Computer Science and Information Systems 17(1):229–252 https://doi.org/10.2298/CSIS180717038L

Visualization of path patterns in semantic graphs ⋆

José Paulo Leal

CRACS & INESC-Tec LA, Faculty of Sciences, University of Porto
Porto, Portugal
zp@dcc.fc.up.pt

Abstract. Graphs with a large number of nodes and edges are difficult to visualize. Semantic graphs add to the challenge since their nodes and edges have types and this information must be mirrored in the visualization. A common approach to cope with this difficulty is to omit certain nodes and edges, displaying sub-graphs of smaller size. However, other transformations can be used to summarize semantic graphs and this research explores a particular one, both to reduce the graph's size and to focus on its path patterns.
A-graphs are a novel kind of graph designed to highlight path patterns using this kind of summarization. They are composed of a-nodes connected by a-edges, and these reflect respectively edges and nodes of the semantic graph.
A-graphs trade the visualization of nodes and edges by the visualization of graph path patterns involving typed edges. Thus, they are targeted to users that require a deep understanding of the semantic graph it represents, in particular of its path patterns, rather than to users wanting to browse the semantic graph's content. A-graphs help programmers querying the semantic graph or designers of semantic measures interested in using it as a semantic proxy. Hence, a-graphs are not expected to compete with other forms of semantic graph visualization but rather to be used as a complementary tool.
This paper provides a precise definition both of a-graphs and of the mapping of semantic graphs into a-graphs. Their visualization is obtained with a-graphs diagrams. A web application to visualize and interact with these diagrams was implemented to validate the proposed approach.
Diagrams of well-known semantic graphs are presented to illustrate the use of a-graphs for discovering path patterns in different settings, such as the visualization of massive semantic graphs, the codification of SPARQL or the definition of semantic measures.
The validation with large semantic graphs is the basis for a discussion on the insights provided by a-graphs on large semantic graphs: the difference between a-graphs and ontologies, path pattern visualization using a-graphs and the challenges posed by large semantic graphs.

Keywords: Semantic Graph Visualization, Semantic Graph Summarization, Linked Data Visualization, Path Pattern Discovery, Semantic Graph Transformation

1. Introduction

Graphs, like most mathematical entities, are inherently visual. In fact, our mathematical intuition relies heavily on our ability to visualize angles, functions, vectors or geometric figures. It fails us, for instance, when we try to visualize a hypercube. However, the

⋆ This is an extended version of an article presented at SLATE'18 [16].

projection of multidimensional solids in 3- or 2-dimensional spaces gives us an idea of these entities' shape. The visualization of the hypercube is an apt metaphor of the key insight that drives this research: the understanding of a complex entity may be improved by looking at the shadow it casts.

Graphs have a particular role in visualization since they are the data model of most diagrams. Diagrams enrich graphs in two ways: a graphical syntax for nodes and edges; and the layout of nodes and edges on a surface. Different kinds of diagrams have been developed for many purposes. These diagrammatic languages are used for modeling and visualizing relationships among entities, even when they are purely abstract.

In spite of their ability to show relationships and symmetries, large diagrams are difficult to visualize. Too many nodes and too many entangled edges reduce our perception on the underlying graph. This is particularly the case of the semantic web, where graphs are growing increasingly larger and denser. Section 2 presents different attempts to provide visualizations of large semantic graphs, with efficient approaches to process large quantities of data and methods to abstract them, mostly by omitting nodes and edges with certain features. These diagrams represent the graphs themselves, and the layout may highlight general properties, such as symmetry, but usually they do not reveal specific features such as patterns formed by nodes and edges.

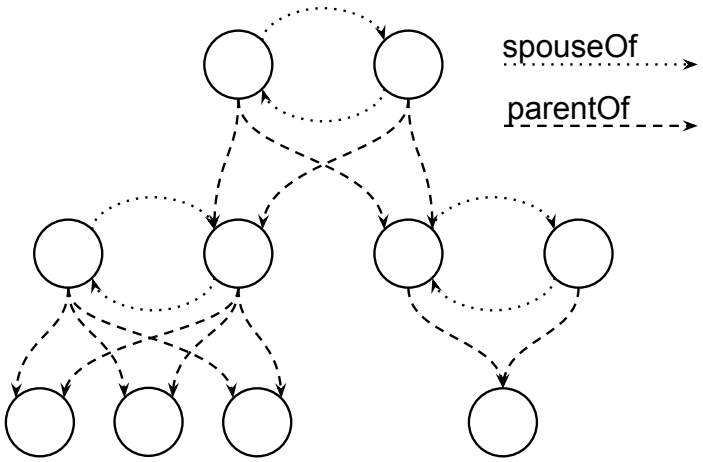

Fig. 1. Immediate family relationships graph

Consider the diagram in Figure 1 that depicts immediate family relationships. Nodes represent persons and there are two types of edges: the dotted edges represent the *spouseOf* relationship and the dashed edges represent *parentOf*. The edges in this graph can be summarized by the *a-graph* in Figure 2 that represents edge *types* as nodes. This kind of graph resembles *line graphs* [12] that represent adjacencies between edges of a graph, although a-graphs represent adjacencies between edge types. For that reason, the corresponding a-graph is much smaller than the original graph. Moreover, even if the number of nodes and edges increases in the original graph, the structure of the a-graph may remain unchanged.

For instance, if the immediate family relationship graph was extended to consider all the humans that ever lived, the corresponding a-graph would still be similar to the one depicted in Figure 2.

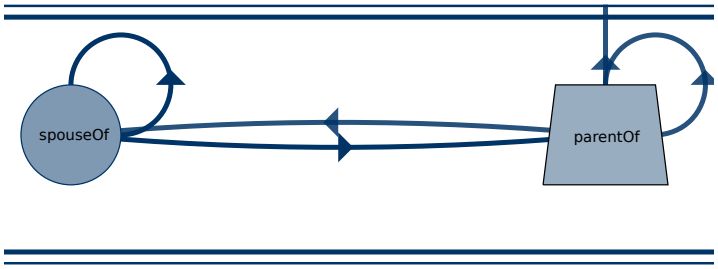

Fig. 2. Immediate family relationships a-graph

The novelty of the a-graph approach is the abstraction of large semantic graphs into a much smaller graph highlighting its *path* patterns. The abstraction process consist on mapping semantic graphs into a-graphs, a particular kind of graph with an associated diagram type. Sets of edges with the same type are mapped into nodes and sets of nodes into edges. Section 3 defines a-graphs and their relationship with semantic graphs. It also introduces the a-graph diagrams used for their visualization and provides small examples of this kind of diagram. The final subsection compares a-graphs with other forms of representing the properties of semantic graphs, such as ontologies.

An implementation of the mapping and diagram layout is described in Section 4. It is deployed as a web application for browsing a-graphs and exporting them as vector images, such as the diagram shown in Figure 2. It is available online[1] and is useful to validate the proposed approach. Section 5 presents examples of diagrams produced with this tool to illustrate different techniques to create meaningful visualizations of large semantic graphs and discover relevant path patterns. Section 6 discusses the relationship between a-graphs and ontologies, the efficacy and efficiency of a-graphs to display path patterns and the ability and limitations of the current implementation to manage large semantic graphs. The last section summarizes the a-graph proposal, highlights its main contributions and identifies opportunities for future research.

2. Related Work

A-graphs are abstract representations of semantic graphs and thus related to *graph summarization* [18]. This research field covers a wide variety of graph types, from plain graphs described by an adjacency matrix to dynamic graphs described by data streams, encompassing also labelled graphs. Semantic graphs are *typed* and types of nodes and edges can be seen as labels.

The most popular approaches to graph summarization involve compression and grouping techniques. The former is inspired by information theory concepts, namely minimum

[1] `http://quilter.dcc.fc.up.pt/antigraph`

description length (MDL), to detect and compress frequent structures in graphs. The latter aggregates nodes into supernodes connected by super edges to create a new structure, a supergraph. In the case of labelled graphs, grouped nodes are structurally close to each other. Many of these graph summarization approaches are database-centred and only a few emerged from semantic graphs [18]. For instance, Song et al. [19] proposed an approach to summarize knowledge graph characterized by graph patterns, called *d*-summaries, that produce a supergraph as summarization. None of these approaches reverses edges for nodes.

Graphs summarization has several applications. Some of these applications are outside the scope of this research, such as to reduce data volume in storing of massive graphs, to speedup graphs algorithms and queries. Other applications are in tune with the objectives of a-graphs, such as to reduce noise in graphs or to support interactive visualization of massive graphs. However, most approaches to semantic graph visualization described in the literature do not rely on graph summarization.

Knowledge bases such as WordNet[2] [8], Yago[3] [15] and DBpedia[4] [4], have a massive amount of information. A typical representation of these knowledge bases are node-link multigraphs, where each node has a type and nodes are connected by links representing the relationship between them.

A convenient way to analyze this data is using data visualization. The most common type of visualization is focused on the analysis of resources, in particular, those with a high outdegree. The main challenge of semantic graph visualization and management is related to the graph size. This type of graphs has several thousands of nodes and edges and is usually very dense.

The literature presents several approaches to handle the visualization and management of node-link graphs. Most of the related work on massive graphs visualization is handled through hierarchical visualization. This type of approach has low memory requirements, however, it depends on the characteristics of the graph. The graph hierarchy can be extracted using different kinds of methods. Tools such as ASK-GraphView [1], Tulip [3] and Gephi [5] explore clustering and partitioning methods, creating an abstraction of the original graph, using graph summarization, that is easier to visualize. Another technique used to build hierarchies is based on the combination of edge accumulation with density-based node aggregation [21]. Visual complexity can also be reduced by hub-based hierarchies, where the graph is fragmented into smaller components, containing many nodes and edges, making meta nodes, as described in [17]. GrouseFlocks [2] allows users to manually define their own hierarchies.

There are specific tools when the semantic graph is in Resource Description Framework (RDF) format, however, they require loading the full graph. Some desktop-based tools, such as Protégé [5] and RDF Gravityare mainly used with purpose of aiding developers to construct their ontologies, providing also complex graph visualizations. Of all available tools for linked data visualization the most notable ones are the following. Fenfire [13] is a generic RDF browser and editor that provides a conventional graph representation of the RDF model. The visualization is scalable by focusing on one central

[2] `https://wordnet.princeton.edu/`
[3] `https://www.mpi-inf.mpg.de/yago-naga/yago`
[4] `http://wiki.dbpedia.org/`
[5] `http://protege.stanford.edu/`

node and its surroundings. RelFinder[6] [14] is a tool that extracts from a Linked Open Data (LOD) source the graph of the relationships between two subjects. It provides an interactive visualization by supporting systematic analysis of the relationships, such as highlighting, previewing and filtering features. ZoomRDF [20] is a framework for RDF data visualization and navigation that uses three special features to support large scale graphs. It uses *space-optimized* visualization algorithms that display data as a node-link diagram using all visual space available. *Fish-eye zooming* is another feature that allows the exploration of selected elements details, while providing the global context. The last feature is the *Semantic Degree of Interest* assigned to all resources that consider both the relevance of data and user interactions. LODeX [6] produces a high-level summarization of a LOD source and its inferred schema using SPARQL endpoints. The representative summary is both visual and navigable. The platform graphVizdb[7] [7] is a tool for efficient visualization and graph exploration. It is based on a *spatial-oriented* approach that uses a disk-based implementation to support interactions with the graph.

3. A-graph definition

The most distinctive feature of a-graphs is that nodes and edges are reversed relatively to the semantic graphs that generated them. Subsection 3.1 explains the motivation behind this decision and characterizes the main components of a-graphs, namely a-nodes and a-edges, as well as their features.

The proposed approach to the visualization of semantic graphs can be divided into two parts. Firstly, the semantic graph is abstracted to another graph – the *a-graph* – that promotes types of edges. Secondly, this abstracted graph is visualized using a special kind of diagram – the *a-graph diagram* – that emphasises path patterns. The following two subsections detail each facet of the a-graph approach.

3.1. Motivation

Nodes have the main role in a graph. Edges connect nodes and establish relationships among them. The goal of a-graphs is to abstract a given graph, highlighting edges and reducing its size. Hence, in an a-graph nodes and edges are reversed, i.e. an *a-node* abstracts edges and an *a-edge* abstracts nodes.

It is important to note that an a-node abstracts an edge *type* rather than a single edge. Hence the order (the number of nodes) of an a-graph is in general much smaller than that of the graph it abstracts. For instance, the graph of WordNet 2.1 has about 2 million edges with 27 edge types, hence 27 is the order of the reductions that abstract it.

An a-edge expresses a relationship between a pair of a-nodes, namely that the edge types it represents can be connected to form a length 2 path. Two edges form a length 2 path when the target of the first is the source of the second. Since an a-edge represents a set of nodes, the size (the number of edges) of an a-graph is much smaller than the size of the graph it abstracts. Considering that a-edges can be loops, the number of a-edges is less or equal to n^2, where n is the number of a-nodes. For instance, the size of the WordNet

[6] `http://www.visualdataweb.org/relfinder.php`
[7] `graphvizdb.imis.athena-innovation.gr/`

2.1 graph is about half a million but the size of its a-graph is only 214, well below the maximum of $27^2 = 729$.

The expressiveness of a-nodes and a-edges is increased by adding weights to them. The weight of an a-node is the percentage of edges with the type it abstracts. For instance, if a graph has half of its edges of type t then the a-node reflecting t has weight $1/2$. Hence, a-nodes with higher weight reflect edge types that are more frequent in the graph. Obviously, the sum of a-node weights must be 1.

By the same token, the weight of an a-edge is the percentage of nodes that participate in length 2 paths involving edge types they have as source and target, respectively. For instance, if an a-node reflects the edge type t_1 and another the edge type t_2, and $1/3$ of the nodes are target of t_1 and source of t_2, then the weight of the a-edge $t_1 \rightarrow t_2$ is $1/3$.

One would expect every node to be reflected by an a-edge, but for that to happen the nodes that are just sources (not the target of any edge) or just targets (not the source of any edge) must also be abstracted by a-edges. To ensure that all nodes are reflected by a-edges it is necessary to introduce two special a-nodes: *bottom*, denoted as $\perp$; and *top*, denoted by $\top$. The bottom a-node represents a nonexisting edge type that would come before the start of a path. Conversely, the top a-node represents a nonexisting edge type that would come after the end of a path. Both special a-nodes have weight 0, thus maintaining the invariant that the sum of all weights is 1.

The two special a-nodes – bottom and top – allow the definition of a-edges that abstract nodes that are only source or target of edges. These a-nodes are considered *special* to differentiate them from *regular* a-nodes, that have an associated edge type. The a-edge $\perp \rightarrow t$ abstracts the nodes with a null indegree that are sources of edges with type t, and the a-edge $t \rightarrow \top$ abstracts the nodes with a null outdegree that are targeted by edges of type t.

In fact, both the indegrees and outdegrees of nodes must be taken into consideration in the weight of all a-edges. Consider a node n with indegree 2 and outdegree 3. For instance, if the two incoming edges and the three outgoing are of different types then the contribution of that node to each a-edge is $1/6$. Thus, the weight of an a-edge is the percentage of connecting nodes in paths formed by the edge types, pondered by their in(out)degrees. With this definition, the sum of a-edges weights is also 1.

As explained above, the introduction of the special a-nodes bottom and top is essential to abstract all the nodes of the original graph in a-edges connecting them. One may wonder what other a-nodes types should be considered. It should be noted that a-nodes may have a-edge loops if the graph contains homogeneous paths, i.e. paths formed by a single type of edge. Since the goal of a-graphs is to highlight path patterns, it is important to distinguish different cases that would be amalgamated by generic a-nodes with loops.

Certainly, not all a-nodes have loops. These are considered *shallow* a-nodes since they have at most paths of length 1. In contrast a *deep* a-node has homogeneous paths of higher length through its loop. Special cases of deep a-nodes can be also considered: *cyclical*, where the loops contain homogeneous cycles, i.e. cycles using only the type of edge represented by the a-node; and *hierarchical*, where the loops represent confluent paths, i.e. where the nodes in homogeneous paths have branching factor above or equal to 2. These types provide information on the kind of paths formed "within" an a-node, similar to the information that can be extracted from other a-edges relating different a-nodes.

In summary, an a-graph is an abstraction of a semantic graph. This does *not* mean that an a-graph is a sort of schema. A semantic graph does not comply with its a-graph, it is the other way round: a-graphs have a functional dependency to semantic graphs. Thus, the information provided by an a-graph is of a different nature of that of an RDF or OWL ontology, as discussed in Subsection 6.1.

3.2. Abstraction map

The previous subsection introduced the concepts of a-node, a-edge and their weights, as well as the map between a semantic graph and the a-graph that it abstracts. This subsection formalizes those concepts, starting by precising the concept of semantic graph. A semantic graph $\mathbb{G}$ can be defined as a tuple $\mathbb{G} = (N, E, T_N, T_E, \mathrm{t_N}, \mathrm{t_E})$ where:

set of nodes N
set of edges $E \subseteq \{(s,t) : s \in N \wedge t \in N\}$
set of types of nodes[9] T_N
set of types of edges[10] T_E
types of nodes $\mathrm{t_N} : N \to T_N$
types of edges $\mathrm{t_E} : E \to T_E$

The *a-graph* $\mathbb{A}$ of graph $\mathbb{G}$ is produced by a map defined as

$$\begin{aligned} \mathrm{abstract}\colon \mathbb{G} \qquad & \to \mathbb{A} \\ (N, E, T_N, T_E, M_N, M_E) \qquad & \mapsto (N', E', \mathcal{W}'_N, \mathcal{W}'_E, \mathrm{t}_{N'}) \end{aligned}$$

It should be noted that the a-graph $\mathbb{A}$ is an abstraction of a semantic graph $\mathbb{G}$, not itself a semantic graph. The a-graph $\mathbb{A}$ is a tuple where:

set of a-nodes $N' = T_E \cup \{\top\ \bot\}$
set of a-edges $E' = E'_0 \cup E'_\bot \cup E'_\top$
weight of a-nodes $\mathrm{w_N} : N' \to [0,1]$
weight of a-edges $\mathrm{w_E} : E' \to]0,1]$
type of a-nodes $\mathrm{t}_{N'} : N' \to T'$ where
$T' = \{\text{shallow deep cyclic hierarchical top bottom}\}$

As defined above, the set of a-nodes is the union of types of edges of G with special nodes $\top$ (top) and $\bot$ (bottom). The definition of the set of a-edges is also the union of three sets, namely the set of regular a-edges E'_0, the set of bottom a-edges $E'_\bot$, and the set of top a-edges $E'_\top$.

An ordered pair $(\bot, t')$ is in the set $E'_\bot$ if there is an edge of type t' where the source node s has a null indegree ($\deg^-(s) = 0$).

[9] In an RDF graph, this would be set of URIs referring to resources rather than a set of RDFS classes

[10] In an RDF graph, this would be set of URIs referring to properties

$$E'_{\perp} = \{(\perp, t') \colon t' \in T_E \wedge (\exists (s,t) \in E \colon \mathrm{t_E}((s,t)) = t') \wedge \deg^{-}(s) = 0 \}$$

Similarly, an ordered pair $(s', \top)$ is in the set $E'_{\top}$ if there is an edge of this type where the target node t has a null outdegree ($\deg^{+}(t) = 0$).

$$E'_{\top} = \{(s', \top) \colon s' \in T_E \wedge (\exists (s,t) \in E \colon \mathrm{t_E}((s,t)) = s') \wedge \deg^{+}(t) = 0 \}$$

Finally, an ordered pair (s', t') of a-nodes is in a set of regular a-edges if there is a path in the graph involving the two edge types.

$$E'_0 = \{(s', t') \colon s' \in T_E \ \wedge \ t' \in T_E \wedge (\exists (s,m) \in E \colon \mathrm{t_E}((s,m)) = s') \wedge (\exists (m,t) \in E \colon \mathrm{t_E}((m,t)) = t') \}$$

By definition, the weight of the special a-nodes, top and bottom, is null; and these are the only a-nodes with null weight. The nonnull weight of a regular a-node n' is the ratio between the number of edges with that type and the total number of edges

$$\mathrm{w_N}(n') = \frac{\sharp\{e \colon e \in E \wedge \mathrm{t_E}(e) = n'\}}{\sharp E}$$

The weight of an a-edge must be computed differently when its source s' or target t' have special a-nodes. If the source $s' = \perp$ then the n-tuple $B_{t'}$ of nodes to consider contains those that are sources of an edge of type t' with a null indegree. The reader should note that this is an n-tuple rather than a set, where each node s may appear more than once. The order of the nodes in the n-tuples is immaterial. The purpose of the n-tuples is merely to count the number of nodes. In all n-tuples $B_{t'}$, as defined below, each node s is repeated as many times as its outdegree $\deg^{+}(s)$.

$$B_{t'} = (s \colon (s,t) \in E \wedge \mathrm{t_E}((s,t)) = t' \wedge \deg^{-}(s) = 0)$$

Similarly, if the target $t' = \top$ then the n-tuple of nodes to consider is those that are the target of an edge of type s' with a null outdegree. In all n-tuples $T_{s'}$ each node t appears repeated as many times as its indegree $\deg^{-}(t)$.

$$T_{s'} = (t \colon (s,t) \in E \wedge \mathrm{t_E}((s,t)) = s' \wedge \deg^{+}(t) = 0)$$

Otherwise, if none of them is a special a-node then the nodes to consider are those that participate in paths of length 2 where the first edge has type s' and the second has type t'. In this case the node m appears repeated $\deg^{-}(m)\deg^{+}(m)$ times.

$$R_{s'}^{t'} = (m\colon (s,m) \in E \wedge (m,t) \in E \wedge \mathrm{t_E}((s,m)) = s' \wedge \mathrm{t_E}((m,t)) = t')$$

The weight of an a-edge $\mathrm{w_N}((s',t'))$ sums the contribution of n-tuples sets according to each case. The contribution of each node is pondered with the inverse of its indegrees or outdegrees, when these are not null. Thus, the definition of weight function is the following,

$$\mathrm{w_N}((s',t')) = \begin{cases} \frac{1}{\sharp E} \sum_{s \in B_{t'}} \frac{1}{\deg^+(s)} & \text{if } s' = \bot \\ \frac{1}{\sharp E} \sum_{t \in T_{s'}} \frac{1}{\deg^-(t)} & \text{if } t' = \top \\ \frac{1}{\sharp E} \sum_{m \in R_{s'}^{t'}} \frac{1}{\deg^-(m)\deg^+(m)} & \text{otherwise} \end{cases}$$

Finally, the $\mathrm{t}_{N'}$ function maps a-nodes to a type in T'. The definition of this function is based on the concept of *graph reduction*. A reduction of the graph $\mathbb{G}$ by the type $t \in T$ is the largest subgraph of $\mathbb{G}$ that has only edges of type t and without disconnected nodes; i.e. nodes that are not the source or target of edges of type t are removed. It should be noted that in general a graph reduction $\mathbb{G}_t$ has many strongly connected components, in some cases as many as the number of edges (a value property, for instance). Consider the following functions defined over the set of $\mathcal{G}$ of all graph reductions:

size of the largest path $\mathrm{slp} : \mathcal{G} \to \mathbb{N}$
number of cycles $\mathrm{nc} : \mathcal{G} \to \mathbb{N}$
average branching factor $\mathrm{abf} : \mathcal{G} \to \mathbb{R}$

Using these functions over graph reductions the a-node type function is defined as follows.

$$\mathrm{t}_{N'}(n') = \begin{cases} \text{shallow} & \text{if } n' \in T_E \wedge \mathrm{nc}(\mathbb{G}_{n'}) = 0 \\ & \quad \wedge \mathrm{slp}(\mathbb{G}_{n'}) < 3 \\ \text{deep} & \text{if } n' \in T_E \wedge \mathrm{nc}(\mathbb{G}_{n'}) = 0 \\ & \quad \wedge \mathrm{slp}(\mathbb{G}_{n'}) >= 3 \wedge \mathrm{abf}(\mathbb{G}_{n'}) < 2 \\ \text{hierarchic} & \text{if } n' \in T_E \wedge \mathrm{nc}(\mathbb{G}_{n'}) = 0 \\ & \quad \wedge \mathrm{slp}(\mathbb{G}_{n'}) >= 3 \wedge \mathrm{abf}(\mathbb{G}_{n'}) >= 2 \\ \text{cycle} & \text{if } n' \in T_E \wedge \mathrm{nc}(\mathbb{G}_{n'}) > 0 \\ \text{top} & \text{if } n' = \top \\ \text{bottom} & \text{if } n' = \bot \end{cases}$$

The codomain of $\mathrm{t}_{N'}$ captures a number of elemental path patterns involving a single edge type. Shallow a-nodes correspond to paths with a single edge. Deep a-nodes correspond to larger paths such as those resulting from transitive edges. Hierarchic a-nodes are in fact a special case of deep a-nodes, where paths converge to a single, or a set of, root nodes; these edges form hierarchies and are particularly interesting for discovering edges types for path based semantic measures [10]. Finally, cyclic a-nodes are created by reflexive edges.

The constants in the definition of $t_{N'}$ require some explanation. The threshold of 0 in the number of cycles [$nc(\mathbb{G}_{n'})$] to distinguish a cycle from other regular types is rather obvious, but *not* the threshold of 3 in the size of largest path [$slp(\mathbb{G}_{n'})$]. A threshold of 2 was, in fact, the first choice, but there are cases where paths of size 3 occur without changing the type of an a-node. The most notable example is *rdf:type*. Since RDF types have themselves a type (*rdfs:Class*), length 3 paths are usual in semantic graphs, but they should be considered shallow and not hierarchic. Finally, with an average branching factor [$abf(\mathbb{G}_{n'})$] above or equal to 2 the paths in a graph reduction form a hierarchy that is suitable for classification. That is usually the case of *rdfs:subClassOf*, for instance, that in conjunction with *rdf:type* creates a taxonomic relationship [10].

3.3. Diagram language

As explained in the previous Subsection, an a-graph is an abstraction of a semantic graph. The a-graph diagram language is a visual representation of an a-graph intended to highlight the path patterns of the abstracted semantic graph. An a-graph has a-nodes of different types connected by a-edges.

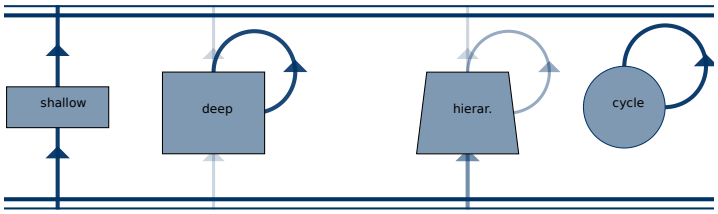

Fig. 3. Catalog of a-node types

The type of an a-node is conveyed by its shape. A shallow a-node is represented by a horizontal rectangle, while a deep a-node is represented by a vertical rectangle. The height of these rectangles is a visual cue of the path sizes contained in these a-nodes. A cyclical a-node is represented by a circle or an ellipse, and a hierarchical a-node is represented by an isosceles trapezoid. The position of these shapes is their geometric center. The a-graph depicted in Figure 3 is a sort of catalog of a-node types, where the label of each regular a-node is the type's name.

The bottom and top a-nodes are represented by a pair of parallel lines rather than shapes. As can be seen also in Figure 3, the parallel lines that represent each of these a-nodes have different widths. The bottom a-node has a larger upper line and the top a-node is the reverse. The bottom and top a-nodes are located respectively at the bottom and top of the diagram, as their names suggest. This way the paths created by a-edges tend to be directed upwards.

Unlike a-nodes, a-edges have a single type. Hence, they are represented all by solid lines with an arrowhead positioned in their middle pointing to the target. Lines connecting from the bottom a-node, or to top a-node, are straight. All the others are curved so that a-edges with opposite directions do not overlap.

A-nodes and a-edges have weights in the $[0, 1]$ interval. Actually, both regular a-nodes and a-edges have always nonnull weights; special a-nodes (top and bottom) have null weights by definition. The nonnull weights of regular a-nodes and a-edges are conveyed graphically too. The weight of an a-node is shown as a transparency, making dimmer the a-edges representing a smaller number of edges in the abstracted graph. The same principle applies to weights of a-edges. In this case, the weight is also shown as line width, making thicker the a-edges that represent a larger number of nodes. The semantic graph that originated the a-graph in Figure 3 has all edge types with the same number of edges, hence all a-nodes have the same weight, thus they all have the same shade. A different thing happens with a-edges; each has a different shade, reflecting their different weights.

The regular a-nodes in the catalog diagram are not connected to each other, just to bottom and top (with the exception of the cycle). This means that they do not form "joins". Using a syntax borrowed from SPARQL, it can be said that the semantic graph that generated it lacks triple patterns of the form

```
?a ?p ?b .
?b ?q ?c .
```

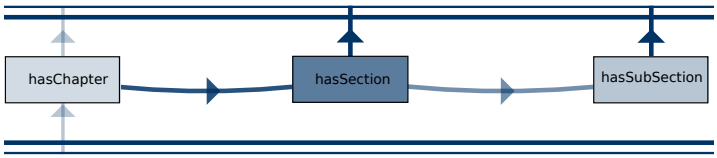

Fig. 4. Book structure

The example in Figure 4 represents the structure of books, where a book has chapters and these have sections. The a-graph of such semantic graph has the properties *hasChapter*, *hasSection* and *hasSubSection*.

In this case "joins" are created using multiple edge types hence the a-nodes have a-edges connecting them. In particular *hasChapter* is connected to *hasSection* and this to *hasSubSection*. The reader should note that the three regular a-nodes are connected to the top, meaning that there are chapters, sections and subsections that are not subdivided, and that only *hasChapter* is connected from bottom, meaning that only these are connected from root elements of the hierarchy.

The previous example reflects a hierarchical structure, although with a different type of edge for each layer. The example in Figure 2 reflects a semantic graph with a couple of family relationships, namely *spouseOf* and *parentOf*. Their associated a-nodes both have loops, which means that paths with a single type of edges can be created. The *parentOf* a-node has hierarchic as type, meaning that paths of length greater or equal to 3 can be created and has an average branching factor greater or equal to 2.

The simple patterns identified in the small examples above occur also in larger semantic graphs. Section 4 presents an a-graph browser that allows us to discover combinations of these patterns in larger examples, as those analyzed in Section 5.

4. A-graph browser

This section describes the design and implementation of a web application developed to validate the concept of a-graph. This web application – the a-graph browser – produces interactive a-graph diagrams from several data sources and is freely available online[11].

The a-graph browser is a Java web application developed with the Google Web Toolkit (GWT). It is composed of a client front-end running on a web browser and a server back end. The server is responsible for transforming a semantic graph in RDF format into an a-graph that is sent to the client. The front end is responsible for laying out diagrams and managing user interaction, as explained in the following subsections.

4.1. Back end processing

The mapping of semantic graphs into a-graphs is performed in two stages by the back end. Firstly, a set of graph reductions is produced from the semantic graph triples. Secondly, the a-graph data is computed by processing these graph reductions.

A graph reduction instance aggregates edges of a single type, that is, the semantic graph obtained by considering only the edges of that type. It records the nodes that are sources and those that are targets, and computes their in and outdegrees. The links between these nodes are also recorded to compute aggregate measures on the reduction such as the number of cycles, depth and branching factor.

Graph reductions are computed by processing a stream of RDF triples. For each subject-predicate-object triple the reduction corresponding to its predicate is selected, with the subject recorded as source and the object as target.

Each reduction corresponds to an a-node. Thus the second stage creates an a-node for each reduction found in the first stage, assigning it a weight computed as the percentage of edges in the graph. The top and bottom a-nodes, with null weight, are also created. Then it iterates over the pairs of reductions to create a-edges.

The computation of a-edges' weights is more complex than that of a-nodes, as it involves determining the intersection of the targets and source sets of nodes respectively of the source and target a-nodes of each a-edge. Also, the contribution of each of these nodes depends both on their in(out)degrees on the reduction. The pairs of a-nodes with nonnull weights create a-edges.

A-edges connecting a-nodes to top and bottom need also to be considered. These are created with the nodes that are not fully consumed to create a-edges among regular a-nodes, following the same approach to compute weights. It should be noted that links between top and bottom are impossible.

4.2. Diagram layout

A-graphs serialized in JSON are sent to the front end where they are visualized as diagrams. The layout of these diagrams is computed using a force-directed algorithm [11]. A-nodes repel each other according to Coulomb's law as if they were electrically charged particles with the same signal. A-edges bind them together as springs following Hooke's law.

[11] `http://quilter.dcc.fc.up.pt/antigraph`

The top and bottom a-nodes, as well as the a-edges connecting them, are ignored in this process. The layout is performed in a rectangular area that acts as a boundary that confines regular a-nodes. Top and bottom a-nodes are positioned respectively at the top and bottom of this rectangle, and a-edges connecting then are plotted perpendicularly to them.

One of the advantages of a force-directed algorithm is that it adjusts to changes, either of window dimensions or in the number of nodes. This enables the selection of a-nodes, choosing which to display and which to hide, with the quick readjustment of the layout. When an a-node is hidden so are the a-edges that link to it. Nevertheless, this algorithm may introduce undesirable changes due to user actions. To remedy this issue, the incremental layout can be toggled on or off, as explained in the next subsection.

A-graphs with a large number of a-nodes tend to have an even larger number of a-edges, cluttering the layout. In this case, the natural candidates to hide are those with smaller weight since they represent a smaller number of edges in the semantic graph. To simplify this kind of selection the a-graph browser provides a node weight threshold. If this threshold is provided then a-nodes are sorted by weight and their accumulated weight is computed in this order. When this value exceeds the threshold the remaining a-nodes are hidden, as well as the a-edges linking to them.

4.3. User interface

Figure 5 depicts the user interface of the a-graph browser available online. The main part is the left central region where the diagram is displayed, following the approach described in the previous subsection. Above this area, there is a toolbar with tools for controlling the diagram layout. The smaller panel on the right contains a data source selector and displays the current data source features. The remainder of this subsection describes these panels in detail.

The a-graph browser has a number of features to control the diagram layout. These features are accessible through the icons on the header toolbar. To start with, the incremental layout can be toggled on and off using the traffic light icon, on the left of the toolbar. The icons to its left provide ways to show and hide a-nodes, as well as the a-edges connecting them. The most relevant (with higher weight) hidden a-node is shown by pressing the outward spiral icon. Using this tool it is possible to gradually enlarge the diagram. The reverse tool, bound to the inward spiral icon, hides the least relevant shown a-node.

The following two icons operate on the currently selected a-node: to show all currently hidden a-nodes connected to it, or to hide all a-nodes connected to it. A-nodes are selected just by clicking on them. Clicking an a-node with the mouse's middle button also toggles a tool tip hovering the node. This tool tip displays the characteristics of the a-node, such as label, type and weight.

The hide all and show all tools allow the user to set the layout at the two extremes. These tools are respectively bound to the icons with an a-graph with no a-nodes and the a-graph with several a-nodes and a-edges. The header toolbar includes two other icons on its right side: the camera icon and the life saving icon. The latter opens a help window expanding the information in this paragraph.

The camera icon produces a *vector* image of the diagram presented in the browser. Using the normal browser features, it is possible to obtain a raster image of the diagram.

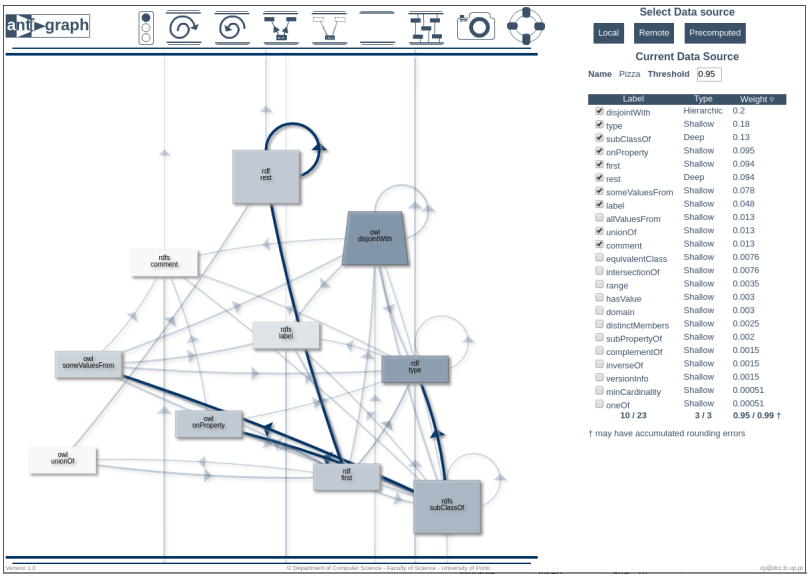

Fig. 5. A-graph browser

However, this kind of image is inadequate for publication since it has a fixed and typically low resolution. The camera icon activates a feature that produces an SVG file with the diagram, using the same layout algorithm described above. This conversion uses the SVGKit[12] package, that works well for graphic primitives (e.g. lines, rectangles, ellipses) but has some limitations regarding fonts and shadows. The vector images look slightly different from their raster counterparts, but have better quality when printed. The diagrams of the next section, as well as those of Subsection 3.3, were produced using this tool.

The a-graph browser presents a second panel next to the diagram. Depending on the width of the web browser's window, this panel may be placed either to the right side (as in Figure 5) or below the diagram. The panel contains a data source selector and displays the main features of the current data source.

The upper part of the side panel is used for selecting a semantic graph as data source for generating an a-graph. It provides three kinds of semantic graph sources: local, remote and precomputed.

Local sources include small examples for testing the basic features of a-graphs, and were presented in Subsection 3.3. The dialog box for the selection of local graphs presents the RDF triples that will be processed to produce the a-graph. These triples are in N-Triples format in an editable window. The user may modify, add or delete these triples, to better understand how these changes are reflected on the a-graph diagram.

The remote sources are RDF graphs available on the web in XML/RDF format. This dialog box presents each graph's URLs and a threshold – the weight above which a-nodes are included in the diagram. The last entry of this dialog box allows the user to enter a

[12] `http://svgkit.sourceforge.net/`

URL to any RDF/XML file available on the web, and assign it an initial threshold. This threshold may be changed later on the current data source panel.

Both the local and remote data sources are processed on the fly by the server. The precomputed data sources provide access to larger semantic graphs that require long processing times and are already available on the client side. Most of these examples are analyzed in detail in the next section.

The current data source panel displays its name, threshold and a grid listing its a-nodes. This grid lists all the a-nodes in the a-graph, showing which are currently visible, their type and weight. By default, this information is ordered by descending a-node weight, but the user can change it. The user can also (de)select the visible a-nodes, which immediately changes the diagram layout. Also, changes in the diagram resulting from the tools described above are also immediately reflected in this grid.

5. Validation

This section shows with concrete examples how a-graph diagrams emphasize the most relevant path patterns of a semantic graph. It also explains how the tools in the a-graph browser help the discovery of path patterns in large semantic graphs, by temporarily hiding some of their a-nodes and the a-edges connecting them, thus producing meaningful diagrams with a reasonable small size.

The a-graph browser has an example selector on its right top corner. These examples are divided according to their availability: local examples, with their source text available for inspection and editing, including the possibility of creating one from scratch; remote examples, published on the web in RDF formats, including the possibility of providing a URL; precomputed examples, of large freely available semantic graphs, that require a much larger processing time. The data sources for these examples must be in RDF format, either serialized in RDF/XML or in NTriples. The local examples include all a-graphs presented in Subsection 3.3.

5.1. WordNet

Wordnet[8] 2.1, whose a-graph diagram is depicted in Figure 6, is a much larger graph than those presented in Subsection 3.3. However, this figure refers only to 95% of Wordnet 2.1 since the 5% least representative edges are omitted. By default, when this example is selected the threshold is set to 95%, but this value may be edited or removed in the corresponding field.

The WordNet 2.1 graph has 27 types of nodes and their corresponding a-nodes would clutter this figure. This approach quickly produces a simple visualization by temporarily hiding the $2/3$ least representative a-nodes, i.e. edge types. It is important to point out that this is not specific of WordNet. All the semantic graphs tested with the a-graph browser have most of their paths concentrated in a fairly small number of edge types, hence this approach can be systematically used to improve the a-graph visualization.

This diagram immediately shows that the edges types that participate in most triples are from imported namespaces – *rdf:type* and *rdfs:label* – since the corresponding a-nodes are darker. Two a-nodes of the *wn20schema* namespace stand out from the pack for having links to several others, namely *hyponymOf* and *containsWordSense*, but the former participates in more “joins”, as evidenced by the darker a-edges.

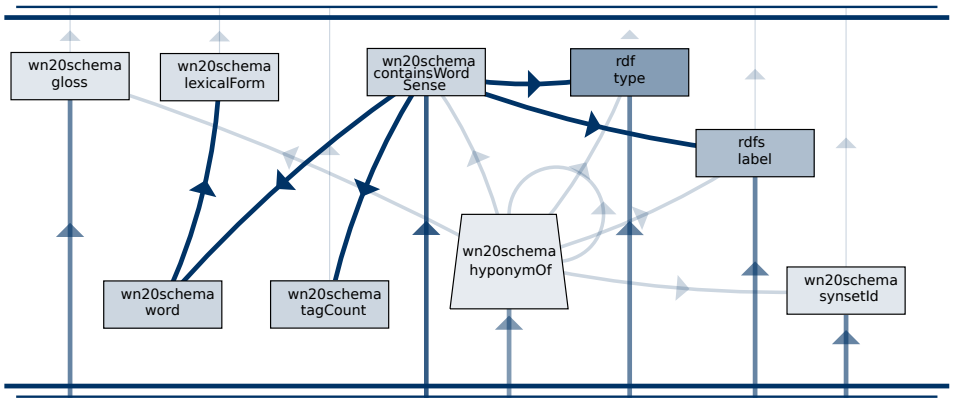

Fig. 6. An a-graph diagram of WordNet 2.1

WordNet is frequently used as a semantic proxy by path based semantic measures [10]. These measures rely on taxonomic relationships to identify a least common ancestor between two concept nodes and compute the shortest path between them. Taxonomic relationships are created using *partOf* (hierarchical) and *isA* relationships. For instance, the RDF and RDFS vocabularies provide the `rdf:type` and `rdfs:subclassOf` properties that can be used to create a taxonomic relationship between typed resources. However, in this version of WordNet `rdf:type` is available, but `rdfs:subclassOf` is missing.

The a-graph diagram in Figure 6 shows how the hierarchic relationship `hyponymOf`, complemented with another relationship, can be used to create a taxonomic relationship. The `rdfs:label` relationship is connected by an a-edge with `hyponymOf`, hence they can be combined to create a taxonomic relationship on words.

Of course, this is not new knowledge. It is well known that WordNet can be used as a semantic proxy using `hyponymOf` and another property to create a taxonomic relationship. The point is that the a-graph diagram highlights the most promising candidates to create a taxonomic relationship. This should be useful to discover candidates for taxonomic relationships in even larger semantic graphs, such as DBpedia [4].

5.2. Yago

Yago[13] [15] is a well known semantic knowledge base derived from several sources, such as DBPedia, WordNet, and GeoNames. It has over 10 million entities but for this study, only the core was used and labels were omitted. Still, this corresponds to over 20 million triples with 60 property types. Hence it produces an a-graph with that order and size 487. Even with a threshold of 80%, as it is by default on the a-graph browser, it is difficult to grasp.

The diagram in Figure 7 was obtained by selecting a single a-node, *hasArea*, the second most frequent edge type in this graph. Afterward, it was used the unhide tool to show a-nodes connected to the one currently selected. The point is to find property types related to concepts that have an area. Examples of such concepts would be cities, regions or

[13] `https://www.mpi-inf.mpg.de/yago-naga/yago`

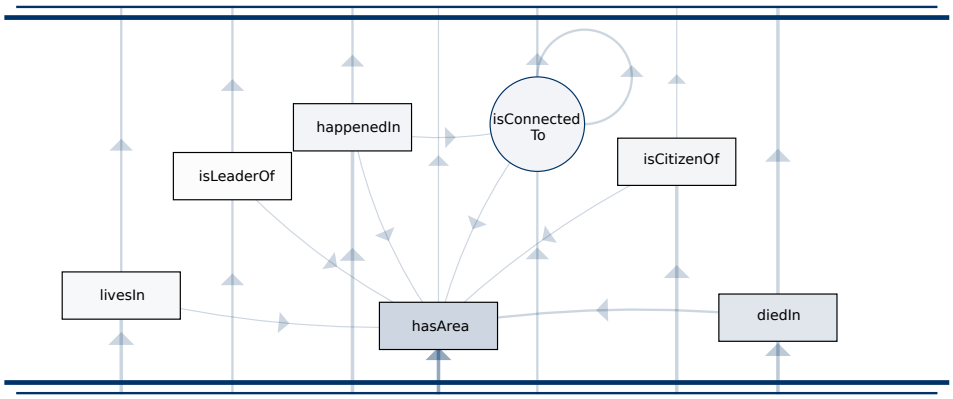

Fig. 7. Yago core - a-nodes connecting to *hasArea*

Listing 1.1. SPARQL query to count leaders of geographic areas

```
SELECT COUNT(*)
WHERE {
  ?p  yago:isLeaderOf ?g.
  ?g  yago:hasArea ?a.
}
```

countries. In a sense, *hasArea* can be seen as a defining property for a class of geographic concepts, although that is not explicit. The diagram shows that these geographic concepts are connected to other properties, such as *livesIn*, or *isLeaderOf*. That is, it is possible to retrieve information about who lives in or who is the leader of an concept that has an area. The SPARQL query in Figure 1.1 should produce a nonempty result set. In fact, it was checked on a Yago SPARQL endpoint[14] and the result is 5666.

Listing 1.2. Counting places connected to where something happened

```
SELECT COUNT(*)
WHERE {
  ?s yago:happenedIn ?g .
  ?g yago:isConnectedTo ?p .
}
```

Also, one can determine the area of entities where something happened, *happenedIn*, or that are connected to each other. The type of this last a-node is cyclic, meaning that its corresponds to a reflexive edge type. These two a-nodes are the only that are directly connected without using *hasArea*. Hence, it must be possible to obtain a non empty answer

[14] https://linkeddata1.calcul.u-psud.fr/sparql

to the query “what places are connected to the place where something happened?”, using the query in Listing 1.2, and it actually returns 888 solutions.

Surprisingly, the graph also indicates that one should not expect results for the query “what places are connected to the place of citizenship of x” since these two a-nodes are not connected. Running the SPARQL query in Listing 1.3 verifies that conclusion as the result is zero.

Listing 1.3. Are citizens connected to other places?

```
SELECT COUNT(*)
WHERE {
  ?s yago:isCitizenOf ?g .
  ?g yago:isConnectedTo ?p .
}
```

5.3. DBLP

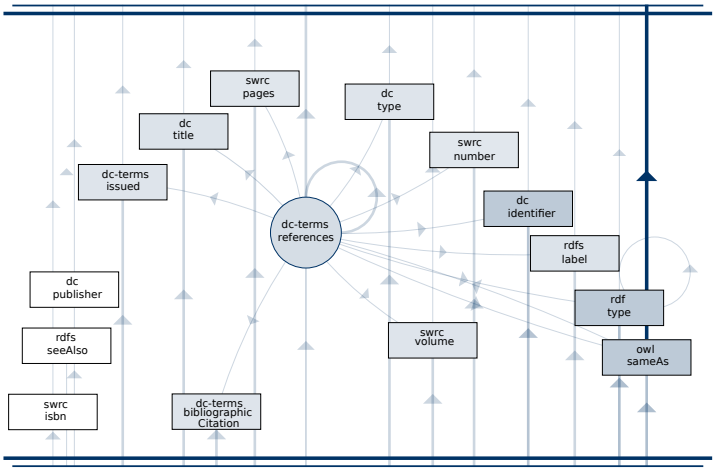

Fig. 8. DBLP open bibliographic information on computer science publications

DBLP[15] is an on-line reference for open bibliographic information on computer science journals and proceedings that publish its data in RDF format. Although it has a massive size, about 134 million triples, it can be processed by sampling since its RDF file has a regular structure. It is a sequence of blocks of triples, each block corresponding to a publication. The diagram depicted in Figure 8 was obtained by processing the initial 1%

[15] http://dblp.uni-trier.de/

of DBLP's RDF file. This approach was possible since this file keeps in consecutive lines the triples related to a single author. If it were the case that lines were sorted by property URL, for instance, then the first 1% of the file would not provide a meaningful representation of the complete graph. Thus, this approach cannot be applied systematically to any semantic graph since it assumes a uniform distribution of property URLs in the stream of triples.

The a-graph of DBLP reveals different patterns. The most simple are the a-nodes on the left side of the diagram, such as *swrc:isbn*, *dc:publisher* and *rdf:seeAlso* that simply connect the bottom to the top and not to any other a-node. Most of the other a-nodes have also an a-edge from the a-node *dc-terms:references* that thus assumes a pivotal role in this diagram and is the only cyclic a-node. The only other a-node with a loop is *rdf:type*, most probably due to the type of a class. Apart from these cycles formed by edges of the same type, there are also cycles with mixed types, composed of *owl:sameAs* and *dc-term:references*.

6. Discussion

The goal of a-graphs is to provide visualizations of path patterns in order to give new insights on large semantic graphs. In this section, we discuss if the concept of a-graph and the current implementation of the a-graph browser meet this goal. Three main questions are analyzed: how do a-graphs compare with ontologies in describing semantic graphs? do a-graphs provide more information regarding paths patterns than classical visualization tools? can a-graphs be computed in a reasonable time for large semantic graphs?

6.1. A-graphs and ontologies

Ontologies and a-graphs are somehow related in the sense that they both abstract semantic graphs. Thus, it is relevant to question if these two concepts – ontologies and a-graphs – overlap or compete in any way.

Semantic graphs are frequently encoded as sets of triples in the Resource Description Framework (RDF). This framework supports multiple vocabularies, including a vocabulary to describe other vocabularies – RDF Schema (RDFS) – which in turn lays the foundations for a richer ontological language – OWL. RDFS and OWL describe vocabularies in terms of classes and properties, where classes provide types for nodes and properties types for edges of semantic graphs, and define hierarchical relationships among those types.

The definition of semantic graph presented in Subsection 3.2, on which the definition of a-graphs relies, is also based on types. However, these types are of a different nature. These node and edge types are not RDFS or OWL classes and properties, and they are not hierarchically related among themselves. The node and edge types in the definition of a-graphs are the actual URIs used to label them.

The concept of ontology varies for different communities [9]. In the semantic web, an ontology is usually understood as a formal definition of a domain of discourse. It declares a taxonomy of concepts and relationships among them. For instance, an ontology may declare *cat* and *dog* as classes, both as subclasses of *pet*, and the property *hasName* associating pets to their names (strings). RDFS and OWL ontologies are themselves RDF

graphs, although not all RDF graphs are ontologies. In fact, most RDF graphs assert facts on resources using types and properties, such as "Rex is a dog"[16], but they do not define hierarchies of classes (concepts) and properties (relationships).

By using inference with an ontology it is possible to entail new facts from existing ones, such as "Rex is a pet". The reverse, to induce an ontology from a collection of facts, is much more complex. It is possible to process statements such as "Rex is a dog" and "Fifi is a cat", "Rex is a pet" and "Fifi is a pet" and induce an ontology similar to the example in the previous paragraph. However, ontologies are not usually created this way.

Ontologies prescribe how certain semantic graphs must be. They are not summarizations of existing semantic graphs. Also, if an ontology is applicable to a particular semantic graph then the latter should be consistent with the former; and as more facts are added to the graph that consistency should be preserved without changing the ontology.

An a-graph is, in fact, a summarization of a semantic graph. It maps edges into a-nodes and nodes into a-edges in a way that the a-graph paths condense several paths of the semantic graph it abstracts. However, only paths that actually exist in the semantic graph are abstracted into a-graph paths, not all the paths that would be consistent with the ontology. Moreover, since a-nodes and a-edges have weights, the path frequency is also presented by the a-graph, which has no parallel in ontologies. As a semantic graph evolves and new nodes and edges are added (or removed), its a-graph may change to reflect it. In some cases, only the weights will be affected, if no kinds of path are created. In other cases, new a-nodes result from edge types that did not exist before.

In summary, a-graphs and ontologies are different kinds of abstractions. A-graphs abstract paths, highlighting the most frequent ones. Ontologies abstract relationships among concepts. The two abstractions are non-overlapping and are in fact complementary.

6.2. Path visualization

A-graphs were designed to discover path patterns in semantic graphs. In this particular point, a-graphs are quite different from other forms of graph visualization, including semantic graph visualization, surveyed in Section 2. These approaches display the actual paths, connecting a sequence of nodes, but do not reveal patterns in these paths.

A path in an a-graph corresponds to a *collection* of paths in the semantic graphs it reflects. Since edges types are replaced by a-nodes, a-graphs put edge types in evidence and aggregate a large number of nodes in a-edges. Paths built from a single edge type are condensed in a-nodes types. Hence, a-node types such as hierarchic or cyclic already condense path patterns.

Weights of a-nodes and a-graphs provide information on their relevance. This extra information can be used both for visualization and browsing. Weights are translated to colors and line widths in a-graphs to indicate the relevance of particular path patterns. Moreover, weights can be used to browse a-graphs and hide the least relevant a-nodes.

The examples analyzed in Section 5 illustrate the ability of a-graphs to reveal information about path patterns. Figure 6 immediately shows the existence of a hierarchic edge

[16] "Rex is a dog" are two RDF facts. Assuming `ex` as an alias for a namespace, that sentence would be represented by the RDF facts

```
ex:rex ex:hasName ''Rex''
ex:rex rdf:type ex:dog
```

type and the edge types that connect it, which is relevant information for someone building a semantic measure. Figure 7 displays a number of edge types related to a particular one that characterizes geographic places. It provides information on the type of path patterns that can be used in SPARQL queries, estimating the possibility of returning non empty result sets, which is relevant to someone interested in extracting information from a triple store.

Surely the kind of insight provided by a-graphs is relevant to a specialized group of users, those interested in path patterns rather than in individual paths. This is an issue since it makes more difficult a thorough validation of the a-graph browser involving actual users. The author acknowledges that such validation is necessary to unequivocally prove the usefulness of a-graphs. However, this kind of users are difficult to find, hence this validation has yet to be done.

6.3. Visualization of large semantic graphs

Discovering path patterns is particularly important in large semantic graphs. Hence, it is important to be able to plot a-graph diagrams for such graphs within a reasonable time. Plotting a-graph diagrams using a force-directed algorithm takes only a few seconds. The time-consuming part is the mapping between semantic graphs and a-graphs.

The computational complexity of the mapping process described in the previous sections is linear on the number of triples, nodes, and property types. The first stage has complexity $O(t)$ where t is the number of triples in the semantic graph and the second stage as complexity $O(p*n)$ where p is the number of property types and n the number of nodes. Nevertheless, to process larger semantic graphs some optimizations are required.

Although the computational complexity of the a-graph mapping is small, processing large sources takes several days. As explained before, the process has two stages, each with a different complexity. Consider a large semantic graph with k triples, t edge types and n different resource nodes, with k and n large (hundred of millions) and t fairly small (less than a thousand). For the first stage, the complexity is linear on the number of triples, and the only data are the t reductions. In contrast, for the second stage the complexity is $n \times t$, both in time and memory. Thus, for larger semantic graphs, the complexity is an issue, particularly if the graph is too large to keep in memory.

The first stage of the mapping can be parallelized by splitting triples according to the predicate (edges type). This can achieve a considerable speedup on the first stage but does not reduce either the number of nodes or the number of types of edges, hence it has no impact on the second stage.

An obvious way to improve efficiency would be to avoid computing weights altogether, since this is the most time consuming task of the a-graph mapping. Of course, this would reduce the informative value of a-graphs, that would not highlight the most relevant path patterns.

An alternative to improve efficiency is sampling. Mapping a small enough subset (sample) of all the triples in the graph has a significant improvement in performance. Sampling is acceptable if it produces the same a-graph structure – a-nodes and a-edges – with a small error on their weights.

A naive approach to sampling would be the random selection of triples. However, a small (around 1%) random sample of triples typically has an impact on the structure of

the a-graph, by not correctly identifying all the a-nodes (a-edges are not usually affected). A larger sample (around 50%) solves that problem but significantly reduces the efficiency.

The DBLP example presented in Subsection 5.3 shows that this approach can be used if a small sample of the triples is representative of the complete set. In this case, this was achieved by considering the subset of triples related to the publications of a few authors. Unfortunately, in most cases this is impossible, since triples in RDF files are usually either grouped by edge type or just unordered.

Both approaches, dropping weight evaluation and sampling, have disadvantages and cannot be used systematically. However, they may be more effective if combined. More precisely, instead of dropping weight evaluation, an approximation may be computed using sampling. In this approach, the first stage is processed exactly in the way described in Subsection 4.1 but the second stage is modified. Hence, the a-nodes and their weights are computed exactly as they are defined in Subsection 3.2. This means that a-node weights maintain as invariant that their sum is 1. The same is not true for the second stage, where a-edges are determined as pairs of a-nodes with nonnull weight.

Computing a-edges weights is the major contributor to the computational complexity of the second stage. This is due to the fact that, in general, a single node may contribute to the weight of several a-edges, as explained in Subsection 3.2. If these weights are computed approximately using sampling then this complexity may be curbed.

Relaxing the computation of a-edge weights may have an impact on the structure of the a-graph, since some low weight a-edges may be missed. Additionally, it will be difficult, if not impossible, to maintain invariant the sum of a-edge weights. Everything considered, it is preferable to risk missing the least relevant a-edges than any a-node, and the weights of a-edges are less important than those of a-nodes to the visualization and browsing of a-graph diagrams. In any event, although promising, this approach of relaxing the computation of a-edges is not yet available and will require further research.

7. Conclusions and future work

Semantic graphs are hard to visualize due to a large number of typed nodes and edges. The a-graph approach to abstract semantic graphs maps edge information into a-nodes and node information into a-edges. The abstraction mapping produces a smaller graph that is easier to visualize and highlights the patterns of paths in the original semantic graph.

A-nodes and a-edges are assigned with weights that reflect the relevance of the edges and nodes they represent, and that can be used for further abstractions. For instance, a-nodes with small weights, corresponding to types of edges that seldom occur in the semantic graph, can be omitted to unclutter large a-graph diagrams.

The a-graph diagram is the proposed graphical syntax to represent a-graphs, and thus visualize the semantic graphs. This kind of diagrams uses different shapes to represent a-nodes according to their types, and transparency to denote weights. The special a-nodes top and bottom are represented as parallel lines respectively on the top and bottom of the diagram. In an a-graph, diagram paths are in general upwards, which facilitates their detection.

The web application for visualizing and interacting with a-graphs is also an important contribution of this research. It uses a force-directed algorithm, which allows the incremental layout of the diagram after reposition or removal of a-nodes. This application can

use data from different sources: local data entered on the interface, remote data available on the web and precomputed data for a few preprocessed semantic graphs.

The proposed approach still faces the challenge of dealing with massive semantic graphs with millions of triples, such as those of Yago and DBpedia. The major problem is due to the computational complexity involving a-edge weights. However, there are approaches to curb this complexity that are currently being researched. After tackling this issue, the a-graph browser will be easier to evaluate with real users interested in discovering path patterns in large semantic graphs.

Acknowledgments. I am in debt to the anonymous reviewers for their careful reading of this manuscript and their many insightful comments and suggestions.

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, by National Funds through the FCT Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961, and by FourEyes. FourEyes is a Research Line within project TEC4Growth Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01- 0145-FEDER-000020 financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

References

1. Abello, J., Van Ham, F., Krishnan, N.: Ask-graphview: A large scale graph visualization system, Visualization and Computer Graphics, IEEE Transactions on 12 (5), 669–676 (2006)
2. Archambault, D., Munzner, T., Auber, D.: Grouseflocks: Steerable exploration of graph hierarchy space, Visualization and Computer Graphics, IEEE Transactions on 14 (4), 900–913 (2008)
3. Auber, D.: Tulipa huge graph visualization framework, in: Graph Drawing Software, Springer, 105–126 (2004)
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data, Springer, (2007)
5. Bastian, M., Heymann, S., Jacomy, M., et al.: Gephi: an open source software for exploring and manipulating networks., International Conference on Web and Social Media - ICWSM 8, 361–362 (2009)
6. Benedetti, F., Po, L., Bergamaschi, S.: A visual summary for linked open data sources, in: International Semantic Web Conference (2014)
7. Bikakis, N., Liagouris, J., Kromida, M., Papastefanatos, G., Sellis, T.: Towards scalable visual exploration of very large RDF graphs, in: The Semantic Web: ESWC 2015 Satellite Events, Springer, 9–13 (2015)
8. Fellbaum, C.: WordNet, Wiley Online Library, (1999)
9. Guarino, N., Oberle, D., Staab, S.: What is an ontology?, in: Handbook on ontologies, Springer, 1–17 (2009)
10. Harispe, S., Ranwez, S., Janaqi, S., Montmain, J.: Semantic similarity from natural language and ontology analysis, Synthesis Lectures on Human Language Technologies 8 (1), 1–254 (2015)
11. Kobourov, S. G. , Spring embedders and force directed graph drawing algorithms, CoRR abs/1201.3011.
    URL http://arxiv.org/abs/1201.3011
12. Harary, F.: Graph Theory, Massachusetts: Addison-Wesley (1972)

13. Hastrup, T., Cyganiak, R., Bojars, U.: Browsing linked data with fenfire., in: Linked Data on the Web (2008)
14. Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., Stegemann, T.: Relfinder: Revealing relationships in rdf knowledge bases, in: Semantic Multimedia, Springer, 182–187 (2009)
15. Hoffart, J.,Suchanek, F. M.,Berberich, K., Weikum,G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia, in: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, AAAI Press, 3161–3165 (2013)
16. Leal, J.P.: Path Patterns Visualization in Semantic Graphs, in: 7th Symposium on Languages, Applications and Technologies, SLATE 2018, OASIcs, vol. 62, 15:1–15:15 (2018)
17. Lin, Z., Cao, N., Tong, H., Wang, F., Kang, U., Chau, D. H.: Demonstrating interactive multi-resolution large graph exploration, in: Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on, IEEE, 1097–1100 (2013)
18. Liu, Y. et al: Graph summarization methods and applications: A survey. ACM Computing Surveys (CSUR), 51.3: 62, (2018)
19. Song, Q. et al.: Mining summaries for knowledge graph search, in IEEE Transactions on Knowledge and Data Engineering 30, no. 10, 1887–1900 (2018)
20. Zhang, K., Wang, H., Tran, Yu, D. T., Y.: Zoomrdf: semantic fisheye zooming on rdf data, in: Proceedings of the 19th international conference on World wide web, ACM, 1329–1332 (2010)
21. Zinsmaier, M., Brandes, U., Deussen, O., Strobelt, H.: Interactive level-of-detail rendering of large graphs, Visualization and Computer Graphics, IEEE Transactions on 18 (12), 2486–2495 (2012)

José Paulo Leal graduated in mathematics from the Faculty of Sciences of the University of Porto and earned a Ph.D. in Computer Science from the same institution. His main research interests are technology enhanced learning, web adaptability, and semantic web.

Received: July 17, 2019; Accepted: October 6, 2019.