LDBC

Cooperative Project

FP7 – 317548

D3.3.1 Use case analysis and choke point analysis

Coordinator: [Alex Averbuch, Norbert Martinez]

With contributions from: [Renzo Angles, Josep Lluis Larriba, Francesc Escalé]

1st Quality Reviewer: Andrey Gubichev (TUM)
2nd Quality Reviewer: Orri Erling (OGL)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M9
Actual delivery date:	M9
Version:	1.1
Total number of pages:	32
Keywords:	graph query language, graph database

Abstract

Due largely to the Web, an exponentially increasing amount of data is generated each year. Moreover, a significant fraction of this data is unstructured, or semi-structured at best. This has meant that traditional data models are becoming increasingly restrictive and unsuitable for many application domains – the relational model in particular has been criticized for its lack of semantics. These trends have driven development of alternative database technologies, including graph databases.

The proliferation of applications dealing with complex networks has resulted in an increasing number of graph database deployments. This, in turn, has created demand for a means by which to compare the characteristics of different graph database technologies, such as: performance, data model, query expressiveness, as well as general functional and non-functional capabilities.

To fairly compare these technologies it is essential to first have a thorough understanding of graph data models, graph operations, graph datasets, graph workloads, and the interactions between all of these.

Executive Summary

Due to the recent and rapid increase of graph database deployments in production environments demand has risen for tools, methods, and processes that assist in comparing the functional and non-functional aspects of these databases. These demands - the creation and maintenance of such tools, methods, and processes - are yet to be addressed, and are the main objective of the LDBC project. To achieve this objective it is necessary to acquire a thorough understanding of graph: data models, operations, datasets, workloads, and how each affects the other. Those are the topics covered in this topics.

The graph data models used in real-world applications are rich, they are directed, attributed multi-graphs, and the numerous node types they comprise are often connected by an equally large number of edge types.

Many graph datasets used to model the natural world share structural characteristics; structural aspects such as degree distribution, connectedness, the number of components, and the emergence of clusters/communities, tend to reappear in many different graphs from many different domains. In the context of graph database benchmarking, it is important that synthetic graph dataset generators take these characteristics into consideration, and attempt to mimic them as closely as possible.

Finally, the type of computations and queries commonly performed on graphs is perhaps even more diverse than the variety of data models or variations in topologies. To effectively characterize the different types of access patterns performed on graphs, regardless of domain or use case, it is important to understand the building blocks that graph operations and queries are built up from. These characterizations are presented in Chapter 2.

Graph querying and graph query languages have been an active field of research for many decades, resulting in the rich foundation of literature that exists today. In spite of this, both academia and industry are still unsettled regarding graph data models, query mechanisms, and query language design. As they are the focus of continual research and development efforts, these areas still evolving rapidly, which perhaps explains why it has thus far been difficult for standardization to occur. Unlike RDF in the semantic world or SQL in the relational world, at the time of writing there is no standard graph database language - nor a proposal for one.

To evaluate systems under appropriate data, operations and queries - to ensure obtained results are representative of system behavior under real situations - benchmarking use cases must be representative of real world situations in which the technologies are applied.

Two use cases are presented in depth here, Social Network Analysis and Information Technologies Analysis, both of which have significant relevance to the industrial community today. In addition, eight further use cases are briefly introduced, illustrating the broad market acceptance of graph database management systems and providing greater insight into their uses.

Document Information

IST Project Number	FP7 – 317548	**Acronym**	LDBC
Full Title	LDBC		
Project URL	http://www.ldbc.eu/		
Document URL	`https://svn.sti2.at/ldbc/trunk/wp3/deliverables/D3.3.1_Use_case_analysis_and_choke_point_analysis/`		
EU Project Officer	Carola Carstens		

Deliverable	**Number**	D3.3.1	**Title**	Use case analysis and choke point analysis
Work Package	**Number**	WP3	**Title**	Graph Choke Point Analysis

Date of Delivery	**Contractual**	M9	**Actual**	M9
Status	version 1.1		final ☐	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	Alex Averbuch (NEO), Norbert Martínez (UPC)			
Responsible Author	**Name**	Alex Averbuch	**E-mail**	alex.averbuch@neotechnology.com
	Partner	NEO	**Phone**	+46765832760

Abstract (for dissemination)	Due largely to the Web, an exponentially increasing amount of data is generated each year. Moreover, a significant fraction of this data is unstructured, or semi-structured at best. This has meant that traditional data models are becoming increasingly restrictive and unsuitable for many application domains – the relational model in particular has been criticized for its lack of semantics. These trends have driven development of alternative database technologies, including graph databases. The proliferation of applications dealing with complex networks has resulted in an increasing number of graph database deployments. This, in turn, has created demand for a means by which to compare the characteristics of different graph database technologies, such as: performance, data model, query expressiveness, as well as general functional and non-functional capabilities. To fairly compare these technologies it is essential to first have a thorough understanding of graph data models, graph operations, graph datasets, graph workloads, and the interactions between all of these.
Keywords	graph query language, graph database

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
06/03/2013	0.1	Alex Averbuch, Norbert Martinez	First draft
10/06/2013	1.0	Alex Averbuch, Norbert Martinez	Apply reviewer recommendations
01/07/2013	1.1	Alex Averbuch, Norbert Martinez	add Limitations of Graph Databases

Table of Contents

List of Tables

1 Introduction

Due largely to the Web, an exponentially increasing amount of data is generated each year. Moreover, a significant fraction of this data is unstructured, or semi-structured at best. This has meant that traditional data models are becoming increasingly restrictive and unsuitable for many application domains – the relational model in particular has been criticized for its lack of semantics. These trends have driven development of alternative database technologies, including graph databases [14]. Graph database models are applied in areas where information about data inter-connectivity or topology is as important as the data itself. The development of graph databases was driven by applications where data complexity exceeded the capabilities of the relational data model: flatness of permitted data structures, difficulty of discovering data connectivity, and challenges of modeling complex domains. Examples of such application domains are those working with complex networks: social networks, information networks, technological networks, and biological networks.

- Social networks – the representation of people/groups and their relationships (friendship, business connection, sexual contact, research collaborator, etc.).
- Information networks – information flow or connectivity in domains such as the World Wide Web and peer-to-peer networks.
- Technological networks – networks in which the spatial aspects of data are dominant: the Internet, power grids, transportation networks, etc.
- Biological networks – biological information such as networks that occur in gene regulation, metabolic pathways, chemical structure, and homology relationships among species.

The proliferation of applications dealing with complex networks has resulted in an increasing number of graph database deployments. This, in turn, has created demand for a means by which to compare the characteristics of different graph database technologies, such as: performance, data model, query expressiveness, as well as general functional and non-functional capabilities.

To fairly compare these technologies it is essential to first have a thorough understanding of graph data models, graph operations, graph datasets, graph workloads, and the interactions between all of these. The purpose of this report is to investigate each of these dimensions.

2 Preliminaries

2.1 Graphs

A graph G is an ordered pair $G = (V, E)$ consisting of a set V of nodes (vertices) together with a set E of relationships (edges), where $E \subseteq V \times V$. In addition to the plain definition of graph, there are some characteristics of them which are relevant to mention and that will be useful for defining graph data models upon which we develop this work. We summarize them in the following.

- **Attributes**: Different types of information may be associated to nodes and edges in order to enrich the graph-based representation. Such information is typically a string or numerical values, which indicate some properties of nodes or their edges. However any other type of information such as enumerated values or vectors might be also used. For instance, for the particular case of edges, some graphs include numerical attributes that quantify the relation, which is usually interpreted as the length, weight, cost or intensity of the relation. Moreover, many applications assign a unique identifier for each node and edge of the graph (this could be interpreted as an attribute called "ID"), useful for enumeration purposes. The information attached to nodes and edges can be very influential in the result of an algorithm or analysis, thus taking into account this information is very critical.

- **Directed**: The relation between two nodes can be symmetric or not, depending on the problem at hand. If the relation is symmetric, it has no particular direction, it is then called *undirected*. On the contrary, if the relation is not symmetric, edges differentiate between the head and the tail. The tail of the edge is the node from which the edge starts, and the head of the edge is the node which the edge points to. In this case the edges are said *directed*. Since an undirected edge can be always represented as two directed edges, each one in a reverse direction of the other, undirected graphs are a particular case of directed graphs. This property will be very determinant on how some measures over graphs, such as connectivity or path lengths are computed.

- **Labels**: In certain applications, different labels (or types) of nodes and edges may be considered. Such labeling or typing impacts the result of operations. For example, in a social network scenario, friendship relationships may be either “positive” or “negative” [41], drastically changing the outcome of certain algorithms.

- **Multigraphs**: Multigraphs are graphs in which two nodes can be connected by more than one edge. This situation commonly appears when two nodes are connected through different types of connections. For instance, in a mobile telephone network, where phone numbers are represented by nodes and telephone calls by edges, each call between two phones (nodes) might be represented by a particular edge, thus leading to have nodes connected with more than one edge when more than one call exists between two telephone numbers.

- **Hypergraphs**: Hypergraphs are graphs in which the edges are substituted by hyperedges. In contrast to regular edges, a hyperedge connects an arbitrary number of nodes. Hypergraphs are used, for example, for building artificial intelligence models [31]. Although Hypergraphs appear commonly along different types of networks, in practice they are usually represented as bipartite networks [51], since it facilitates its representation and posterior treatment by the algorithms.

- **Hypernodes**: A Hypernode graph is a graph whose nodes can themselves be graphs or hypernodes, thus allowing nesting of graphs [47]. Hypernodes can be used to represent both simple and complex objects such as hierarchical, composite and cyclic objects, as well as mappings and records. A key feature is that they have the inherent ability to encapsulate information.

Unless otherwise stated, in this work we will suppose directed attributed multigraphs. We will denote the number of nodes in a graph by n and the number of edges by m.

2.1.1 Graph Characterization

Real graphs are typically very different from graphs following the Erdös-Renyi model (random graphs) [25]. Leskovec et al. [43], analyzed over 100 real-world networks belonging to the following fields: social networks, information/citation networks, collaboration networks, web graphs, Internet net- works, bipartite networks, biological networks, low dimensional networks, actor networks, and product-purchaser networks. The size of those networks varied from a few hundred nodes to millions of nodes, and from hundreds of edges to more than one hundred million. We note that although they might seem large, the graph data sets of some current real applications are significantly larger: for example Flickr accounts more than 4 billion photographs that can be tagged and rated [2], and Facebook is publishing more than 25 billion pieces of content each month. For these large graphs, one of the most interesting aspects is that in general most graph metrics (such as the node degree or the edge weight) follow power law distributions [26, 44, 49], and hence some areas of the graph are significantly denser than others.

With respect to the characterization of graphs, we summarize some properties that often appear in these real graphs [19], and that will be useful to characterize the graphs used in the use cases.

- **Large component**: This property states that for undirected graphs, there is typically a large component that fills most of the network (usually more than 50% and not infrequently 90%), while the rest of the network is divided into a large number of small components disconnected from the rest. There can be networks where there is only one component filling all the network (for instance internet, or WWW if acquired from one single crawler). For directed networks, there is usually one large weakly connected component and other small ones in a similar way as in the undirected case. For strongly connected components there is typically one strongly connected component and a selection of small ones (for instance, in WWW network, the largest strongly connected component fills about 25% of the network). Associated with each strongly connected component there is an out-component and an in-component. Acyclic networks do not have strongly connected components. Citation networks for instance, which are considered almost acyclic, have few small strongly connected components of 2-3 nodes but no larger ones.

- **Small-world property**: A small-world network is a type of network in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps. In other words, the average diameter of each connected components from the graph is small. That is, from a given node there is a short path to reach the majority of the remaining nodes in the connected component. One interesting property of the small-world networks is that the distance L between two randomly chose nodes in the same component grows proportionally to the logarithm of the number of nodes n in the network, i.e. $L \propto log(n)$, which means that even for huge networks the diameter remains very low compared to the number of nodes in the network.

- **Scale-free networks**: Roughly speaking, the degree k of a node is the number of edges attached to it. Then, the proportion of nodes of the graph that have degree k is $p_k = \frac{\#\ nodes\ with\ degree\ k}{n}$. pk can also be seen as the probability that a random chosen node has degree k. For $k = 0 \ldots N$ (with N arbitrary), all p_k form the degree distribution. Although we cannot tell the complete structure of the network, the degree distribution can reveal interesting properties. To give an example, in the Internet network[51], most of the nodes have small degrees but there is a tail containing some nodes with high degree (the highest degree is 2407, which means that such node is connected to about 12% of the nodes in the network). Such well connected nodes are called hubs. Degree distributions adhere to the following law: $p_k = Ck^{-\alpha}$ are said to be power laws (if we plot the degree distribution on a log-log plot then $ln(p_k) = -\alpha \cdot ln(k) + c$. Values of α are typically $2 \leq \alpha \leq 3$ (see [51] for a complete list of networks with the corresponding values of α). Networks with power-law degree distributions are called scale-free networks.

- **Sparse**: Given a simple graph (i.e. a graph with neither multiple edges nor self-edges) win n nodes, the maximum number of edges is $m_{max} = \binom{n}{2} = \frac{1}{2}n(n-1)$. The *connectance* or *density* ρ of a graph is the fraction of these edges that are actually present in the graph. That is, $\rho = \frac{m}{m_{max}}$. A sparse graph is a graph in which $\rho \to 0$ as $n \to \infty$. In other words, the number of edges in the graph is far away from the maximum number of edges it can have.

- **Communities**: A graph is said to have communities if there are sets of nodes where the density of edges among the nodes in the set is larger than the edges going outside the set.

2.1.2 Graph Operations

Another important aspect to be considered for characterizing a use case is the type of operations one can perform on graphs. These types are generic enough to be fitted into any of the typical graph applications and in particular in the use cases presented later in this work. In other words, any application where graphs are used requires of any of these operation types to obtain results from the graph.

A first set of operations, are generic operations. These are common operations that may be necessary in a wide range of applications and contexts, thus not focused to a single specific domain. Such generic operations include (i) get atomic information from the graph such as getting a node, getting the value of an attribute of an edge, or getting the neighbor nodes of a specific node, and (ii) create, delete and transform any graph.

Then, any more complex query or transformation of the graph will necessarily use these operations. In the following, we introduce different types of higher level graph operations commonly built upon that generic ones.

- **Traversals**: By traversals we understand operations that, given a set of starting nodes, explore recursively the neighborhood of those nodes until a terminating condition, such as the depth or visiting of a target node, is fulfilled. Consider for instance, the operation of computing the *shortest path* between two nodes, which is the shortest sequence of edges (or the smallest addition of edge weights in the case of weighted graphs) that connects two nodes. In a directed graph the direction is restricted to outgoing edges from the tail to the head. Note that shortest paths may be constrained by the value of some node or edge labels/attributes, as in the case of finding the shortest route from two points, avoiding a certain type of road, for instance. Another typical traversal operation is the computation of *k-hops*. That is, given a source node, such an operation returns all the nodes that are at a distance of k edges from such source node. A particular case that is worth to mention because it is widely used in other operations is the 1-hops (i.e $k = 1$). In this case, the operation returns all the neighbors of the source node, also known as the neighbors of the node. Examples of operations using 1-hops include calculating the nearest neighborhood in recommender systems, obtaining a particular user's neighborhood with similar interest, or in web ranking using hubs and authorities.

- **Graph Metrics**: The objective is basically the study of the topology of the graph in order to analyze their complexity and to characterize graph objects. It is used for instance to verify some specific data distributions, to evaluate a potential match of a specific pattern, or to get detailed information of the role of nodes and edges. In several situations graph measurement is the first step of the analytical process and it is widely used in social network analysis and protein interaction analysis. Typical graph metrics include: the *hop-plot*, which, given a source node, measures the rate of increase of the neighborhood depending on the distance to such source node; the *diameter*, that is, the greatest distance between any pair of vertices in the graph; the *effective diameter*, which is defined as the minimum number of hops in which 90% of all connected pairs of nodes can reach each other; the *density*, i.e. the portion of all possible edges currently present in the graph; or the *clustering coefficient*, which measures the degree of transitivity of the graph.

- **Component Finding**: A *connected component* is a subgraph of the original graph in which there exists a path between any pair of its nodes. With this definition at hand, it is straightforward to see that a node only belongs to a single connected component of the graph. Finding the connected components of a graph is of capital importance in many operations, and it is usually used in pre-processing steps in order to help posterior operations. Related to connected components, there are some operations helpful to study the vulnerability of a graph, or the probability to separate a connected component into two other components.

For instance, finding *bridges*, that are edges whose removal would imply separating a connected component, is important in many applications. Another example is the *cohesion* of the graph which can be computed by finding the minimum number of nodes that disconnect the component if removed.

- **Community Detection**: A community (or cluster) is generally considered to be a set of nodes densely connected among them and poorly connected to nodes outside the community. This effect has been found in many real-world graphs, especially social networks, where people tend to form compact groups having similar profiles in terms of hobbies, jobs, etc. Algorithms for finding communities include the *minimum-cut* method, dendograms (communities formed through hierarchical clustering), methods based on *clique* detection or other clustering techniques, such as the k-means clustering algorithm.
- **Centrality Calculation**: Within the scope of graph theory and network analysis, centrality measures aim at determine the relative importance of a vertex within the graph, based on how well this node connects the network. For instance, in a social network, centrality of a node would mean how influential a person is within the social network, or how well-used a road is within an urban network. The most well-known centrality measures are the *degree* (number of links incident upon a node), *closeness* (which measures the mean distance from a vertex to other vertices) and *betweenness* (that quantifies the number of times a node acts as a bridge along the shortest path between two other nodes) centrality.
- **Pattern matching**: Graph matching is the specific process of evaluating the structural similarity of two graphs, and is usually categorized into *exact* and *approximate* graph matching. Exact matchings may include finding *homomorphisms* or *(subgraph) isomorphisms*. Approximate matchings may *include error-correcting (subgraph) isomorphisms*, *distance-based matching*, etc. Thus pattern matching operations aim to answer whether a given pattern (graph), matches (in one of the different matching variants) a part of another graph.
- **Graph Anonymization**: The anonymization process generates a new graph with properties similar to the original one, avoiding potential intruders to reidentify nodes or edges. This problem gets more complex when the nodes and edges contain attributes. The anonymization of graphs becomes important when several actors exchange datasets that include personal information. To give a couple of examples, two anonymization procedures are the the k-degree anonymity of vertices, or the k-neighborhood anonymity, which guarantees that each node must have k others with the same (one step) neighborhood characteristics.
- **Other operations**: There are other operations related to the applications presented later in this work. For instance, finding similarity between nodes in a graph has shown to be very important in social network analysis. An example of this is structural equivalence, which refers to the extent to which nodes have a common set of linkages to other nodes in the system. Also, specially for recommendation systems, ranking the nodes of a graph is an important issue (for instance PageRank).

We observe that over a small set of generic operations that are shared by all scenarios, applications compute a rich set of more specific graph operations.

2.2 Graph databases

A graph database is any storage system that uses graph structures with nodes, edges, and properties to represent and store data. Some graph database industrial projects are, for example, Neo4J[1], a Java-based open-source graph database engine; DEX[2], a multi-platform graph database management system for efficient graph management in memory constrained environments; HyperGraphDB[3], an embeddable graph database with generalized

[1]http://neo4j.org
[2]http://www.sparsity-technologies.com/
[3]http://www.hypergraphdb.org/index

hypergraphs; OrientDB[4], an open source document-graph database; or InfiniteGraph[5], a distributed and cloud-enabled graph database. In these systems, data manipulation is performed by means of graph operations and types. Operations (queries) are characterized by different aspects ranging from the extension of the graph being accessed to the answer they give.

2.2.1 Query Categorization

The computational requirements of graph queries are characterized by their heterogeneity. For instance, some queries may access the full graph, while others may only request the degree of a single node. In this section, we build up a set of categories to classify the different operations that can be issued to a graph database. See also the summary of this section in Table 2.2.1.

- **Transformation (mutating)/Analysis (non-mutating)**: We distinguish between two types of operations to access the database: transformations and analysis operations. The first group comprise operations that alter the graph database: bulk loads of a graph, adding/removing nodes or edges to the graphs, create new types of nodes/edges/attributes or modify the value of an attribute. The rest of queries are considered analysis queries. Although an analysis query does not modify the graph, it may need access to secondary storage because the graph or the temporary results generated during the query resolution are too large to fit in memory.
- **Cascaded/non-cascaded access**: We differentiate two access patterns to the graph: cascaded and not cascaded. We say that an operation follows a cascaded pattern if the query performs neighbor operations with a depth at least two. For example, a 2-hop operation follows a cascaded pattern. Thus, a non cascaded operation may access a node, an edge or the neighbors of a node. Besides, an operation that does not request the neighbors of a node, though it may access the full graph, is a non cascaded operation. For instance, an operation that returns the node with the largest value of an attribute accesses all nodes, but since it does not follow the graph structure is a non-cascaded operation.
- **Global/neighborhood scale**: Depending on the number of nodes accessed, we distinguish two types of queries: global and neighborhood queries. The former type corresponds to queries that access the complete graph structure. In other words, we consider as global queries those that access to all the nodes and/or the edges of the graph. The latter queries only access to a (small) portion of the graph. Examples of global operations may include finding the node with the highest degree, or the number of edges in the graph. Neighborhood operations may include a k-hop operation from one node, for instance.
- **Attributes accessed**: Graph databases do not only have to manage the structural information of the graph, but also the data associated to the entities of the graph. Here, we classify the queries according to the attribute set that it accesses: edge attribute set, node attribute set, mixed attribute set or no attributes accessed.
- **Result**: We differentiate three different types of results: graphs, aggregated results, and sets. The most typical output for a graph database query is another graph, which is ordinarily a transformation, a selection or a projection of the original graph, which includes nodes and edges. An example of this type of result is getting the minimum spanning tree of a graph, or finding the minimum length path that connects two nodes. The second type of results build up aggregates, whose most common application is to summarize properties of the graph. For instance, a histogram of the degree distribution of the nodes, or a histogram of the community size are computed as aggregations. Finally, a set is an output that contains either atomic entities or result sets that are not structured as graphs. For example, the selection of one node of a graph or finding the edges with the greatest weight are set results.

[4]http://www.orientdb.org/index.htm
[5]http://www.infinitegraph.com/

Table 2.1: Graph operations and query categorization

Graph Operations		Query Categorization				
Group	Operation	Analytical	Cascaded	Scale[a]	Attr.[b]	Result[c]
Generic Operations						
Local Information Extraction	Get Node/Edge	✓	✗	N	✗	S
	Get Node/Edge Attribute	✓	✗	N	✗	S
	Get Neighborhood	✓	✗	N	✗	S
	Node degree	✓	✗	N	✗	A
Transformations	Add/delete node/edge	✗	✗	N	✗	S
	Add/delete/update attribute	✗	✗	N	✓	S
Application dependent Operations						
Traversals	(Constrained) Shortest Path	✓	✓	G	E	G
	k-hops	✓	✓	G/N	✗	G
Graph Metrics	Hop-plot	✓	✗	G	✗	A
	Diameter	✓	✓	G	E	S
	Eccentricity	✓	✓	G	E	A
	Density	✓	✗	G	✗	A
	Clustering coefficient	✓	✓	G	✗	A
Components	Connected components	✓	✓	G	✗	G
	Bridges	✓	✓	G	✗	S
	Cohesion	✓	✓	G	✗	S
Communities	Dendogram	✓	✓	G	✗	G
	Max-flow min-cut	✓	✓	G	E	G
	Clustering	✓	✓	G	✗	G
Centrality Measures	Degree centrality	✓	✗	G	✗	S
	Closeness centrelity	✓	✓	G	✗	S
	Betweeness centrality	✓	✓	G	✗	S
Pattern Matching	Graph/subgraph matching	✓	✓	N	✗	G
Graph Anonymization	k-degree anonymization	✓	✗	G	✗	G
	k-neighborhood anonymization	✓	✓	G	✗	G
Other Operations	Structural equivalence	✓	✓	G	✗	G
	PageRank	✓	✗	G	N	S

[a]N=Neighborhood, G=Global
[b]✓=Node and edge, ✗=Neither nodes nor edges, N=Nodes, E=Edges
[c]S=Set, A=Aggregate, G=Graph

2.2.2 Limitations of Graph Databases

Though graph databases offer a very rich data model and, as illustrated in Section 2.2.1, support diverse query types, they are not without limitations. The following list highlights the limitations that apply to graph databases (from one or more vendors) today.

- Declarative interface: most commercial graph databases can not be queried using a declarative language. All vendors provide an imperative programming interface, often with multiple bindings in different languages, but few also offer a declarative query interface. For more on this see Section 3.1.
- Vectored operations (e.g. scatter/gather, map/reduce, etc.): a method of input and output by which a procedure sequentially writes data from multiple buffers to a single data stream or reads data from a data stream to multiple buffers. To horizontally scale it is essential a database support this type of data access.

  To our knowledge, no graph databases support vectored operations today. Current graph databases (like relational databases) tend to prioritize low-latency query execution over high-throughput data analytics. As such, the omission of this functionality is likely the result of a conscious design decision.

  Graph analytics frameworks [50, 48, 32, 59, 34] - designed for high-throughput processing of large data volumes - do offer this functionality. However, these systems are never transactional, rarely persistent, and most often prioritize throughput at the cost of latency - they are therefore not considered graph *databases*.
- Data partitioning: most graph databases do not include the functionality to partition and distribute data across multiple networked computers. This, too, is essential for supporting horizontal scalabilty.

  There are many reasons [17] for this, including the rapidly reducing cost of main memory, making vertical scaling a viable solution for larger installations than was previously possible. Many of the other reasons can be reduced to the non-functional requirement of providing low-latency query execution. As, by definition, graph data is composed of data dependencies, it is difficult to partition a graph in a way that would not result in most queries having to access multiple partitions.

  In contrast nearly all graph analytics frameworks do have inbuilt support for partitioning. This is largely due to the workloads they target. Whereas graph databases aim to provide low-latency query execution, graph analytics frameworks are optimized for high-throughput processing of massive data volumes, making it significantly easier for the latter to mask the cost of network latency.
- High throughput data ingestion: due to lacking support for *vectored operations* and *data partitioning*, the data ingest performance of most graph databases is limited by the write throughput of a single harddrive.
- Query optimization: the ability of the system to transparently optimize the execution plan for any given query. Naturally, most graph databases can not do this as they lack a declarative interface.
- Data schema and constraints: the schema of a database system is its structure described in a formal language. Schema refers to the organization of data, which describes how the database will be constructed. The formal definition of schema is a set of formulas, a language, which describes the integrity constraints imposed on a database. In effect, a populated database can be considered an instance of its schema.

  Schema can make application development a less error-prone task, but is also beneficial as it enables a number of other powerful features, including the ability for the database to perform enhanced query optimization. On the other hand, strict schema enforcement is sometimes considered disadvantageous by those who develop applications for dynamic domains - for example, domains dealing with user-generated content, where the structure of data may change from one day to the next. For precisely this reason, many graph database vendors have opted to either support a weaker notion of schema or to forgo it entirely.

3 Graph Query Languages

3.1 Overview of Graph Query Languages

Associated with graphs are specific graph operations in the query language algebra [14], which refer directly to the underlying graph structure. Existing graph database and query processing research can be classified into two main categories [56].

1. Graph databases consisting of many small graphs, such as: bioinformatic applications, cheminformatics applications, and repositories of business process models. Two types of queries are commonly studied in this category.
    - Subgraph query which aims to find all the graphs in the database such that a given query graph is a subgraph of them.
    - Supergraph query that aims to find all the graphs in the database that are subgraphs of the given query graph.
2. Graph databases consisting of few large graphs, such as: social networks, bibliographical networks, and knowledge bases. Three query types are commonly studied in this category.
    - Pattern match queries, which search for instances of a pattern graph (e.g. path, star, subgraph) in the large graph.
    - Reachability queries, a special case of pattern matching, find the path(s) between any two vertices in the large graph.
    - Shortest path queries, a special case of reachability queries, return only the shortest path(s) between any two vertices in the large graph.

The databases and languages discussed here mostly fall into the second of these categories - those that model large graphs, and query for patterns and the existence of paths. Moreover, special focus will be given to languages designed for querying attributed graphs; applications in this category commonly require the ability to express pattern matching queries based on path patterns [56], where patterns combine both structural (topological) predicates and value (attributes of vertices and edges) predicates.

When working with graph query languages, database transformations are regarded as graph transformations. Based on graph-pattern matching, they allow insertions and deletions to be specified graphically - often regarded to be more intuitive. A significant strength of the graph database model is that it makes graphs and graph operations explicit; that is, it allows users to express queries at a higher level of abstraction, easing the task of working with data in the graph/network domain. This contrasts with query languages for deductive databases (i.e. the relational data model), where complex rules must be written to perform graph manipulations. In particular, when using deductive databases it is difficult for the query language to explore the underlying graph structure, to compute: paths, neighborhoods, patterns, reachability, and graph statistics (e.g. diameter or centrality). On the other hand, computing reachability of information, which translates to path problems characterized by recursive queries, is natively supported by graph query languages like GraphLog [22], Gram [13], Cypher [5], and G-SPARQL [56], and partially supported by GOAL [38], Hyperlog [53], GraphDB [35], G-Log [52], and HNQL [45].

Though rapidly maturing, graph databases and their related technologies are still relatively young. In industry, various query language implementations are in use, most undergoing active development. In academia, numerous works of related research exist, and many more are produced each year - graph query language research spans over 30 years, and has received contributions from many different fields. However, unlike the world of relational databases, where SQL[1] has become ubiquitous, a standard query language for querying graph databases is yet to emerge - the field is still vast and unsettled. As such, this section will first briefly summarize the field,

[1] `http://www.sql.org`

including the most influential academic contributions, and a number of graph query languages already used in industry today.

3.1.1 Imperative vs Declarative

One useful way of categorizing query languages is by separating them into one of the two following groups, imperative and declarative.

In imperative languages algorithms are implemented in terms of explicit steps. A program defines some sequence of commands for the computer to perform; computation is described in terms of statements, which change the state of a program. In contrast, declarative languages express computation logic without describing its control flow; they specify what a program should accomplish rather than the sequence of actions to be taken - the how (specific sequence of actions) is left to the language implementation. Additionally, such languages often attempt to minimize, or eliminate, side effects.

Although two of the most successful database query languages, SQL and SPARQL, are both declarative, each approach has its unique set of advantages. In particular, there are clear tradeoffs regarding: expressiveness, simplicity, performance, optimization, and portability.

Generally, imperative languages are more expressive, i.e., it is possible to implement arbitrary algorithms. Moreover, they provide greater control over how a task is executed, giving skillful, experienced developers the ability to manually optimize the implementation of a query. The advantages of declarative query languages are manifold: using the correct level of abstraction simplifies the task of expressing queries; removing the need to describe how a query should be performed results in more concise syntax; they are easier to port across different database systems; and, perhaps most importantly, it allows the database to create its own query plan, making it possible to apply optimizations based on access patterns, graph structure, etc.

Native graph database systems (e.g., Neo4j [6], DEX [1], OrientDB [7], InfiniteGraph [4]), are designed for efficient querying of persisted graphs. However, many such systems currently lack declarative language support, limiting portability and optimization opportunities, and increasing programmer burden. In the absence of declarative language support the only means of querying these databases is via proprietary, low-level, language-specific, imperative interfaces. As a result, the efficiency of executing a graph query not only becomes programmer-dependent, but assumes that the programmer has sufficient knowledge and skill, which is rarely the case.

3.2 Graph Query Languages from Academia

G [23] (1987) A graph query language for labeled directed graphs with expressive power equal to that of other relational query languages, it was proposed as a compliment to general purpose relational query languages, to make recursive queries more simple to formulate. G is based on regular expressions that allow the simple formulation of recursive queries, which are more general than transitive closures. A query in G is a set of labeled directed multigraphs, where nodes may be either variables or constants, and edges can be labeled with regular expressions. The result of a query is the union of all query graphs that match subgraphs from the instance.

G+ [24] (1988) G evolved into a more powerful language called G+, in which a query graph remains as the basic building block. A simple query in G+ has two elements, a query graph specifying the patterns to search, and a summary graph, which represents how to restructure the query result. The primary use case influencing the design of G+ was to find all pairs of nodes connected by a simple path, such that the concatenation of the labels along the path satisfies a given regular expression. Although the problem is shown to be generally intractable, an algorithm was presented that runs in polynomial time in the size of the graph (when the regular expression and the graph are free of conflicts).

GraphLog [22] (1989) GraphLog is a query language for hypertext that extends G+ by adding negation and unifying the concept of a query graph. A query is now only one graph pattern containing one distinguished edge, which corresponds to the restructured edge of the summary graph in G+. In GraphLog queries are formulated by drawing graph patterns, and their results include all occurrences of those patterns. Edges in the queries represent edges or paths in the database, and regular expressions are used to qualify these paths. A query locates all instances of the pattern that occur in the database, and for each of them defines a virtual link represented by the distinguished edge.

The expressive power of GraphLog is equivalent to first order logic with transitive closure - GraphLog includes an implicit transitive closure operator, which replaces the usual recursion mechanism.

The query languages G, G+, and GraphLog all integrate a family of related graphical languages defined over a general simple graph model. In addition, both G+ and GraphLog support the notion of shortest path.

Gram [13] (1992) The Gram model proposes an SQL-style query language with explicit path expressions. A schema in Gram is a directed labeled multigraph, where nodes are labeled with a type (associated domain of values), and edges are assigned a label (representing a relation between types).

Gram presents a query algebra where regular expressions over data types (node types and edge labels) are used to define walks (paths) in a graph. In Gram walks are the basic objects. A walk expression is a regular expression without union, whose language contains only alternating sequences of node and edge types, starting and ending with a node. When a walk (alternating sequence of nodes and edges) combines with other walks, special objects called hyperwalks are formed.

The Gram query language defines a hyperwalk algebra, which presents unary operations (projection, selection, renaming) and binary operations (join, concatenation, set operations), all closed under the set of hyperwalks.

PaMaL [28] (1993) PaMaL is a graphical data manipulation language based on pattern matching, oriented towards a graphical data model for describing schema and instances of object-databases. Thanks to its three main programming constructs (loop, procedure, and program), PaMaL is computationally complete language.

It presents operators for addition and deletion of nodes and edges, and an additional operation that reduces instance graphs by deleting repeated data. These operations uses graph patterns to specify the parts of an instance on which the operation has to be executed.

Among the highlights of the model are the explicit definition of sets and tuples, multiple inheritance, and the use of graphics to describe queries.

GOAL [38] (1993) GOAL is a graph-based language, capable of expressing all constructive database transformations. Additionally, GOAL includes the notion of fix-points, enabling it to handle the recursion derived from a finite list of additions and deletions. Finally, its pattern-based graphical manipulation syntax makes GOAL more natural to use than text-based interfaces.

GraphDB [35] (1994) The GraphDB model defines special operators for functional definition and querying of graphs. Querying can be done in a familiar style, similar to SQL.

The model allows for explicit representation of graphs by partitioning object classes into simple classes, link classes, and path classes; these are analogous to nodes, edges, and explicitly stored paths of a graph. Additionally, for graph querying, the derive statement can also refer to subgraphs of the database graph.

GraphDB includes a rewrite operation, to manipulate heterogeneous sequences of objects that often occur as a result of accessing the database graph. Graph-specific operations, such as determining a shortest path or a subgraph, are also provided by GraphDB.

An example query in GraphDB consists of several steps, each of which computes operations that specify argument subgraphs in the form of regular expressions over edges that dynamically extend or restrict the database graph.

WEB [33] (1994) WEB was designed specifically for the modeling and querying of genome-graph data, but can also be used to query generic graph databases. It is a declarative query language based on graph logic, which targets the labeled graph model. WEB programs define graph templates for creating, manipulating, and querying objects and relations in the database. Results of these operations are the located matching graphs, in valid instances. An interesting feature of WEB is its support for packaging graphs as encapsulated nodes.

G-Log [52] (1995) G-Log is a declarative language for complex objects with identity, and uses the logical notion of rule satisfaction to evaluate queries that are expressed as G-Log programs. G-Log programs are sets of graph-based rules, which specify how the schema and instance of the database will change. It is a non-deterministic complete query language, thus allowing the expression of a large variety of queries. G-Log may be used in a totally declarative way, as well as in a more procedural manner, providing an intuitive, flexible graph-based formalism for non-expert database users.

HNQL [45] (1995) HNQL defines a set of operators for declarative querying and updating of the hypernode graph model. It includes operators for, assignment, sequential composition, conditional (for making inferences), for loop, and while loop constructs, and uses compounded statements to produce HNQL programs.

Lorel [10] (1997) Lorel was designed for the querying of semi-structured data, where: data is partial (some is missing), similar concepts are represented using different types, heterogeneous sets are present, or object structure is not fully known. It is well suited to cases where such data is queried, whereas traditional data models and query languages are inappropriate.

Lorel is a user-friendly language in the SQL/OQL style, and can be viewed as an extension of OQL [11]. Among the contributions of the Lorel are: extensive use of coercion to avoid the strict typing of OQL, which is inappropriate for semi-structured data; and powerful path expressions that permit a flexible form of declarative navigational access, which are particularly suitable when details of the structure are not known. Lorel also includes a declarative update language.

Hyperlog [53] (2001) Hyperlog is a declarative, logic-based graph query language for the Hypernode [46] model, that supports database query and update. The single data structure of the Hypernode model is the Hypernode, a graph whose nodes can themselves be graphs. Hypernodes are typed, and types, too, are nested graphs.

Hyperlog visualizes schema, data, and query output as sets of nested graphs, which can be stored, browsed and queried in a uniform manner. In Hyperlog, queries consist of a set of graphs that are matched against the database; queries are expressed as sets of hypernode rules called hypernode programs, and database updates are supported via these programs. In addition, Hyperlog defines an operator that infers new hypernodes from the instance, using the set of rules in a hypernode program. Finally, by adding composition, conditional constructs, and iteration, Hyperlog was made Turing-complete.

Evaluation of this language is intractable in the general case, however, in certain cases evaluation can be performed efficiently.

GDM [37] (2002) A graph data model where database instances and schema are described by certain types of labeled graphs, called instance graphs and schema graphs. GDM defines two graph-manipulation operations, addition and deletion, based on pattern matching and represented graphically.

Glide [30] (2002) The Glide query language uses a method called GraphGrep, based on sub-graph matching, to answer queries. GraphGrep is a method of querying graphs to find all occurrences of a sub-graph from within a database of graphs.

Glide, the interface to GraphGrep, is a regular expression graph query language that combines features from a number of other query languages, include XPath [21]; queries are expressed as regular expressions - a linear notation of labels and wildcards (both single node and variable-length wildcards).

GraphQL [36] (2008) Designed with the goal of querying and manipulating graphs with heterogeneous attributes and structures, GraphQL is a declarative graph database query language that supports arbitrary attributes on nodes, edges, and graphs. GraphQL was developed as a general language for querying, both, large sets of small graphs and small sets of large graphs. Graphs are the basic unit of abstraction, each query manipulates one or more collections of graphs, and the output of every expression takes the form of a graph structure.

To support graph compositions, the notion of formal languages is extended from strings to the graph domain. The graph algebra for GraphQL is an extension of relational algebra, where the selection operator is generalized to graph pattern matching, and a composition operator is introduced for rewriting matched graphs.

One potential limitation of GraphQL, it is difficult to express queries about arbitrary path structures, meaning that reachability queries may be difficult or impossibe to expressed using GraphQL.

To improve efficiency of the selection operation, GraphQL uses a combination of techniques: use of neighborhood subgraphs and profiles, joint reduction of the search space, and optimization of the search order.

SNQL [57] (2011) SNQL was designed as a flexible means of representing, transforming, and querying graphs, specifically within the domain of social networks. It is based on GraphLog and second-order tuple generating dependencies, allowing expressiveness for graph querying and node creation as required by a social network, while keeping the complexity of query evaluation in non-deterministic logarithmic space.

G-SPARQL [56] (2011) Motivated by shortcomings in the initial SPARQL specification, G-SPARQL is a SPARQL-like language for querying attributed graphs. G-SPARQL queries combine both structural and value-based (node and edge attributes) predicates, and can express query types that are particularly interesting when working with large graph databases: pattern matching, reachability, shortest path.

Although a number of other extensions (SPARQ2L [15], PSPARQL [12]) have been proposed, and the latest specification (SPARQL 1.1) also addresses various limitations, drawbacks still exist. For example, in SPARQL 1.1 variables can be used at paths ends, but not in the path itself. As a result, queries return all matches of a given path expression - there is no ability to filter further. However, two vertices may be connected by many paths. As such, G-SPARQL extends on this by adding support for filtering conditions on the attributes of vertices and edges participating in the matched path(s).

The implementation of G-SPARQL includes an efficient, hybrid memory/disk representation for the storage of large attributed graphs, where only the graph topology is maintained in memory, while graph data (node and edge attributes) is stored in a relational database. The execution engine splits parts of the query plan to be pushed inside the relational database (using SQL) while the execution of other parts of the query plan are processed using memory-based algorithms.

3.3 Graph Query Languages from Industry

With a few notable exceptions [56, 36], most graph query methods from academia focus on querying the topological structure of graphs, but few consider the attributes on graph elements (nodes and edges). However, graph datasets used in modern applications are not only directed and labeled, but also attributed, and the queries of large graph database applications (e.g. social or citation networks) often consider both these attributes and the graph topology. There is clear a disconnect between the graph query work being performed by academia and industry.

As querying these graphs has received limited attention in the literature, there is little theoretical foundation for building query engines that query such graphs efficiently, especially at the scale (volume of data and load) of modern applications.

SPARQL & RDF [54, 39] (2004) RDF is a recommendation of the W3C, originally designed to represent metadata - a mechanism for describing resources that makes no assumptions about the application domain. A key advantage of the RDF model is its ability to interconnect resources in an extensible way. It models information with graph-like structure, using basic notions of graph theory like node, edge, path, neighborhood, connectivity, distance, and degree.

An atomic RDF expression is a *triple* consisting of subject (resource being described), predicate (property), and object (property value). To draw an analog with the graph domain, in an RDF triple (subject, predicate, object): a subject represents an entity/vertex, a predicate represents a relationship/edge, and an object represents either another entity/vertex or a literal value. Each triple represents a relationship between a subject and object. A general RDF expression is a set of such triples, and can be considered a labeled graph.

Several languages for querying RDF data have been proposed and implemented, which follow the lines of database query languages like SQL, OQL, and XPath [21]. SPARQL [54] is a protocol and query language designed to access RDF stores in a simple way. It defines a query language with a SQL-like style, where a query is based on graph patterns, and query processing consists of the binding variables to generate pattern solutions (graph pattern matching).

The SPARQL specification supports defining query graph patterns and expressing various restrictions on the entities and relationships that participate in those (triple) patterns. Each query defines a query graph pattern `P` that is matched against an RDF graph `G`, where each variable in `P` is replaced by matching elements of `G`, such that the resulting graphs are contained in `G`. Each part of this triple pattern can represent either a constant value or a variable. Hence, a set of concatenated triple patterns represents the query graph pattern.

Gremlin [3] (2010) Unlike most other languages covered here, Gremlin is an imperative, domain-specific language for traversing property graphs. This makes Gremlin very expressive, while still remaining concise thanks to its terse syntax. It is part of Tinkerpop [8] - an open source software stack for working with property graphs. As a result, Gremlin works over any graph database that implements the Tinkerpop interfaces, including: Neo4j [6], OrientDB [7], DEX [1], InfiniteGraph [4], and Titan [9].

OrientQL & OrientDB [7] (2010) OrientDB is an open source document and graph database that defines a query language called OrientQL. Although there is little in the way of literature and documentation, OrientQL reportedly supports the full SQL language standard, and three schema models: schema-full, schema-less and mixed-schema.

Finally, OrientQL includes graph-specific language extensions such as the `TRAVERSE` command. Essentially, `TRAVERSE` introduces the concept of recursion, making it possible to retrieve connected documents/nodes by traversing the relationships between them (to an unbounded depth), in a more performant manner than executing `JOIN` commands.

Cypher & Neo4j [6, 5] (2011) Neo4j is an open-source graph database that defines a query language called Cypher [5]. Cypher is a declarative graph query language, designed to be "humane" and intuitive. Its constructs are based on English prose and neat iconography (ASCII art), making it easier to understand.

Cypher is inspired by a number of approaches and builds upon established practices for expressive querying. Most keywords like `WHERE` and `ORDER BY` are inspired by SQL, and its pattern matching functionality borrows expression approaches from SPARQL. In short, Cypher is comprised of several clauses: `MATCH`, graph pattern to match; `WHERE`, filtering criteria; `CREATE`, creates nodes and relationships; `DELETE`, removes nodes, relationships and properties (attributes); `SET`, sets values of properties (attributes); `FOREACH`, performs updating actions once per element in a list; `WITH`, divides a query into multiple, distinct parts; `RETURN`, specifies what to return.

4 Use Cases

4.1 Introduction

Graph databases are becoming popular recently due to the increase of applications that handle data in form of graphs. However, one aspect that has not been addressed yet is the benchmarking of such technology, which is the main objective of the LDBC project.

Graph database benchmarking requires, among other aspects, to have clear scenarios where the benchmarks are executed. Such use cases, must be representative of the real world situations where the benchmarked technologies are applied in order to test the critical points of the given database. In addition, defining good use cases for database benchmarking is crucial in the sense that in this way, the system under evaluation is tested under appropriate data, operations and queries, so that the obtained results are representative of the behavior of the system under near real situations.

The main objective of this section is to present different use cases potentially useful for graph database benchmarking. The use cases presented in this work are not an exhaustive list of possible real scenarios where graph databases could be applied, but rather a selection of the most representative ones in terms of impact in the industrial community, the type of operations performed and the kind of graphs managed. In particular, we detail a use case related to social network analysis and one use case related to information technology networks. In each use case, we introduce the kind of graphs the scenario typically handles, its characteristics and the operations and queries that are typically associated with it. Also, we present several real projects implemented by the LDBC vendor partners that show different use cases for graph databases.

4.2 Social network analysis

In this section, we introduce the first of our use cases, the Social Network Analysis (SNA). First, we give a brief introduction about SNA. After that, we characterize the use case, giving the underlying graph model and its characteristics, and introducing the types of operations performed on social networks. This way, we fully characterize the use case in order to better understand which characteristics a benchmark should have when run on this kind of data.

4.2.1 Introduction

In social networks nodes typically represent people and edges represent some form of social interaction between them, such as friendship, co-authorship, etc. Although the study of the characteristics of social networks known as *Social Networks Analysis* (SNA) (formerly known as *sociometry*), has its starting point in the early 30s, it has become very popular in recent years because of the digital techniques and internet. SNA techniques have been effectively used in several areas of interest like social interaction and network evolution analysis, counter-terrorism and covert networks, or even viral marketing. Due to the Web and increasing use of Internet applications, which facilitate interactive collaboration and information sharing, many social networks of different kinds have appeared, like Facebook and LinkedIn for social interaction, or Flickr for multimedia sharing. Other web portals that contain human interactions can also be considered social networks, like in bibliographic catalogs such as Scopus, ACM or IEEE, where the scientific community is sharing information and establishing de facto relationships. In all these cases, there is an increasing interest in the analysis of the underlying networks, to obtain a better knowledge of the patterns and the topological properties. This may be used to improve service to users or even to provide more profit to the information providers in the form of direct advertising or personalized services.

4.2.2 Graph model

As we have seen before, there are many different kind of social networks. Therefore, the nature of the underlying graph model of these networks may differ from one to another. However, the following characteristics are common to many of the existing social networks:

- **Attributed:** Graphs belonging to social networks are attributed graphs. We can find attributes both in the nodes and in the edges. Node attributes may include personal data about the user, their preferences, activity log, comments, etc. Edges may be attributed with the number of times two persons have interacted, comments, etc.
- **Labeled:** Social graphs may be labeled in both the nodes and the edges. For example, interactions between two different users may have different forms, such as like/dislike something, request for something, comments about a post, etc. In the same way, nodes may represent different entities, such as persons or companies in a professional social network.
- **Directed:** Graphs representing social networks are usually directed graphs, since the interactions between the actors in the network are not always symmetric. For instance, a user may like/dislike a comment of another user, and this is a form of asymmetric or directed interaction, since the user of the comment has no activity in the opposite direction.
- **Multigraph:** Social interactions are usually recurrent. That is, people linked through a social network usually have more than a single interaction, moreover their interactions are unlikely to be limited to the same types. For instance, two friends may have several interactions, some of them being comments about one user post and some of them sending a private message. This multiplicity in the interactions may be represented in a multigraph.

4.2.3 Statistical properties

In the following, we summarize the statistical properties that characterizes the graph. Since social networks are essentially evolving networks, we distinguish between static and dynamic properties.

Static properties:

Static properties are those appearing in snapshots of the network at a certain point in time.

- **Community structure:** Real-world social graphs are found to exhibit a modular structure, with nodes forming groups, and possibly groups within groups [27, 29, 58]. In addition to that, in [51], it is shown along several social network examples, that in most cases there is a **large component** which includes more than 80% of the total nodes of the network.
- **Small-world property:** Social networks exhibit the **small-world property**. That is, even in the case where the network is composed of millions (or even billions) of nodes, the average geodesic distance between connected vertex pairs is relatively very low, approximately log the number of nodes in the network (around 5 in most of the examples given in [51]).
- **Degree distribution:** The degree distribution of many social network obey a power law of the form $f(d) \propto d$, with the exponent > 0, and $f(d)$ being the fraction of nodes with degree d. Therefore, they can be considered **scale-free networks**.
- **Sparse:** Real social networks are almost always **sparse**, meaning that only a small portion of the total possible number of edges appear in the network. The examples given in [51], show that the portion of existing edges with respect to the total possible edges is less than 1%.

Dynamic properties:

Dynamic properties are those characteristics that the graph exhibits with respect to a change in time. These are typically studied by looking at a series of static snapshots and seeing how measurements of these snapshots compare.

- **Shrinking diameter:** Leskovec. et al. [42] showed that not only is the diameter of real social graphs small, but it also shrinks and then stabilizes over time [42]. Briefly, at the beginning of time, the nework is composed by several small components. As time evolves those small components grow and connections between them lead to bigger connected components and a growing diameter. At some point (the *gelling point*), many of these components merge and the large component emerges and the diameter spikes. After this point, the **diameter keeps shrinking until it reaches an equilibrium**.
- **Densification power law:** Time-evolving social graphs show the following relation between the number n of nodes and the number m of edges at all time ticks t: $m(t) \propto n(t)^\beta$, with $\beta > 1$, which is known as the **densification power law**. Examples of social networks shown in [51], discover a mean value of β around 1.12.

4.2.4 Graph operations and queries

One characteristic of social networks is that the operations performed on them are extremely diverse, they cover much of the spectrum of operations known to be performed on graphs. Some examples in different workflows are:

- **Transactional:** Insertions, updates and deletions are usually small and affect a few entities (nodes) and relationships (edges). The most usual operation is the insertion of new data, a very frequent action with a high degree of isolation with respect other update operations. Updates are not frequent because the SN grows and information is more evolving than changing. Deletes also are not usual and, in general, information is timestamped when it is deleted to denote the end of its availability instead of begin removed.
- **Lookups:** The basic queries are the most frequent: look for a node, look for the neighbors (1-hop), scan edges in several hops (layers), retrieve an attribute, etc. In general, these operations are small and affect only a few nodes, edges and attributes in the graph. When the graph schema is complex, most of the lookup queries follow a few query patterns where the underlying lookup operations are in general the same with different arguments. Concurrency is one of the most important issues due to the high amount of small queries executed at the same time in sparse areas of the graph.
- **BI:** While the SN graph data contains a lot of useful information for bussiness intelligence, this is not usually explored due to privacy concerns and restrictions. Aggregate computations or multidimensional analysis using edge adjacencies as dimensions are in general only performed after an anonymization process.
- **Analytics:** Graphs metrics, centrality measures or community finding are tools used to analyze the SN to observe the behaviour, predict the evolution or to identify the shape in order to split a very large graph in smaller units more easy to manage. Pattern matching is often used to extract groups of nodes and edges that match a specific pattern, for example for marqueting purposes, data cleansing or integrity validation.

4.3 Information Technologies Analysis

Organizations use a significant amount of internal and external data to obtain added value information that provides them with an understanding of the positioning of the world in relation to their knowledge and objectives. However, they employ a significant amount of time to complete this search and analysis cycle because of the lack of quality in the data and the lack of flexible technologies to extract and integrate multimedia and multilingual features from the sources, having to use the skills of experts in a slow, error prone and inspiration dependent process.

4.3.1 Introduction

Knowledge, which sits in the digital core of organizations like SMEs, large companies and public institutions, is not fully exploited because data inside the organization is stored in separate unconnected repositories: the documents written (internal reports, patents filed, meeting minutes, usage manuals, papers published, collaboration reports of funded projects, etc.), strategy reports, financial audits, managerial structure, the electronic mail generated, the relationships with other organizations expressed by means of contracts and agreements and by means of IP ownership and mercantile transactions, the media content produced through courseware and marketing material, and more. Moreover, the international nature of many organizations implies that multiple languages are used in the data they generate. The dispersion and unlinked multimodal nature of those sources leads to a significant lack of corporate self-knowledge[1] that is hidden behind the internal repositories. On the other hand, the Internet offers a huge amount of relevant outside data about the organization: web pages, social networks, product opinions, cloud services... Although organizations usually know which are those interesting data sources, they currently need huge human driven efforts to retrieve and analyse them in the multiple languages and multiple formats they are provided: textual and video blogs and microblogs criticizing or praising their achievements, other companies assessing their performance, newspapers telling stories about them, open data in the form of patents or scientific papers explaining inventions related to their knowledge, videos and images making apparent to the eye events where the organizations are involved, etc. The use of the Internet content in many cases is not only enriching but necessary for the adequate growth of those organizations, and in particular for SMEs.

The integration of inside and outside organization data can merge in a single vision the collection of internal partial perspectives of the company business departments as well as the view of the company in the external world. Corporate data can be analysed and information can be extracted, interpreted and summarized in the form of added value knowledge and linked relationships among documents (either textual or media), people in the organization, concepts and keywords in the different languages of the organization providing a network between the sources of knowledge and the actual linked information that describes them. Moreover, from the linked relationships, further analysis can be done to create multilingual ontologies that organize the knowledge of the organization. In all those cases, the relationships and ontologies can be exploited to obtain added value information like, for instance, who knows more and is more reputed within or outside the organization about a topic to find are placement for a person who quit the company, what is the most relevant internal and external IP and how they are related for a specific research being done and who are the most relevant inventors, what internal and external media content is available for the next marketing campaign, what are the documents that describe the products to be announced better and who are the employees with better knowledge for those, etc. Thus, from the need to analyse a combination of structured data and contents it is necessary to integrate data from different sources to obtain composite views, including image metadata, blogs, and web pages sourced from within and outside the enterprise.

4.3.2 Dataset Integration

When integrated in a single graph-based framework, the information extracted from the multimedia and multilingual repositories is merged in such a way that the identification of relations and similarities within and across different media will be easier. This way, the internal data sources can be linked, and enriched information is extracted providing added value ground information to increase the ability to detect and exploit meaning from where it was hidden before with analytical queries. The linked information and ontologies created is constantly enriched by the new documents being created within the organization providing a circle of constant improvement of the corporate self-knowledge.

Integration techniques are applied to intelligently aggregate the result sets of the different data providers by means of entity identification techniques [18]. Data linkage typically uses weak identifiers (name, family name, country, etc.) to link data across different data sources. In the case of graphs the integration target are

[1]By self-knowledge, we understand the analytical capability that allows an organization to extract added value information from the integrated view of the data in their different applications and repositories.

the vertices of the graph, and hence, the entity data linkage deals with finding those vertices that represent the same entity. In order to obtain a perfect recall, the problem becomes quadratic because it is necessary to perform all pairwise comparisons. Since this is prohibitive for large volumes of information one of the main research topics is finding techniques on how to scale them [20]. Some data integration frameworks are available from the research community that facilitate the integration of data. They can be classified in three main groups, based on the interface of the framework: rule, numerical and workflow based. Rule based approaches give users the freedom to state sets of rules that are applied sequentially to integrate datasets [16]. Such rules are not a static set, and can change over time in order to increase the flexibility of the system [62]. Furthermore, such rules can express even exceptions to stated rules, which facilitate the design of the system and the resolution of inconsistencies among previously ingested rules [61]. Numerical approaches compute complex similarity functions based on a set of features among a pair of entities. Those entities with a numerical value over certain threshold are considered as the same entity. The construction of the numerical function and the threshold setting can be programmed by the users of the system [40], or helped with the aid of a training set [55]. Workflows allow users to define complex data flows where combinations of matchers, conditions and loops [60]. A graph-based framework include functionalities to integrate easily graph features (such as transitive relations or graph patterns among others) during the integration process to compute the similarity of entities that are in a graph. It allows also the scalability of the system in order to support the large graphs coming from different data sources

4.3.3 Graph Analytics

Once the datasets have been integrated inside a single graph. the goal is to provide a set of techniques to analyse the the relationships among the entities. Some examples of self-knowledge services are:

- A document search engine that is be able to return the most relevant documents for a given topic. It allow the analysts to explore the contents of the documental data stored in the graph. The result is a set of documents that had been obtained from outside or inside the information network.
- A reputation algorithm to rank the most relevant persons and organizations in a network according to a search topic. The algorithms take into account that real networks are not hierarchic and consider the cycle shapes to deduce the most reputed individuals. The results are able to return people that are relevant for a query with respect to the information extracted from the graph.
- A sentiment analysis summarization procedure to evaluate multimodal data that talks about a brand name. The query aggregates the sentiment analysis results obtained for a brand name, in order to show to the analysts which is the perception of a product among customers.

In particular, for the different workflows some of the required graph query capabilities are:

- **Transactional:** The graph is built like a large data warehouse of entities and relationships. There are few updates and, in general, all new data is inserted in massive bulk loads of preprocessed, deduplicated and interrelated data. This process can be executed also over a snapshot in such a way that updates are not in conflict with read-only query. This relaxes the locking and concurrency requirements of the graph database engine.
- **Lookups:** Queries are more analytical than exploratory. Simple lookup queries are used only to validate the content of the generated graph or to generate reports of the data.
- **Analytics:** This represents the most important group of queries for this use case. Analysis is made in several steps by combining different techniques. For example, reputation requires the construction of communities or clusters based on search topic; then the graph is improved with weighted relationships of the involved people; finally, a recommendation algorithm based on connectivity returns the relevent nodes.

4.4 Other Use Cases

In addition to the detailed use case descriptions provided in Section 4.3 and Section 4.2, this section contains short descriptions of many more graph data management use cases.

4.4.1 Master Data Management

Master Data Management (MDM) refers to managing the inherent complexity of the master data sets themselves, including building and maintaining an accurate model of complex hierarchical data sets. Master data such as organization and product master are inherently shaped like graphs: deep hierarchies with top-down, lateral, and diagonal connections. Managing such data models with a relational database results in complex and unwieldy code that is slow to run, expensive to build, and time-consuming to maintain. Graph databases are increasingly used in this space, for managing these types of data sets, as they provide greater agility and performance benefits. As graph databases allow relationships to be attributed with properties, MDM applications ascribe relationships with effective dates, types, qualities, and other descriptive information.

4.4.2 Network and Data Center Management

The graph techniques originally devised to analyze physical routing problems are nearly identical to those used for electronic networks. As the electronic equivalent of geographical routing (the application for which graph theory was invented), Network Management is among the applications that can benefit most from graph databases. Examples of such applications are network failure and degradation analysis, quality of service mapping, OSS network inventory mapping, and network asset management.

4.4.3 Geographical

What sparked the invention of graph theory in 1736 by Leonhard Euler was a geographic routing problem. It is therefore no surprise that graph databases are well suited for business applications involving geography, routing, and optimization, including: road, airline, rail, or shipping network. Graph databases excel at mapping physical locations against the various ways that one might get people or things to and from those locations. Graph algorithms enable extremely fast execution of common problems, such as "shortest path".

4.4.4 Social

More and more organizations across a variety of sectors are gaining competitive and operational advantage by leveraging social computing. Plugging into the brave new world of social media is a common way to do this, and is a common driver for adopting graph databases. Social computing refers to the use of information about relationships and social context in business and operational decisions, recognizing the impact of social context on individual preferences and behavior.

4.4.5 EU Projects Analysis

Applying for a European project is not a straightforward task. Many aspects must be considered before starting to write a proposal, such as "*Is an idea original?*", "*What research has been done in the same field?*", and "*Which are the best partners in order to successfully achieve the objectives?*". Moreover, coordinator institutions must take into consideration the chances of the project to be approved before enrolling other partners for the proposal, saving valuable time for their organizations.

The solution is to integrate different public databases in a single graph database for the benefit of a better project proposal and analysis, linking EC official data with bibliographic data, and allowing your search for the best partners and the best bibliographic items for the State of the Art of projects. The analysis approach also helps to understand the progress of projects, institutions and regions, even compared to other institutions and regions.

The exploration of the integrated graph data allows the analyst to:

- Recommend the most effective consortium for a new idea: potential partners from their knowledge in previous European projects, as well as the number of successful projects where they were involved.
- Look for similar projects to a new idea: just provide the abstract of the proposal and any similarity with previous EC funded projects will be detected.
- Recommend the best bibliography for the State of the Art of a new project
- Analyze relationships between consortiums, with profiles of the companies and the most adequate groupings for the topics selected, either with keywords or abstracts.
- Analyze relationships between projects by understanding how well different companies fit the project, and how well they fit with other companies in the past. It also allows finding how consortiums are connected through projects and topics.
- Analyze historic data to discover the insights of the data from the already approved projects, how it is related and how it evolves through time

4.4.6 Syntactical Structures Search Engine

A Syntactical Structures Search engine browser is an interesting tool addressed to linguists, which contains a tree-bank with sentences syntactically annotated. Dependency grammar is the formalism used to represent the syntactic information. Such formalism allows seeing a sentence as a graph, therefore all the syntactic information in the corpus is represented as a directed graph, nodes being the words in the corpus and edges being the dependencies - dependencies are the annotations among related words.

All sentences in the corpus are semi-automatically analyzed using a grammar with a predefined set of dependency relations such as subject, direct object, specifier, modifier, and punctuation. The tree-bank browser allows searching for sentences in the corpus that satisfies user defined patterns. Such patterns take into consideration both dependencies and word information; the latter may include any combination of part-of-speech, word form, and lemma.

Taking advantage of the nature of the graph, there are no restrictions in the position of the elements in a search. Therefore, it will find a solution independently of the relative position of each item of the query in the sentences (e.g. subjects/modifiers in pre-verbal or post-verbal position).

4.4.7 Detecting Social Capitalists in a Social Network

Social capitalists are those users that try to gain visibility by following users regardless of their content. Social capitalists are not healthy for social networks as they help spammers to gain visibility and may mislead influence detection. Some of the techniques used by social capitalists are "*follow me and I follow you*" or "*I follow you, follow me*", insisting that the majority of users they follow should follow them back (overlap). On the other side, spammers wish to accumulate followers as then spread spam links. Social capitalists can be detected inside a Social Network using similarity measures, even without analyzing the messages of the users, as data of the graph topology is often sufficient for this purpose.

Graph databases are better suited than other technologies to quickly answer questions like retrieving the neighborhood of the nodes, which is essential in the computation of the similarity algorithms.

4.4.8 Detection of threats of insiders

Insiders are those people who work, or have previously worked, in a company and intentionally misuse the access to compromise available information. A popular example is Wikileaks, and how the threat of insiders should be a concern for any company. Nowadays, with the outsourcing done with cloud computing, it is more important to detect insider attacks than ever. One approach to avoid it is to detect deviations of users from normal behavior while accessing the systems, using a graph database in order to benefit from its capabilities to store huge volumes of data to be analyzed.

The first step of the process is to extract all the access logs of the applications and to integrate them in a single graph database. For a large corporation this usually means thousands of connections and billions of log entries. User commands are also integrated in the database and correlated with the log data. Then, a cluster analysis algorithm detect communities, based on the accessed resources, projects and the daily access patterns. These communities are useful to discover that a deviation on the daily pattern can be an alert of a possible insider threat.

References

[1] Dex - graph database. http://www.sparsity-technologies.com/dex.php.

[2] Flickr: Four billion (jun 2010). http://blog.flickr.net/en/2009/10/12/4000000000.

[3] Gremlin - graph traversal language. http://gremlin.tinkerpop.com.

[4] Infinitegraph - graph database. http://objectivity.com/infinitegraph.

[5] Neo4j cypher documentation. http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html.

[6] Neo4j: The neo database (2006). http://neo4j.org.

[7] Orientdb - open source graph database. http://www.orientdb.org.

[8] Tinkerpop - open source property graph software stack. http://www.tinkerpop.com.

[9] Titan - open source graph database. http://thinkaurelius.github.com/titan.

[10] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The lorel query language for semistructured data. *International journal on digital libraries*, 1(1):68–88, 1997.

[11] AM Alashqur, S.Y.W. Su, and H. Lam. Oql: a query language for manipulating object-oriented databases. In *Proceedings of the 15th international conference on Very large data bases*, pages 433–442. Morgan Kaufmann Publishers Inc., 1989.

[12] F. Alkhateeb, J.F. Baget, and J. Euzenat. Extending sparql with regular expression patterns (for querying rdf). *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):57–73, 2009.

[13] B. Amann and M. Scholl. Gram: a graph data model and query language. In *Proceedings of the ACM conference on Hypertext*, pages 201–211. ACM, 1992.

[14] R. Angles Rojas and C. Gutiérrez. Survey of graph database models. *ACM COMPUTING SURVEYS*, 40(1), 2008.

[15] K. Anyanwu, A. Maduko, and A. Sheth. Sparq2l: towards support for subgraph extraction queries in rdf databases. In *Proceedings of the 16th international conference on World Wide Web*, pages 797–806. ACM, 2007.

[16] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 952–963. IEEE, 2009.

[17] Alex Averbuch and Martin Neumann. Partitioning graph databases-a quantitative evaluation. *arXiv preprint arXiv:1301.5121*, 2013.

[18] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Computing Surveys (CSUR)*, 41(1):1, 2008.

[19] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys (CSUR)*, 38(1):2, 2006.

[20] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.

[21] J. Clark, S. DeRose, et al. Xml path language (xpath) version 1.0, 1999.

[22] M.P. Consens and A.O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proceedings of the second annual ACM conference on Hypertext*, pages 269–292. ACM, 1989.

[23] I.F. Cruz, A.O. Mendelzon, and P.T. Wood. A graphical query language supporting recursion. *ACM SIGMOD Record*, 16(3):323–330, 1987.

[24] I.F. Cruz, A.O. Mendelzon, and P.T. Wood. G+: Recursive queries without recursion. In *Proceedings of the Second International Conference on Expert Database Systems*, pages 355–368, 1988.

[25] Paul Ërdos and Alfreéd Rényi. On random graphs. *Mathematicae*, 6:290–297, 1959.

[26] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 251–262. ACM, 1999.

[27] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3):66–71, 2002.

[28] M. Gemis and J. Paredaens. An object-oriented pattern matching language. *Object Technologies for Advanced Software*, pages 339–355, 1993.

[29] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[30] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 112–115. IEEE, 2002.

[31] Ben Goertzel. Opencogprime: A cognitive synergy based architecture for artificial general intelligence. In George Baciu, Yingxu Wang, Yiyu Yao, Witold Kinsner, Keith Chan, and Lotfi A. Zadeh, editors, *IEEE ICCI*, pages 60–68. IEEE Computer Society, 2009.

[32] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 17–30, 2012.

[33] M. Graves, E.R. Bergeman, and C.B. Lawrence. Querying a genome database using graphs. In *Proceedings of the 3th International Conference on Bioinformatics and Genome Research*, 1994.

[34] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514. International World Wide Web Conferences Steering Committee, 2013.

[35] R.H. Güting. Graphdb: Modeling and querying graphs in databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 297–297. Citeseer, 1994.

[36] H. He and A.K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 405–418. ACM, 2008.

[37] J. Hidders. Typing graph-manipulation operations. *Database TheoryâĂŤICDT 2003*, pages 394–409, 2002.

[38] J. Hidders and J. Paredaens. Goal, a graph-based object and association language. 1993.

[39] O. Lassila, R.R. Swick, et al. Resource description framework (rdf) model and syntax specification. 1998.

[40] Luís Leitão, Pável Calado, and Melanie Weis. Structure-based inference of xml similarity for fuzzy duplicate detection. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 293–302. ACM, 2007.

[41] Jure Leskovec, Daniel P. Huttenlocher, and Jon M. Kleinberg. Signed networks in social media. In Elizabeth D. Mynatt, Don Schoner, Geraldine Fitzpatrick, Scott E. Hudson, W. Keith Edwards, and Tom Rodden, editors, *CHI*, pages 1361–1370. ACM, 2010.

[42] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *KDD*, pages 177–187. ACM, 2005.

[43] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *WWW*, pages 695–704. ACM, 2008.

[44] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *Proceeding of the 17th international conference on World Wide Web*, pages 695–704. ACM, 2008.

[45] M. Levene and G. Loizou. A graph-based data model and its ramifications. *Knowledge and Data Engineering, IEEE Transactions on*, 7(5):809–823, 1995.

[46] M. Levene and A. Poulovassilis. The hypernode model and its associated query language. In *Information Technology, 1990.'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No. 90TH0326-9)*, pages 520–530. IEEE, 1990.

[47] Mark Levene and Alexandra Poulovassilis. The hypernode model: A graph-theoretic approach to integrating data and computation. In *FMLDO*, pages 55–77, 1989.

[48] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[49] Mary McGlohon, Leman Akoglu, and Christos Faloutsos. Weighted graphs and disconnected components. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 524–532, 2008.

[50] Jacob Nelson, Brandon Myers, Andrew H Hunter, Preston Briggs, Luis Ceze, Carl Ebeling, Dan Grossman, Simon Kahan, and Mark Oskin. Crunching large graphs with commodity processors. In *Proceedings of the 3rd USENIX conference on Hot topic in parallelism*, pages 10–10. USENIX Association, 2011.

[51] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[52] J. Paredaens, P. Peelman, and L. Tanca. G-log: A graph-based query language. *Knowledge and Data Engineering, IEEE Transactions on*, 7(3):436–453, 1995.

[53] A. Poulovassilis and S.G. Hild. Hyperlog: A graph-based system for database browsing, querying, and update. *Knowledge and Data Engineering, IEEE Transactions on*, 13(2):316–333, 2001.

[54] E. Prud'Hommeaux, A. Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008.

[55] Vibhor Rastogi, Nilesh Dalvi, and Minos Garofalakis. Large-scale collective entity matching. *Proceedings of the VLDB Endowment*, 4(4):208–218, 2011.

[56] S. Sakr, S. Elnikety, and Y. He. G-sparql: A hybrid engine for querying large attributed graphs. 2011.

[57] M. San Martın, C. Gutierrez, and P.T. Wood. Snql: A social networks query and transformation language. *cities*, 5:r5, 2011.

[58] Michael F. Schwartz and David C. M. Wood. Discovering shared interests among people using graph analysis of global electronic mail traffic. *Communications of the ACM*, 36:78–89, 1992.

[59] Philip Stutz, Abraham Bernstein, and William Cohen. Signal/collect: Graph algorithms for the (semantic) web. In *The Semantic Web–ISWC 2010*, pages 764–780. Springer, 2010.

[60] Andreas Thor and Erhard Rahm. Moma-a mapping-based object matching system. 2009.

[61] Steven Euijong Whang, Omar Benjelloun, and Hector Garcia-Molina. Generic entity resolution with negative rules. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 18(6):1261–1277, 2009.

[62] Steven Euijong Whang and Hector Garcia-Molina. Entity resolution with evolving rules. *Proceedings of the VLDB Endowment*, 3(1-2):1326–1337, 2010.