Exploring Importance Measures for Summarization on Graph Databases

Alexandros Pappas

Thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes Campus, GR-70013, Heraklion, Crete, Greece

Thesis Advisors: Prof. *Dimitris Plexousakis*

This work was partially supported by **Institute of Computer Science, Foundation of Research and Technology Hellas**

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

Exploring Importance Measures for Summarization on Graph Databases

Thesis submitted by
Alexandros Pappas
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Alexandros Pappas

Committee approvals: _____
Dimitris Plexousakis
Professor, Thesis Supervisor

_____
Georgios Georgakopoulos
Associate Professor, Committee Member

_____
Giorgos Flouris
Professor, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, 10^{th} February 2017

Abstract

The real world is richly interconnected. As such the natural properties of graphs, render them extremely useful in modeling real world, understanding a wide diversity of data-sets and offering applied solutions in different fields of industry. A graph database is an on-line, operational database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. Alternative to traditional relational databases, graph databases are being optimized and designed predominantly for graph workloads, traversal performance and executing graph algorithms on complex hierarchical structures.

Given the explosive growth in the size and the complexity of the Data Web, it is estimated that by the end of 2018, 70% of leading organizations will have one or more utilizing graph databases. Triple stores are a subcategory of graph databases, modeled around the Resource Description Framework (RDF) specifications and designed as labeled, directed multi-graphs.

To this direction, there is now more than ever, an increasing need to develop methods and tools in order to facilitate the understanding and exploration of RDF/S Knowledge Bases (KBs). Given the fact that the human brain can only interpret at most a few hundred nodes in one chart it becomes obvious that current data size and schema complexity are far beyond the exploration capability that any automated layout can provide.

Summarization approaches try to produce an abridged version of the original data source, highlighting the most representative concepts. Central questions to summarization are: how to identify the most important nodes and then how to link them in order to produce a valid sub-schema graph. In this thesis, we try to answer the first question by revisiting several measures covering a wide range of alternatives for selecting the most important nodes and adapting them for RDF/S KBs. Then, we proceed further to model the problem of linking those nodes as a graph Steiner-Tree problem (GSTP). Since the GSTP is NP-complete, we explore three approximations (SDIST, CHINS and HEUM) employing heuristics to speed up the execution of the respective algorithms. Our detailed experiments show the added value of our approach since a) our adaptations outperform current state of the art measures for selecting the most important nodes and b) the constructed summary has a better quality in terms of the additional nodes introduced to the generated summary as GSTP approximations outperform past approaches.

Περίληψη

Ο πραγματικός κόσμος είναι πλούσια διασυνδεδεμένος. Ως εκ τούτου οι φυσικές ιδιότητες των γραφημάτων, τα καθιστούν εξαιρετικά χρήσιμα στη μοντελοποίηση του πραγματικού κόσμου και την κατανόηση μια ευρείας ποικιλίας συνόλων δεδομένων, προσφέροντας παράλληλα εφαρμόσιμες λύσεις σε διάφορους τομείς της βιομηχανίας. Μια βάση δεδομένων γραφημάτων, είναι ένα επιχειρησιακό σύστημα διαχείρισης βάσεων δεδομένων, το οποίο μπορεί να εκτελέσει μεθόδους δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής, οι οποίες εκθέτουν ένα μοντέλο δεδομένων γράφου. Διαφέροντας από τις παραδοσιακές σχεσιακές βάσεις δεδομένων, οι βάσεις δεδομένων γραφημάτων έχουν βελτιστοποιηθεί και σχεδιαστεί κυρίως για διεργασίες πάνω σε δεδομένα γράφων, αποδοτικότερη διάσχιση των δεδομένων και εκτέλεση αλγορίθμων γράφων σε πολύπλοκες ιεραρχικές δομές.

Με δεδομένη την εκθετική αύξηση στο μέγεθος και την πολυπλοκότητα των δεδομένων του διαδικτύου, εκτιμάται ότι μέχρι το τέλος του 2018, το 70% των κορυφαίων οργανισμών θα αξιοποιεί μία ή περισσότερες βάσεις δεδομένων γραφημάτων. Οι τριπλέτες αποθήκευσης αποτελούν μια υποκατηγορία των βάσεων δεδομένων γραφημάτων, η οποία διαμορφώθηκε και μοντελοποιήθηκε βασισμένη στις προδιαγραφές του Resource Description Framework (RDF) και σχεδιάστηκε ως ένας επισημασμένος, κατευθυνόμενος, πολυγράφος.

Προς αυτή την κατεύθυνση, υπάρχει τώρα περισσότερο από ποτέ ανάγκη για την ανάπτυξη μεθόδων και εργαλείων, προκειμένου να διευκολυνθεί η κατανόηση και η εξερεύνηση των RDF γνωσιακών βάσεων δεδομένων. Λαμβάνοντας υπόψη το γεγονός ότι ο ανθρώπινος εγκέφαλος μπορεί να ερμηνεύσει μόνο μερικές εκατοντάδες κόμβους σε ένα γράφημα, τότε είναι προφανές ότι το μέγεθος των σημερινών δεδομένων και η πολυπλοκότητα του σχήματος είναι εκτός των δυνατοτήτων εξερεύνησης που μπορούν να προφέρουν οι μέθοδοι αυτοματοποιημένων σχεδιασμών.

Ως προς την επίλυση αυτού του προβλήματος, οι μέθοδοι συνόψισης επιδιώκουν την παραγωγή μιας συνοπτικής έκδοσης της αρχικής πηγής δεδομένων, αναδεικνύοντας τις πιο αντιπροσωπευτικές έννοιες. Βασικά ερωτήματα για την παραγωγή μιας συνόψισης είναι: το πως θα προσδιοριστούν οι σημαντικότεροι κόμβοι ενός συνόλου και εν συνεχεία, το πώς θα συνδεθούν προκειμένου να παραχθεί ένας έγκυρος υπογράφος. Σε αυτή την εργασία, προσπαθούμε να απαντήσουμε το πρώτο ερώτημα με την χρήση και την προσαρμογή σε γνωσιακές βάσεις δεδομένων, μέτρων σημαντικότητας τα οποία έχουν ήδη ερευνηθεί στο παρελθόν, ώστε να καλύψουν ένα ευρύ φάσμα διαφορετικών δεδομένων για την επιλογή των πιο σημαντικών κόμβων. Έπειτα μοντελοποιούμε το πρόβλημα της διασύνδεσης των κόμβων ως ένα Δέντρο Στάινερ σε γραφήμα, το οποίο ανήκει σε προβλήματα συνδυαστικής βελτιστοποίησης, με κοινό ζητούμενο να βρεθεί η συντομότερη διασύνδεση για ένα ορισμένο σύνολο κόμβων. Δεδομένου ότι το πρόβλημα αυτό ανήκει στην κατηγορία των δυσεπίλυτων προβλημάτων, διερευνήσαμε τρεις προσεγγιστικούς αλγορίθμους, χρησιμοποιώντας ευρηστικά τεχνάσματα τα οποία επιταχύνουν την εκτέλεση τους, για την επίλυση του προβλήματος σε πολυωνυμικό χρόνο. Μέσω της διεξαγωγής λεπτομερών πειραμάτων εμφανίζουμε την προστιθέμενη αξία της προσέγγισης μας, δεδομένου ότι α) οι προσαρμογές

μας ξεπερνούν τις τρέχουσες τεχνικές υψηλού επιπέδου μέτρων σημαντικότητας για την επιλογή των πιο σημαντικών κόμβων και β) η παραγόμενη σύνοψη έχει καλύτερη ποιότητα, εισάγοντας μικρότερο αριθμό πρόσθετων κόμβων, καθώς οι προσεγγιστικοί αλγόριθμοι του Δέντρου Στάινερ αποδίδουν καλύτερα από τις μεθόδους οι οποίες έχουν χρησιμοποιηθεί στο παρελθόν.

Acknowledgements

This work was funded by means obtained through the activities of the Information Systems Laboratory of the FORTH-ICS.

First of all, I would like to thank my thesis advisor, Professor Dimitrio Pleksousaki and co-advisor, Professor Haridimo Kondylaki for their irreplaceable dedication during these years. Their office door was always open for advice and help. I would also like to thank Professor Ioannis G. Tolli and Professor Georgio Georgakopoulo for their academic guidance during my undergraduate and graduate years, that was more than helpful. Furthermore I need to express my gratitude to the ISL group members of the Institute of Computer Science of the Foundation for Research and Technology for supporting and helping me, as well as the staff of the University of Crete. Especially Georgia Troullinou, Giorgo Flouri and Gianni Roussaki for their priceless time, work and effort on the Experiments of this thesis and their valuable comments. In addition, the formal framework and the query evaluation framework was created in cooperation with them. Moreover, I would like to thank my family and my friends for their unconditional encouragement and support. People come into your life for a reason, a season or a lifetime. This thesis is dedicated to everyone who is a part of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The recent explosion of the Data Web and the associated Linked Open Data (LOD) initiative have led to an enormous amount of widely available RDF datasets. These datasets often have extremely complex schemas which are difficult to comprehend, limiting the exploration and the exploitation potential of the information they contain. As a result, there is now, more than ever, an increasing need to develop methods and tools in order to facilitate the quick understanding and exploration of these data sources.

To this direction, approaches for ontology modularization [1] and partitioning [2] try to minimize and partition ontologies for better understanding but without preserving the important information. Other works try to provide overviews on the aforementioned ontologies [3, 4, 5] maintaining however the most important ontology elements. Such an overview can also be provided by means of an ontology summary. Ontology summarization [6] is defined as the process of distilling knowledge from an ontology in order to produce an abridged version. While summaries are useful, creating a good summary is a non-trivial task. A summary should be concise, yet it needs to convey enough information to enable a decent understanding of the original schema. Moreover, the summarization should be coherent and provide an extensive coverage of the entire ontology.

So far, although a reasonable number of research works tried to address the problem of summarization from different angles, a solution that simultaneously exploits both the structure and the semantics provided by the schema and the data instances is still missing.

In this thesis, we focus on RDF/S ontologies and explore efficient and effective methods to automatically create high-quality summaries. We view an RDF/S Knowledge Base as two distinct and interconnected graphs, i.e. the schema and the instance graph. As such, a summary constitutes a valid sub-schema graph containing the most important nodes, summarizing the instances as well. Central questions to the process of summarization is how to identify the most important nodes and then how to link those nodes to produce a valid sub-schema graph. For answering the first question various importance measures have been proposed trying to provide real-valued functions on the nodes of a graph, where the values produced are expected to provide a ranking which identifies the most important

nodes. Importance has a wide number of meanings, leading to many different definitions, usually conceived in relation to a type, flow or transfer across the network.

1.2 Methodology

In this thesis we explore eight diverse measures covering a wide range of alternatives for identifying importance. Then we try to answer the second question by modelling the problem of selecting a valid sub-schema graph as a Steiner-Tree problem which we resolve using approximations with heuristics.

More specifically our contributions are the following:

- To identify the most important schema nodes we explore six measures that have been proposed already for identifying importance in generic graphs. Those measures are the *Degree* , the *Betweeness*, the *Bridging Centrality*, the *Harmonic Centrality* and the *Radiality* and the *Ego Centrality*. Besides measures proposed for generic graphs that exploit only the schema graph of the RDF/S KB we explore hybrid measures combining both the schema and the instance graph of the RDF/S KB such as the *KCE importance* and the *Relevance*.

- Next we try to identify the proper paths connecting those nodes. We achieve this by modelling the problem as a graph Steiner-Tree Problem trying to minimize the total number of nodes of the selected subgraph. Since the problem is NP-complete and the exact algorithms proposed require significant execution time, we proceed further to explore three approximations, the SDIST, the CHINS and the HEUM trying to optimize either the insertion of a single component or the connection of the components using their shortest paths. On top of these approximations we implement an improvement procedure using heuristic the I-MST, ensuring that all leaves are terminal nodes.

- Finally, we perform a detailed two-stage experimental evaluation using eight diverse ontologies: the BIOSPHERE, the Financial, The Aktors Portal, the CRMdig, the LUBM, the eTMO, the DBpedia 3.8 and the Dbpedia 3.9 ontologies. In the first stage we compare the applicability of the selected measures for identifying the nodes' importance. To this direction, initially we use the Spearman's rank correlation coefficient to identify the statistical dependence between the produced ranking of the nodes. Then we identify that overall the Ego Centrality and the Betweeness outperform the other important measures in the examined ontologies, without being however the winners in all cases. In the second stage, we evaluate the quality of the selected sub-graphs showing that CHINS outperforms the current state of the art in terms of quality without too much overhead in the execution time.

To the best of our knowledge, this is the first time that eight diverse importance measures are compared for summarization purposes. In addition although other recent works focus on using the maximum cost spanning tree [7, 5] for linking the selected nodes, this

is the first time the problem of summarization is formulated as a Steiner-Tree problem using approximations for the fast identification of the corresponding summaries with many benefits as we shall show in the sequel.

1.3 Related Work

The latest years summarization approaches for linked data are constantly gaining ground.

For example, a wide variety of research works [8, 9, 10, 11] focus on extracting statistics and producing visual summaries of linked datasets. To do that they exploit statistical information using the data instances and the frequencies of the links that appear there. Other approaches try to create mainly instance summaries, exploiting the instances' semantic associations and they propose algorithms that do not take into consideration the schemata of the graphs. Jiang et.al. [12], Navlakha et al. [13], and Tian et al. [14] try to construct instance-focused graph summaries of unweighted graphs by grouping similar nodes and edges to super-nodes and super-edges. Hasan [15] focus on summarizing metadata and large graphs, by proposing a method to summarize the explanation of the related metadata over a set of Linked Data, based on user specified filtering criteria and producing rankings of explanation statements. However, our system differs from the above in terms of both goals and techniques.

More closely related works are Peroni et al. [3], Wu et al. [16], Zhang et al.[6], Queiroz-Sousa et al.[17], Pires et al. [4] and Troulinou et al. [7, 5].

Peroni et al. [3] try to automatically identify the key concepts in an ontology, combining cognitive principles, lexical and topological measurements such as the density and the coverage. The goal is to return a number of concepts that match as much as possible those produced by human experts. On the other hand Wu et al. [16] use similar algorithms, named Concept-And-Relation-Ranking, to identify the most important concepts and relations in an iterative manner. However, both of these works focus only on returning the most important nodes and not on returning an entire graph summary.

In Zhang et al. [6] the authors use measures such as the degree-centrality, the betweenness and the eigenvector centrality (weighted Page Rank and HITS) to identify not the most important nodes but the most important RDF sentences. The notion of RDF Sentence is the basic unit for the summarization and corresponds to a combination of a set of RDF statements. Then they link those sentences to produce the final summary. However, in this approach, the overall importance of the entire graph is not considered and many important nodes may be left out.

In Queiroz-Sousa et al. [17] the authors try to combine user preferences with the degree centrality and the closeness to calculate the importance of a node and then they use the Broaden *Relevant Paths* algorithm to find paths that include the most important nodes in the final graph. However the corresponding algorithm prioritizes direct neighbors ignoring that the selection of other paths could maximize the total importance of the selected summary.

Pires et al. [4], propose an automatic method to summarize ontologies that represent schemas of peers participating in a peer-to-peer system. In order to determine the rele-

vance of a concept, a combination two measures, centrality and frequency is used. Then, the adjacent nodes are grouped together and paths between those groups are identified in order to produce the final summary. Since multiple paths might exist, precision and recall are used to determine the level of coverage and consiseness of each path. In our case however, a more deterministic approach is used to identify the selected paths.

Finally Troulinou et al. [7, 5] employee relevance for identifying the most important nodes and then they try to connect those nodes by generating and pruning appropriately the maximum cost spanning tree. However, many additional nodes might be introduced and the selected summary does not guarantee to maximize the total importance of the selected sub-graph.

In this thesis, we employee and compare eight measures in determining the nodes' importance. In addition, modelling the problem of linking those nodes as a graph Steiner-Tree problem ensures that the selected summary minimizes the number of the additional nodes that are introduced. The high quality of the result is verified by our experiments.

Chapter 2

Background

2.1 Graph Theory

The history of graph theory begins in 1736 with the paper, its titles Seven Bridges of Königsberg, written by the Swiss mathematician Leonhard Euler. Unexpectedly it took two hundreds years before the first textbook was published in 1936. Its titles was as "Theorie der endlichen und unendlichen Graphen" and was written by the Hungarian mathematician Dénes Kőnig, since the term "graph" was developed into an extensive and popular branch of mathematics. Over the years many real world problems was introduced and solved in the field of graph theory, with many of them having a huge impact on our lives. Graph theory is widely used to study and model various applications, in diverse fields which include biology, electrical engineering, computer science, sociology, economy, operations research etc. Graph Databases is one of the these applications which constitute an overlap between traditional databases and graph theory. Researchers in this area benefit from the rich background on graph theory, by exploiting a large baseline of concepts and algorithms developed over the last fifty years.

2.1.1 Graph Theory Basic Definitions

A network describes an object composed of elements and interactions or connections between these elements. We are using structures called graphs, in order to model networks mathematically. Graphs is an abstract data type that can provide a natural representation of a wide array Structured data.

Graph: A graph $G = (V, E)$ is an abstract object, comprising a set V of vertices and a set E of edges that connect (join) pairs of vertices. In computer science, the vertices of a graph, may also be called "nodes" or "points" and the edges connecting the vertices, may also be called "arcs" or "links".

Node: A node v is an object of a graph and the total number of nodes in a graph is often denoted as $|V|$ or n. On Graph databases nodes represent entities such as people, businesses, accounts etc and they are roughly the equivalent of the record, relation or row

in a relational database, or the document in a document database.

Edge: An edge represent the relationship between the objects and the total number of edges in a graph is often denoted as $|E|$ or m. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in other systems.

Adjacent: Two vertices u, v are adjacent if they are joined by an edge $e(u, v)$ and we call them neighbors.

Self-loop: An edge $e(u, v)$ that links a vertex to itself is called as a self-loop or reflexive tie.

Sub-Graph: A graph $G' = (V', E')$ called sub-Graph of a graph $G = (V, E)$ if the graph vertices V' and graph edges E' are a subset of V,E.

Neighborhood: The neighborhood of a vertex v is the induced sub-Graph $G' = (V', E')$ of $G = (V, E)$ consisting of all vertices adjacent to v and all edges that connect two such vertices.

Super-Graph: if a graph $G' = (V', E')$ is a sub-Graph of $G = (V, E)$, then G is a super-Graph of G'.

Path: A path in a graph (or a path graph P_g) is a finite sequence of edges which connect a sequence of vertices. Several algorithms exist to find shortest and longest paths in graphs for directed, undirected, weighted and unweighted graphs.

Connected component: A connected component of a graph $G = (V, E)$ is defined as a maximal sub-Graph $G' = (V', E')$ in which any two vertices are connected to each other by one or more paths.

2.1.2 Graph Data Formats

There are different ways to manage and store graphs in a computer system. Using the correct data structure with graph problems is critical and depends on its properties and the algorithms, we have to deal with.

Adjacency Matrix The adjacency matrix of a graph G is a n $\times$ n matrix A, where n is the number of nodes of G and each element $A_{i,j}$ corresponds to an edge $e_{i,j}$. The value of an element $A_{i,j}$ defines the weight of the edge that connects the node V_i to node V_j. If this value is undefined or 0, means that node V_i and V_j are not connected. For a graph without loops, the value of diagonal elements of A is 0 and if the graph is Undirected that matrix A has to be symmetric.

Adjacency List Adjacency list is a collection of unordered lists, equal to number of vertices. Each entry i of the collection represents the linked list of vertices adjacent to the ith vertex. There are many variations of the implementation differing in collection structure (array,set,list), the association between vertices and collections, whether to include both vertices and edges or only vertices as first class objects and in structure of objects that used to represent the vertices and edges.

Trade-offs Adjacency list,Adjacency Matrix

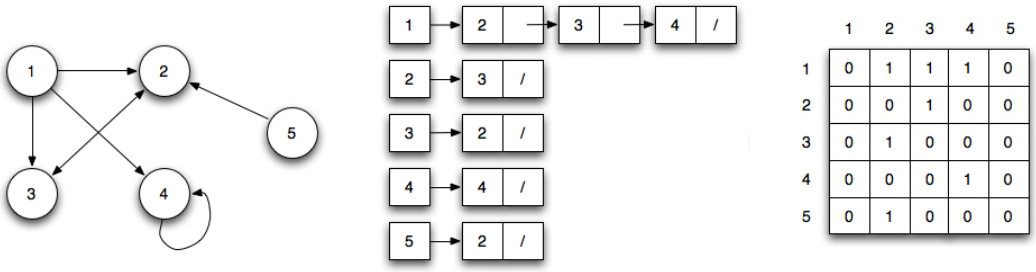


Figure 2.1: Different Representations of an undirected Graph $G = (V, E)$.

The usage of an Adjacency list and an Adjacency Matrix depends on our graph structure and the problems we have to deal with. It is more efficient to use an adjacency list for a sparse graph, where most pairs of vertices are not connected, because it requires significantly less space. Because of their structure adjacency list space usage is proportional to the number of edges and vertices in the graph, i.e. An adjacency list requires E space for a directed graph and $2 \cdot E$ for an undirected graph, as each edge (u, v) appears exactly twice. An adjacency matrix will takes $\Theta(V^2)$ space even if the graph is directed, undirected and has no edges. In contrast if the graph is dense it will be more efficient to use an adjacency matrix, especially if the graph is undirected because the matrix is symmetric. That means 2 bit per pair of vertices for an Adjacency Matrix, rather than 16 bits per edge for an Adjacency List.

The other important trade off between adjacency lists and adjacency matrices lies in the efficiency of the operations they perform. If we want to find all vertices that are adjacent to a vertex v, with an adjacency list the neighbors of each vertex are listed efficiently in time proportionally to the degree of vertex v, while for an adjacency matrix we have to iterate all V entries in row i that represents the edges of vertex v. But if we want to find out whether an edge (u, v) is in the graph (i.e. whether two vertices are adjacent) then in a adjacency matrix, it takes O(1) time, while in an adjacency list, it requires time proportional to the minimum degree d of the two vertices, with a worst case time complexity $\Theta(d)$. Figure 2.1[1], providing a clear representation of the Adjacency list and The Adjacency Matrix representation of an undirected graph $G = (V, E)$.

[1] `http://ycpcs.github.io/cs360-spring2015/lectures/lecture15.html`

2.1.3 Types of Graphs

The first step in any graph problem is to determine witch type of graph we are dealing with, because of the big impact on which algorithms are appropriate and efficient to use. Even more many solutions, algorithms or important measures, can be implemented only on specific graph types.

Directed graphs A graph $G = (V, E)$ is a Directed graph (or digraph) if the edges have a direction associated with them and do not consists multiple edges. A directed graph is strongly connected if it contains a directed path from u to v and a directed path from v to u for every pair of vertices u,v. They are commonly used to define a hierarchy or a causal effect from one vertex to other.

Weighted graphs A graph $G = (V, E)$ called Weighted if the edges of the graph are assosiated with a weight $w(e)$ [18]. The weights can be positive or negative, integers or decimal and these properties affects the performance and the complexity of many graph algorithms. Graphs with weights

Multigraphs A graph $G = (V, E)$ is a multigraph if it contains multiple edges between the same vertices. For some authors, the terms pseudograph and multigraph are synonymous. For others, a multigraph can not contain loops and a pseudograph is a multigraph with loops. More formaly a graph $G = (V, E)$ is a multigraph if V is a set and E is a multiset of 2-element subsets of V.

Hypergraphs A graph G is a hypergraph if an edge can join any number of vertices. Formally, a hypergraph H is a pair $H = (X, E)$ where X is a set of vertices, and E is a set of hyperedges or edges. Therefore, E is a subset of $\mathcal{P}(X) \setminus \{\emptyset\}$, where $P(X)$ is the power set of X [19].

2.1.4 Important Graph Classes

Bipartite graph A graph $G = (V, E)$ is bipartite if its vertices can be partitioned into two disjoint subsets V_1 and V_2 such that each edge connects a vertex from V_1 to one from V_2. Equivalently, a graph $G = (V, E)$ is bipartite if do not contain any odd length cycles. Bipartite graphs are arise naturally, when modelling relations between two mutually disjoint classes of objects.

Complete graph A graph $G = (V, E)$ is complete if each vertex is connected to all others vertices of the graph with one edge (all its nodes are interconnected). If the graph G is undirected every pair of distinct vertices is connected by a unique edge and if G is directed every pair of distinct vertices is connected by a pair of unique edges (one in each direction).

Cycle graph A graph $G = (V, E)$ is a cycle graph (also called as cyclic,circular) if

consists at least one cycle, i.e. a number of vertices of the graph are connected in a closed chain. The number of vertices in Cycle, equals the number of edges, and every vertex has degree 2 in an undirected graph and in-degree 1,uniform out-degree 1 in a directed graph.

Planar graph A graph $G = (V, E)$ is planar if can be embedded in the plane, i.e. it can be drawn in a plane without graph edges crossing. Every graph that can be drawn on a plane can be drawn on the sphere as well, and vice versa. For a simple, connected, planar graph with $|V|$ vertices and $|E|$ edges, it is possible to determine in time O(n) whether the graph be planar or not if the following simple conditions hold:

Condition 1. *If* $v \geq 3$ *then* $e \leq 3v - 6$.

Condition 2. *If* $v \geq 3$ *and there are no cycles of length* 3*, then* $e \leq 2v - 4$.

Simple graph A simple graph as his name, is the simplest existing graph class. A unweighted, undirected graph $G = (V, E)$, containing no graph loops or multiple edges is defined as Simple Graph. In addition may be either connected or disconnected.

Tree Tree is a connected graph $G = (V, E)$ without having any cycle. If the edges of the tree are directed, then it also can be considered as a special case of a digraph with the constrains, that a node may have at most one parent, and that no cycles are allowed. Trees are commonly used to store, manage and represent hierarchical data (e.g. sorted lists, work-flows etc) as their data structure is extremely fast for traversal operations.

2.1.5 Graph Properties

Graph properties are the basic characteristics of a graph $G = (V, E)$. They are also having a big impact on which algorithms and important measures are appropriate and efficient to use but they are more deleted to define the structure of a network, in order to contrast two or more of them.

Density of a graph $G = (V, E)$ measures how many edges are in set E compared to the maximum possible number of edges between vertices in set V. The Density values ranging between 0 (a graph having no edges Null-Empty Graph) and 1 (Complete graph) and is defined as:

Definition 1. *For undirected simple graphs, the graph density is defined as:*

$$D = \frac{2|E|}{|V|\,(|V| - 1)}$$

Definition 2. *For directed simple graphs, the graph density is defined as:*

$$D = \frac{|E|}{|V|\,(|V| - 1)}$$

where E is the number of edges and V is the number of vertices in the graph.

The distinction between sparse and dense graphs is rather vague, and depends on the context but generally, a graph $G = (V, E)$ is defined as a dense graph if the number of edges is close to the maximal number of edges, i.e. the number of edges is greater than or equal to $V \cdot logV$. While, a graph $G = (V, E)$ is defined as sparse if number of edges is lower than $V \cdot logV$.

Average degree of a graph $G = (V, E)$ is closely related to the density by forming an another measure of how many edges are in set E compared to number of vertices in set V. More formally Average degree for undirected simple graphs, is defined as: $\frac{2|E|}{|V|}$ and for directed simple graphs, is defined as: $\frac{2|E|}{|V|}$

Average path length of a graph $G = (V, E)$ is defined as the average length of the shortest paths over all possible pairs of vertices in G. It constitutes one of the most robust measures of graph topology by distinguish an easily from a complicated negotiable network.

Diameter of a graph $G = (V, E)$, is the maximum eccentricity of any vertex in the graph, i.e. the longest shortest path between any two graph vertices (u,v) of a graph.

Radius of a graph $G = (V, E)$, is the smallest eccentricity over all the vertices in the graph. i.e. the shortest path between any two graph vertices (u,v) of a graph.

2.1.6 Importance Measures

Importance (also known as Centrality) measures, produce rankings which seek to identify the role and importance of any vertex in a graph. Depending on what we mean by importance, there are various measures of centrality that have been proposed over the years, in order to quantify such notions of importance. According to Freeman in 1978 [20] "There is certainly no unanimity on exactly what centrality is or on its conceptual foundations, and there is little agreement on the proper procedure for its measurement". In order to better understand the nature and importance of these measures, we can divide them in three basic categories, Geometric, Path-based and Spectral.

2.1.6.1 Geometric Measures

Degree The simplest importance measure for a node is the Degree, that is defined as the number of edges incident to a node (i.e. how many adjacent nodes, a node has).

Definition 3. ***(Degree)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Degree of a node $v \in V_S$ is defined as follows:*

$$DE(v) = deg(v) \tag{2.1}$$

where $deg(v)$ is the number of edges incident to the node.

The corresponding algorithm needs $O(V_S + E_S)$ time to compute the degree for all nodes in the RDF/S schema graph.

Ego Centrality The Ego Centrality measure was first introduced in the iManage Cancer project [2] for identifying important nodes. In order to compute the Ego centrality of a node v we need the induced sub-graph of G which contains v, its neighbors, and all the edges between them. Egotism is the characteristic that defines a person referred to his own views and interests as the most important. With Ego Centrality we want to show how important a node is to his neighborhood.

Definition 4. ***(Ego Centrality)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Ego Centrality of a node $v \in G_S$ is defined as follows:*

$$EC(v) = \sum_{i=1}^{i=n_{in}} W_i * 1/DE_{out}(v_i) + \sum_{i=1}^{i=n_{out}} W_i * DE_{in}(v_i) \tag{2.2}$$

where:

$$W = \sum_{i=1}^{i=n_{in}} 1/DE_{out}(v_i) + \sum_{i=1}^{i=n_{out}} 1/DE_{in}(v_i) \tag{2.3}$$

n_{in} is the set of the neighbors from the incoming and n_{out} from outgoing edges to a node. $DE_{in}(v)$ is the incoming and $DE_{out}(v)$ is the outgoing degree of node v.

The complexity of the corresponding algorithm for computing the Ego Centrality of all nodes is $O(E_S + V_S)$.

Closeness The Closeness centrality was introduced by Bavelas [21] for undirected, connected networks as the reciprocal of the sum of distances from a given node to any other vertex in a graph $G_S = (V_S, E_S)$. Nodes of the graph with lower mean distance to others are defined as more central.

Definition 5. ***(Closeness Centrality)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Closeness Centrality of a node $v \in V_S$ is defined as follows:*

$$CC(v) = \sum_{u \neq v} \frac{1}{d(u, v)} \tag{2.4}$$

where d(u,v) is the distance between vertices u and v.

[2] http://imanagecancer.eu/

Harmonic Centrality The Harmonic Centrality was initially defined for undirected graphs by Rochat [22] in 2009 and later for directed graphs by Boldi and Vigna [23]. Essentially, it is a modification of the *Closeness* [23], designed to take unreachable nodes into account, by replacing the average distance with the harmonic mean of all distances. For graphs with a small diameter or infinite distances, harmonic mean behaves better than the arithmetic mean. Similar to the Closeness, the Harmonic Centrality requires the computation of the shortest paths between all nodes.

Definition 6. ***(Harmonic Centrality)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Harmonic Centrality of a node $v \in V_S$ is defined as follows:*

$$HC(v) = \frac{1}{\sum_{u \neq v} d(u, v)} \tag{2.5}$$

where d(u,v) is the distance between vertices u and v.

The algorithm for computing the Harmonic Centrality for all nodes requires $O(V_S(V_S + E_S))$ time.

Radiality The Radiality is a closeness-based measure and was first proposed by Valente and Foreman [24], in order to provide information on how close a node is to all other nodes in a graph (i.e. the integration measure of a node to a graph). In order to compute the diameter of a graph we need to compute the shortest paths between all nodes.

Definition 7. ***(Radiality)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Radiality of a node $v \in V_S$ defined as:*

$$RA(v) = \frac{1}{\sum_{u \neq v} (\Delta_G - (1/d(u, v))}$$

where Δ_{G_S} is the Diameter of graph G_s.

Obviously, the algorithm for computing the Radiality for all nodes requires $O(V_S(V_S + E_S))$ time.

2.1.6.2 Path-based Measures

Path-based or more veritably Shortest Paths-based measures are based on the assumption that information is transmitted along shortest paths. The requirements of running time and space for Path-based measures are usually far more than all the rest.

Betweenness The Betweenness measure is equal to the number of the shortest paths from all nodes to all others, that pass through that node, divided by the total number of possible shortest paths. The development of the Betweenness was proposed and published by Freeman [25] in 1977 and generalized by Brandes [26] in 2001, for weighted, unweighted, directed and undirected networks. Calculating the Betweenness for all nodes in a graph requires the computation of the shortest paths between all nodes.

Definition 8. ***(Betweenness)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Betweenness of a node $v \in V_S$ is defined as follows:*

$$BE(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2.6}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

The complexity of the Brandes algorithm is $O(V_S \cdot E_S)$ for an RDF/S schema graph $G_S = (V_S, E_S)$.

Stress centrality Stress centrality was introduced by Alfonso Shimbel [27] at 1953 and was the first centrality index, based on enumeration of shortest paths. An element is the more central the more shortest paths run through it. Calculating the Stress Centrality for all nodes in a graph requires the computation of All possible shortest paths between all nodes.

Definition 9. ***(Stress)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The Stress of a node $v \in V_S$ is defined as follows:*

$$SE(v) = \sum_{s \neq v \neq t} \sigma_{st}(v) \tag{2.7}$$

where $\sigma_{st}(v)$ is the total number of shortest paths that pass through v.

Bridging Centrality The Bridging Centrality [28] tries to identify the information flow and the topological locality of a node in a network. It is widely used for clustering or in order to identify the most critical points interrupting the information flow for network protection and robustness improvement purposes. A node with high Bridging Centrality is a node connecting densely connected components in a graph. The bridging centrality of a node is the product of the betweenness centrality and the bridging coefficient, which measures the global and local features of a node respectively.

Definition 10. ***(Bridging Centrality)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The bridging centrality of a node $v \in V_S$ is defined as follows:*

$$BC(v) = B_C(v) \cdot BE(v) \tag{2.8}$$

where $B_C(v)$ is the bridging coefficient of a node which determines how well the node is located between high degree nodes and $BE(v)$ is the betweenness centrality. The bridging coefficient of a node v is defined:

$$B_C(v) = \frac{DE(v)^{-1}}{\sum_{i \in N(v)} \frac{1}{DE(i)}} \tag{2.9}$$

where $DE(v)$ is the degree of node v, and $N(v)$ is the set of the neighbors of that node.

The algorithm for computing the Bridging Centrality for a node requires $O(V_S \cdot E_S)$ time.

2.1.6.3 Spectral Measures

Algorithms used to define Spectral measures, compute the left dominant eigenvector of a non-negative matrix that describes the link structure of the given graph and use the entries of this eigenvector as the node weights. Kleinberg's HITS algorithm, the PageRank algorithm of Sergey Brin and Larry Page, and the SALSA algorithm of Lempel and Moran are the most famous and commonly used algorithms that assign weights to each node of a network by using the link structure.

Katz's Index Katz's Index centrality of a node was introduced by Leo Katz in 1953 and is used to measure the relative degree of influence of an actor (node) within a social network (graph) [29]. Katz centrality of a node v, computed by measuring the number of the immediate neighbors and all other nodes of the network that can reach node v, penalized by an attenuation factor α. Each path between a pair of nodes is assigned a weight determined by $alpha$ and the distance between nodes as α^d. The linear algebra formulation of the Katz's Index is:

$$\overrightarrow{C}_{katz} = ((I - \alpha A^T)^{-1} - I)\overrightarrow{I} \tag{2.10}$$

where I is the identity matrix, A^T denotes the transposed matrix of A and $(I - \alpha A^T)^{-1}$ denotes matrix inversion of the term $(I - \alpha A^T)$.

PageRank PageRank was developed by Larry Page in 1998 [30], one of the founders of Google and was used by Google Search to rank websites in their search engine results. It is providing a global ranking of all web pages, regardless of their content, based solely on their location in the Web's graph structure. It is also one of the most discussed and quoted spectral indices in use today. Pagerank of a node can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web. Thus PageRank is the unique vector p satisfying

$$p = \alpha \cdot p \cdot \bar{A} + (1 - \alpha)v \tag{2.11}$$

where $\bar{A}$ is the l_1-normalized adjacency matrix of the graph, $\alpha \in [0..1)$ is a damping factor, and v is a preference vector (which must be a distribution).

PageRank is now regularly used in bibliometrics, social and information network analysis, system analysis, road networks, as well as biology, chemistry, neuroscience, physics and for link prediction and recommendation [31]. For a long time, PageRank was an important metric relating to the quality of a site but in april of 2014 Google announced that will stop using PageRank by replacing it with the Domain Authority method.

HITS Hyperlink-Induced Topic Search (HITS; also known as hubs and authorities) is a link analysis algorithm that rates Web pages. It was introduced by Jon Kleinberg in 1999 [32], to identifies good authorities and hubs for a topic by assigning two numbers to a page: an authority and a hub weight. The algorithm that calculates the hub and authority values for all nodes in a graph requires a series of iterations, where each consisting of two basic phases, the Authority and the hub update. On the first phase each node's Authority score be equal to the sum of the Hub Scores of each node that points to it. That is, a node is given a high authority score by being linked from pages that are recognized as Hubs for information. On the second phase each node's Hub Score to be equal to the sum of the Authority Scores of each node that it points to. That is, a node is given a high hub score by linking to nodes that are considered to be authorities on the subject. The initial hub and authority score of each node is equal to 1 and it is necessary to normalize the matrix after every iteration.

SALSA Stochastic Approach for Link-Structure Analysis (SALSA) is a web page ranking algorithm designed by R. Lempel and S. Moran [33], to assign high scores to hub and authority web pages based on the quantity of hyperlinks among them. SALSA combines key ideas from HITS and PageRank by computing the neighborhood graph as HITS but defines hub score and authority score by performing two independent random walks on the neighborhood graph, a hub walk and an authority walk, thus adopting a key idea of PageRank. The approach is based upon the theory of Markov chains, and relies on the stochastic properties of random walks performed on our collection of pages. Furthermore SALSA can be seen as an improvement of HITS because it is computationally lighter since its ranking is equivalent to a weighted in/out degree ranking.

2.1.7 Graph Algorithms

Graph algorithms are the intersection of computer science and graph theory, i.e. Graph algorithms solve problems related to graph theory. In this thesis we have to solve the shortest path problem and the minimum Spanning tree Problem with exact algorithms, in order to provide an approximation (close to optimal) solution for the Graph Steiner Tree Problem. A look back at the history, will help us to understand why these problems are closely related. The first solution of minimum spanning tree problem was published in 1926 by Czech mathematician Otakar Borůvka [34], as a method of constructing an efficient electricity network for Moravia. In 1930 Vojtěch Jarník [35] thought of an improvement on Borůvka's algorithm, publish a new (but with no complexity improvement) solution of the problem with one more generic algorithm. This algorithm is the well known Prim algorithm for the Minimum Spanning Tree problem, who was re-discovered by Prim in 1957 [36]. Then independently of all these, in 1956 Edsger W. Dijkstra provided a solution for the single shortest path problem by rediscovering Jarník, Prim's algorithm. Finally Dijkstra published the algorithm two years later in 1959 [37], because he thought this may not be very important.

2.1.7.1 Shortest path problem

The shortest path problem is the problem of finding a path between two vertices in a graph $G = (V, E)$, such that the sum of the weights of its constituent edges is minimized.

The single-source shortest path problem, find shortest paths from a source vertex v to all other vertices in the graph. The fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights is the Dijkstra's algorithm, based on a Fibonacci heap and a running time complexity $O(E + VlogV)$ [38].

The all-pairs shortest path problem, find shortest paths between every pair of vertices (u, v) in the graph. This problem was introduced by Shimbel in 1953 [27], with the fastest known algorithm be the Floyd–Warshall [39],[40], with polynomial running time $O(V^3)$.

In the case of Unweighted Graphs using a Breadth-first search(BFS) requires $O(V + E)$ time (where E is $O(V)$) for each iteration, which is fairly better than $O(E + VlogV)$, the time complexity of the Dijkstra's algorithm. Exploring the Graph is structurally the same in both algorithms, with the main difference the data structure these algorithms employee. Dijkstra's implementation is based on a priority queue with amortized running time $O(logn)$ for the delete operation whereas BFS is based on a regular queue with $O(1)$ delete operation time. All-pairs shortest paths for unweighted undirected graphs can be computed in $O(V * E)$ time on a pointer machine [41] or in $O(V \cdot (V + E))$ time by running the BFS algorithm for each node of the graph.

2.1.7.2 Minimum spanning tree

A minimum spanning tree (MST) of a weighted graph $G = (V, E)$, is a subset E' of the edges E that connects all the vertices V together, with the minimum total edge weight and without any cycles. A solution of this problem can be provided by Borůvka's, Prim's and Kruskal's algorithm in polynomial time $O(E \cdot logV)$. When a graph is unweighted (all edges have the same weight), any spanning tree is a minimum spanning tree. In this case, any algorithm that solves graph reachability, like BFS or DFS, solves MST in time $O(V + E)$ linear in the number of edges. The computation time for finding a minimum cost spanning tree (MST) can be reduced from $O(E \cdot logE + V \cdot logV)$ to $O(E + V \cdot logV)$ if the edge weights are integers in the range 1 to $|V|$. Kruskal's algorithm can also be implemented with a Counting-Sort ($O(V+E)$ running time) instead of a Comparison-Sort ($O(E \cdot logE)$) to sort the edges. Then the problem can be solved in $O(V + E + V \cdot logV) = O(E + V \cdot logV)$ time. Also the edge weights can be represented in binary to be further used by determenistic algorithms that provide a solution with $O(V + E)$ integer operations [42].

2.1.8 Np Completeness

In computational complexity theory, A problem is NP-complete when it is both in NP (verifiable in nondeterministic polynomial time) and NP-hard (any NP-problem can be translated into this problem). Differentially from the problems discussed above, there are many NP-complete problems defined in graph theory and one of them is the Graph Steiner Tree Problem (GSTP). When we have to deal with NP-complete problems, we cannot

expect to find polynomial time algorithms to solve them exactly. Most of these problems have been solved with exact algorithms, that take exponential time to the number of nodes, to provide a solution. This means that a solution for a non-small graph can not been found or it takes too many time. Usually in these cases we have to develop heuristics in order to provide an approximation solution with a good lower bound.

2.1.9 Steiner Tree Problem

The Steiner tree problem, or minimum Steiner tree problem, named after Jakob Steiner, is a fundamental combinatorial optimization problem applicable on VLSI design, computational biology, transportation, relation databases etc. [43]. The Steiner tree problem in graphs (also called Graph Steiner Tree Problem (GSTP)) can be seen as a generalization of the non-negative shortest path problem and the minimum spanning tree problem. Generally, we have to find a minimum spanning tree for a given subset of vertices of a graph $G = (V, E)$. The GSTP is NP-hard [44] and remains NP-complete if all edge weights are equal, even if the graph is a planar or bipartite [43]. For further exploration, a Compendium on Steiner Tree and related optimization problems is available on-line from M. Hauptmann and M. Karpinski [45]. Formally the GSTP is defined as:

Definition 11. ***(The Graph Steiner-Tree problem (GSTP))*** *Given an undirected graph $G = (V, E)$, with edge weights $w : E \to \mathbb{R}^+$ and a node set of terminals $S \subseteq V$, find a minimum-weight tree T in G such that $S \subseteq V_t$ and $E_t \subseteq E$.*

2.2 NoSQL Databases

NoSQL (also known as "Not Only SQL", "Non Relational") is a widely used approach in big data and real-time web applications, as it has the ability to solve the scalability and big data performance issues that relational databases were not designed to address. The schema-less (schema-free) format of NoSQL Databases allows them to scale vertically and horizontally. Depending on the design of each application, can be scaled by adding more power (CPU, RAM) to an existing machine (vertically) and by adding more machines into your pool of resources (horizontally). In this chapter, we are going to introduce four primary types of NoSQL databases, Key-value, Wide-column, Document and Graph databases. NoSQL databases are by far the fastest growing database segment. Figure 2.2 from the independent site `http://db-engines.com/` shows the popularity changes of each category starting with January 2013.

2.2.1 Key-value Stores

The Key-value stores are the simplest of NoSQL databases and probably the simplest form, of database management systems. This type uses an associative array constructed by a single table with two columns, the Primary Key and the Value, where each key is associated with one and only one value. This model is highly scalable and simple, with extremely fast writing,reading and update operations but any more complex operation on

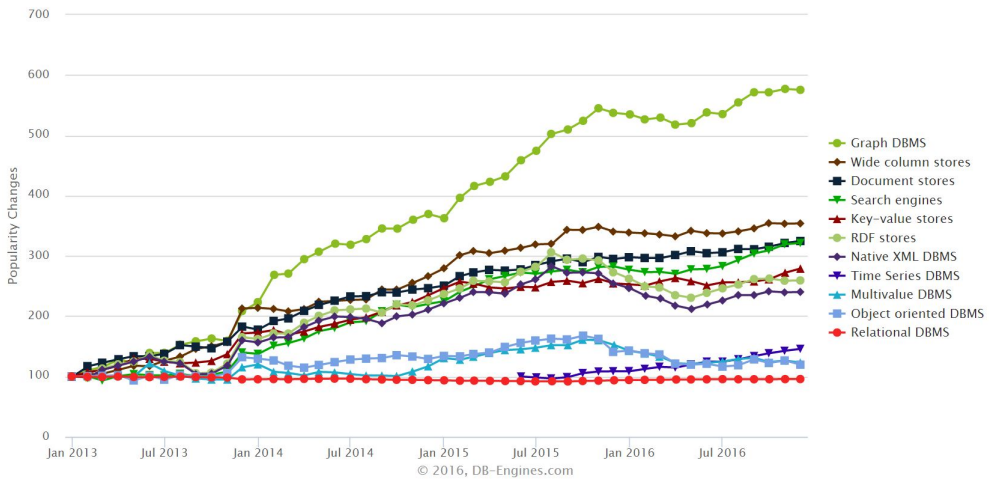


Figure 2.2: The historical trend of the databases categories' popularity.

the key value pairs is not supported and should be done outside the database. Because of their nature (Global HashMap) Key-value Stores haves a poor applicability to cases that require processing of key ranges. In order to avoid this limitation we can use Ordered Key-Value which significantly improves aggregation capabilities.

2.2.2 Document Stores

A document store database is a database that uses a document-oriented model to store data. Each record of the database is associated with a document, a structured format with schemes of arbitrary complexity, not just a map-of-maps. The structure of Document-oriented databases can be considered as an extension of the key-value store model by managing document-oriented information that the database engine uses for further optimization, like flexible schema and automatic or manual indexes. Especially with nested structures, this model has a very powerful query expressivity but is limited to keys and indexes.

2.2.3 Wide-column stores

Wide-column stores are type of key-value stores databases with the declarative characteristics of relational databases. They are using tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. Wide-column stores seem to store data in related rows, but actually, data is serialize into sections of columns of data, so Map and Reduce functions can be applied. This structure gives the ability to hold very large numbers of dynamic columns, since a record can have billions of columns. Thus sparse (semi-structured) data can be efficiently stored, indexed and analyzed over a wide column store model. However if the relationships between the data (interconnected data) are as important as the data itself, then this

model is unsuited.

2.2.4 Native Graph Databases

Graph databases are a rising tide, and big data is getting bigger. A graph database is an on-line database management system with Create, Read, Update and Delete (CRUD) operations working on a graph data model, where the data is stored as nodes and relationships. Native graph storage is specifically designed to store and manage graphs, the most generic of data structures, capable of elegantly representing any kind of data in a highly accessible way. Thus Graph Databases are very powerful for interconnected data, since they are extremely efficient and easy to use, to deal with complex but relational information (degrees,connection etc). In addition Graph databases allows a more natural modeling of data. Because of their structure Graphs are powerful representation formalism for both structured and unstructured data. In order to understand how a native graph database facilitates performant graph traversals we can see the structure of nodes and relationships on disk as shown in Figure [3] 2.3

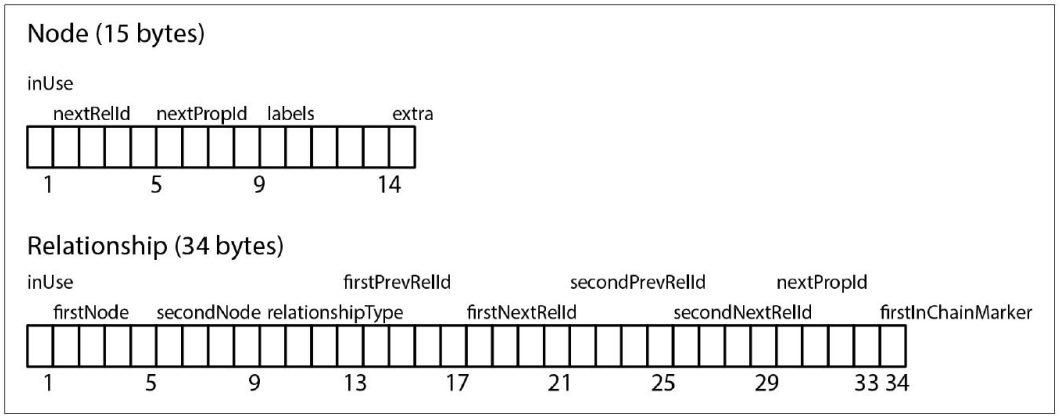


Figure 2.3: Neo4j node and relationship store file record structure.

Triple Stores Triple stores are a subcategory of graph databases, modeled around the Resource Description Framework (RDF) specifications, designed by W3C for representing data in the Web. An RDF graph is a set of RDF triples, each consisting of a subject, a predicate and an object. The set of nodes of an RDF graph is the set of subjects and objects of triples and the set of Edges are the predicates, witch denotes the relationship over the subject and the object and always points toward the object. In a triple store, the data tends to be very atomic, because they store just triples (subject, predicate, object), where the vertices of the graph tend to be primitive data types and the edges link those primitives together.

[3]Book: Graph Databases By Ian Robinson, Jim Webber, Emil Eifrem

Property Graph The property graph model is quite similar to the labeled graph but this model haves the advantage of directionality on the edges in the graph. A property graph contains by a set of vertices and a set of edges. Each vertex has a: unique identifier, set of outgoing edge and incoming edges, set of labels that denotes the roles of the node in the graph and a collection of properties defined by a map from key to value (usually quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths). Each edge has a: unique identifier, outgoing tail vertex, incoming head vertex, label that denotes the type of relationship between its two vertices and a collection of properties defined by a map from key to value. Figure [4] 2.4 provides a representative example of the property graph model.

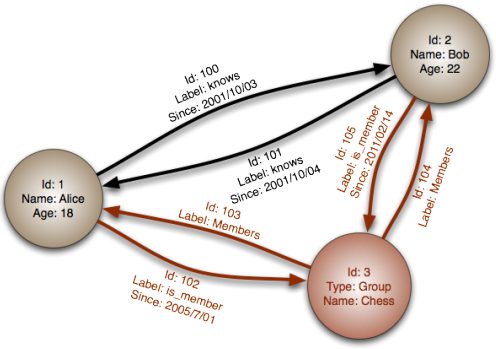


Figure 2.4: Graph databases employ nodes, properties, and edges.

Comparison Graph over Triple Store Databases Graph databases and Triple Stores focus on the linked data but they are having a different nature. Graph databases are optimized for graph traversals and can store various types of graphs, including directed graphs, undirected graphs, weighted graphs, unweighted, hyper graphs etc but Triple stores creates a graph from triples (node-edge-node) with only directed edges. This does Triples stores edge centric and Graph databases node centric. On the other hand triple stores provide inferences and are standardized with common data formats and exchange protocols for the Semantic Web.

Comparison NoSql over Relational Databases Relational database-management systems (RDBMS) model the data as a set of tables and columns, carrying out complex joins and self-joins when the dataset becomes more interrelated. But the Graph databases are optimized for connected data, so they are performing particularly well when the relationships inside your data are important and your queries depend on exploring and exploiting them. Unlike other database management systems, which require to infer connections between entities using special properties such as foreign keys, graph databases store relationship information as a first-class entity. These relationship records are organized by type and direction and may hold additional attributes. So they can provide direct access

[4] `https://en.wikipedia.org/wiki/Graph_database`

to the connected nodes as running an equivalent JOIN operation on a relation Database, that has an expensive search / match computation [46]. NoSQL databases are usually extremely faster with very large sets of data, because they can easily scale horizontally and work with(in) clusters. The schema-less format does not require structure from the beginning and offers a large amount of flexibility. Finally there is a variety of NoSQL models to suit your needs and to get the most out of the database management system - depending on your data type. We can easily understand in figure[5] 2.6 that the resulting data models in a graph database, are much simpler and at the same time more expressive than those produced using relational databases 2.5.

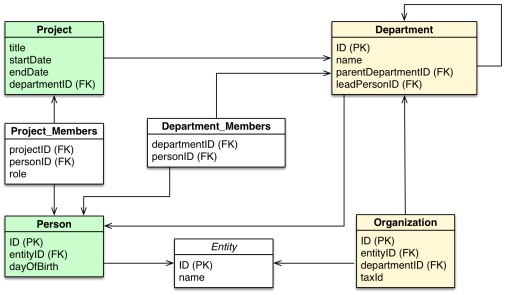


Figure 2.5: Organizational domain, modeled in a relational database.

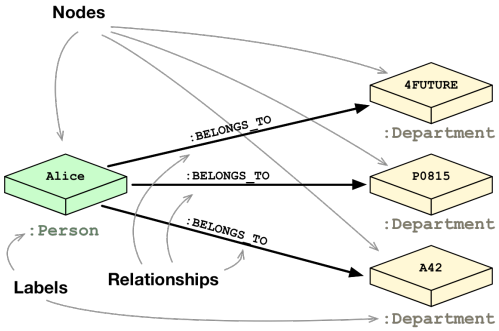


Figure 2.6: Organizational domain, modeled in the graph database.

2.3 State of the Art Importance Measures in Ontology Summarization

2.3.1 KCE Importance

In the context of ontology summarization, Peroni et al. [3] try to identify automatically the key concepts in an ontology, combining cognitive principles, lexical and topological

[5] https://neo4j.com/developer/graph-db-vs-rdbms/

measurements such as density and coverage, as well as, statistical lexical measures (popularity). In particular, the authors use the notion of natural category, which is drawn from cognitive psychology, to identify concepts that are information-rich in a psycho-linguistic sense. Two other criteria are drawn from the topology of an ontology: the notion of density highlights concepts which are information-rich in a formal knowledge representation sense, i.e., they have been richly characterized with properties and taxonomic relationships, while the notion of coverage states that the set of key concepts identified by the corresponding algorithm should maximize the coverage of the ontology with respect to its is-a hierarchy. Finally, the notion of popularity, drawn from lexical statistics, is introduced as a criterion to identify concepts that are likely to be most familiar to the users.

Definition 12. ***(KCE Importance)*** *Let $G_S = (V_S, E_S)$ be an RDF/S schema graph with V_S nodes and E_S edges. The KCE importance of a node v in G_S is defined as follows:*

$$KC(v) = D(v) + P(v) + NC_{Value}(v)$$

where $D(v)$ the weighted density of the node, $P(v)$ the weighted popularity of the node, and $NC_{Value}(v)$ a weight calculated according to psycho-linguistic criteria (see [3] for more details).

The final algorithm for computing the KCE importance for a node requires $O(V_S^2 + E_S)$ time.

2.3.2 Relevance

Another measure that has been recently proposed for identifying the most important nodes of an RDF/S KB is the Relevance [7, 5]. The Relevance tries to determine initially the importance of a node judging from the instances it contains by calculating its relative cardinality. The Relative Cardinality $RC(p(v_i, v_j))$ of an edge $p(v_i, v_j)$ is the number of the specific instance connections divided by the total number of the connections of the instances of these two nodes v_i, v_j. After that, in order to combine the notion of centrality in the schema and the distribution of the corresponding dataset, a variation of the degree centrality is defined, called in/out centrality (C^{in}/C^{out}). This is the sum of the weighted relative cardinalities of the incoming/outgoing edges. Finally to determine the importance of a node, the centrality of the other nodes is considered as well:

Definition 13. ***(Relevance of a node)***. *Assume a node $v \in \mathbf{C} \cap V_S$ in a dataset $V = \langle G_S, G_I, \lambda, \tau_c \rangle$. Assume also that $p(v_i, v) \in E_S$, $1 \leq i \leq n$ are the incoming edges of v and $p(v, v'_j) \in E_S$, $1 \leq j \leq k$ are the outgoing edges of v. Then, the relevance of v, denoted by $RE(v)$, is the following:*

$$RE(v) = \frac{C_V^{in}(v) \cdot n + C_V^{out}(v) \cdot k}{\sum_{i=1}^{n} C_V^{in}(v_i) + \sum_{j=1}^{k} C_V^{out}(v'_j)}.$$

where C^{in}/C^{out} the in and out centralities of the corresponding nodes.

Obviously, the relevance of a schema node in an RDF/S KB is determined by both its connectivity in the schema and the cardinality of the instances. Thus, the number of instances of a node is of vital importance in this measure. When the data distribution significantly changes, the focus of the entire data source is shifted as well, and as a result, the relevance of the nodes changes. The complexity of the corresponding algorithm for computing the relevance of all nodes is $O(E_S + V_S + V_I + E_I)$ [5].

Chapter 3

Methodology

3.1 Graph Summarization

"Graphs are everywhere" is a phrase that's often heard in the recent years. Social Networks, Transportation Networks, Biological Pathways and a variety of information can be represented as a Graph. This growing trend creates huge volume of data stored as graphs. The first key to solve a graph related problem is to recognize it, as a graph problem. Effective graph summarization methods are required to help users extract and understand the underlying information in large networks. Clear and precise Visualization of data can provide important and detailed information (discover relations, causality etc). A common problem that arises when the size of a graph becomes a little large (more than 100 nodes), is the viewability and usability issues. Graph visualization is a well studied field of computer science with a plethora of publications and surveys but even the state of the art algorithms fails to provide a clear representation of arbitrary large graphs. Coming ahead to this problem graph summarization becomes inevitably necessary nowadays, where data are growing exponentially.

3.2 Preliminaries

In this thesis, we focus on RDF/S KBs, as RDF is among the widely-used standards for publishing and representing data on the Web. The representation of knowledge in RDF is based on triples of the form (subject, predicate, object). RDF datasets have attached semantics through RDFS [47], a vocabulary description language. Here, we will follow an approach similar to [48], which imposes a convenient graph-theoretic view of RDF data that is closer to the way the users perceive their datasets.

A knowledge base (KB) is a repository of knowledge used to store complex structured and unstructured information in a computer system. It is a machine-readable resource for the dissemination of information, supporting human decision-making, learning and action. More generally a KB promotes the collection, organization and retrieval of knowledge, that lead users to solutions of problems they have.

Representation of RDF data is based on three disjoint and infinite sets of *resources*,

namely: URIs ($\mathcal{U}$), literals ($\mathcal{L}$) and blank nodes ($\mathcal{B}$). We impose typing on resources, so we consider 3 disjoint sets of resources: classes ($\mathbf{C} \subseteq \mathcal{U} \cup \mathcal{B}$), properties ($\mathbf{P} \subseteq \mathcal{U}$), and individuals ($\mathbf{I} \subseteq \mathcal{U} \cup \mathcal{B}$). The set $\mathbf{C}$ includes all classes, including RDFS classes and XML datatypes (e.g., xsd:string, xsd:integer). The set $\mathbf{P}$ includes all properties, except rdf:type, which connects individuals with the classes they are instantiated under. The set $\mathbf{I}$ includes all individuals, but not literals. In addition, we should note that our approach adopts the unique name assumption, i.e. that resources that are identified by different URIs are different.

In this work, we separate between the schema and instances of an RDF/S KB, represented in separate graphs (G_S, G_I, respectively). The schema graph contains all classes and the properties they are associated with; note that multiple domains/ranges per property are allowed, by having the property URI be a label on the edge (via a labelling function λ) rather than the edge itself. The instance graph contains all individuals, and the instantiations of schema properties; the labelling function λ applies here as well for the same reasons. Finally, the two graphs are related via the τ_c function, which determines which class(es) each individual is instantiated under. Formally:

Definition 14. ***(RDF/S KB)*** *An RDF/S KB is a tuple $V = \langle G_S, G_I, \lambda, \tau_c \rangle$, where:*

- *G_S is a labelled directed graph $G_S = (V_S, E_S)$ such that V_S, E_S are the nodes and edges of G_S, respectively, and $V_S \subseteq \mathbf{C} \cup \mathcal{L}$.*
- *G_I is a labelled directed graph $G_I = (V_I, E_I)$ such that V_I, E_I are the nodes and edges of G_I, respectively, and $V_I \subseteq \mathbf{I} \cup \mathcal{L}$.*
- *A labelling function $\lambda : E_S \cup E_I \mapsto 2^{\mathbf{P}}$ determines the property URI that each edge corresponds to (properties with multiple domains/ranges may appear in more than one edge).*
- *A function $\tau_c : \mathbf{I} \mapsto 2^{\mathbf{C}}$ associating each individual with the classes that it is instantiated under.*

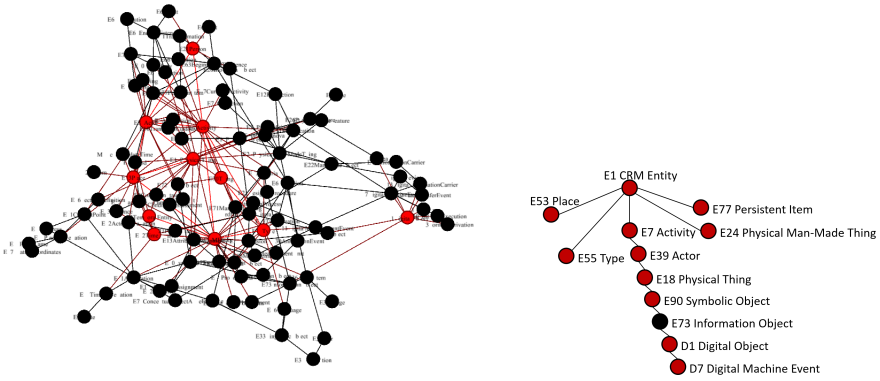
For simplicity, we forego extra requirements related to RDFS inference (subsumption, instantiation) and validity (e.g., that the source and target of property instances should be instantiated under the property's domain/range, respectively), because these are not relevant for our results below and would significantly complicate our definitions.

Our approach is working only for ontologies respecting the aforementioned definition/requirements.

In the following, we will write $p(v_1, v_2)$ to denote an edge e in G_S (where $v_1, v_2 \in V_S$) or G_I (where $v_1, v_2 \in V_I$) from node v_1 to node v_2 such that $\lambda(e) = p$.

In addition, a path from a schema node v_s to v_i, denoted by $path(v_s, v_i)$, is the finite sequence of edges, which connect a sequence of nodes, starting from the node v_s and ending in the node v_i. The length of a path, denoted by $d_{path(v_s,v_i)}$, is the number of the edges that exist in that path whereas $d(v_s, v_i)$ is the number of the edges that exist in the shortest path linking v_s and v_i.

Finally, having a schema graph G_S, the closure of G_S, denoted by $Cl(G_S)$, contains all triples that can be inferred from G_S using inference. From now on when we use G_S



(a) (b)

Figure 3.1: The schema graph of the CRMdig ontology (a) and the corresponding schema summary (b).

we will mean $Cl(G_S)$ for reasons of simplicity unless stated otherwise. This is to ensure that the result will be the same, independent of the number of inferences applied an input schema graph G_s.

Now as an example, consider the CRMdig[1] ontology shown in Fig. 3.1(a) used to encode metadata about the steps and methods of production of digitization products and synthetic digital representations. Obviously, it is really difficult to examine all the nodes in order to understand the schema. However, examining only the schema summary as identified by our algorithms shown in Fig. 3.1(b), allows the user to get a quick overview on the contents of the ontology, identifying and linking the most important nodes. In the result summary, the nodes in red are the most important nodes as identified by the Ego Centrality measure (as we shall see in the sequel) and the node in black is the node introduced in the summary in order to link the most important nodes and to produce a valid sub-schema graph. We have to note in addition, that our approach handles OWL ontologies as well, considering however only the RDF/S fragment of these ontologies.

3.3 Summarization of the Graph in Terms of Important Graph Structures

Schema summarization aims to highlight the most representative concepts of a schema, preserving important information and reducing the size and the complexity of the schema [17]. Despite the significance of the problem, there is still no universally accepted measurement on the importance of nodes in an RDF/S graph. In this section, we describe eight alternative measures that have been proposed for capturing importance in directed graphs and RDF/S KBs that we intend to explore for summarization purposes. We se-

[1] http://www.ics.forth.gr/isl/index_main.php?c=656

Table 3.1: The complexities of the examined importance measures.

Measure	Complexity
Degree	$O(V_S + E_S)$
Betweeness	$O(V_S \cdot (V_S + E_S))$
Bridging Centrality	$O(V_S \cdot (V_S + E_S))$
Harmonic Centrality	$O(V_S \cdot (V_S + E_S))$
Radiality	$O(V_S \cdot (V_S + E_S))$
Ego Centrality	$O(V_S + E_S)$
KCE Importance	$O(V_S^2 + E_S)$
Relevance	$O(E_S + V_S + V_I + E_I)$

lected the *Betweeness*, the *Bridging Centrality*, the *Degree*, the *Harmonic Centrality*, the *Radiality* and the *Ego Centrality* as they constitute the state of the art geometric measures for generic graphs [23] and the *KCE importance* and the *Relevance* as they constitute the state of the art in ontology summarization [7, 5]. We do not compare aforementioned measures, with spectral measures (HITS, PageRank etc.) because they are based on external factors and spectral properties and they are commonly used for other purposes (describe the network, identify subgraphs, cliques, chromatic number etc.). The complexities of all aforementioned measures are shown in Table 3.1. New definitions of directed centrality measures are provided from CentiScaPe authors [49], in network analysis, centrality measures. Importance of those measures is a function of distances, that identifies the most central or important nodes on a graph.

3.4 Summarized Importance Value

In order to take into consideration the instances of each class, we adapt the aforementioned importance measures. To achieve that we first normalize each importance measures IM_i on a scale of 0 to 1:

$$normal(IM_i(v)) = \frac{IM_i(v) - \min(IM_i(g))}{\max(IM_i(g)) - \min(IM_i(g))} \tag{3.1}$$

Where i is one of the DE, BE, BC, HC, RA, EC. $IM_i(v)$ is the importance value of a node v in the schema graph g, $min(IM_i(g))$ is the minimum and $max(IM_i(g))$ is the maximum importance value in the graph. Similarly, we normalize the number of instances (InstV) that belong to a schema node. As such the *summarized importance value* of each node is the sum of the normalized values of the importance measures and the instances.

$$SIM_i(v) = normal(IM_i(v)) + normal(InstV(v)) \tag{3.2}$$

3.5 Construction of the RDF/S Summary Schema Graph

Having selected the most important nodes of a directed schema graph (also known as *terminals* in graph problems), it is now time to focus on the paths that link those nodes, trying to produce a valid sub-schema graph. We have to note that in the stage of constructing the final RDF/S summary schema graph we are not interested in the direction of the edges since we only want to get a connected schema graph.

The latest approaches in the area [5] are trying to identify a maximum cost spanning tree in the graph and to link the selected nodes using paths from a selected maximum-cost spanning tree:

Definition 15. ***(The Maximum-Cost Spanning Tree (MST) problem*** *Given an undirected graph $G = (V, E)$, with edge weights $w : E \rightarrow \mathbb{R}^+$ find a spanning tree T in G of maximum total edge cost such that $E_t \subseteq E$.*

In [5] the authors consider as G the G_S ignoring the direction in the edges. In addition, the edge weight is the sum of the weights of the nodes linked by that edge. However, the main problem there is that although the selected paths maximize the total edge cost, they might not maximize the total weight of the selected subgraph - the summary. A second problem there is that many additional nodes are introduced in the result, since there is only one path to be selected between two nodes and in this path many other not important nodes might appear as well.

Besides the aforementioned approach, the problem can also be modelled as a variation of the well-known graph Steiner-Tree problem (GSTP) exploiting the optimal solutions there proposed by Hakimi [50] and Levin [51] independently.

In our case, we consider as G the G_S ignoring as well the direction in the edges. In addition, the set of terminals is the set of the most important nodes as they are selected using the measures from Section 3 whereas all nodes have equal weights. As such we try to minimize the weight of the selected tree, i.e. to minimize the number of the additional nodes introduced in the selected summary schema graph.

3.5.1 Algorithms, Approximation & Heuristics

There had been various exact algorithms for the GSTP. Hakimi [50] proposed the first brute force algorithm that enumerates all minimum spanning trees of sub-networks of G included by super-sets of terminals that runs in $O(2^{V-t} \cdot V^2 + V^3)$. The first dynamic programming algorithms were proposed independently by Dreyfus & Wagner [52] and by Levin [51]. The former runs in $O(3^t \cdot V + 2^t \cdot V^2 + V^3)$ whereas the latter in $O(3^t \cdot V + 2^t \cdot V^2 + t^2 \cdot V)$ and they are based on the optimal decomposition property by creating two node sets, removing one node at each step and solving the GSTP by connecting each set. Levin's method uses a recursive optimization approach that pre-computes the possible sub-trees.

Since all aforementioned algorithms have an exponential running time, various approximations such as [53, 54, 55, 56] have been proposed in order to find good approximate solutions for large networks. The approximation quality of the algorithms summa-

Table 3.2: Performance ratios for known approximation algorithms for the Steiner tree problem in graphs.

Authors	Year	Ratio
Moore	1968	2.000
Zelikovsky	1990	1.834
Berman, Ramaiyer	1991	1.734
Zelikovsky	1995	1.694
Pramel, Steger	1996	1.667
Karpinski, Zelikovsky	1996	1.644
Hougardy, Pramel	1999	1.598
Robins, Zelikovsky	2000	1.550

rized on Table 3.2, shows that the development improved substantially. The quality of an approximation algorithm A is usually measured by its performance ratio R_A, which is the maximum ratio between the solution of algorithm A and the optimum solution [57]. More formally, performance ratio is defined as:

Definition 16. $R_A = \left\{ \frac{A(I)}{Opt(I)} \mid \textit{all instanses } I \right\}$

A central theme in these approximations, is the use of some principles known from the two classic algorithms for solving the minimum spanning tree problem, Prim's and Kruskal's [56]. The generic idea is to build up a feasible solution by inserting the shortest paths. Two main ideas can be distinguished:

Single Component Insertion: Start with a partial solution $T = (w, 0)$ consisting of a single terminal node. T will be expanded to a feasible solution by successively inserting all terminal nodes e.g. through the computation of at most $|Q|$ shortest paths (Based on Prim's minimum spanning tree algorithm).

Component Connecting: Start with a partial solution $T = (Q, 0)$ consisting of $|Q|$ singleton components. T will be expanded to a feasible solution by repeatedly selecting components which are connected by shortest paths (Based on Kruskal's minimum spanning tree algorithm). Using these ideas, we will use the following top-three well-known and good-performing methods SDISTG, CHINS and HEUM [56]. These approximations have a worst case bound of 2, i.e., $Z_T/Z_{opt} \leq 2 \cdot (1 - l/|Q|)$, where Z_T and Z_{opt} denote the objective function values of a feasible solution and an optimal solution respectively, Q the set of terminals and l a constant [58].

SDISTG (Shortest distance graph)

1. Construct a complete graph G' for the node set Q (set of terminal nodes) with each edge having the weight of a shortest path between the corresponding nodes in G.

2. Construct a minimum spanning tree T' of G'.

3. Replace each edge of the tree T' by its corresponding shortest path in G.

CHINS (Cheapest insertion)

1. Start with a partial solution $T = (w, 0)$ consisting of a single terminal node w.
2. While T does not contain all terminal nodes do
   find the nearest nodes $u* \in V_t$ and $p*$ being a terminal node not in V_t. Construct the Tree by adding the vertices and the edges of path $p(u, p*)$.

HEUM (Heuristic measure)

1. Start with a partial solution $T = (Q, 0)$ consisting of Q singleton components (terminal nodes).
2. While T is not connected do
   choose a node u using a heuristic function F and unite the two components of T which are nearest to u by combining them with u via shortest paths (the nodes and edges of these paths are added to T).
   Up to now the most promising way is to choose F according to:
   $\min_{i \le t \le \sigma, } \left\{ \frac{1}{t} \cdot \sum_{i=0}^{t} d(u, T_i) \right\}$ where $T_0, \ldots, T_\sigma$ are the components of T such that $d(u, T_i) \le d(u, T_i) \forall i, j \in \sigma, i < j$.

Besides these approximations, many heuristics can be employed to improve even more the corresponding algorithms. The most promising ones are the I-MST+P and the TRAFO [56]. I-MST+P is a pruning routine that ensures that all leaves are terminal nodes whereas TRAFO transforms a feasible solution to another one trying to overcome the deficiency of bad local optima by allowing the temporary deterioration of the actual solutions. In this paper we use only the I-MST+P since TRAFO requires considerable more time to run and the improvements are insignificant - due to the sparsity of the examined ontologies.

I-MST + P (Improvement procedure with MST+P)

1. Let $T = (V_t, E_t)$ be a feasible solution of the GSTP. The subgraph of G induced by V_t will be defined as G_t.
2. Construct a minimum spanning tree $T = (V_t', E_t')$ of G_t.
3. While there exists a leaf of T' being a terminal do
   delete that leaf and its incident edge.

TRAFO (Transformation)

1. Let $T = (V_t, E_t)$ be a feasible solution of SP. The subgraph of G induced by V_t will be defined as G_t.
2. Select randomly an edge $(i, j) \in E_t$ and delete it from E_t.

3. For $h = i, j$ do
   Select a path in T connecting h with a vertex $h*$ s.t. all vertices of this path but $h*$ are Steiner vertices of degree 2 (with respect to T) and remove this path from T

4. Let T_i and T_j be the two disconnected components of T. Find nearest vertices $i*$ and $j*$ belonging to T_i and T_j, respectively, and add the vertices and edges of a shortest path between $i*$ and $j*$ to T.

3.5.1.1 Complexities

Distance Networks and Shortest paths. Using a Breadth-first search(BFS) requires $O(V + E)$ time (where E is $O(V)$) for each iteration, which is fairly better than $O(E + VlogV)$, the time complexity of the Dijkstra's algorithm. Exploring the Graph is structurally the same in both algorithms, with the main difference the data structure these algorithms employee. Dijkstra's implementation is based on a priority queue with amortized running time $O(logn)$ for the delete operation whereas BFS is based on a regular queue with $O(1)$ delete operation time. All-pairs shortest paths for unweighted undirected graphs can be computed in $O(V * E)$ time on a pointer machine [41] or in $O(V \cdot (V + E))$ time by running the BFS algorithm for each node of the graph.

Maximum cost spanning tree. Maximum cost spanning tree can be computed by negating the weights for each edge of the graph and applying Kruskal's algorithm [59] which takes $O(E \cdot logV)$ time. When a graph is unweighted(all edges have the same weight), any spanning tree is a minimum spanning tree. In this case, any algorithm that solves graph reachability, like BFS or DFS, solves MST in time $O(V + E)$ linear in the number of edges. The computation time for finding a maximum cost spanning tree (MST) can be reduced from $O(E \cdot logE + V \cdot logV)$ to $O(E + V \cdot logV)$ if the edge weights are integers in the range 1 to $|V|$. Kruskal's algorithm can also be implemented with a Counting-Sort ($O(V + E)$ running time) instead of a Comparison-Sort ($O(E \cdot logE)$) to sort the edges. Then the problem can be solved in $O(V + E + V \cdot logV) = O(E + V \cdot logV)$ time. Also the edge weights can be represented in binary to be further used by determenistic algorithms that provide a solution with $O(V + E)$ integer operations [42].

SDISTG, CHINS and HEUM. The complexity of those algorithms differs, due to the usage of different heuristics. Table 3.3 provides the worsts case complexity of those algorithms for weighted and un-weighted graphs.

Table 3.3: Worst-case complexities of the algorithms employed for linking the most important nodes in a graph.

Algorithm	Weighted graph	Un-weighted graph
MST	$O(E \cdot logV)$	$O(\|V + E\|)$
SDISTG	$O(Q \cdot \|VlogV\|)$	$O(Q \cdot \|V + E\|)$
CHINS	$O(Q \cdot \|VlogV\|)$	$O(Q \cdot \|V + E\|)$
HEUM	$O(V \cdot \|VlogV\|)$	$O(V \cdot \|V + E\|)$

Chapter 4

Evaluation

Despite the significance of the problem, there is still no universally accepted measurement on the importance of vertices in an RDF/S graph. In our approach, we try to elicit this information from the structure of the graph and the instances of the KB. Our goal is to produce a simple and expressive graph that presents an overview of the schema and also provides an intuition about the corresponding stored data.

4.1 Evaluation of Measures for Assessing Vertices' Ranking.

4.1.1 Spearman's Rank Correlation Coefficient

Initially we tried to understand the statistical dependence between the ranking of the aforementioned measures using the Spearman's rank correlation coefficient [60], a non-parametric measure of rank correlation. It assesses how well the relationship (measures the strength and direction of association) between two variables can be described using a monotonic function. The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the ranked variables.

Definition 17. ***(Spearman's rank correlation coefficient)*** *For a sample of size n, the n raw scores X_i,Y_i are converted to ranks rgY_i, rgX_i and r_s is computed from:*

$$r_s = \rho_{rg_X, rg_Y} = \frac{\mathrm{cov}(rg_X, rg_Y)}{\sigma_{rg_X, rg_Y}}$$

where:

- *ρ denotes the usual Pearson correlation coefficient, but applied to the rank variables.*
- *$\mathrm{cov}(rg_X, rg_Y)$ is the covariance of the rank variables.*
- *σ_{rg_X} and σ_{rg_Y} are the standard deviations of the rank variables.*

Spearman correlation indicates the direction of the association between two variables X, Y. It can vary between -1 and 1, where 1 is total positive correlation, 0 is no correlation, and -1 is total negative correlation. If Y tends to decrease when X increases, the Spearman correlation coefficient is negative. A Spearman correlation of zero indicates that there is no tendency for Y to either increase or decrease when X increases. When X and Y are perfectly monotonically related, the Spearman correlation coefficient becomes 1.

4.1.2 The Similarity Measure

Next, we would like to evaluate the measures identified in Section 2.1.6, 3.3 for their quality with respect to identifying the vertices' importance. Measures like precision, recall and F-measure, used by the previous works [3, 4, 17] are limited in exhibiting the added value of a summarization system because of the "disagreement due to synonymy" [61] meaning that they fail to identify closeness with the ideal result when the results are not exactly the same with the reference ones. On the other hand, content-based metrics compute the similarity between two summaries in a more reliable way [6]. In the same spirit, Maedche et al. [62] argue that ontologies can be compared at two different levels: lexical and conceptual. At the lexical level, the classes and the properties of the ontology are compared lexicographically, whereas at the conceptual level the taxonomic structures and the relations in the ontology are compared. To this direction, we use the *similarity measure*, denoted by $Sim(G_S, G_R)$, in order to define the level of agreement between an automatically produced graph summary $G_S = (V_S, E_S)$ and a reference graph summary $G_R = (V_R, E_R)$:

$$Sim(G_S, G_R) = \frac{|V_S \cap V_R| + a \cdot \sum_{i=k}^{p} \frac{1}{d_{p(c_i, c'_i)}} + b \cdot \sum_{i=m}^{n} \frac{1}{d_{p(c_i, c'_i)}}}{|V_R|}$$

where $c_k, ..., c_p$ are the classes in V_R that are sub-classes of the classes $c'_k, ..., c'_p$ of V_S and that $c_m, ..., c_n$ are the classes in V_R that are super-classes of the classes $c'_m, ..., c'_n$ of V_S.

In the above definition a and b are constants assessing the existence of sub-classes and super-classes of G_S in G_R with a different percentage. In [5] the ideal weights for RDF/S KBs have been identified to be $a = 0.6$ and $b = 0.3$ which we use in this paper as well, giving more weight to the super-classes. The idea behind that is that the super-classes, since they generalize their sub-classes, are assessed to have a higher weight than the sub-classes, which limit the information that can be retrieved. Consequently, the effectiveness of a summarization system is calculated by the average number of the similarity values between the summaries produced by the system and the set of the corresponding summaries by experts and users' queries.

4.2 Evaluating Summaries

To evaluate not only the vertices selected by the various measures but also the whole returned summary, next we evaluate the RDF/S graph edit distance between the reference summaries and the summaries generated by our algorithms. Then, we compare them with respect to the number of additional vertices they introduce in order to produce a linked schema-graph summary. For reasons of completeness, in each case we compare our approximation algorithms with the corresponding algorithm that uses the maximum-cost spanning tree (MST) for linking the most important vertices.

4.2.1 RDF/S Schema Graph Edit Distance

In this section, we evaluate as a whole the result of the approximation algorithms comparing them to the reference summaries generated by the experts and users most common queries. We only use the LUMB, the CRMdig and the eTMO ontologies since only for those ontologies we have complete reference graph summaries - for the first three ontologies only the most important vertices were selected by the experts. To evaluate the distance between the generated summaries and the reference summaries we use a measure similar to the graph edit distance:

Definition 18. ***(RDF/S schema graph edit distance)*** *Let G_S and G_R two RDF/S schema graphs. The RDF/S schema graph edit distance, denoted by $SGED(G_S, G_R)$ is defined as follows:*

$$SGED(G_S, G_R) = \min_{(e_1,\ldots,e_k)\in P(G_S,G_R)} \sum_{i=1}^{k} c(e_i)$$

where $P(G_S, G_R)$ denotes the set of edit paths transforming G_S into (a graph isomorphic) G_R and $c(e)$ the cost of each graph edit operation e.

In our case the elementary graph edit operators e and the corresponding costs are the following:

- class/property substitution by a superclass/superproperty - $c(e) = 0.6$
- class/property substitution by a subclass/subproperty - $c(e) = 0.3$
- class/property insertion - $c(e) = 1$
- class/property deletion - $c(e) = 1$

4.2.2 Additional Vertices Introduced

Next we would like to identify the overhead imposed by the algorithms for linking the most important vertices in terms of the additional vertices that are introduced. Since our approximation algorithms are based on calculating the shortest paths, we expect that they will have quite similar results on sparse graphs which do not have high diversity - as the number of paths and spanning trees are highly depended on the number of edges.

Next we try to calculate the deviation from the optimum solution for our results using as cost the number of total vertices that a solution produces - we would like this to be as close as possible to the number of the identified most important vertices. The deviation is calculated by:

$$deviation = (Z_t - Z_{opt})/Z_{opt} \cdot 100$$

where: Z_t is the cost value of a feasible solution of the algorithm and Z_{opt} is the cost value of the optimal solution. Whenever an optimal solution is unknown for a graph, Z_{opt} is defined as the cost of the best known feasible solution [56].

4.3 Execution Time

We implemented main memory-based versions of important measures and GSTP heuristics algorithms in JAVA. We focus only on the running times for computations, by ignoring processing steps of loading data and initialization, as we are interested in performance and scalability with graph sizes. Finally, to test the efficiency of our system, we measured the average time of 50 executions in order to produce the corresponding summaries of the aforementioned ontologies. The experiments run on a Intel(R) Xeon(R) CPU E5-2630 running at 2.30GHz with 64GB memory running Ubuntu 12.04 LTS.

As we can observe, the execution times of the various measures can be divided into three categories. The measures that need to compute the shortest paths of all pairs, the measures that need to iterate only the vertices and the edges of the graph and the measures that need to execute queries on external databases or combine complex measures.

The Betweenness, the Bridging Centrality, the Harmonic Centrality and the Radiality belong to the first category since they assign weights by calculating the shortest paths between all pairs of vertices. As such they have similar execution times. The Betweenness differentiates from the rest since the set of all shortest paths should be computed for each pair of vertices. The Bridging Centrality uses the Betweenness and as such it takes allmost the same time.

In the second category we find the Degree and the Ego Centrality. The Degree needs only to iterate over all edges of the graph and "submit" the weight to each node. The Ego Centrality needs one more iteration over all vertices and edges of the graph.

Finally in the third category there are complex measures such as the Relevance and the KCE importance which execute complex queries on triple stores or use complex psyco-cognitive measures and as such they need significantly more time for their execution.

For linking the most important vertices, the complexity of the MST and the SDISTG and CHINS approximation algorithms show that there is a linear function relationship between their execution time and the input data size (the number of the vertices and the edges). HEUM is the only one that has a quadratic time, and this is due to the fact that it has to construct the shortest paths of all pairs. As such SDISTG and CHINS have a better execution time as the number of terminal vertices is small and HEUM the worst execution time, which grows linearly to the number of vertices. MST is slightly faster than other algorithms because it depends on the size of graph (vertices and edges), in contrast to

CHINS and SDISTG that are highly dependent on the number of the terminals and the shortest paths between them.

4.4 Experiments

To evaluate our algorithms we performed extensive experiments using two sets of ontologies. One set where human experts were used to construct the reference summaries and on set where we used the most common queries to identify the most important nodes. More specifically:

1. **Expert Summaries:** Human experts with a good experience in ontology engineering which were familiar with the aforementioned ontologies.
2. **Users Most Common Queries:** To identify the most important vertices of those ontologies we rely on the corresponding SPARQL endpoint query logs, created by users queries.

Details on the generation of these reference summaries are given below in section 4.4.1 and section 4.4.2.

4.4.1 Experts' Summaries

To evaluate our system, we used in total six ontologies:

- **BIOSPHERE**[1]: The BIOSPHERE ontology is consisted of 87 classes and 3 properties and models information in the domain of bio-informatics.
- **Financial**[2]: The Financial ontology in consisted of 188 classes and 4 properties and describes information on the financial domain.
- **Aktors Portal**[3]: The Aktors Portal ontology describes an academic computer science community and is consisted of 247 classes and 327 properties.
- **CRMdig**[4]: The CRMdig is an ontology to encode metadata about the steps and methods of production ("provenance") of digitization products and synthetic digital representations created by various technologies. It is consisted of 126 classes and 435 properties. In addition, for our experiments we used 900 real instances provided by the 3D-SYSTEK project.
- **LUBM**[5]: The Lehigh University Benchmark (LUBM) is a widely used benchmark for evaluating semantic web repositories. It contains 43 classes and 32 properties

[1] http://www.aiai.ed.ac.uk/project/biosphere/downloads.html
[2] http://139.91.210.38:5000/ontologies/BankOntology.owl
[3] http://www.daml.org/ontologies/322
[4] http://www.ics.forth.gr/isl/index_main.php?c=656
[5] http://swat.cse.lehigh.edu/projects/lubm/

modeling information about universities and is accompanied by a synthetic data generator. For our tests, we used the default 1555 instances coming from a real dataset.

- **eTMO**[6]: This ontology has been defined in the context of MyHealthAvatarEU project [63] and is used to model various information within the e-health domain. It is consisted of 335 classes and 67 properties and it is published with 3861 real instances coming from the aforementioned project.

Table 4.1: Ontology characteristics.

Ontology	Classes	Properties	Instances
BIOSPHERE	87	3	-
Financial	188	4	-
Aktors Portal	247	327	-
CRMdig	126	435	900
LUBM	43	32	1555
eTMO	335	67	3861

Table 4.2: Graph Structural characteristics.

Ontology	Density	Diameter	Avg path length
BIOSPHERE	0.011	4	1.94
Financial	0.005	10	3.94
Aktors Portal	0.002	24	7.96
CRMdig	0.033	8	3.65
LUBM	0.017	10	4.21
eTMO	0.007	9	2.65

The characteristics of those ontologies are shown in Table 4.1 and Table 4.2.The variety on the size, the domain and the structure of these ontologies offers an interesting test-case for our evaluation. We have to note that some of these ontologies are actually OWL ontologies (BIOSPHERE, Financial, Aktors Portal, eTMO) however we consider only their RDF/S fragment.

To proceed with the evaluation for the first three ontologies we are using the reference summaries published in [3] used also by Queiroz-Sousa et al. [17] in their evaluation.

[6] http://www.myhealthavatar.eu/

The reference summaries were generated by eight human experts with a good experience in ontology engineering which were familiar with the aforementioned ontologies. The experts were requested to select up to 20 concepts, which were considered as the most representative of each ontology. The level of agreement among experts for the three ontologies had a mean value of 74% meaning that the experts did not entirely agree on their selections.

For the next three ontologies we are using the reference summaries published in [5] generated each by three human experts with an extensive experience in ontology engineering and which were familiar with the aforementioned ontologies. In this case, the experts were requested to select not only the most important vertices but also the most representative schema graph summary containing the 10% of the classes for each ontology. The level of agreement among experts for the vertices of the three ontologies had a mean value of 60% meaning that the experts in this case as well did not completely agree on their selections.

4.4.1.1 Spearman's Rank Correlation Coefficient

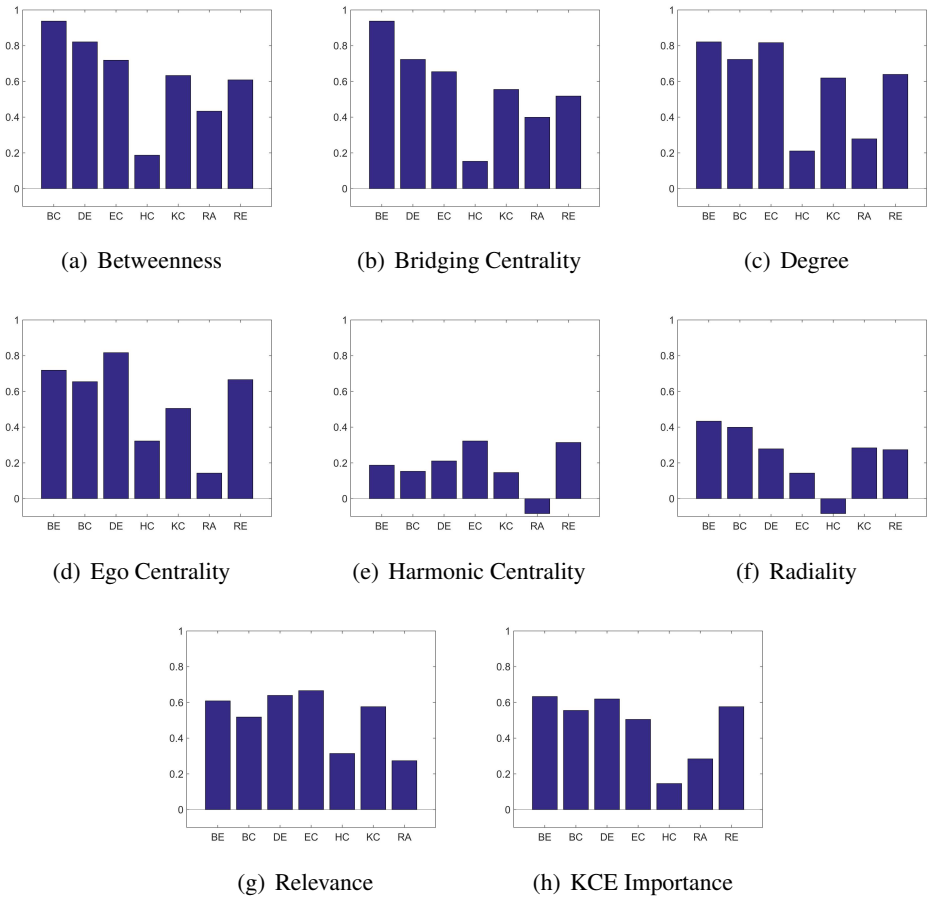
The results of our experiments for Spearman's rank correlation coefficient are shown in Fig. 4.1. As expected the Betweenness, the Bridging Centrality, the Degree and the Ego Centrality are high correlated. This is due to the fact that the Bridging Centrality is a product of the Betweenness and the Bridging coefficient, and the Ego Centrality has a strong tie relationship with the Degree according to it's definition. The Betweenness and the Bridging Centrality are based on identifying the shortest paths, while the Degree and the Ego Centrality are based on the neighborhood of each node. Since the graphs used in our experiments are too sparse, the shortest paths are limited and have a tendency to the vertices with a high degree.

The Harmonic Centrality and the Radiality are independent to all other centrally measures - the correlation is lower than 0.5. The vertices with high Harmonic Centrality are really close to their neighbors i.e. the neighborhoods of those vertices are more compact. Since the Harmonic Centrality can reflect vertices that are very close, it is not specific to the nature of the node couples and should be compared with the Radiality [64]. The Radiality on the other hand, is calculated similarly to the Harmonic Centrality, but with respect to the diameter. As such, vertices with a small number of reachable vertices and a lower Harmonic Centrality have higher Radiality values than vertices with a big number of reachable vertices and a higher Harmonic Centrality.

Finally the KCE Importance and the Relevance try to explore metrics more specific to the RDF data model combining them with other structural measures. As such the seem to have a minor correlation between themselves and with the Betweenness, the Degree and the Ego Centrality.

4.4.1.2 The Similarity Measure

The results of our experiments are shown in Fig. 4.2 . Overall, the Ego Centrality has a better similarity followed by the Betweenes and then the Relevance, however with close

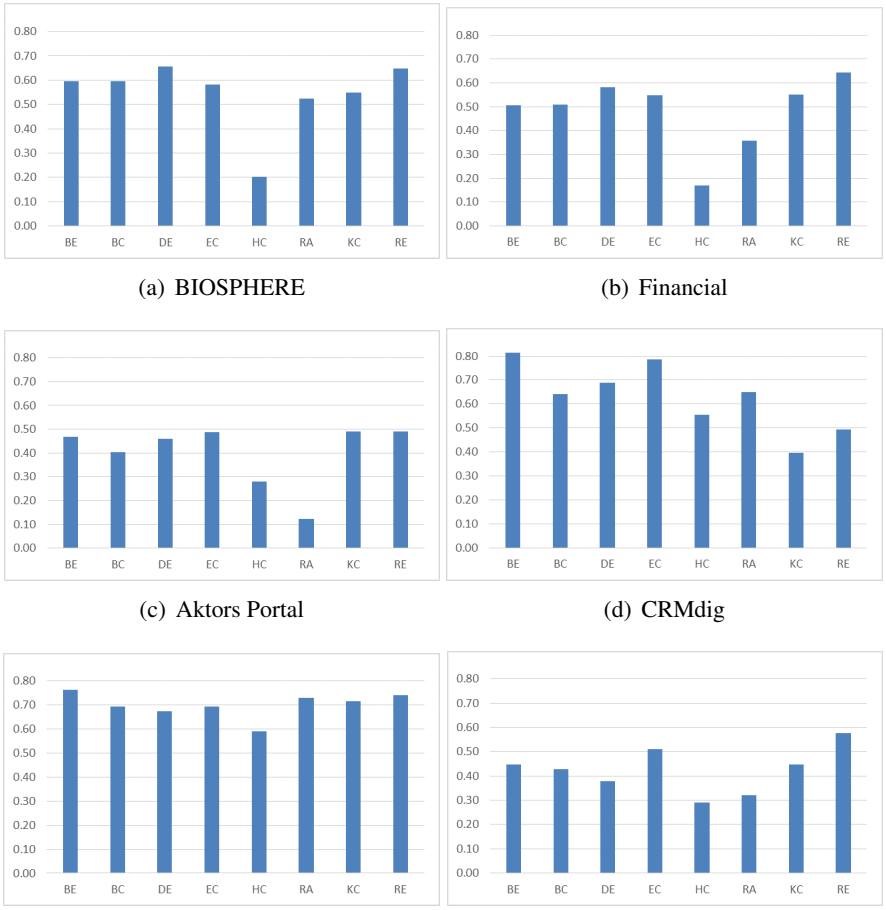


(a) Betweenness (b) Bridging Centrality (c) Degree

(d) Ego Centrality (e) Harmonic Centrality (f) Radiality

(g) Relevance (h) KCE Importance

Figure 4.1: Spearman's rank correlation coefficient for the eight importance measures (Betweenness (BE), Bridging Centrality (BC), Degree (DE), Ego Centrality (EC), Harmonic Centrality (HC), Radiality (RA), KCE importance (KC),Relevance (RE)).



(a) BIOSPHERE (b) Financial

(c) Aktors Portal (d) CRMdig

(e) LUBM (f) eTMO

Figure 4.2: Comparing the similarity of the importance measures using the six ontologies.

Table 4.3: Deviation from the optimum solution

Algorithm	Max dev	Average dev	Best
MST	32.3	3.10	61.1
HEUM	20.8	2.48	68.0
SDISTG	10.0	0.74	87.5
CHINS	8.3	0.26	94.4

mean values 0.601, 0.599, 0.599 accordingly. For hierarchical ontologies without instances (BIOSPHERE and FINANCIAL) the Relevance and the Degree seem to be the best choices whereas for ontologies with many instances the Relevance seems to be a better choice. This is reasonable, since the Relevance is the only measure considering also instance information - along with the KCE. For more dense ontologies such as the CRMdig, the Betweenness gives the best results followed by the Ego Centrality. Finally, the results for all measures, seem to be better for more dense ontologies such as the CRMdig and the LUBM.

4.4.1.3 RDF/S Schema Graph Edit Distance

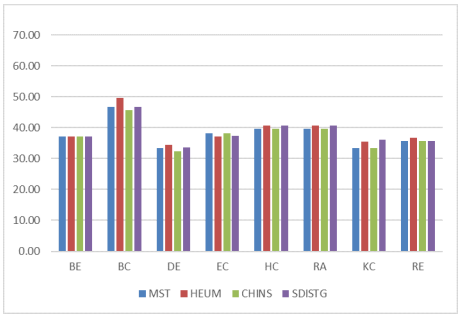
The results are shown in Fig. 4.3. As shown in the first three subfigures, the bigger the ontology the bigger the number of the edit operations that are required to get the reference summary. This is reasonable, since the bigger the ontology, the bigger the summary of the 10% and as such more edit operations will be required to get the reference ones.

When comparing the algorithms for identifying the most important vertices with respect to the whole summary schema graph constructed, the KCE Importance and the Betweeness perform better, producing summaries with on average 30.31 and 31.42 edit distance from the reference ones. As such, the Betweeness is distinguished not only as a good measure for identifying the most important vertices but for creating good schema summaries as well.

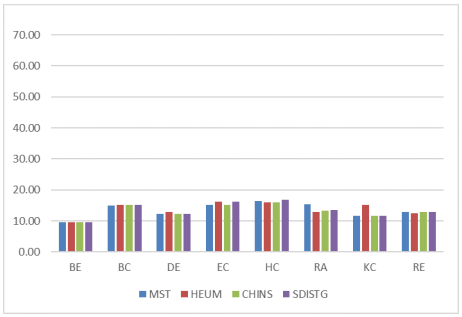
In addition, when comparing the algorithms for linking the most important vertices overall, CHINS is better producing summaries with an average edit distance of 34.32.

4.4.1.4 Additional Vertices Introduced

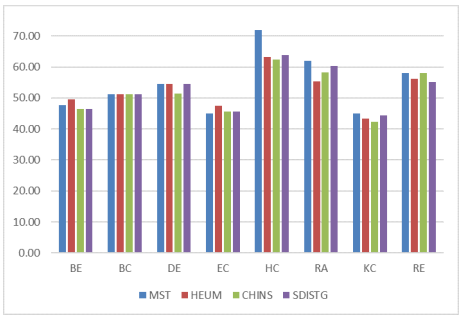
The results are shown in Table 4.3 produced by analyzing the average and max(the worst case) deviation of each algorithm thought the different examined ontologies and terminal sets as produced by the eight importance measures. Again CHINS seems to perform better, whereas CHINS and SDISTG have a low max deviation, indicating that they do not deviate singificanlty from the optimum solution. The average percentage of additional vertices introduced per algorithm is shown in Fig.4.4. We can observe that MST introduces on average 25.6% additional vertices, whereas CHINS introduces 21.8% (3.8% less) additional vertices.



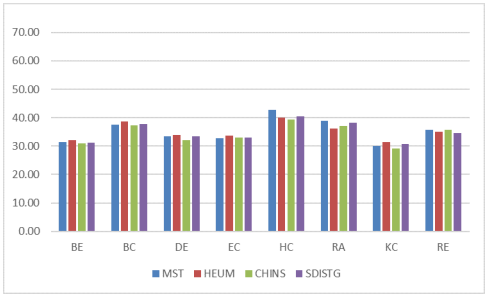
(a) CRMdig



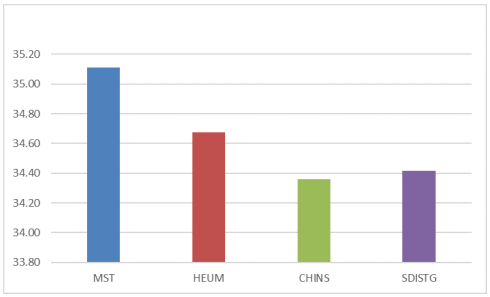
(b) LUBM



(c) eTMO



(d) Overall all Measures



(e) Overall

Figure 4.3: Edit distance results for the three ontologies with available reference summaries.

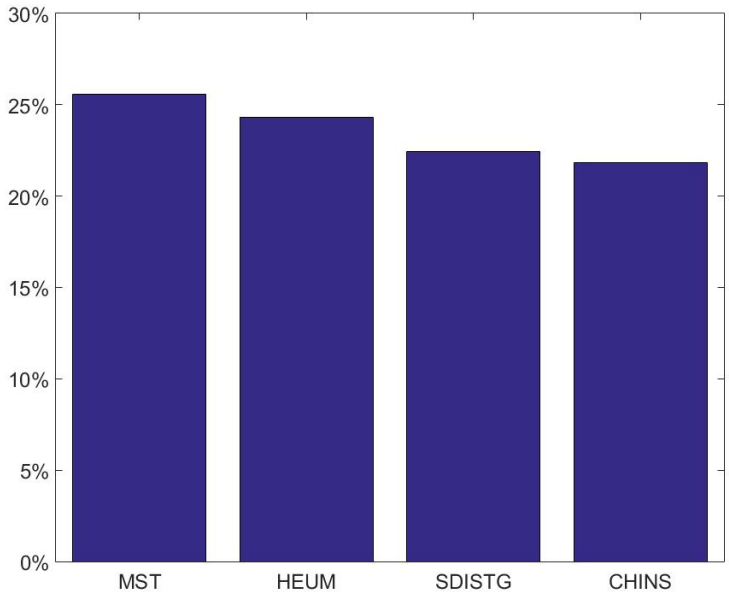


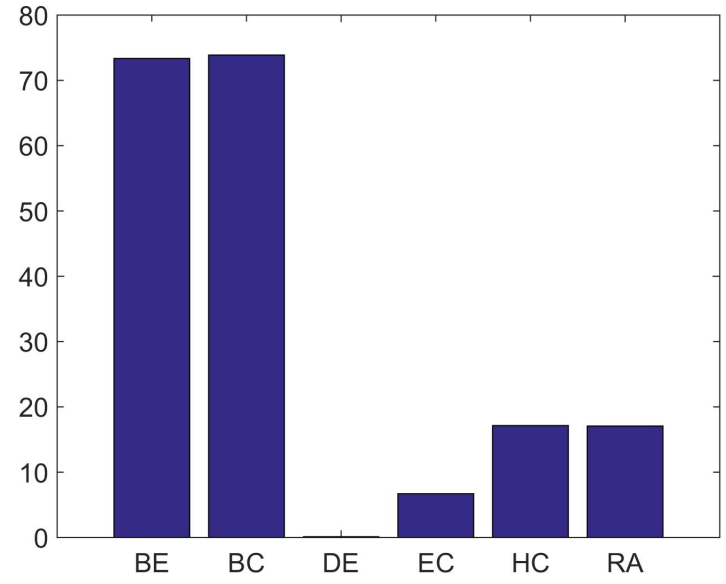
Figure 4.4: The average percentage of extra nodes introduced by the algorithms for linking the most important nodes.

4.4.1.5 Execution Time

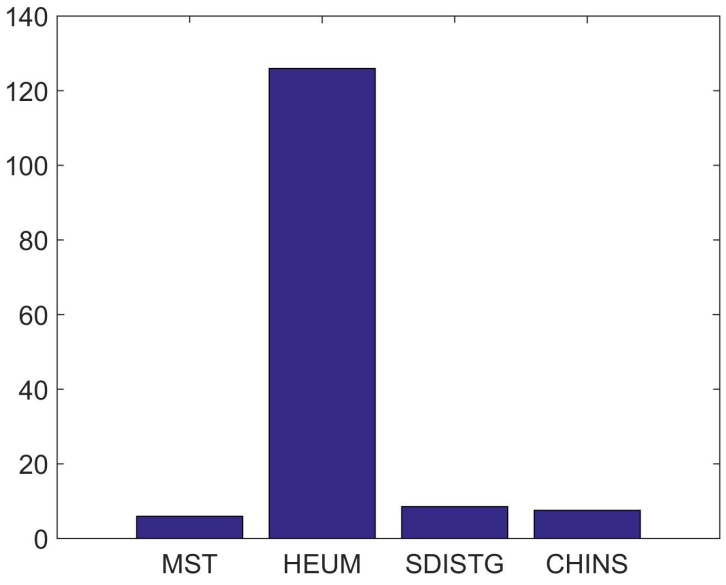
The mean execution times for identifying the most important vertices and constructing the corresponding summaries are shown in Fig. 4.5. As we can observe, the execution times confirms the complexities of the examined importance measures 3.1 and the algorithms 3.3 employed for linking the most important nodes in a graph. In addition conclusions from 4.3 are confirmed, in cases where the complexities are the same but the architecture and operations of processing steps differs. Betweenness centrality and HEUM are quite slower, as they have to compute All Pairs Shortest Paths, with that meaning quadratic complexity with respect to the number of nodes.

4.4.2 Users Most Common Queries

DBpedia: The DBpedia is an open, large-scale, multilingual Knowledge Base Extracted from Wikipedia. This structured information is made available on the Web using Semantic Web and Linked Data standards that allow users to ask sophisticated queries to Wikipedia resources, including links to other related data-sets. It was started at the Free University of Berlin and Leipzig University, in collaboration with OpenLink Software in 2006 and meanwhile attracted ample interest in research and practice. Considering that Wikipedia is the most widely used encyclopedia, it consists of over 400 million facts that describe 3.7 million things and DBpedia can be used to directly answer fact questions about a wide range of topics then we can can imagine why DBpedia is so important.



(a)



(b)

Figure 4.5: The average execution times of the importance measures (msec) (a) and the approximation algorithms (msec) (b)

To evaluate our approach, we used two versions of the DBpedia [7]. DBpedia 3.8 consist of 359 classes, 1323 properties and more that 2.3M instances, whereas DBpedia 3.9 consist of 552 classes, 1805 properties and more than 3.3M instances and offer an interesting use-case for exploration. To identify the most important vertices of those two versions we do not rely on a limited amount of domain experts with subjective opinions as past approaches do. Instead, we exploit the query logs from the corresponding DBpedia end points trying to identify the schema nodes that are more frequently queried. For DBpedia 3.8 we were able to get access to more than 50K queries whereas for 3.9 we were able to get access to more than 110K queries. The main graph characteristics of those ontologies are shown in Table 4.4. In order to indicate the difficulty of understanding an Ontology Schema with more than a hundred of Classes and the diversity of importance measures, we provide visualizations of the ontology graph DBpedia 3.8 on figures 4.6, 4.7, 4.8 and 4.9.

Table 4.4: Graph Structural characteristics.

Ontology	Density	Diameter	Avg path length
DBpedia 3.8	0.00472	9	3.80
DBpedia 3.9	0.00298	13	4.36

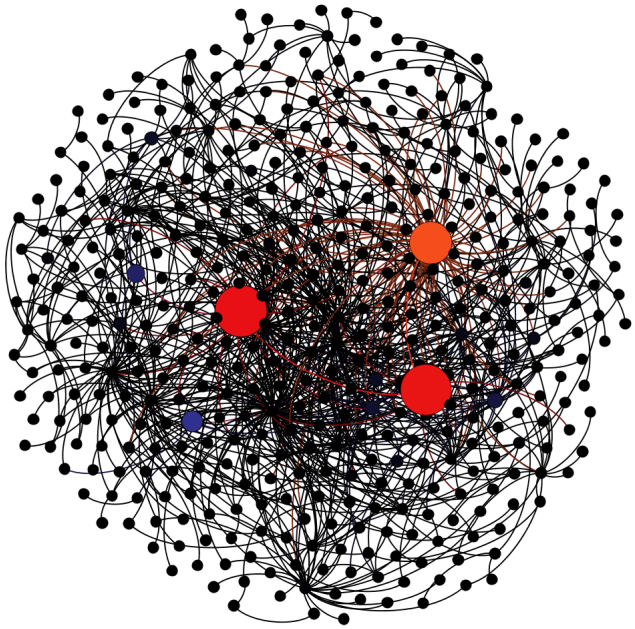
For each examined version, we considered the corresponding query log trying to identify the most important classes. We assess as the most important, the ones that have higher frequency of appearance in the queries. A class appears within a query either directly or indirectly. Directly when the said class appears within a triple pattern of the query and undirectly when a) the said class is the type of an instance or the domain/range of a property that appear in a triple pattern of the query.

To generate the reference ranking initially we filtered the query logs by removing the mistyped, syntactically incorrect queries which may lead to misleading results.As a result, we rely only on the valid SPARQL queries. In addition, classes may appear within a query either directly, or indirectly (as variables mapped to classes). In both cases the appearance of a class in a query increases its appearance frequency by one. Finally, the classes of each version are sorted by their frequencies as they were computed wrt. the corresponding query log. Consider for example the following query from the query log:

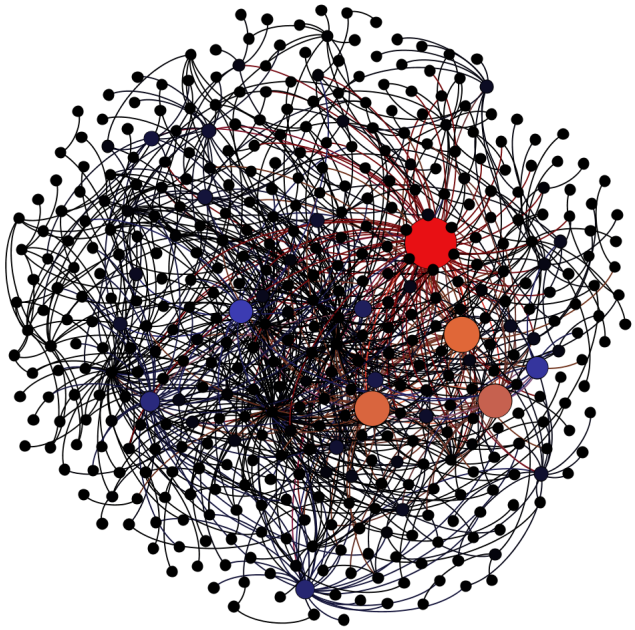
```
1 select ?p where {
2 ?p a <http://dbpedia.org/ontology/Person>.
3 } limit 10
```

Obviously the user is interested in the class Person thus, the corresponding algorithm should increase its appearance frequency by one. Note that, even if this class appears more than one times within the query, its appearance frequency is increased as if it appears one time. This holds because our analysis is based on the queries and not on the triple patterns. Now consider the following query:

[7] http://wiki.dbpedia.org/

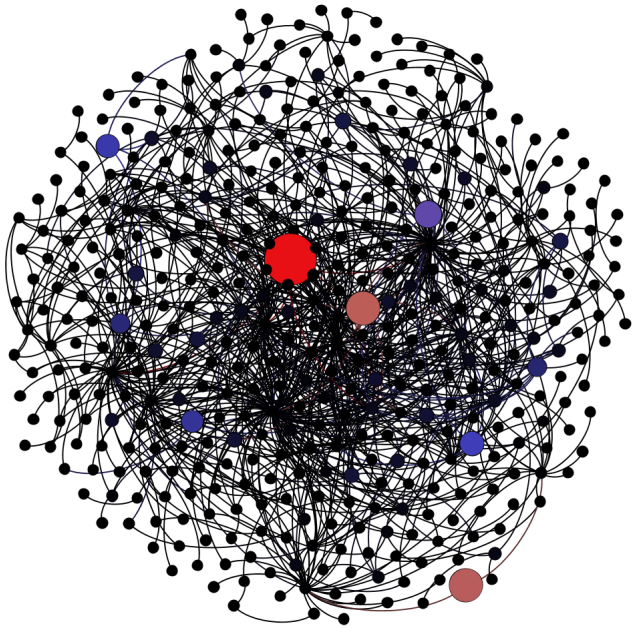


(a)

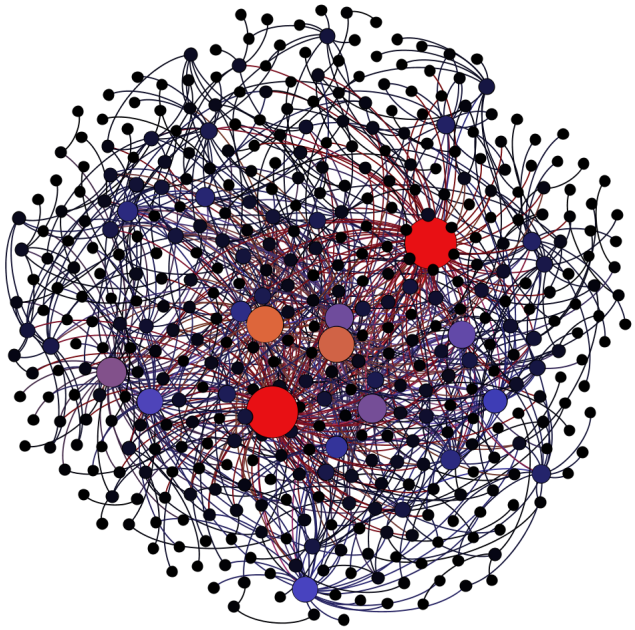


(b)

Figure 4.6: Frequency of queries (a), Betweenness Centrality (b)

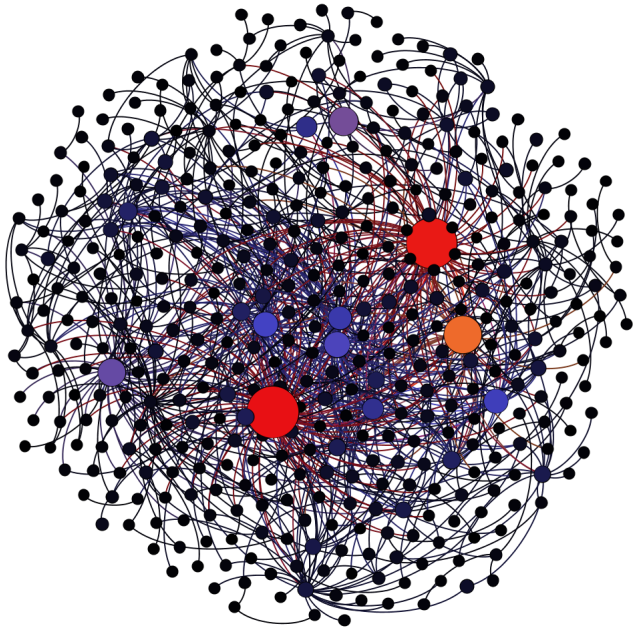


(a)

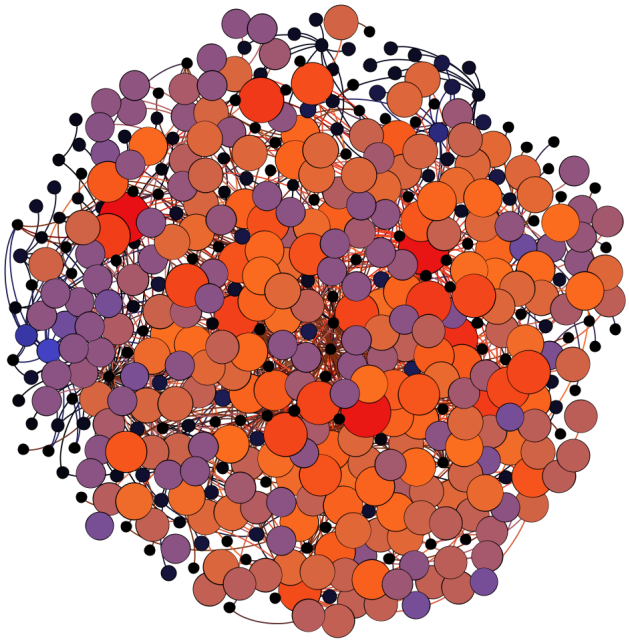


(b)

Figure 4.7: Bridging Centrality (a), Degree Centrality (b)

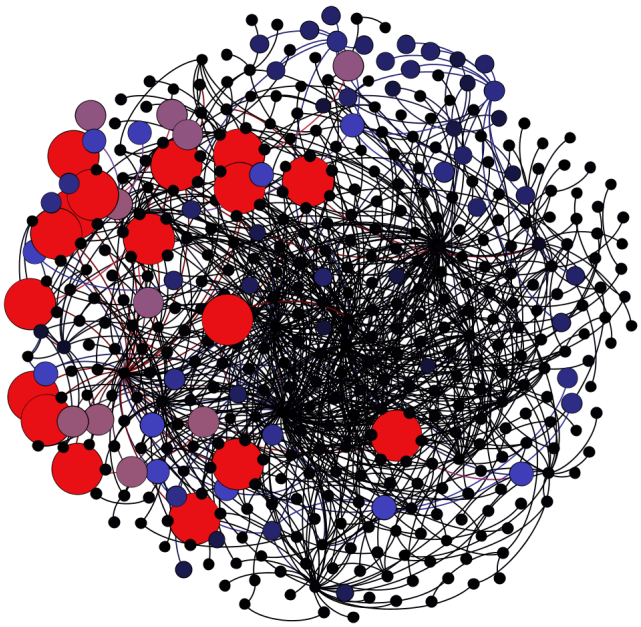


(a)

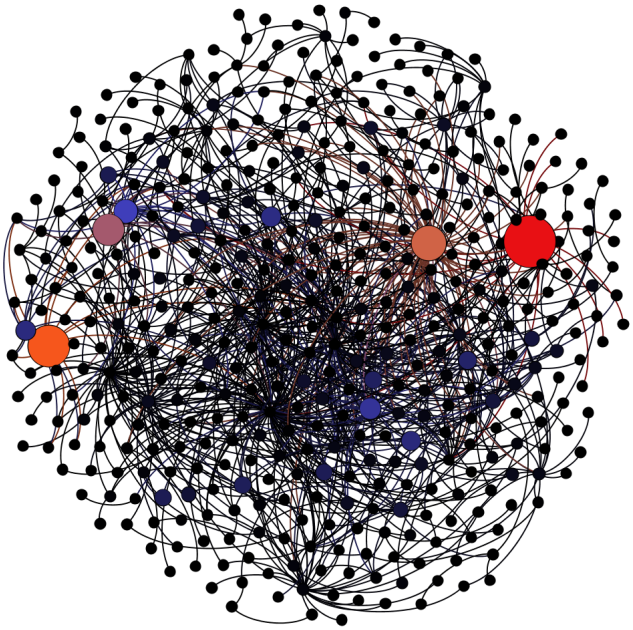


(b)

Figure 4.8: Ego Centrality (a), Harmonic Centrality (b)



(a)



(b)

Figure 4.9: Radiality Centrality (a), Relevance Centrality (b)

```
1 select ?s, ?o where {
2 ?s <http://dbpedia.org/ontology/Person/height> ?o.
3 } limit 10
```

In the above query, the end-users are interested in the heights of some individual persons. As a result, the user is also interested in the classes which are connected by this property in the schema level (ie., its domain and range). As such the appearance frequency of these classes should be increased as well. Finally consider the following query:

```
1 select ?type where {
2 <http://dbpedia.org/resource/Ainslie_Roberts> a ?type.
3 }
```

In the above query, the users are interested in the types of a specific individual named "Ainslie Roberts". As a result, the user might also be interested in all the classes which are instantiated by this individual. As such their appearance frequency should increase by one. To conclude, we parse all queries and we increase the appearance frequency of a class in the following cases:

- If the class is explicitly mentioned in the query.
- If an individual appears in the query that is instantiated under that class.
- If the class is a domain or a range of a specific queried property.

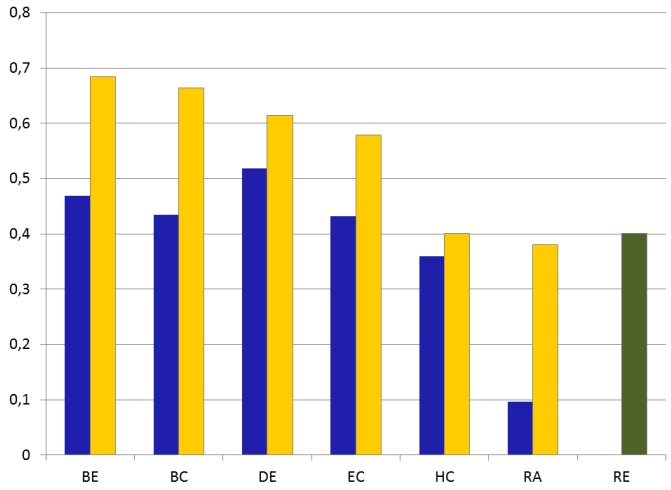
In addition we compare our approach with another measure recently published, relevance (RE), combining both syntactic and semantic information, shown to outperform past approaches in the area [7, 5].

4.4.2.1 Spearman's rank correlation coefficient

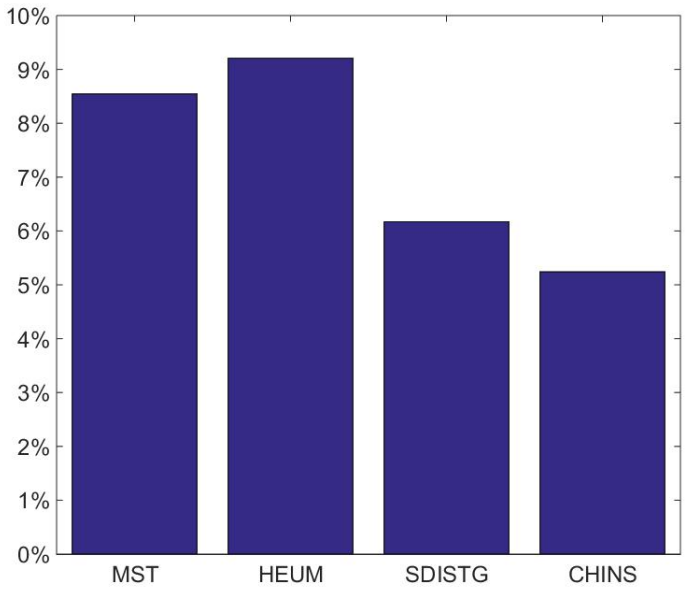
The results of our experiments for Spearman's rank correlation coefficient among the important measures are shown in Fig. 4.10. As shown, our adapted importance measures show a higher similarity than the pure structural measures with the identified frequency ranking. In addition we can see that measures like the Betweenness, and the Bridging Centrality show a really high correlation with the frequency ranking. Finally, we can see that all adapted measures - except Radiality - show a better correlation than Relevance.

4.4.2.2 The Similarity Measure

The results of our experiments are shown in 4.13 and present the average similarity values for generating summaries from 1% to 50% of the corresponding schema graph size. As shown again our adapted measures (in yellow) outperform the pure structural ones (in blue) in all cases. In addition, all measures but AIM_{RA} outperform Relevance showing again the high value of our adaptations. When comparing between the ontology versions we can observe that although AIM_{BE} is the clear winner in all cases, the second best in DBpedia 3.8 is the AIM_{BC} whereas in DBpedia 3.9 is the AIM_{DE}. To interpret these



(a)



(b)

Figure 4.10: (a) Spearman's rank correlation for the adapted (yellow) and the non-adapted (blue) importance measures and (b) the percentage of additional nodes introduced

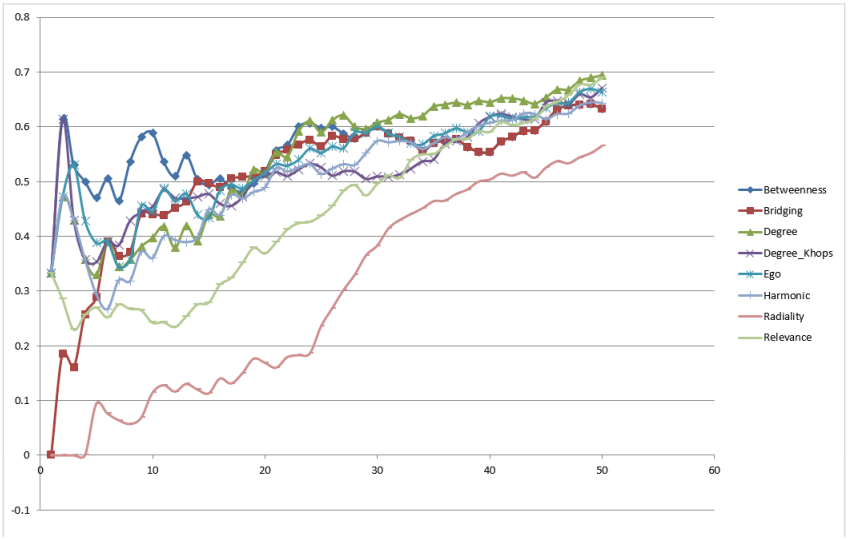
results we shall consider that 193 more classes were added in DBPedia 3.9 introducing only a small number of new edges. This results in a reduction of 37% of the density and an increase of the diameter from 9 to 13. As such, only a few number of nodes have more than one out-going edge and the degree performs better in this case as it captures more effectively the importance of more sparse graphs. A more detailed information about similarity of importance measures according to percentage size of summarization is provided on 4.11 and 4.12.

4.4.2.3 Additional vertices Introduced

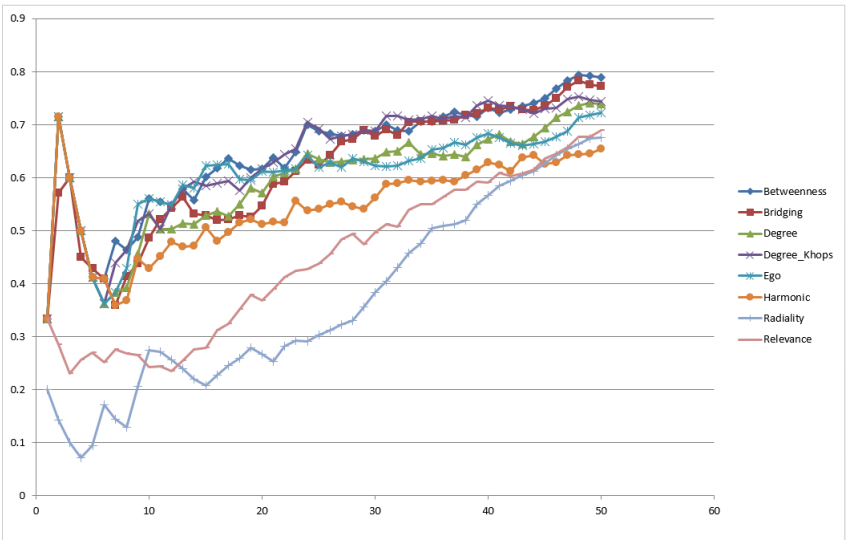
The average number of additional nodes introduced per algorithm is shown in Fig. 4.10(b). We can observe that MST introduces on average 8.5% of additional nodes, whereas CHINS introduces almost 4.7% additional nodes. For example, for DBpedia 3.9 this corresponds to19 additional nodes using MST over CHINS when requesting a summary of 10% of the nodes. This is reasonable, since the Steiner-Tree approximations have the objective of minimizing the additional nodes introduced in the selected sub-graph confirmed by our experiments. An example summarization of DBpedia 3.8 with Betweenness Centrality and CHINS is shown on figure 4.14, in order to understand the complexity of this Ontology.

4.4.2.4 Execution Time

The mean execution times for identifying the most important nodes and constructing the corresponding summaries are shown in Fig. 4.15. Both in this and in previous experiments the assumptions from 4.3 are confirmed. DBpedia 3.8 and 3.9 constitutes bigger graphs with the variance of execution times getting bigger linearly to the number of nodes and edges of the graph.

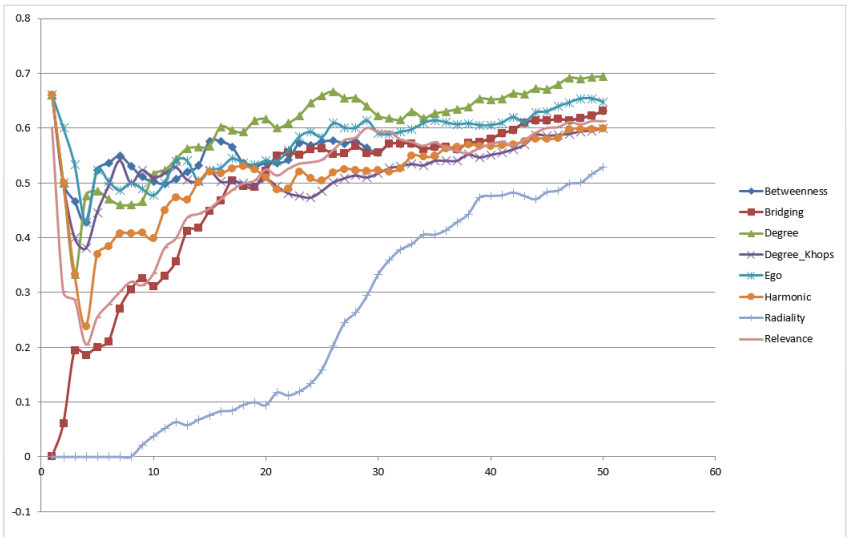


(a)

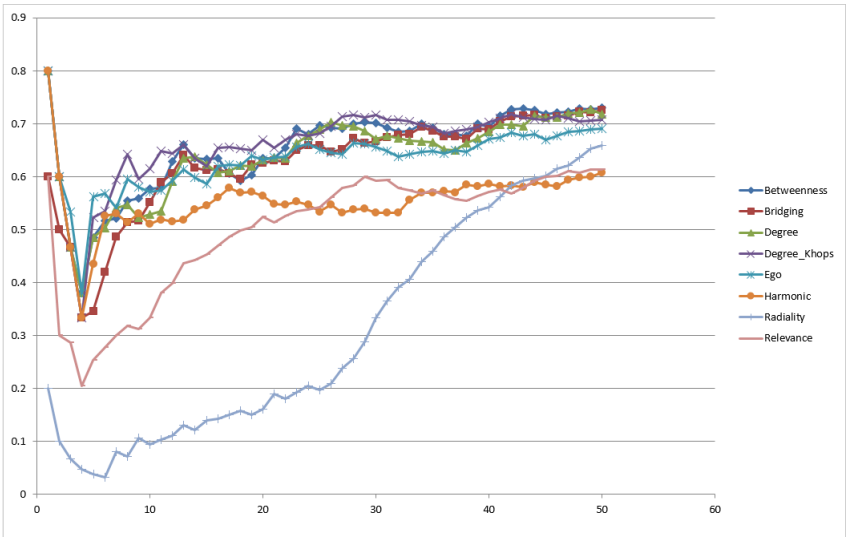


(b)

Figure 4.11: Similarity of importance measures (y-axes) according to size of percentage of summarization (x-axes) for non-adapted (a) and adapted (b) case for DBpedia 3.8

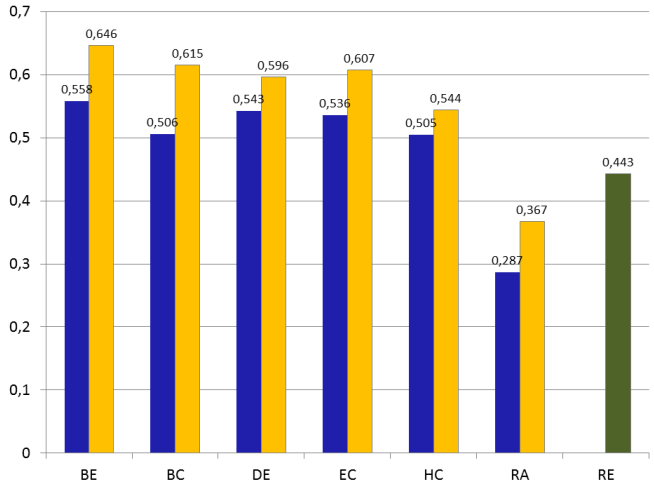


(a)

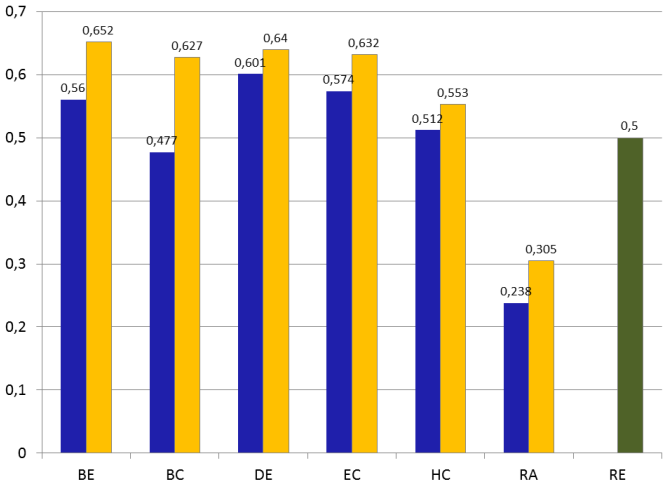


(b)

Figure 4.12: Similarity of importance measures (y-axes) according to size of percentage of summarization (x-axes) for non-adapted (a) and adapted (b) case for DBpedia 3.9

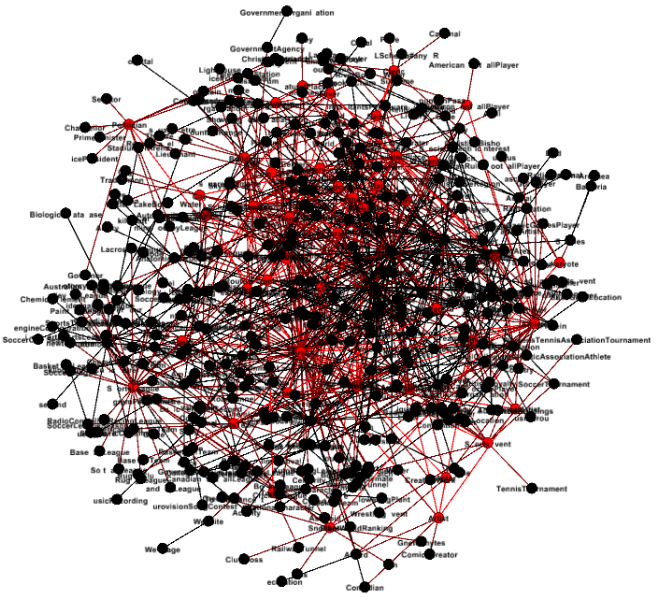


(a) DBpedia 3.8

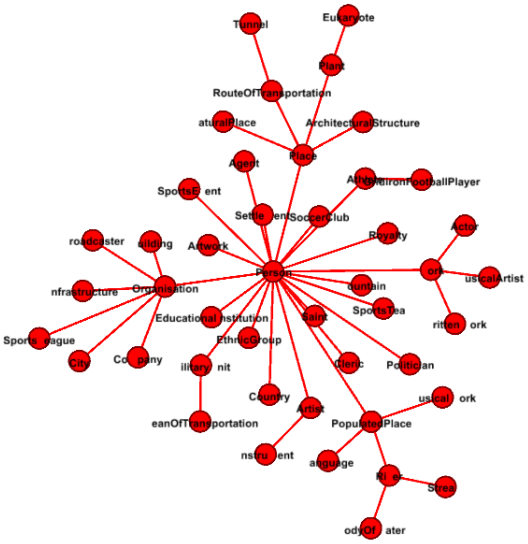


(b) DBPedia 3.9

Figure 4.13: Comparing the average similarity of the adapted (in yellow) and the non-adapted (in blue)importance measures in (a) DBpedia 3.8 and (b) DBpedia 3.9 for a summary of 1-50%.

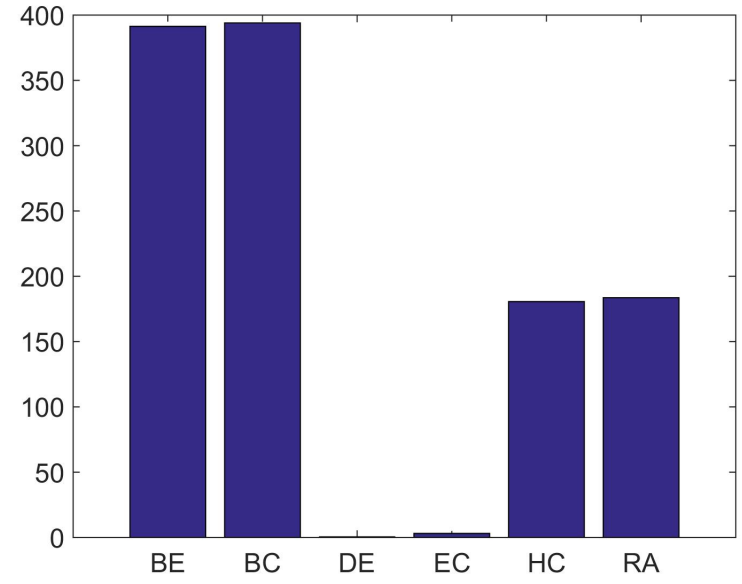


(a)

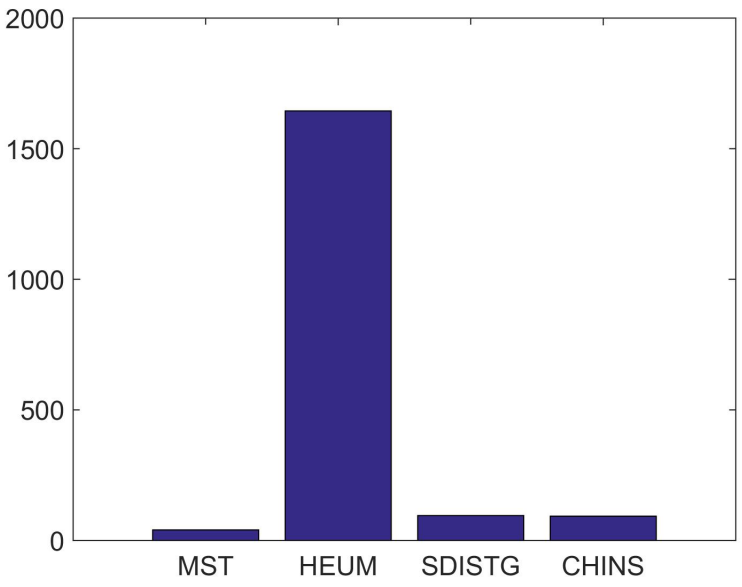


(b)

Figure 4.14: The schema graph of the DBpedia 3.9 ontology (a) and the corresponding Schema Summary (b).



(a)



(b)

Figure 4.15: The average execution times of the importance measures (msec) (a) and the approximation algorithms (msec) (b)

Chapter 5

Conclusions and Future Work

"Graphs Are in More Places than You Think"

5.1 Discussion & Conclusion

In this thesis, we try to provide answers to the two main questions in constructing RDF/S summaries: how to identify the most important nodes and how to link the selected nodes to create the final summary. We explore eight diverse measures to identify importance and we implement three graph Steiner-Tree approximations in order to link those nodes. The performed evaluation shows the feasibility of our solution and demonstrates the advantages gained by producing high quality summaries.

Obviously there is no measure for identifying the most important nodes that outperforms the rest in terms of quality in all cases. Surprisingly, structural measures however like the Betweenness and the Ego Centrality seem to have better results in most cases. The performed evaluation shows that the adapted measures outperform the pure structural ones and past approaches in the area in all cases. Besides selecting the most important nodes, the Betweeness has good results when producing a complete summary schema graph (along with the KCE and Ego). In addition, we show that the Steiner-Tree approximation algorithms produce better summaries and introduce less additional nodes to the result schema graph. CHINS seems to be the best choice in terms of the quality of the generated summary. With respect to the execution time, the MST is the fastest one, however with an impact on the quality of the results, since it introduces more additional nodes with low quality as it is verified by the corresponding RDF/S schema graph edit distances. Overall CHINS seems to achieve an optimal trade-off between quality and execution time.

5.2 Possible Methodological Limitations

All studies have limitations. In this thesis lack of available data is the most important. More data sets and types of databases will expand the reliability and the scope of our analysis. The RDF model of KBs, considers specific cases of graphs by following a subject–predicate–object structure. Moreover a strict measure for accuracy of summarization

does not exists because it relies on human thinking, usability and preference on each data set and process. Another variation that is not considered is when the graph is disconnected but a summarization is expected from separated graphs that constitute a KB.

5.3 Future work

As the size and the complexity of schema's and data increase, ontology summarization is becoming more and more important and we expect more challenges to arise in the near future. There are several interesting open questions related to our work. As future work, an interesting topic would be to extend our evaluation to spectral properties as well or to focus on how to combine the various measures in order to achieve the best results according to the specific characteristics of the input ontologies. An another interesting topic would be to extend our approach to handle more constructs from OWL ontologies such as class restrictions, disjointness and equivalences. Finally, instead of relying on reference summaries for the evaluation of the automatically produced summaries, another idea would be to check if these summaries are able to answer the most common queries formulated by the users i.e. index coverage in the sense of frequent sub-graph problem has to be further examined. Index coverage means that a query Q does not have to hit the original table. A user-specified query Q on a graph database G want to retrieve the set of sub-graphs of G, each of which is isomorphic to Q. This is the well studied sub-graph isomorphism problem, which has proven to be NP-complete [65]. Smaller index size and improvement of query performance over graph databases constitute a growing need, in order to be able to solve this problem on large networks [66]. Scalable graph indexing mechanisms and graph query optimizers have to be explored for a vast improvement of querying response time on graph databases.

Bibliography

[1] H. Stuckenschmidt, C. Parent, and S. Spaccapietra, Eds., *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5445. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01907-4

[2] H. Stuckenschmidt and M. C. A. Klein, "Structure-based partitioning of large concept hierarchies," in *ISWC*, 2004, pp. 289–303. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30475-3_21

[3] P. Silvio, M. Enrico, and d'Aquin Mathieu, "Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures," in *Asian Semantic Web Conference*, 2008, pp. 242–256.

[4] C. E. Pires, P. Sousa, Z. Kedad, and A. C. Salgado, "Summarizing ontology-based schemas in pdms," in *ICDEW*, 2010, pp. 239–244.

[5] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis, "Ontology understanding without tears: The summarization approach," *SWJ*, p. (in press), 2016. [Online]. Available: http://www.semantic-web-journal.net/content/ontology-understanding-without-tears-summarization-approach-0

[6] X. Zhang, G. Cheng, and Y. Qu, "Ontology summarization based on rdf sentence graph," in *WWW*, 2007, pp. 707–716.

[7] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis, "RDF digest: Efficient summarization of RDF/S kbs," in *ESWC*, 2015, pp. 119–134. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-18818-8_8

[8] M. Dudáš, V. Svátek, and J. Mynarz, "Dataset summary visualization with lodsight," in *European Semantic Web Conference*. Springer, 2015, pp. 36–40.

[9] S. Khatchadourian and M. P. Consens, "Explod: summary-based exploration of interlinking and rdf usage in the linked open data cloud," in *ESWC*, 2010, pp. 272–287.

[10] S. Khatchadourian and M. Consens, "Exploring RDF usage and interlinking in the linked open data cloud using explod," in *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*, 2010. [Online]. Available: http://ceur-ws.org/Vol-628/ldow2010_paper05.pdf

[11] M. Palmonari, A. Rula, R. Porrini, A. Maurino, B. Spahiu, and V. Ferme, "Abstat: linked data summaries with abstraction and statistics," in *ESWC*, 2015, pp. 128–132.

[12] X. Jiang, X. Zhang, F. Gao, C. Pu, and P. Wang, "Graph compression strategies for instance-focused semantic mining," in *China Semantic Web Symposium and Web Science Conference*. Springer, 2013, pp. 50–61.

[13] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *ACM SIGMOD*. ACM, 2008, pp. 419–432.

[14] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *ACM SIGMOD*. ACM, 2008, pp. 567–580.

[15] R. Hasan, "Generating and summarizing explanations for linked data," in *ESWC*, 2014, pp. 473–487.

[16] G. Wu, J. Li, L. Feng, and K. Wang, "Identifying potentially important concepts and relations in an ontology," in *International Semantic Web Conference*. Springer, 2008, pp. 33–49.

[17] P. O. Queiroz-Sousa, A. C. Salgado, and C. E. Pires, "A method for building personalized ontology summaries," *Journal of Information and Data Management*, vol. 4, no. 3, p. 236, 2013.

[18] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. New York: Elsevier, 1976.

[19] C. Berge, *Hypergraphs*, ser. North-Holland Mathematical Library. North-Holland, 1989, vol. 45, combinatorics of Finite Sets.

[20] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, p. 215, 1978.

[21] A. Bavelas, "Communication Patterns in Task-Oriented Groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, Nov. 1950. [Online]. Available: http://dx.doi.org/10.1121/1.1906679

[22] Y. Rochat, "Closeness centrality extended to unconnected graphs: The harmonic centrality index," in *Applications of Social Network Analysis (ASNA)*, 2009.

[23] P. Boldi and S. Vigna, "Axioms for centrality," *Internet Mathematics*, vol. 10, no. 3-4, pp. 222–262, 2014. [Online]. Available: http://dx.doi.org/10.1080/15427951.2013.865686

[24] T. W. Valente and R. K. Foreman, "Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network," *Social Networks*, vol. 20, no. 1, pp. 89 – 105, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378873397000075

[25] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977. [Online]. Available: http://www.jstor.org/stable/3033543

[26] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[27] A. Shimbel, "Structural parameters of communication networks," *The bulletin of mathematical biophysics*, vol. 15, no. 4, pp. 501–507, 1953. [Online]. Available: http://dx.doi.org/10.1007/BF02476438

[28] W. Hwang, T. Kim, M. Ramanathan, and A. Zhang, "Bridging centrality: graph mining from element level to group level," in *ACM SIGKDD*, 2008, pp. 336–344. [Online]. Available: http://doi.acm.org/10.1145/1401890.1401934

[29] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953. [Online]. Available: http://dx.doi.org/10.1007/BF02289026

[30] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," in *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998, pp. 161–172. [Online]. Available: citeseer.nj.nec.com/page98pagerank.html

[31] D. F. Gleich, "Pagerank beyond the web," *CoRR*, vol. abs/1407.5107, 2014. [Online]. Available: http://arxiv.org/abs/1407.5107

[32] J. M. Kleinberg, "Hubs, authorities, and communities," *ACM Comput. Surv.*, vol. 31, no. 4es, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/345966.345982

[33] R. Lempel and S. Moran, "Salsa: The stochastic approach for link-structure analysis," *ACM Trans. Inf. Syst.*, vol. 19, no. 2, pp. 131–160, Apr. 2001. [Online]. Available: http://doi.acm.org/10.1145/382979.383041

[34] B. O., "O jistém problému minimálním," *Práce mor. přírodověd. spol. v Brně*, vol. 3, pp. 37–58, 1926.

[35] V. Jarník, "O jistém problému minimálním," *Práce moravské přírodovědecké společnosti*, vol. 6, pp. 57–63, 1930.

[36] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1389–1401, 1957.

[37] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390

[38] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *25th Annual Symposium onFoundations of Computer Science, 1984.*, Oct 1984, pp. 338–346.

[39] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962. [Online]. Available: http://doi.acm.org/10.1145/321105.321107

[40] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 345–, Jun. 1962. [Online]. Available: http://doi.acm.org/10.1145/367766.368168

[41] T. M. Chan, "All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time," *ACM Trans. Algorithms*, vol. 8, no. 4, p. 34, 2012. [Online]. Available: http://doi.acm.org/10.1145/2344422.2344424

[42] M. L. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *J. Comput. Syst. Sci.*, vol. 48, no. 3, pp. 533–551, 1994. [Online]. Available: http://dx.doi.org/10.1016/S0022-0000(05)80064-9

[43] F. K. Hwang and D. S. Richards, *Steiner Tree Problems*. Wiley, 1992.

[44] R. M. Karp, "Reducibility among combinatorial problems," in *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, 2010, pp. 219–241. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68279-0_8

[45] M. Hauptmann and M. Karpinski. From relational to neo4j. [Online]. Available: http://theory.cs.uni-bonn.de/info5/steinerkompendium/netcompendium.html

[46] Neo4j. From relational to neo4j. [Online]. Available: https://neo4j.com/developer/graph-db-vs-rdbms/

[47] "Rdf schema 1.1," https://www.w3.org/TR/rdf-schema/, accessed: 2016-08-30.

[48] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl, "RQL: a declarative query language for RDF," in *WWW*, 2002.

[49] G. Scardoni, G. Tosadori, M. Faizan, F. Spoto, F. Fabbri, and C. Laudanna, "Biological network analysis with centiscape: centralities and experimental dataset integration."

[50] S. L. Hakimi, "Steiner's problem in graphs and its implications," *Networks*, vol. 1, no. 2, pp. 113–133, 1971. [Online]. Available: http://dx.doi.org/10.1002/net.3230010203

[51] A. Y. Levin, "Algorithm for the shortest connection of a group of graph vertices," *Soviet Mathematics Doklady*, vol. 12, pp. 1477–1481, 1971.

[52] S. E. Dreyfus and R. A. Wagner, "The steiner problem in graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1971. [Online]. Available: http://dx.doi.org/10.1002/net.3230010302

[53] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, pp. 55–89, 1992. [Online]. Available: http://dx.doi.org/10.1002/net.3230220105

[54] J. Plesnik, "Worst-case relative performances of heuristics for the steiner problem in graphs," 1991.

[55] V. J. Rayward-Smith and A. Clare, "On finding steiner vertices," *Networks*, vol. 16, no. 3, pp. 283–294, 1986. [Online]. Available: http://dx.doi.org/10.1002/net.3230160305

[56] S. Voß, "Steiner's problem in graphs: Heuristic methods," *Discrete Applied Mathematics*, vol. 40, no. 1, pp. 45–72, 1992. [Online]. Available: http://dx.doi.org/10.1016/0166-218X(92)90021-2

[57] C. Gröpl, S. Hougardy, T. Nierhoff, and H. J. Prömel, *Approximation Algorithms for the Steiner Tree Problem in Graphs*. Boston, MA: Springer US, 2001, pp. 235–279. [Online]. Available: http://dx.doi.org/10.1007/978-1-4613-0255-1_7

[58] in *Advances in Steiner Trees*, D.-Z. Du, J. M. Smith, and J. H. Rubinstein, Eds. Kluwer Academic Publishers, 2000.

[59] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," in *Proceedings of the American Mathematical Society, 7*, 1956.

[60] C. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904. [Online]. Available: http://www.jstor.org/stable/1412159

[61] R. L. Donaway, K. W. Drummey, and L. A. Mather, "A comparison of rankings produced by summarization evaluation measures," in *Proceedings of the 2000 NAACL-ANLP Workshop on Automatic Summarization*, 2000, pp. 69–78. [Online]. Available: http://dl.acm.org/citation.cfm?id=1567564.1567572

[62] A. Maedche and S. Staab, "Measuring similarity between ontologies," in *EKAW*, 2002, pp. 251–263. [Online]. Available: http://dx.doi.org/10.1007/3-540-45810-7_24

[63] H. Kondylakis, E. G. Spanakis, S. Sfakianakis, V. Sakkalis, M. Tsiknakis, K. Marias, X. Zhao, H. Yu, and F. Dong, "Digital patient: Personalized and translational data management through the myhealthavatar EU project," in *EMBC*, 2015, pp. 1397–1400. [Online]. Available: http://dx.doi.org/10.1109/EMBC.2015.7318630

[64] J. M. Winkler and H. S. Fox, "Transcriptome meta-analysis reveals a central role for sex steroids in the degeneration of hippocampal neurons in alzheimer's disease," *BMC Systems Biology*, vol. 7, no. 1, pp. 1–11, 2013. [Online]. Available: http://dx.doi.org/10.1186/1752-0509-7-51

[65] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[66] P. Zhao and J. Han, "On graph query optimization in large networks," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 340–351, Sep. 2010. [Online]. Available: http://dx.doi.org/10.14778/1920841.1920887