Introduction to Graph Databases

Josep Lluís Larriba-Pey[1], Norbert Martínez-Bazán[2], and David Domínguez-Sal[2]

[1] DAMA-UPC, BarcelonaTech, Barcelona, Spain
larri@ac.upc.edu
[2] Sparsity Technologies, Barcelona, Spain
{nortbert,david}@spartisty-technologies.com

Abstract. The use of graphs in analytic environments is getting more and more widespread, with applications in many different environments like social network analysis, fraud detection, industrial management, knowledge analysis, etc. Graph databases are one important solution to consider in the management of large datasets. The course will be oriented to tackle four important aspects of graph management. First, to give a characterization of graphs and the most common operations applied on them. Second, to review the technologies for graph management and focus on the particular case of Sparksee. Third, to analyze in depth some important applications and how graphs are used to solve them. Fourth, to understand the use of benchmarking to make the requirements of the user compatible with the growth of the technologies for graph management.

1 Introduction to 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 that are relevant to mention and that will be useful for defining graph data models upon which we develop this paper. We summarize them in the following points.

– **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 the 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 relationship, which is usually interpreted as its corresponding length, weight, cost or intensity. 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.

M. Koubarakis et al. (Eds.): Reasoning Web 2014, LNCS 8714, pp. 171–194, 2014.

- **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 determine 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" [1], 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 relationships. 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 nodes connected with more than one edge when more than one call exists between two telephone numbers.
- **Hypergraphs:** Hypergraphs are a generalization of graphs, where edges are substituted by hyperedges. In contrast to regular edges, a hyperedge connects an arbitrary number of nodes instead of two. Hypergraphs are used, for example, for building artificial intelligence models [2]. Although Hypergraphs appear commonly along different types of networks, in practice they are usually represented as bipartite networks [3], since it facilitates its representation and posterior treatment by the algorithms.
- **Hypernodes:** A hypernode graph is another graph generalization, where nodes are substituted by hypernodes. A hypernode is an entity that contains a set of nodes and edges (i.e. a graph). A regular node is equivalent to an hypernode that contains a single node and no edges [4]. Hypernodes are used to nest graphs inside graphs. They 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.

Hypergraphs and hypernodes are formally well defined, but their popularity is limited because of their additional complexity. Unless otherwise stated, in this paper 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.

1.1 Graph Characterization

Real graphs are typically very different from graphs following the Erdös-Renyi model (random graphs) [5]. Leskovec et al. [6], analyzed over 100 real-world networks belonging to the following fields: social networks, information/citation networks, collaboration networks, web graphs, Internet networks, 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 to more than one hundred million edges. 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 6 billion photographs that can be tagged and rated [7], 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 [6, 8, 9], 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 [10], and that will be useful to characterize the graphs from in the use cases below.

- **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 not 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 component 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 chosen 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:** Although the exact connection patterns between nodes may differ between graphs, the macroscopic structure of the degrees is often very regular and fits known statistical distributions. We denote the degree of a node as k, or in other words the number of edges attached to that node. Then, the proportion of nodes of the graph that have degree k is $p_k = \frac{\#\ nodes\ with\ degree\ k}{n}$. Thus, p_k can also be seen as the probability that a randomly chosen node has degree k.
  Degree distributions from graphs typically follow power law distributions, that correlate exponentially the number of nodes with a given degree to its frequency. The most popular statistical distribution is the Zipf that defines $p_k = C \cdot k^{-\alpha}$, where C is a normalization factor. These distributions are often plot in logarithmic scale because they trace a straight line since $ln(p_k) = -\alpha \cdot ln(k) + ln(C)$. The most common values for α are between $2 \leq \alpha \leq 3$ (see [3] for a complete list of networks with the corresponding values of α). We note that, for some graphs, the Zipf distribution does not model well the nodes with few connections (do not fit well the straight line), and an alternate process called Zipf with cut-off is used. This procedure removes the small degree nodes when the α for a Zipf is estimated.
  Networks that follow power-law degree distributions are called scale-free networks because their degree distribution look similar for all graph sizes. To give an example, in the Internet network[3], 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.
- **Small Average Degree:** The maximum average degree of a graph is $(n - 1) = O(n)$, which corresponds to a structure called clique that connects all pairs of nodes. These graphs are described as dense because they have many edges. However, the study of real graphs has determined that such dense graphs are not common for real datasets [11]. Graphs that represent real world data have an average degree that is small compared to the number of nodes in the graph. The average degree typically remains in the range between 3 and 100, even for graphs with millions of nodes [12]. For graphs that grow over time, it has been found that the average degree tends to increase slightly faster than the number of nodes [11], but the growth is so slow that it is rare to find real graphs with average degrees over a thousand [12].
  Graphs with small average degree are also referred as sparse graphs. The notion of sparse comes from the matrix representation of the graph, which indicates with one the presence of an edge, and zero otherwise. Sparse matrices are those that have a large number of zeros, and similarly, sparse graphs are those whose matrix representation has a large number of zeroes.
- **Large Clustering Coefficient:** Graphs from real datasets have often a clustering coefficient larger than expected by pure chance. This is an effect of transitive relations among members of the network. In other words, "the friends of my friends are also my friends". Graphs usually have observable communities that are groups of nodes structurally strongly related among them, but not structurally related to the rest of the graph.

1.2 Graph Operations and Queries

A graph operation is a computation on the graph that is directly interpreted by the engine of the GDB. A graph query is a user statement that requests a piece of information from the database, which requires one or more operations to be computed.

There is a set of basic operations that is available in most GDBs, which includes: (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/update/delete the nodes/edges/attributes of the graph.

Then, there are graph queries that are more complex and which are built on top of those basic operations. Some GDBs implement subsets of graph queries, such as graph traversals, as operations that are directly interpreted by the GDB. Therefore, depending on the software under analysis some queries can be referred as operations, too. The most common families of graph queries are the following:

- **Traversals:** Traversals are queries that, given a set of starting nodes, explore recursively the neighborhood of those nodes until a terminating condition, such as a fixed number of steps or the arrival to a target node, is fulfilled. Consider for instance, the computation of 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 query is the computation of *k-hops*. That is the query returns all the nodes that are at a distance of k edges given a source node. A particular case that is worth to mention because it is widely used in other queries is the 1-hops (i.e $k = 1$). In this case, the query returns all the neighbors of the source node, also known as the neighbors of the node. Examples of queries 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 largest 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 queries, and it is usually used during pre-processing steps in order to help further computations. Related to connected components, there are some helpful queries 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 determining the relative importance of a vertex within the graph, based on how well this node connects the network. For instance, in a social network, the 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 queries aim at answering 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 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 Queries:** There are other queries related to the applications presented later in this paper. 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 summarize the previously described operations and queries in Table 1. We note that these graphs operations and queries are not homogeneous from the computational complexity point of view, because they range from constant time to NP-complete complexity. We observe that applications compute a rich set of complex graph queries, using a small set of basic operations that are shared by all scenarios.

2 Graph Databases

A graph database (GDB) 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; Sparksee[2], a multi-platform graph database management system for efficient graph management in memory constrained environments; HyperGraphDB[3], an embeddable graph database with generalized 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 are characterized by different aspects ranging from the extension of the graph being accessed to the answer they give.

2.1 Operation 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.

[1] http://neo4j.org
[2] http://www.sparsity-technologies.com/
[3] http://www.hypergraphdb.org/index
[4] http://www.orientdb.org/index.htm
[5] http://www.infinitegraph.com/

Table 1. Graph operations and queries

Graph Operations		**Categorization**				
Group	Operation	Analytical	Cascaded	Scale[a]	Attr.[b]	Result[c]
Basic 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
Complex operations / Queries						
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	Degmore ree 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 Queries	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

- **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 operation does not modify the graph, it may need access to secondary storage because the graph or the temporary results generated during the operation 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 operation 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 distinctive output for a graph database operation 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.

3 Case Study: The Sparksee Graph Database

Sparksee[6] is an efficient GDB implementation based on bitmap representations of the entities. It is devised to directly handle labeled and directed multigraphs containing an undetermined number of attributes in both nodes and edges. In [13, 14], the authors propose a logic bitmap-based organization to store a graph that does not fit in memory and has to be handled out-of-core. In this scenario, several aspects must hold:

- Computing an operation should not imply loading the whole graph into memory.
- The graph organization must be as compact as possible in order to fit as many graph structures in memory as possible.
- The most commonly used graph-oriented operations, such as edge navigation, should be executed as efficiently as possible.
- Attributes in the graph should be accessed very fast.

In Sparksee, all the nodes and edges are encoded as collections of objects, each of which has a unique oid that is a logical identifier. Sparksee converts a logical adjacency matrix into multiple small indexes to improve the management of out-of-core workloads, with the use of efficient I/O and cache policies. It encodes the adjacency list of each node in a bitmap, which for the adjacent nodes has the corresponding bit set. Given that bitmaps of graphs are typically sparse, the bitmaps are compressed, and hence are more compact than traditional adjacency matrices.

3.1 Sparksee Structures

The basic logical data structure in Sparksee is a labeled and directed attributed multigraph. In this system, nodes and edges are uniquely identified by a set of ids separated into two disjoint domains (oids and eids), and the whole graph is built using a combination of two different types of structures: bitmaps and maps.

A bitmap or bit-vector is a variable-length sequence of presence bits that denotes which objects are selected or related to other objects. They are essential for speeding-up the query execution and reducing the amount of space required to store and manipulate the graph. In a bitmap, each bit is only set to 1 if the corresponding oid is selected. The first bit in a bitmap is always considered to be in position 1 (the first valid oid) and the last one is the last bit set (the highest oid considered in the bitmap). In order to know the length of the bitmap, the number of actual bits set to 1 in the structure is kept updated. The main advantage of this structure is that it is very easy to manage, operate, compress, iterate, etc.

A map is an inverted index with key values associated to bitmaps, and it is used as an auxiliary structure to complement bitmaps, providing full indexed access to all the data stored in the graph.

[6] Available at `http://www.sparsity-technologies.com/`

These two types of structures are combined to build a more complex one: the link. A link is a binary association between unique identifiers and data values. It provides two basic functionalities: given an identifier, it returns the value; and given a value, it returns all the identifiers associated to it.

3.2 Graph Representation Using Bitmaps

A Sparksee graph is built using a combination of links, maps and bitmaps to provide a logical view of a labeled and directed attributed multigraph.: each node or edge type has a bitmap which contains the oids of all the objects (nodes or edges) that belong to the type; each attribute of a type is a link; and, finally, the edges are decomposed into two different links, one for the tails, where for each node contains all the edges outgoing connected to it, when this node acts as their tail or origin, and in the same way another one for the heads which contains the ingoing edges. Thus, an edge is represented as a double join, one between the tail and the edge, and the other one between the head and the edge. If the edge is undirected, then both nodes of the edge are set as tails and heads for the edge.

All Sparksee graphs are built as a collection of bitmaps: one for each type to store the objects in the database, one for each distinct value of each attribute, one for each node that is the tail of one or more edges, and finally one for each node that is the head of one or more edges. With these bitmaps, solving distinct operations such as selecting all the objects of a type, retrieving the objects that have a specific value for an attribute or finding the number of edges or degree of a node, becomes straightforward.

4 Limitations of Graph Databases

Though graph databases offer a very rich data model and, as illustrated in Section 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.
- **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 that a database supports 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.

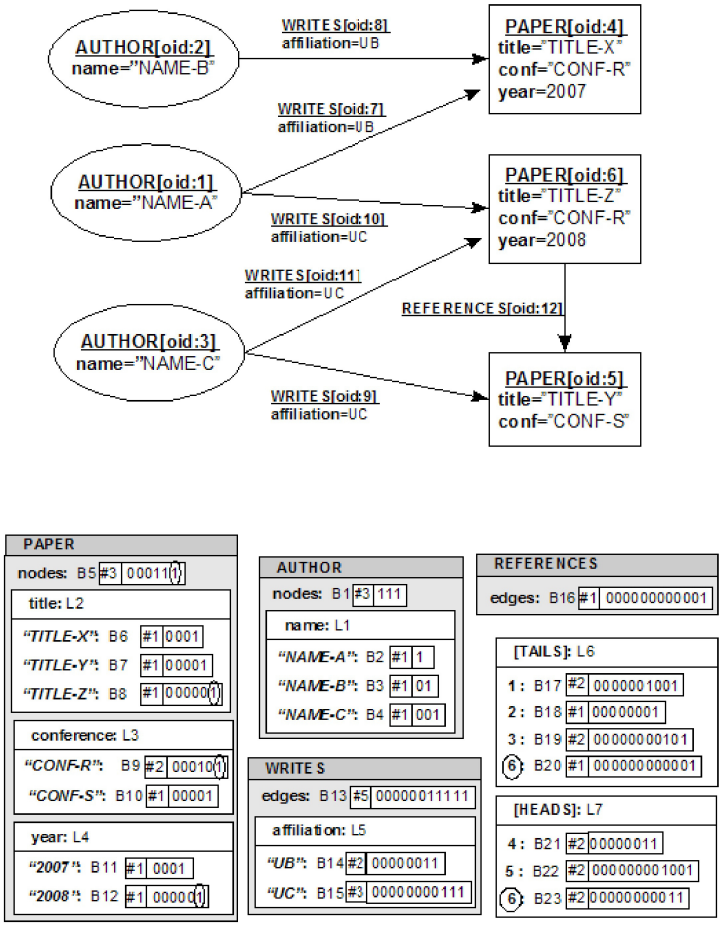

Fig. 1. Sample representation of a graph in Sparksee

Graph analytics frameworks [15–19] - 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 is essential for supporting horizontal scalability, too.
There are many reasons for this [20] , 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 has a significant amount 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 storage device (either a hard drive, a RAID or any distributed storage).
- **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 avoid it entirely.

4.1 Sparksee Example

Figure 1 shows an example of a graph extracted from a bibliographic data source (upper side), and the mapping of the previous graph into the internal structures as defined above (lower side).

The graph contains four object types: author (ovals) and paper (boxes) nodes, writes and references edges. These types are represented in the gray boxes at the right, where each type has one bitmap with its collection of objects, represented in the lighter gray frame, and their inner boxes represent the attributes, with one link each. Bitmaps are variable length sequences of 0's and 1's prefixed with the number of bits set. Links have the name, the collection of distinct values and the bitmap for each value. Maps are hidden in this representation because they are only used to index collections, for example the types or the attribute values.

For example, if we look at the node type paper, we can see that there are 3 bits set in bitmap B5, one for each node. There is also a bitmap for each distinct attribute value (L1 to L5 in the example) which indicates the oids of the objects containing this value in the attribute. If an object does not have any value then it will not appear in any bitmap of the attribute. Thus, the union of all the bitmaps of all the values of an attribute is equal to or a subset of the bitmap of the objects of the type. For example, $(B6 \cup B7 \cup B8) = B5$ in attribute title of paper, $(B11 \cup B12) \subset B5$ but because node 5 has no value for the attribute year.

There are two extra links at the rightmost side: one for the tails and the other for the heads. Each one has one value for each node that has edges, with its corresponding bitmap containing the edges where the node is connected. Again, the union of all the bitmaps of each of these links is equal to the union of all the collections of edge types, because all edges have one tail and one head. We can verify that $(B17 \cup B18 \cup B19 \cup B20) = (B21 \cup B22 \cup B23) = (B13 \cup B16)$.

As an example of the meaning of the structures, in the bitmaps we have marked all the occurrences of the oid 6, which identifies the node of the PAPER with title 'TITLE-Z'. These are the value '6' in L6 and L7, and the bit '6' in bitmaps B5, B8, B9 and B12. Note that B5 tells us it is a node PAPER; and B8, B9 and B12 show which are the values for the attributes of this node (title, conference and year respectively). Finally, L6 has the edges where this node is the tail, and L7 which are the edges where it is the head.

As we can see, with these structures now it is very easy to define graph-based operations just by combining one or more bitmap and map operations. For example:

- Number of authors: $|B1| = 3$
- Papers in conference 'CONF-R' of year 2007: $B9 \cap B11 = 4$.
- In-degree of paper 'TITLE-Y': $|B22| = 2$

In conclusion, this representation presents some advantages inherent to the structures and others more subtle that appear as a consequence of how the structures are being used. For example, the use of bitmaps directly provides some statistics without extra penalties, like the number of objects for each type, or the number of objects that have the same value for an attribute, the equivalent of a clustering or a GROUP BY / COUNT(*) operation of the relational model. The out-degree and in-degree of nodes are also the count of a bitmap stored into the tails or heads collections respectively. Also, the capability to add or remove attributes becomes easier because they are independent from the object storage. This is crucial for graph mining algorithms that typically require the creation of temporary attributes like weights or distances.

5 Use Case 1: 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.

5.1 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, counterterrorism 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 services to users or even to provide more profit to the information providers in the form of direct advertising or personalized services.

5.2 Graph Model

As we have seen before, there are many different kinds 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.

5.3 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 [21–23]. In addition to that, in [3], 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. An efficient algorithm to locate communities is available in [24].
- **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 [3]).
- **Degree distribution:** The degree distribution of many social networks obey a power law of the form $f(d) \propto d^{\beta}$, with the exponent $\beta < 0$, 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 [3], 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. [11] showed that not only the diameter of real social graphs is small, but it also shrinks and then stabilizes along

time [11]. Briefly, at the beginning of time, the network 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 [3], discover a mean value of β around 1.12.

5.4 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 are also 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 operation 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 business 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 behavior, predict the evolution or to identify the shape in order to split a very large graph in smaller units, easier to manage. Pattern matching is often used to extract groups of nodes and edges that match a specific pattern, for example for marketing purposes, data cleansing or integrity validation.

6 Use Case 2: 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.

6.1 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[7] 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 analyzed and information can be extracted, interpreted and summarized in the form of added value knowledge and linked relationships among

[7] 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.

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.

6.2 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 [25]. 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 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 [26]. 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 [27]. Such rules are not a static set, and can change over time in order to increase the flexibility of the system [28]. 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 [29]. 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 [30], or helped with the aid of a training set [31]. Workflows allow users to define complex data flows where combinations of matchers, conditions and loops [32]. 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

6.3 Graph Analytics

Once the datasets have been integrated inside a single graph. the goal is to provide a set of techniques to analyze 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 operations. 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 relevant nodes.

7 Graph Database Benchmarking

Early efforts: Popular database benchmarks, such as TPC-C or TPC-H [33], focus on evaluating relational database queries that are typical of a business application. These benchmarks emphasize queries with joins, projections, selections, aggregations and sorting operations. However, since Graph Databases aim at different types of queries, these widespread benchmarks are not adequate for evaluating their performance. Graph use cases often involve recursive steps, e.g. graph neighborhoods within n steps or even an undetermined number of steps. Graph queries may involve structural similarity, e.g. comparing structures of chemical compounds for similarity with a similarity score quantifying the deviations. Graph analytics often produce large intermediate results with complex structure, e.g. edge weights or iteratively calculated ranks (e.g. Page rank). Widespread relational benchmarks do not contain such operations. All those are operations that, in some cases, are difficult to imagine in RDBMSs and that find a good alliance in the RDF area and GDB area since the former adhere to a graph data model.

Object oriented databases (OODB) share some similarities with GDBs. The data of an OODB also conforms a graph structure, where the entities that are represented as objects draw [34] relationships among them. The OO1 benchmark, one of the earliest proposals, is a very simple benchmark that emphasizes three basic operations for OODB: (a) lookup, which finds the set of objects for a given object identifier; (b) traversal, which performs a 7-hop operation starting from a random node; and (c) insertion, which adds a set of objects and relations to the database. OO1 defines a dataset that only contains one type of objects with a fixed number of outgoing edges per object. Since the links mostly go to objects with a similar document identifier, the graphs are very regular. Another popular benchmark for OODB is the OO7 [35] proposed by Carey et al. In OO7, the database contains three types of objects, which are organized as a tree of depth seven. The connectivity of the database is also very regular because objects have a fixed number of relations. The benchmark is made up by a rich set of queries that can be clustered into two groups: (a) traversal queries, which scan one type of objects and then access the nodes connected to them in the tree, and (b) general queries, which mainly perform selections of objects according to certain characteristics.

Graph benchmarking: The graphanalysis.org initiative started a project to evaluate graph performance. After some preliminary benchmark proposals, which refined the queries in the system, the project released the final version of the benchmark as the "HPC Scalable Graph Analysis Benchmark v1.0[36]. The benchmark is compound by four separated operations on a graph that follows a power law distribution generated with the R-MAT generator [37]: (a) insert the graph database as a bulk load; (b) retrieve the set of edges with maximum weight; (c) perform a k-hops operation; and (d) calculate the betweenness centrality of a graph, whose performance is measured as the number of edges traversed per second. However, this benchmark does not evaluate some features expected from

a GDB such as object labeling or attribute management. In [38], this benchmark is evaluated on four representative graph data management alternatives (Neo4j, DEX, Jena and HypergraphDB) giving some insights about the strengths and weakness of each system. A recent survey has [39] reviewed some of the main operations and uses cases of graph databases, and thus, is a good starting point for the development of graph benchmarks. Other open source initiatives have proposed simple benchmarks to evaluate the performance of graph databases. For instance, Ciglan published a set of traversal oriented queries [40], or Tinkerpop initiated a project (currently stopped) to build a framework for running benchmarks on graph databases [41]. Nevertheless, these initiatives lack a wide acceptance because of their individual approach and limited resources.

Graphs in supercomputers: The performance of supercomputers has been traditionally tested using the Linpack benchmark, which is derived from the Linpack library that computes linear algebra operations. According to the Linpack results, a list of the top 500 computers is published biannually, which determines the most powerful computers in the world. Nevertheless, the use of supercomputers has spread from computationally intensive integer and floating point computation, to memory intensive applications. For such applications, the Linpack is not a good reference and other evaluation methods have been proposed, including graph related computation. Since 2010 an alternative top 500 list is published using the traversed edges per second of a Breadth First Search in a graph [42].

Linked Data Benchmark Council (LDBC): LDBC is a EU funded project that is creating a non profit organization similar to TPC, which will design and support graph database and RDF benchmarks. LDBC benchmarks are innovative because: (i) they will be based on real use cases, and thus be meaningful for users to fairly compare graph databases; (ii) they will motivate graph database vendors to innovate in the development of graph databases to improve its performance and scalability; (iii) they will compile a repository of supporting knowledge for the area of graph database benchmarks that will be used as a reference in the design of benchmarks in this field; and, (iv) they will generate benchmark expertises and rules of fair practice for carrying out and auditing the benchmark of database instances by vendors. The first set of LDBC benchmarks will be published in 2014.

8 Conclusions

We have been describing important aspects of graph charateristics, database implementation, use cases and benchmarking. There is no doubt about the fact that many other aspects concur in the graph area, i.e. graphical representation of large and small graphs, use of graphs in complex systems analysis, etc. However, the objective of this paper was to give a broad overview of the knowledge behind graph management and the technologies around graphs. In the course, we will provide a similar overview with a set of slides provided to the students and the general public through the course web page and through DAMA-UPC web page (`www.dama.upc.edu`).

References

1. Leskovec, J., Huttenlocher, D.P., Kleinberg, J.M.: Signed networks in social media. In: CHI, pp. 1361–1370 (2010)
2. Goertzel, B.: OpenCogPrime: A cognitive synergy based architecture for artificial general intelligence. In: IEEE ICCI, pp. 60–68 (2009)
3. Newman, M.: Networks: An Introduction. Oxford University Press, Inc., New York (2010)
4. Levene, M., Poulovassilis, A.: The hypernode model: A graph-theoretic approach to integrating data and computation. In: FMLDO, pp. 55–77 (1989)
5. Ërdos, P., Rényi, A.: On random graphs. Mathematicae 6, 290–297 (1959)
6. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Statistical properties of community structure in large social and information networks. In: WWW, pp. 695–704 (2008)
7. Flickr Blog: Six billion (retrieved on march 2014), `http://blog.flickr.net/en/2011/08/04/6000000000/`
8. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM, pp. 251–262 (1999)
9. McGlohon, M., Akoglu, L., Faloutsos, C.: Weighted graphs and disconnected components: patterns and a generator. In: KDD, pp. 524–532 (2008)
10. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. ACM Comput. Surv. 38 (2006)
11. Leskovec, J., Kleinberg, J.M., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. TKDD 1 (2007)
12. SNAP: (Stanford large network dataset collection), `http://snap.stanford.edu/data/index.html`
13. Martínez-Bazan, N., Muntés-Mulero, V., Gómez-Villamor, S., Nin, J., Sánchez-Martínez, M.-A., Larriba-Pey, J.-L.: Dex: high-performance exploration on large graphs for information retrieval. In: CIKM, pp. 573–582 (2007)
14. Martínez-Bazan, N., Aguila-Lorente, M.A., Muntés-Mulero, V., Dominguez-Sal, D., Gómez-Villamor, S., Larriba-Pey, J.-L.: Efficient graph management based on bitmap indices. In: IDEAS, pp. 110–119 (2012)
15. Nelson, J., Myers, B., Hunter, A.H., Briggs, P., Ceze, L., Ebeling, C., Grossman, D., Kahan, S., Oskin, M.: Crunching large graphs with commodity processors. In: HotPar (2011)
16. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: SIGMOD, pp. 135–146 (2010)
17. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: Distributed graph-parallel computation on natural graphs. In: OSDI, pp. 17–30 (2012)
18. Stutz, P., Bernstein, A., Cohen, W.: Signal/Collect: Graph algorithms for the (Semantic) web. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 764–780. Springer, Heidelberg (2010)
19. Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., Zadeh, R.: Wtf: The who to follow service at twitter. In: WWW, pp. 505–514 (2013)
20. Averbuch, A., Neumann, M.: Partitioning graph databases-a quantitative evaluation. arXiv preprint arXiv:1301.5121 (2013)
21. Flake, G.W., Lawrence, S., Giles, C.L., Coetzee, F.: Self-organization and identification of web communities. IEEE Computer 35(3), 66–71 (2002)

22. Girvan, M., Newman, M.: Community structure in social and biological networks. National Academy of Sciences 99(12), 7821–7826 (2002)
23. Schwartz, M., Wood, D.: Discovering shared interests among people using graph analysis of global electronic mail traffic. Communications of the ACM 36, 78–89 (1992)
24. Prat-Pérez, A., Dominguez-Sal, D., Larriba-Pey, J.-L.: High quality, scalable and parallel community detection for large real graphs. In: To be published in WWW (2014)
25. Bleiholder, J., Naumann, F.: Data fusion. ACM Computing Surveys (CSUR) 41, 1 (2008)
26. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. on Knowledge and Data Engineering 24, 1537–1555 (2012)
27. Arasu, A., Ré, C., Suciu, D.: Large-scale deduplication with constraints using dedupalog. In: ICDE, pp. 952–963 (2009)
28. Whang, S.E., Garcia-Molina, H.: Entity resolution with evolving rules. PVLDB 3, 1326–1337 (2010)
29. Whang, S.E., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with negative rules. VLDB Journal 18, 1261–1277 (2009)
30. Leitão, L., Calado, P., Weis, M.: Structure-based inference of xml similarity for fuzzy duplicate detection. In: CIKM, pp. 293–302 (2007)
31. Rastogi, V., Dalvi, N., Garofalakis, M.: Large-scale collective entity matching. PVLDB 4, 208–218 (2011)
32. Thor, A., Rahm, E.: MOMA - A Mapping-based Object Matching System. In: CIDR, pp. 247–258 (2007)
33. Transaction Processing Performance Council (TPC): TPC benchmark website, http://www.tpc.org
34. Cattell, R., Skeen, J.: Object operations benchmark. ACM Trans. Database Syst. 17, 1–31 (1992)
35. Carey, M.J., DeWitt, D.J., Naughton, J.F.: The oo7 benchmark. In: SIGMOD Conference, pp. 12–21 (1993)
36. Bader, D., Feo, J., Gilbert, J., Kepner, J., Koetser, D., Loh, E., Madduri, K., Mann, B., Meuse, T., Robinson, E.: HPC Scalable Graph Analysis Benchmark v1.0. HPC Graph Analysis (2009)
37. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: A recursive model for graph mining. In: SDM, pp. 442–446 (2004)
38. Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vañó, A., Gómez-Villamor, S., Martínez-Bazan, N., Larriba-Pey, J.-L.: Survey of graph database performance on the hpc scalable graph analysis benchmark. In: WAIM Workshops, pp. 37–48 (2010)
39. Dominguez-Sal, D., Martinez-Bazan, N., Muntes-Mulero, V., Baleta, P., Larriba-Pey, J.L.: A discussion on the design of graph database benchmarks. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 25–40. Springer, Heidelberg (2011)
40. Ciglan, M., Averbuch, A., Hluchý, L.: Benchmarking traversal operations over graph databases. In: ICDE Workshops, pp. 186–189 (2012)
41. Tinkerpop: Open source property graph software stack, http://www.tinkerpop.com
42. Graph 500 Website: The graph 500 list, http://www.graph500.org/