A Discussion on the Design of Graph Database Benchmarks*

David Dominguez-Sal[1], Norbert Martinez-Bazan[1], Victor Muntes-Mulero[1], Pere Baleta[2], and Josep Lluis Larriba-Pey[1]

[1] DAMA-UPC, Barcelona, Spain,
{ddomings,nmartine,vmuntes,larri}@ac.upc.edu
[2] Sparsity Technologies, Barcelona, Spain
pbaleta@spartisty-technologies.com

Abstract. Graph Database Management systems (GDBs) are gaining popularity. They are used to analyze huge graph datasets that are naturally appearing in many application areas to model interrelated data. The objective of this paper is to raise a new topic of discussion in the benchmarking community and allow practitioners having a set of basic guidelines for GDB benchmarking. We strongly believe that GDBs will become an important player in the market field of data analysis, and with that, their performance and capabilities will also become important. For this reason, we discuss those aspects that are important from our perspective, i.e. the characteristics of the graphs to be included in the benchmark, the characteristics of the queries that are important in graph analysis applications and the evaluation workbench.

1 Introduction

The analysis and storage of data in the form of a graph has increased in the recent years. Analyzing the characteristics of social networks, the use of the Internet, or the interactions among proteins has put graph processing in the eye of the storm. The amount of data managed in most of those cases is huge, and the complexity of the algorithms needed for the analysis as well, leading to a clear need in the market: the Graph Database Management System (GDB).

A GDB is a step forward in the management and analysis of data. As stated by Angles and Gutierrez [1]: *"Graph database models can be defined as those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors"*. Graph databases emphasize the queries that compute results related to the structure of the links in the graphs rather than on the entities themselves: for example detecting link patterns, path analysis, authority relations, etc. However, managing large graphs is a complex issue, and obtaining the best suited analysis algorithms is difficult.

There are a certainly growing number of initiatives to implement and commercialize GDBs, like Neo4j [2], HyperGraphDB [3], Infogrid [4] or DEX [5] and many RDF solutions such as Jena [6] or AllegroGraph [7]. There are other initiatives to create graph querying languages that allow for a simplified view of

* The members of DAMA-UPC thank the Ministry of Science and Innovation of Spain and Generalitat de Catalunya, for grant numbers TIN2009-14560-C03-03 and GRC-1087 respectively.

querying to the user like SPARQL [8] and Gremlin [9]. This shows that the community is very active proposing new technologies, and sets an adequate scenario to reflect on which is the adequate benchmark for a GDB.

The main objective of this paper is to open the discussion on GDB benchmarking. Thus, we describe and discuss important aspects to be considered for benchmarking. We started describing and analyzing the type of applications where it is necessary the use of GDBs. In particular, we review the application of GDBs in Social Network Analysis (SNA), proteomics, recommendation systems, travel planning and routing, which gives a catalog of representative areas where huge graph datasets are appearing.

We believe that the set of applications mentioned are representative of the marketplace for GDBs. Thus, based on those, we discuss the characteristics of the graphs that appear in such applications and how they could influence benchmarking. We also survey different types of operations, why they are important and how they can be categorized in order to produce a wide coverage of issues within a benchmark. Finally, we discuss on the evaluation setup of the benchmark, where issues like the experimental process to follow and the type of measures to be taken are considered. Despite the diversity of applications, we find that the different areas have common features, and we believe that the design of a benchmark based on SNA would become a representative candidate of general GDB applications.

The paper is organized as follows. We start by setting up the state of the art in Section 2. Then, in Section 3, we analyze a broad spectrum of applications demanding for massive graph management. From these scenarios, we extract generic characteristics of graphs and queries, that will be important in order to design a graph database benchmark. Also, we propose a query categorization and we remark the relevant aspects that we should take into account for the experimental settings of a benchmark. Finally, we draw some conclusions.

2 Graph oriented benchmarks

Popular database benchmarks, such as TPC-C or TPC-H [10], 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 GDBs aim at different types of queries, these widespread benchmarks are not adequate for evaluating their performance.

Object oriented databases (OODB) share some similarities with GDBs. The data of a OODB also conforms a graph structure, where the entities that are represented as objects draw relationships among them. The OO1 benchmark [11], 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 proposed by Carey et al [12]. 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.

We observe that although OODB benchmarks create graphs, the graphs have a very different structure from typical graphs in graph analysis applications. As we review in detail in Section 3.2, graphs in GDBs are very irregular: the degree of the nodes exhibit a large variance, nodes are clustered in communities and graphs have small diameters. Furthermore, the applications that interact with GDBs are mainly interested in analyzing the graph structure, i.e. the relationships, instead of the attributes in the objects. For example, operations such as finding the shortest path connecting two objects or finding patterns (e.g. a clique) are common GDB operations that are not considered in OODB.

XML databases also follow a model which relates entities. An XML database is a collection of typed data and attributes organized as a tree. One of the most well known benchmarks for XML databases is XMARK [13], which models an auction site. The queries in the benchmark cover many aspects: selections, sorted access, tree path location, aggregation, etc. Nevertheless, XML only models trees, which are a limited subclass of graphs.

In the recent years, the knowledge management community has made efforts to design a standard for representing the relations between metadata elements, which has derived in the introduction of the Resource Description Framework (RDF). Along with RDF, the community has designed a query language called SPARQL, which is an SQL extension to describe relationship patterns among entities. In order to test the performance of these knowledge bases, Guo et al. proposed a benchmark of 14 SPARQL queries in [14], which is known as LUBM.

To our knowledge, the only benchmark proposed for the evaluation of graph libraries is the HPC Scalable Graph Analysis Benchmark v1.0 [15]. The benchmark is compound by four separated operations on a graph that follows a power law distribution: (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. For an implementation and a discussion of the benchmark over different graph databases see [16].

3 Benchmark considerations

In this section, we discuss several aspects that may affect the design of a GDB benchmark. First, we examine a set of motivating scenarios where graph databases are useful. Second, we describe the most commonly used graph types. From this, we explore common queries run in these scenarios and propose a categorization depending on their interaction with the graph data. Finally, we review some experimental considerations for the design of GDB benchmarks.

3.1 Analysis of Motivating Applications

Over the past years, there has been an increasing interest in multiple disciplines on datasets that can be represented and analyzed as a network or graph. In these networks, the nodes represent the entities and the edges the interaction or relationships between them. For example, the use of Social Network Analysis (SNA) has grown to currently become one of the most successful tools to investigate the structure of organizations and social groups, focusing on uncovering the structure of the interactions between people [17]. 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. The Web 2.0 [18] and the increasing use of Internet applications that facilitate interactive collaboration and information sharing has caused the appearance of many social networks of different kinds, 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.

The rapid growth of the World Wide Web, has caused new graph structured data to be archived and analyzed, such as hypertext and semi-structured data [19]. Also, with RDF, users are allowed to explicitly describe semantic resources in the form of graphs [20]. In this context, and others such as finding web service's connection patterns, algorithms and applications for graph and subgraph matching in data graphs are becoming increasingly important [21]. Pattern matching algorithms are also used to find relationships in social networks [22], find research collaboration partners, or mine the connections among research paper publications in archived bibliography datasets.

The use of graph theory combined with computing analysis has attracted the interest of the graph mining community [23]. Some of the classical analysis in this area are the determination of an actor's centrality, to identify key players, leaders or relevant people in a community; the grouping of individuals in communities or affinity groups, to provide specific services or to improve their connectivity; the identification of weaknesses or singular points in the network, for security or counter-terrorism; or the study of social roles and collaborations, to get the position of an actor in the society or to connect people to others to improve their relationships network in professional environments.

Another area of interest is in proteomics or genetic interactions, where the large-scale study of proteins is considered the next step in the study of biological systems. The fact that most proteins act in collaboration with other proteins is the basis for proteomics to reveal which of those are potentially associated with a certain disease. In protein-protein interaction networks, nodes represent proteins, and edges between nodes represent physical interactions between proteins. Several protein interaction databases are shared between hundreds of research laboratories and companies around the world, such as BioGRID [24] or the Protein Data Bank (PDB) [25]. With this data, some of the most usual analysis in genetic interactions are the study of the complexity of the network topology, the node analysis of centrality and role, the identification of articulation points or bridges between groups of proteins, or the clustering of proteins based on their relationship or neighborhoods. Many scientific software tools are available for the analysis and the visualization of data, like Navigator [26] for network analysis, visualization and graphing.

Network analysis has also become essential for recommendation systems. In this case, the goal is to present information that could be interesting to the users based on previous knowledge extracted from their social environment. Recommendation systems are prior to computers, but the Internet has exploded again the use of recommendation for different purposes, such as on-line sales and catalogs like Amazon, or digital music services in iTune. Even PageRank [27] in Google can be considered a recommendation engine, and the analysis of hubs and authorities to rate Web pages in HITS [28] is an exploration of the network of relationships of a web page with its hyperlinks. Although relational databases have been the storage system of choice in many commercial recommendation engines for collaborative filtering, like Strands [29] for social networks and eCommerce, lately, new approaches have appeared using graph representation and exploration like conceptual graphs [30] or more recently DirectedEdge [31]. Another area of network analysis where graphs may be large is travel planning and routing, where the system tries to find the most efficient path between two points according to some constraints or recommendations given by the user, like in Amadeus [32], or real time analysis of traffic networks in large cities. In these cases, the data is naturally represented as graphs where vertices stand for the location and the edges are the routes with lengths and costs. Then, queries are mainly navigational operations between neighbors or network topology analysis.

3.2 Graph Description

GDBs store entities as nodes, which have relations among them that are set as edges. However, depending on the particular application, the graph model may differ, showing a different degree of complexity. In addition to the plain storage of nodes and edges, we detail the main features required by some applications:

Attributes: In addition to nodes and edges, graph databases store information associated to these nodes and edges. The associated information is typically string or numerical values, which indicate the features of the entity or relation. 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 set a unique identifier for each node and edge of the graph that labels each graph element.

Directed: Depending on the problems the relation between two nodes may be symmetric or not. If the relation is symmetric, the edge might be traversed from any of the adjacent nodes to the opposite one. If the relation is not symmetric, edges differentiate 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. Undirected graphs are a particular case of directed graphs, since an undirected edge can be represented as two directed edges, each one in a reverse direction of the other.

Node and edge labeling: Some applications differentiate different labels (or types) of nodes and edges. Labeling has an important impact because some applications require distinguishing between different kinds of relations. For example, a social network may accept either "positive" or "negative" friendship relations [33].

Multigraphs: Multigraphs differ from graphs in that two nodes can be connected by multiple edges. Multigraphs appear commonly when graphs have typed edges because often two nodes are related by different categories. For example, in a mobile telephone network that represents the cell phones as the nodes and the telephone calls as the edges, two nodes will have multiple connections if they have called more than once.

Hypergraphs: A hypergraph is a generalization of the concept of a graph, in which the edges are substituted by hyperedges. In contrast to regular edges, an hyperedge connects an arbitrary number of nodes. Hypergraphs are used, for example, for building artificial intelligence models [34].

Graph Characterization: Real graphs are typically very different from graphs following the Erdös-Renyi model (random graphs) [35]. Leskovec et al. analyzed over 100 real-world networks in [36] in 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 these networks varies 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 huge, the graph data sets of some current applications are significantly larger: for example Flickr accounts more than 4 billion photographs that can be tagged and rated [37], 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 [36, 38, 39], 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 huge graphs [23]: (a) they contain a large connected component that accounts for a significant percentage of the nodes; (b) they are sparse, which means that the number of edges is far away from the maximum number of edges; (c) the degree distribution follows a power law distribution (i.e scale-free networks), where a few nodes have a number of connections that greatly exceeds the average, usually known as hubs; (d) the average diameter of each connected components from the graph is small, in other words, from a given node there is a short path to reach the majority of the remaining nodes in the connected component (which is also called the small world property); and (e) the nodes are grouped in communities where the density of edges among the members of the community is larger than the edges going outside the community.

Discussion: In this section, we see that graph applications represent their datasets following graphs with different degrees of complexity. Nevertheless, we observe that the structure of the graphs datasets follow power law characterizations and properties, which makes it possible to create generic graphs representative of multiple applications for benchmarking purposes.

According to the previously described applications, we also identify three aspects that a generic GDB should be able to compute (and thus be included in a benchmark): (a) labeled graphs, which enable to identify the nodes and edges of the graph; (b) directed graphs, which set the relation and its direction; (c) attribute support, which are used by applications such as for setting the weight of the edges.

3.3 Graph Operations

In this subsection, we present several types of operations used in the areas presented before. The analysis of this section will be useful to learn different aspects that will be used to fix criteria, in the following subsection, to design relevant operations for a future benchmark. Table 1 lists some of these graph operations, organized by the type of access that is performed on the graph.

First, we define a set of generic operations. These operations are not typical in a single specific domain, but common operations that may be necessary in any context. This set of operations allows us to (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. Any complex query or transformation of the graph will necessarily use these operations.

Afterwards, we extend these operations to other higher level actions typically performed in the scenarios presented before. We group these operations into different types:

Traversals. Traversals are operations that start from a single node and explore recursively the neighborhood until a final condition, such as the depth or visiting a target node, is reached. For instance, we consider the operation of calculating a *shortest path*, 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 attributes, as in the case of finding the shortest route from two

Group	**Operation**	**Social Network**	**Protein Interaction**	**Recom-mendation**	**Routing**	**Analytical**	**Cascaded**	**Scale**	**Attr.**	**Result**
Generic operations										
General Atomic / Local Information Extraction	Get node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get attribute of node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get neighborhood	+	+	+	+	Yes	No	Neigh.	No	Set
	Node degree	+	+	+	+	Yes	No	Neigh.	No	Agr.
General Atomic Transformations	Add/Delete node/edge	+	+	+	+	No	No	Neigh.	No	Set
	Add/Delete/Update attrib.	+	+	+	+	No	No	Neigh.	E/N	Set
Application dependent operations										
Traversals	(Constrained) Shortest Path	+	+		+	Yes	Yes	Glob.	Edge	Graph
	k-hops	+		+	+	Yes	Yes	G/N	No	Graph
Graph Analysis	Hop-Plot	+				Yes	No	Glob.	No	Agr.
	Diameter	+	+			Yes	Yes	Glob.	Edge	Set
	Eccentricity	+				Yes	Yes	Glob.	Edge	Agr.
	Density	+	+			Yes	No	Glob.	No	Agr.
	Clustering coefficient	+				Yes	Yes	Glob.	No	Agr.
Components	Connected Components	+	+			Yes	Yes	Glob.	No	Graph
	Bridges	+	+		+	Yes	Yes	Glob.	No	Set
	Cohesion	+				Yes	Yes	Glob.	No	Set
Communities	Dendrogram	+				Yes	Yes	Glob.	No	Graph
	Max-flow min-cut	+				Yes	Yes	Glob.	Edge	Graph
	Clustering	+	+	+		Yes	Yes	Glob.	No	Graph
Centrality Measures	Degree Centrality	+		+		Yes	No	Glob.	No	Set
	Closeness Centrality	+		+		Yes	Yes	Glob.	No	Set
	Betweenness Centrality	+		+		Yes	Yes	Glob.	No	Set
Pattern Matching	Graph/Subgraph Matching	+	+			Yes	Yes	Neigh.	No	Graph
Graph Anonymization	k-degree Anonym.	+		+		Yes	No	Glob.	No	Graph
	k-neighborhood Anonym.	+		+		Yes	Yes	Glob.	No	Graph
Other Operations (Similarity, ranking,...)	Structural Equivalence	+				Yes	Yes	Glob.	No	Graph
	PageRank			+		Yes	No	Glob.	Node	Set

Table 1. Graph Operations, Areas of Interest and Categorization

points, avoiding a certain type of road, for instance. This operation is also used as a measure to calculate the information loss of graph anonymization methods. Another typical operations is calculating *k-hops*, which returns all the nodes that are at a distance of k edges from the root node. A particular case is when $k = 1$, also known as the *neighbors* of the node. The particular case of *1-hops* is widely used as part of other operations. For example to calculate the *nearest neighborhood* in recommender systems, to obtain a particular user's neighborhood with similar interest, or in web *ranking* using hubs and authorities.

Graph Analysis. Basic graph analysis includes the study of the topology of graphs to analyze their complexity and to characterize graph objects. It is basically used 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 analysis is the first step of the analytical process and it is widely used in SNA and protein interaction analysis. Among this we may calculate the *hop-plot* (a metric to measure the rate of increase of the neighborhood depending on the distance to a source node), the (*effective*) *diameter*, the *density*, or the *clustering coefficient* (to measure the degree of transitivity of the graph), to give some examples.

Components. A *connected component* is a subset of the nodes of the graph where there exists a path between any pair of nodes. Thus, a node only belongs to a single connected component of the graph. Finding connected components is usually crucial in many operations, typically used in a preprocess phase. Also, some operations are helpful to study the vulnerability of a graph, or the probability to separate a connected component into two other components. Finding *bridges*, edges whose removal would imply separating a connected component, is important in many applications. Going further, the *cohesion* of the graph can be computed by finding the minimum number of nodes that disconnect the group if removed.

Communities. A community is generally considered to be a set of nodes where each node is closer to the other nodes within the community than to nodes outside it. This effect has been found in many real-world graphs, especially social networks. Operations related to the creation of a community may be building *dendograms* (communities formed through hierarchical clustering), finding the *minimal-cut* set of edges or other *clustering* techniques.

Centrality Measures. A centrality measure aims at giving a rough indication of the social importance of a node based on how well this node connects the network. The most well-known centrality measures are *degree*, *closeness* and *betweenness centrality*.

Pattern matching. Pattern recognition deals with algorithms which aim at recognizing or describing input patterns. *Graph matchings* are usually categorized into exact or approximate. Exact matchings may include finding homomorphisms or (subgraph) isomorphisms. Approximate matchings may include error-correcting (subgraph) isomorphisms, distance-based matching, etc.

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 and the problem goes beyond the anonymization of the pure graph structure. 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 in Subsection 3.1. For instance, finding similarity between nodes in a graph has shown to be very important in SNA. 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*).

Discussion: 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. Moreover, according to Table 1, SNA provides one of the most rich set of graph operations, which makes social networks a candidate scenario for designing benchmarks that are representative of applications that use GDBs.

3.4 Query Categorization

The computational requirements of graph queries is not homogeneous. For example, some queries may traverse the full graph, while others may request the outdegree of a single node. In order to build a balanced benchmark it must be representative of the different types of operations that can be issued by an application to the graph database. In this section, we build up a set of categories to classify the different operations that are issued to a graph database:

Transformation/Analysis: 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, add/remove 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 to access to secondary storage because the graph or the temporary results generated during the query resolution are too large to fit in memory.

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 3-hops operation follows a cascaded pattern. Thus, a non cascaded operation may access a node, an edge or the neighbours of a node. Besides, an operation that does not request the neighbours 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.

Scale: We classify the queries depending on the number of nodes accessed. We distinguish two types of queries: global and neighbourhood 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 or the edges of the graph. The latter queries only access to a portion of the graph.

Attributes: 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.

Discussion: Queries in a benchmark must represent the workload of the real environment where the application is going to be executed, and thus should be adapted to the particular application profile to be tested. We have seen that graph operations are diverse, but many operations share similar operational patterns. In Table 1, we summarize these patterns categorizing the catalog of popular operations. In this table, we find that although the most basic operations are neither structured nor affect large fractions of the graph, many applications use large scale operations that traverse the graph. Furthermore, we find that most graph operations are accessing the information stored in the edges since the attributes in the edges (and weights in particular) are modeling the relations between entities, which is the main objective of a graph. We also observe that generic GDB must be able to store temporal objects because they are necessary for most operations (e.g: storing a boolean to decide whether a node has been visited or counting the number of paths through a node). Finally, we see that generic GDB must be able to manage different types of result sets because we find operations in many different applications that return sets, aggregates and graphs. We note that although most operations analyze the graph, many applications may store the results as attributes. In other words, although the results of a complex operation such as community detection are analytical, they may be stored as attributes that may be updated periodically.

In general, operations with a complexity larger than linear (over the number of nodes or edges) should be included with care in a GDB benchmark because they may become unfeasible to compute for large graphs. If these operations are very representative of a certain workload, then one possible approach is to accept approximate results. For example, the betweenness centrality measure is very appreciated in the analysis of social networks, but in practice it is seldom computed with an exact algorithm because of its high computational costs [40]. Therefore, if large graphs are considered, benchmarks may also consider approximated implementations though metrics about quality of the result and precise descriptions of the approximated algorithm are recommended.

3.5 Experimental setting

Experimental setup and measurement is one of the most important parts of a benchmark and it must be clearly defined and configured to allow a fair comparison between multiple solutions in different environments. Early database benchmarks only focused in the response time of the queries. As the benchmarks have evolved and platforms are more complex, the measurements have become more sophisticated and, in consequence, the experimental setup and the testing process is more expensive and complicated. For example, early benchmarks like Wisconsin only considered the response time using very simple scale factors [41]. Later, TPC-C or TPC-H introduced a metric with the relationship of the pricing with respect the maximum throughput [42], and more recently LUBM [14], for the benchmarking of RDF graphs, defined a combined metric between the query response time and the answer completeness and soundness in the case of partial results and pattern matching.

The different concepts related to the experimental setup and configuration of a graph database benchmark can be grouped in the following areas: configuration and setup, experimental process, and measures.

Configuration and setup: Modern benchmarks allow for the definition of the dataset size by means of a *scale factor*, which fixes the dataset size generated following precise data distributions. For graphs, the scale factor defines the number of nodes and edges. Additionally, the graph must conform to scale the graph structural properties for different scales such as the vertex degree and hop-plot distributions, the diameter or the community structure [43].

A second set of parameters defines the allowed capabilities to preprocess the dataset. Some of the most important factors to define are: (a) *data partitioning*, which usually allows for a flexible graph partitioning using either horizontal partitioning, (each structure partitioned), vertical partitioning (structures are grouped following some criteria), or hybrid; (b) *indexing*, that allows the GDB to build freely indexes that speedup queries at the expense of slower loads; (c) *redundancy*, that specifies if data may be replicated; and if (d) *data reorganization* procedures that optimize the internal data structures of the GDB are allowed. In order to have precise measures, benchmarks usually distinguish between two parts: (a) a load phase where the data is preprocessed, and (b) a query phase where the ingested data cannot be reindexed or reorganized.

Finally, another aspect is the definition of the data consistency requirements that the database must support during the benchmark. In transactional benchmarks, ACID properties are usually the standard. However, since many graph operations are analytical, more relaxed approaches can be taken, like eventual consistency or simply requiring concurrence for multiple readers.

Experimental process: The experimental process definition minimizes the side-effects produced by the way the experiments are executed, or by some influences that may appear between experiments, like the the processor caches or the execution order of the queries. (a) The *warm-up* sets the state of the computer before the benchmark is executed. It usually allows to populate the memory with a fraction of the graph (or indexes) as a result of some warm up queries. On the other hand, if the benchmarks aims at measuring the I/O overhead, it is necessary to ensure that the operating system's cache is emptied. (b) The *query generation* sets in which order the queries are executed (e.g. in sequential order). For non sequential benchmarks, it also sets the probability distribution that incoming queries fit. (c) The *observational sampling procedure* describes the number of executions for each query and how they are summarized. The observations are collected once a condition happens, which is typically once a query (or set of queries) finishes or a period of time lapses. Optionally, outliers or the fastest and slowest results might be discarded. It should be taken into consideration that when data is modified by a query it is also important to include into the measure the flush of the graph updates to disk.

Measures: Measures are closely related to the benchmark capabilities and to the audience. Some general measures that can be applied to graphs are the following: (a) the *load time*, which measures the elapsed time for loading and preprocess the dataset; (b) the *size* of the graph; (c) the *query response time* that accounts for the time elapsed between the query is issued until the results are output; (d) the *throughput* that measures the number of queries completed in an interval of time; (e) the *price* of the computing site including hardware, license and maintenance costs if applicable; or (f) the *power consumption* that measures the energy requirements of the computing equipment and gives an indirect measure of its cooling requirements. In order to compare such metrics among different platforms, it is common to introduce normalized metrics such as the *price/throughput* or the *power/throughput* that enable an easier comparison of benchmarking results between, for example, a supercomputer and a laptop hardware setup.

In the area of graph benchmarking, some specialized metrics have been proposed. The HPC benchmark [15] defines the *number of traversals per second* (TEPS) as the average number of edges explored during a traversal per second. The TEPS gives a measure of the average effort to navigate through the relationships with respect to the size of the graph. For pattern matching, when it is difficult to find all the results in a deterministic way like in knowledge bases, Guo et al. have proposed to measure the *query completeness* (or recall) or how far is the result respect to all the possible answers [14]. If the GDB is able to return results as soon as they appear, it might be interesting to obtain a plot of the GDB's recall with respect to time.

Discussion: There are many experimental settings and measures that can be selected for a GDB benchmark. The final selection depends on the benchmark characteristics, the goals and the audience. Thus, an industrial-oriented benchmark will probably focus on the throughput and cost of complex queries on very large graphs. However, research oriented benchmarking may be interested in the performance of more specific operations for different hardware configurations. In contrast to other scenarios where ACID properties are mandatory, we believe that many applications of GDB benchmarks allow for more relaxed consistency behaviors.

4 Conclusions

In this paper, we have analyzed important aspects for the design of a GDB benchmark. First of all, there is a significant core of applications that benefit significantly from their management as a graph, like social network analysis, protein interaction, recommendation and routing among others. These applications justify by themselves the existence and evolution of GDBs, and at the same time, justify the existence and evolution of a GDB benchmark. Its correct design and implementation implies the following aspects to be considered:

– The characteristics of the graph to be used in a benchmark are important. Considering the inclusion of directed graphs with attributes, with different node and edge types in the context of multigraphs would be important.
– The characteristics of the graph like the distribution of edges per node, attribute values per edge, etc. depend on the application and should be applied based on the different studies appeared in the literature. Nevertheless, most huge graph datasets follow power law distributions.
– Although not necessarily all the operations appearing in our analysis need to be considered for a benchmark, both analytical and transformation operations should be present.
– The cascaded nature of many graph operations is important, and a benchmark should include a good selection of operations with and without this cascaded nature.
– While there are operations that cover a traversal of all the database, others just affect a few of their components. Such feature should be evaluated taking into consideration the metrics to be used, in order to balance the importance of each case.
– Depending on the application, some operations just evaluate the structure of the graph, while others take the attributes in the nodes and specially in the edges, to be evaluated. A good combination of queries with both characteristics would be of interest for the proper deployment of a benchmark.
– The nature of the result is important because GDBs are capable of offering a good assortment of answers. In particular, operations returning sets of nodes, graphs or aggregational answers would be recommended for a balanced benchmark.
– There are other aspects that influence the fairness of a benchmark. Those are the configuration of the GDB (partitioning, indexing of attributes, data redundancy and reorganization), the way the experimental process is undertaken (warm-up of the database, query generation and the observational

procedure) and the metrics to be considered (load time, repository size, query response time, throughput obtained and the cost of the deployment). However, those aspects are not totally influenced by the GDB environment.

Just to finalize, it would be very important to find an application covering as many of the aspects that we have evaluated in this paper as possible. We believe that social network analysis is very significant because it covers almost all the operation types in graph databases, it is easy to understand by the final user and carries a lot of natural queries with answers that can be conceptually understandable. Any other application with such characteristics would be of use and beneficial for a GDB benchmark.

References

1. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1) (2008)
2. Neo4j: The neo database. Available at http://dist.neo4j.org/neo-technology-introduction.pdf (2006)
3. HypergraphDB: HypergraphDB website. http://www.kobrix.com/hgdb.jsp (Last retrieved in March 2010)
4. Infogrid: Blog. http://infogrid.org/blog/2010/03/operations-on-a-graph-database-part-4 (Last retrieved in March 2010)
5. N. Martínez-Bazan, V. Muntés-Mulero et al : Dex: high-performance exploration on large graphs for information retrieval. In: CIKM. (2007) 573–582
6. Jena-RDF: Jena documentation. http://jena.sourceforge.net/documentation.html (Last retrieved in March 2010)
7. AllegroGraph: AllegroGraph website. http://www.franz.com/agraph/ (Last retrieved in May 2010)
8. E. Prud'hommeaux and A. Seaborne: SPARQL Query Language for RDF. W3C. Available at http://www.w3.org/TR/rdf-sparql-query/ (2008)
9. Gremlin website: Gremlin documentation. http://wiki.github.com/tinkerpop/gremlin/ (Last retrieved in Jun 2010)
10. Transaction Processing Performance Council (TPC): TPC Benchmark. TPC website. http://www.tpc.org (Last retrieved in Jun 2010)
11. Cattell, R., Skeen, J.: Object operations benchmark. TODS **17**(1) (1992) 1–31
12. Carey, M., DeWitt, D., Naughton, J.: The oo7 benchmark. In: SIGMOD Conference. (1993) 12–21
13. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: Xmark: A benchmark for xml data management. In: VLDB. (2002) 974–985
14. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. **3**(2-3) (2005) 158–182
15. 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 (February 2009)
16. Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vañó, A., Gómez-Villamor, S., Martínez-Bazán, N., Larriba-Pey, J.L.: Survey of graph database performance on the hpc scalable graph analysis benchmark. In: IWGD. (2010)
17. INSNA: International network for social network analysis. http://www.insna.org/

18. OReilly, T.: What is Web 2.0: Design patterns and business models for the next generation of software. (2005)
19. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann Publishers Inc. (2000)
20. Brickley, D., Guha, R.V.: Resource description framework (rdf) schema specification 1.0. W3C Candidate Recommendation (2000)
21. Shasha, D., Wang, J., Giugno, R.: Algorithmics and applications of tree and graph searching. In: PODS, New York, NY, USA, ACM (2002) 39–52
22. Anyanwu, K., Sheth, A.: ρ-queries: Enabling querying for semantic associations on the semantic web. In: WWW, ACM Press (2003) 690–699
23. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. ACM Computing Surveys (CSUR) **38**(1) (2006) 2
24. BioGRID: General repository for interaction datasets. http://www.thebiogrid.org/
25. PDB: Rcsb protein data bank. http://www.rcsb.org/
26. NAViGaTOR. http://ophid.utoronto.ca/navigator/
27. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer networks and ISDN systems **30**(1-7) (1998) 107–117
28. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5) (1999) 604–632
29. Strands: e-commerce recommendation engine. http://recommender.strands.com/
30. Chein, M., Mugnier, M.: Conceptual graphs: fundamental notions. Revue d'Intelligence Artificielle **6** (1992) 365–406
31. DirectedEdge: a recommendation engine. Available at http://www.directededge.com (Last retrieved in Jun 2010)
32. Amadeus: Global travel distribution system. http://www.amadeus.net/
33. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: CHI. (2010) 1361–1370
34. Goertzel, B.: OpenCog Prime: Design for a Thinking Machine. Online wikibook, at http://opencog. org/wiki/OpenCogPrime (2008)
35. Erdos, P., Renyi, A.: On random graphs. Mathematicae **6**(290-297) (1959) 156
36. Leskovec, J., Lang, L., Dasgupta, A., Mahoney, M.: Statistical properties of community structure in large social and information networks. In: WWW. (2008) 695–704
37. Flickr: Four Billion. http://blog.flickr.net/en/2009/10/12/4000000000/ (Last retrieved in Jun 2010)
38. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM. (1999) 251–262
39. McGlohon, M., Akoglu, L., Faloutsos, C.: Weighted graphs and disconnected components: patterns and a generator. In: KDD. (2008) 524–532
40. Bader, D., Madduri, K.: Parallel algorithms for evaluating centrality indices in real-world networks. In: ICPP. (2006) 539–550
41. Bitton, D., DeWitt, D., Turbyfill, C.: Benchmarking database systems a systematic approach. In: VLDB. (1983) 8–19
42. Transaction Processing Performance Council (TPC): TPC Benchmark H (2.11). TPC website. http://www.tpc.org/tpch/ (Last retrieved in Jun 2010)
43. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. Journal of Machine Learning Research **11** (2010) 985–1042