our study, we have been able to access a random sample of the actual logs with anonymized or filtered fields to avoid privacy issues, which has caused troubles in order to properly distinguish human search sessions. However, this information is not usually public, not even in this filtered format, which is an obvious downside to apply this kind of metric.

The second group consists of AAT approaches. These kinds of techniques assume that the KG’s topology itself can be an objective indicator of class importance.

The OTT techniques constitute the third group, which compute the subgraph of T-BOX elements to produce a rank of classes. The techniques included in this group could be adapted and executed over the whole graph structure since they are all general techniques of graph theory not necessarily linked to the T-BOX domain (or any semantic domain at all). However, except for Degree, they all have a huge computational cost since they need to calculate the shortest path between every node in the graph.

It seems clear that the AAT techniques perform better than the OTT ones. Even with that, when applying RBO, it seems also that there is not a high similitude between any technique and the reference rankings until we gave enough weight to deep regions of the ranking. When 90% of the importance is given to the top-10 elements, ClassRank, the best-performing technique, has just a score of 0.24445 with EH and of 0.2272 with HH.

In table 5, we show the top 20 elements according to the two best performing approaches, which are ClassRank and IC, to perform a qualitative analysis of these elements. As one can see comparing Tables 5 and 3, despite the rankings seem to contain elements related to very similar domains (mainly arts, sports, geopolitical divisions, and people), the amount of shared elements between them is moderate. In table 6, we show the number of observed matches for the top-10 and the
top-20 of these two techniques with the reference standards. As can be seen, both approaches have similar numbers against EH entries. When they are compared to the HH ranking, ClassRank slightly improves the overlaps of IC.

The number of shared classes in the top-20 positions is not greater than 50% of the elements. Nevertheless, the similarity between these techniques and the reference rankings, especially in the case of ClassRank, keeps constantly improving as long as we keep giving importance to wider prefixes. The techniques share the assumption that there is a correlation between the links of a certain element and its actual importance. This experiment raises two main conclusions about this. On the one side, it seems that this correlation exists, but it is not exact. There is not such a thing as a perfect alignment between what is better connected and what is most used by the users, not mattering the chosen technique. On the other side, it seems that, even if there is not a perfect alignment, the wider is the amount of top elements considered, the stronger the correlation becomes. According to RBO, with a prefix length of 200 over a total of 827 classes, ClassRank reaches a similarity of 0.70. One may conclude that, in the conditions of our experiments, both the reference rankings and ClassRank detect a similar set of important classes, but they do not sort them equally. The more importance we give to wider prefixes, the less significant are the differences between them.

### 5.2. Comparison between the two best performing techniques

Although ClassRank performs better than any other considered technique, IC, the second-best performing technique, consists of a much simpler idea that requires less computational power. IC is a simple algorithm whose execution time may depend on how easy it is to retrieve the instances of a given class in the studied source, but whose base complexity is $O(n)$, where $n$ is the number of classes. The ClassRank algorithm is built on top of PageRank. Once the PageRank scores have been calculated, the complexity of ClassRank is $O(n)$ as well. Nevertheless, the computation of the PageRank scores takes an extra $O(m + o)$, where $m$ is the total number of nodes and $o$ the total number of edges. This can be partially compensated with parallel implementations of PageRank [21, 22]. However, it is interesting to evaluate whether the improvement of ClassRank is worth the computational cost.

According to [23], when comparing two techniques, improvements of less than 5% could be discarded and attributed to the nature of the samples chosen in the experiments; improvements between 5% and 10% are noticeable; improvements greater than 10% can be considered material.

Attending to these criteria, we compared the performance of ClassRank and IC with every considered prefix length and checked the importance of that improvement. The results are shown in Figure 3. As can be seen, the general case is that ClassRank has a noticeable improvement over IC, but rarely a material one. The most extreme cases happen at the top-10 of the different lists. Considering EH, IC slightly improves the score of ClassRank, but both marks are nearly identical. By contrast, when considering HH, ClassRank performs materially better than IC, but this relative difference tends to decrease when we give importance to
wider prefixes. From top-35, it stops being noticeable.
The natural tendency observed when considering EH is the same, but the improvement remains noticeable until depth 190.
When considering EH entries, the average relative improvement at the considered depths is 6.05%. When we limit the study to HH, the average improvement is 5.88%. In both cases, we can say that the improvement is noticeable. Nevertheless, in order to obtain conclusions that could be extrapolated to different domains, sources, and KG structures, more experimentation is required.
However, it is possible to make a qualitative analysis of the main differences between both ranking strategies, whose conclusions can be generalized. As it can be seen in Table 5, the tops of ClassRank and IC share many elements. 14 elements appear in the top-20 of both lists, and some of them have an identical rank. Both metrics are built on top of centrality notions. IC consist of a simple counting of a certain type of links. By contrast, ClassRank accounts not just for the quantity but also the importance of those links. Many incoming links coming from many instances ensure high importance with IC, but they will also cause high importance with ClassRank as well.
A couple of missing elements in the top-20 of IC raise a clue about the kind of classes in which these two techniques heavily disagree: `dbo:SoccerClub` and `dbo:Country`. With ClassRank, `dbo:SoccerClub` ranks 5th and `dbo:SoccerPlayer` ranks 6th. The different soccer players are connected to their club, so some of the importance accumulated by all soccer players goes to their respective teams. With this, even if there are much fewer instances of clubs (21,955) than players (117,619), these two classes can achieve a similar final score of importance. By contrast, `dbo:SoccerClub` descends to position 34th with IC.
The case of `dbo:Country` is more revealing. Instances of countries are frequently key elements to link different topics in cross-domain KGs such as DBpedia. Many different kinds of individuals can be linked to their country, such as people, smaller administrative divisions, or events. With ClassRank, that accumulated importance is good enough to rank 8th. `dbo:Country` is one of the top seeds according to the reference rankings as well (9th with HH and 21st with EH). By contrast, `dbo:Country` descends to 145th with IC.
Similar examples are `dbo:MusicalGenre` (19th with CR vs 22nd with IC), `dbo:Legislature` (42nd vs 192nd), and `dbo:Language` (24th vs 85th). In general, we can say that IC and ClassRank produce rankings that tend to be quite similar. However, ClassRank can capture the importance of classes that do not have too many instances when those instances are really important elements of the KG.

6. Related work

6.1. Importance or relevance of entities or classes in KGs

Several authors have already used centrality metrics to determine entity importance or relevance in KGs, sometimes as a previous task to achieve some other goals such as ontology summarization.
In [14], a study of different techniques to detect class importance is performed. They check the performance of Degree, Betweenness, Bridging Centrality, Harmonic Centrality, Radiality and Ego Centrality against a gold standard built using DBpedia logs in the same manner that we do in this paper. In this case, the authors do not experiment with any technique using A-BOX knowledge. The study is a preliminary stage in order to support graph summarization processes. There are two main differences between this study and ours. First, they do not consider any AAT technique. Second, they use Spearman correlation coefficient to determine the similarity between the rankings, so the top and the tail of each ranking have the same weight on the final results.
Authors in [10] perform another general study of class importance as a previous step for ontology summarization. Again, all the techniques used are applied over the ontologies, not using A-BOX knowledge at any point.
In [11], another case of measuring class importance for ontology summarization is presented. They propose two methods, one inspired in Degree, which is defined in [24], and another one inspired in Closeness. However, those methods require some user input, such as relevant domain-specific relations in ontologies or weights for certain elements.
RDFDigest+ is a tool to perform RDF/S Knowledge Base exploration using summaries [12]. To identify the most important nodes of a source, it allows the user to choose and combine many different centrality algorithms by normalizing their scores. It also uses information related to the instances of each schema element. The adaptation and combination of these algorithms is called Adapted Importance Measure (AIM).
In [25], the authors perform a study of several cross-domain KGs quality, including DBpedia and four more
Figure 3. Relative performance improvement of ClassRank over instance counting

<table>
<thead>
<tr>
<th>Prefix length</th>
<th>Relative improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-0.02%</td>
</tr>
<tr>
<td>20</td>
<td>4.72%</td>
</tr>
<tr>
<td>30</td>
<td>6.69%</td>
</tr>
<tr>
<td>40</td>
<td>7.39%</td>
</tr>
<tr>
<td>50</td>
<td>7.60%</td>
</tr>
<tr>
<td>60</td>
<td>7.96%</td>
</tr>
<tr>
<td>70</td>
<td>7.71%</td>
</tr>
<tr>
<td>80</td>
<td>6.97%</td>
</tr>
<tr>
<td>90</td>
<td>6.75%</td>
</tr>
<tr>
<td>100</td>
<td>6.10%</td>
</tr>
<tr>
<td>110</td>
<td>5.88%</td>
</tr>
<tr>
<td>120</td>
<td>5.67%</td>
</tr>
<tr>
<td>130</td>
<td>5.47%</td>
</tr>
<tr>
<td>140</td>
<td>5.27%</td>
</tr>
<tr>
<td>150</td>
<td>5.08%</td>
</tr>
<tr>
<td>160</td>
<td>4.89%</td>
</tr>
<tr>
<td>170</td>
<td>4.64%</td>
</tr>
<tr>
<td>180</td>
<td>4.39%</td>
</tr>
<tr>
<td>190</td>
<td>3.47%</td>
</tr>
<tr>
<td>200</td>
<td>3.26%</td>
</tr>
</tbody>
</table>

10% http://people.aifb.kit.edu/ath/ Accessed in 2020/03/28

sources. One of the features studied is class coverage for different knowledge domains. They manually classify each class to belong to the different domains. Then, they measure the importance that each class provides to each domain by counting their instances.

In [26], PageRank is applied over a graph of Wikipedia linked entries. Each entry is represented by its DBpedia URI, so they produce a ranking of DBpedia entities based on the Wikipedia link graph. In combination with other methods, the results are being used for entity summarization [27, 28]. In order to merge the information of different Wikipedia chapters, the authors compute a ranking of Wikipedia entities using their Wikidata URI, which is unique for all the languages. In these works, PageRank is used as a base metric to rank entities in a KG. However, they use a voting method w.r.t. different Wikipedia chapters. Thus, this technique cannot be applied over KGs whose elements are not linked (directly or indirectly) to Wikipedia pages.

In [29], an approach to rank classes in DBpedia is presented. As well as ClassRank, this work is also based on aggregation of PageRank scores. Nevertheless, these scores are not obtained from DBpedia’s structure, but each entity receives the PageRank score of its associated page in Wikipedia. Then, this technique cannot be applied over KGs whose entities are not linked to Wikipedia. Also, the authors combine the aggregation of PageRank scores with some other parameters such as Instance Counting.

In [30], PageRank is applied over the DBpedia link structure to mine significant concepts. Given an element in DBpedia, the authors track its most related concepts by exploring its neighborhood in the graph, and they rank those results according to inverse PageRank. They consider that the most related elements are the ones with a lower PageRank score. The authors argue that elements with low PageRank are not so well connected because they are too specific of a given topic. Hence, those URLs in the neighborhood of a concept c with low PageRank may have higher chances of being semantically closer to c than those with high PageRank, which may be too transversal.

Freebase associates a score with ranking purposes to each one of its stored entities. However, this score is not computed with PageRank, but using a simpler formula based on link counts of an entity in Freebase KG and its associated page in Wikipedia [31].

Wikidata Project maintains some special pages offering some metrics of the graph that are frequently up-
determined. Among these results, link counts of the most used elements can be found, but there are no reports about class importance or PageRank-like scores of any element.

6.2. Alternative centrality measures based in PageRank

The idea of using personalized versions of PageRank to adapt the algorithm to different contexts was early suggested in [13, 32]. Since that proposal, many PageRank adaptations have been published. Probably the closest adaptations to our domain are those that compute aggregations of PageRank scores. A representative example of this strategy is BlockRank [33], which divides the target graph into several disjoint blocks of smaller units. An illustrative use of BlockRank-like strategies is HostRank [34], which was thought to be applied over web structures. Nevertheless, most BlockRank-like approaches are not compatible with our domain, since in RDF graphs the very same entity may be an instance of several different classes. Hence, hypothetical blocks formed by instances of the same class would not be disjoint.

Most of the PageRank adaptations have been thought to measure the relevance of an element in a KG w.r.t. a query [35]. These approaches are focused on information retrieval tasks and tend to rank entities using notions of semantic relatedness between query and resource. Some of them measure importance and are used in combination with other notions to produce some result [1, 36]. Some others measure relevance, including strategies such as text similarity or exploration of topic sub-graphs in the algorithm itself [37–41].

OntologyRank [1] is designed to rank Semantic Web Documents (SWD), such as ontologies or RDF files, which are linked to each other through their internal elements. The algorithm uses the semantics of the properties to divide them into four different categories. Then, it computes a version of PageRank where each link can be weighted w.r.t. each category. Although it follows a strategy of aggregation of PageRank-like scores, OntologyRank is designed to rank different SWDs instead of elements within a single SWD.

PopRank [36] is an adaptation of PageRank designed to be applied over a network of objects. It combines two factors to obtain the popularity of an object: a weighted PageRank in which every property has its own weight, and the PageRank of the database/web page which contains the object (Web Popularity). PopRank is thought to assign a score to every entity in the graph, i.e., there is no aggregation or grouping of individuals in some class or cluster, so the algorithms have different domains of application. Also, PopRank has a stage in which some training data should be provided by experts, which may be too costly in graphs with many properties such as DBpedia or Wikidata.

ReConRank [37] is a PageRank adaptation designed to be applied over RDF domains that combines the approaches of ResourceRank and ContextRank. ReConRank is closely related to search and retrieval domains. The ranking of entities is not applied over the whole target graph but over a sub-graph composed of certain elements that are related enough to some keywords. The scores produced are a measure of relevance w.r.t. a query instead of importance.

RareRank [38] makes use of transition scores between entities, as well as PageRank does. However, it proposes a Rational Research model to define transitions between elements aiming to simulate a human strategy of jumping from one document to another. RareRank is thought to be applied in semantic search of research documents. It relies on meta-data associated with scientific papers modeled in an ontological way, as well as topic relatedness computed with Latent Dirichlet Allocation [42]. As well as ReConRank, RareRank produces scores of relevance instead of importance. Also, the model of Rational Research should be adapted to apply it in domains different from scientific documents.

DBpediaRanker [39] describes an algorithm to rank DBpedia entities w.r.t. a query. In this case, the authors do not follow a PageRank-like approach, but they consider several different notions of similarity. This includes textual similarity, proximity to a certain set of seed nodes, or results supported by external resources, such as search engines or tagging systems. Thus, although the main goal is also the ranking of RDF resources, this approach has a specific domain of application and cannot be used to measure class importance.

TripleRank [40] is a HITS-based algorithm to rank entities w.r.t. a subject and a facet (predicate) in RDF environments. TripleRank gives a notion of relevance w.r.t. some other graph elements instead of importance per se.

DWRank [41] ranks ontology concepts in search and retrieval environments. It combines three types of
notions to rank a given element: text similarity with a query, hub score within its own ontology using a reversed PageRank function, and authority of its ontology w.r.t. the rest of ontologies. The goal of the algorithm is to rank ontology members, and it works purely with T-Box elements, i.e., it does not use any instance information to produce its results.

7. Conclusions and future work

In this paper, we have evaluated different techniques to measure class importance in KGs based on the KG’s structure. To compare the approaches, we have performed experiments using logs of the official DBpedia SPARQL endpoint. We have elaborated rankings of importance based on the mentions of the classes in the logs and measured their similarity with the selected techniques with Ranking Biased Overlap.

The experiments raise several conclusions. The techniques that consider just T-BOX statements are outperformed by techniques that compute A-BOX knowledge as well. Among all the analyzed techniques, ClassRank, a novel approach, outperforms the rest of the proposals under the conditions of our experiments in terms of similarity with the reference rankings. Instance Counting has been detected as the second-best performing technique. Its results are not far from the results obtained by ClassRank, and it is computationally cheaper. However, Instance Counting is not able to catch the importance of really well-connected elements that do not have too many instances, such as the class dbo:Country in our target source.

As future work, we plan to extend this experimentation using different logs of different public KGs, including domain-specific sources or cross-domain ones such as DBpedia.

8. Acknowledgments

We would like to thank OpenLink Software for letting us accessing the log files used in our experimentation and for the operation of the DBpedia SPARQL endpoint. This work is supported by the Severo Ochoa Research program, 2017 call, exp. number BP17-88.

References


