A Comparison of Current Graph Database Models

Renzo Angles

Department of Computer Science, Engineering Faculty, Universidad de Talca
Camino Los Niches, Km. 1, Curicó, Chile
renzoangles@gmail.com

***Abstract*— The limitations of traditional databases, in particular the relational model, to cover the requirements of current applications has lead the development of new database technologies. Among them, the Graph Databases are calling the attention of the database community because in trendy projects where a database is needed, the extraction of worthy information relies on processing the graph-like structure of the data. In this paper we present a systematic comparison of current graph database models. Our review includes general features (for data storing and querying), data modeling features (i.e., data structures, query languages, and integrity constraints), and the support for essential graph queries.**

I. Introduction

The limitations of traditional databases, in particular the relational model, to cover the requirements of current application domains, has lead the development of new technologies called *NOSQL databases* [1]. According to its data model, these databases can be categorized as: *Wide-column stores*, which follow the BigTable model of Google (e.g., Cassandra); *Document stores*, which are oriented to store semi-structured data (e.g., MongoDB); *Key-value stores*, which implement a key to value persistent map for data indexing and retrieval (e.g. BerkeleyDB); and *Graph Databases*, which are oriented to store graph-like data.

Activity around graph databases flourished in the first half of the nineties and then the topic almost disappeared [2]. Recently the area is gaining attention because in trendy projects where a database is needed (for example chemistry [3], biology [4], Web Mining [5] and semantic Web [6]), the importance of the information relies on the relations more or equal than on the entities (a basic principle of every graph database). Moreover, the continued emergence and increase of massive and complex graph-like data makes a graph database a crucial requirement. This renascence is showed by the availability of several graph databases systems.

One of the most important elements conforming a database is its *database model* (or simply *data model*). In the most general sense a data model is a collection of conceptual tools used to model representations of real-world entities and the relations among these entities [7]. From a database point of view, a data model consists of three components: a set of data structure types, a set of operators or inference rules, and a set of integrity rules [8].

Graph database models can be characterized as those where 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 [2]. The benefits of using a graph data model are given by: the introduction of a level of abstraction which allows a more natural modeling of graph data; query languages and operators for querying directly the graph structure; and ad-hoc structures and algorithms for storing and querying graphs.

Motivation. In order to choose the most suitable graph database, to be used in some application domain, we need to known its features, advantages and disadvantages. In particular, the data model implemented by a database play an important role in its selection because this ensure (in advance) an innate support for storing and querying the data for a desired application domain.

Related Work. Concerning the origins of graph databases, Angles and Gutierrez [2] developed a survey on graph database models proposed before the year 2002. The authors synthesized the notion of a "graph database model" and compare the proposals available at the moment. It is important to emphasize that most of the works reviewed by the authors followed a theoretical interest more than practical developments. Hence, a practical evaluation of the models is not available.

With respect to the recent developments in the area. Pere Burton [9] reviewed six graph databases (Neo4j, HyperGraphDB, DEX, InfoGrid, Sones and VertexDB) and published a comparison-matrix that included information like software features (e.g., license), schema features (e.g., types of nodes and edges), query features (e.g., language and traversals), general database features (e.g., transactions, indexing), database operation utilities (e.g., protocols), language bindings and operating systems. This work summarizes the features but does not include major discussion nor analysis. Another informal review [10] included an interesting questionnaire about desirable features for graph databases. Domingues-Sal et al.[11] evaluated the performance of three graph databases (Neo4j, HypergraphDB and DEX) and an RDF Database (Jena). The tests, that included the evaluation of several typical graph operations over different graph sizes, shown that DEX and Neo4j were the most efficient implementations.

In summary, a systematic analysis and comparison of the current graph data models is, to the best of our knowledge, not available.

Objective and Contribution. In this paper we compare current graph databases concentrating on their data model features, that is data structures, query facilities, and integrity constraints. We restrict our study to the logical level and avoid physical and implementation considerations. Additionally, and

considering that graph databases are oriented to store and query graph data, we evaluate graph databases in terms of their support for querying a set of essential graph queries.

The paper is organized as follows: In Section II, we surveyed current graph databases. Section III, presents the comparison of graph database models. The support for essential graph queries is discussed in Section IV. Finally, in Section V we draw some conclusions and future work.

II. Current Graph Databases

In the last time there have been an increasing work on graph databases. Next we present a list of current implementations, including a short description of each of them. Considering their level of maturity, in terms of the facilities provided by a database management system, we consider two types of developments: graph databases and graph stores. Additionally, we review several developments related to graph databases.

Graph Databases. We assume that a graph database must provide most of the major components in database management systems, being them: external interfaces (user interface or API), database languages (for data definition, manipulation and querying), query optimizer, database engine (middle-level model), storage engine (low-level model), transaction engine, management and operation features (tuning, backup, recovery, etc.). Among the developments satisfying the above condition, we found AllegroGraph, DEX, HypergraphDB, InfiniteGraph, Neo4J and Sones.

AllegroGraph[12] is one of the precursors in the current generation of graph databases. Although it was born as a graph database, its current development is oriented to meet the Semantic Web standards (i.e., RDF/S, SPARQL and OWL). Additionally, AllegroGraph provides special features for GeoTemporal Reasoning and Social Network Analysis.

DEX[13], [14] provides a Java library for management of persistent and temporary graphs. Its implementation, based on bitmaps and other secondary structures, is oriented to ensure a good performance in the management of very large graphs.

HyperGraphDB [15], [16] is a database that implements the hypergraph data model where the notion of edge is extended to connect more than two nodes. This model allows a natural representation of higher-order relations, and is particularly useful for modeling data of areas like knowledge representation, artificial intelligence and bio-informatics.

InfiniteGraph [17] is a database oriented to support large-scale graphs in a distributed environment. It aims the efficient traversal of relations across massive and distributed data stores. Its focus of attention is to extend business, social and government intelligence with graph analysis.

Neo4j [18] is based on a network oriented model where relations are first class objects. It implements an object-oriented API, a native disk-based storage manager for graphs, and a framework for graph traversals.

Sones [19] is a graph database which provides an inherent support for high-level data abstraction concepts for graphs (e.g., walks). It defines its own graph query language and a underlying distributed file system.

Graph Stores. This category grouped implementations providing basic facilities for storing and querying graphs. Among them, Filament [20] is a project for a graph storage library with default support for SQL through JDB; G-Store [21] is a basic storage manager for large vertex-labeled graphs; redis_graph [22] provides a basic Python implementation for storing graphs; and VertexDB [23] implements a graph store on top of TokyoCabinet (a B-tree key/value disk store). Additionally, we can mention CloudGraph [24], Horton [25] and Trinity [26] as prototypes of graph databases.

Technologies related to graph databases. Graph databases are oriented to store any type of graph, hence they are distinct from specialized data management tools that use graph notions in their implementation. Among them we can mention: *Web-oriented databases*, which use graph structures for modeling data used in Web-oriented applications (e.g., InfoGrid and FlockDB); *Document-oriented databases*, which implement graph algorithms in order to traverse the relations between documents (e.g., OrientDB [27]); and *Triple Stores* (also *called RDF databases*), which are oriented to store RDF graphs consisting in statements of the form subject-predicate-object (e.g., 4Store, Virtuoso and Bigdata). Giraph [28] is a graph processing infrastructure that runs on Hadoop (see Pregel); AnGrapa [29] is an large-scale graph data management framework for analytical processing; GoldenOrb [30] is a cloud-based open source project for massive-scale graph analysis; Phoebus [31] is an implementation of Google's Pregel for distributed processing of very large graphs.

Additionally, we can found several *in-memory graph tools* characterized by their restriction to work with small graphs. For example, we can mention complex analysis tools (e.g., Cytoscape), and graph visualization tools (e.g., JUNG, IGraph, GraphViz, Gephi and NodeXL).

III. Comparison of Graph Database Models

A comparison among databases is typically done by either using a set of common features or by defining a general model used as a comparison basis. The evaluation presented in this section is oriented to evaluate the data model provided by each graph database, in terms of data structures, query language and integrity constraints.

As an initial approach we consider some general features for data storing, operation and manipulation (see Table I). From the point of view of data storing, we review the support for three storing schemas (i.e., main memory, external memory, and back-end storage) and the implementation of indexes. It is important to emphasize that managing a huge amount of data is a important requirement in real-life applications for graph databases. Hence the support for external memory storage is a main requirement. Additionally, indexes are the basis to improve data retrieval operations.

From the point of view of data operation and manipulation, we evaluate whether a graph database implements database languages, application programming interfaces (API) and graphical user interfaces (GUI). We consider three database languages: the *Data Definition Language*, which allows to

modify the schema of the database by adding, changing, or deleting its objects; the Data Manipulation Language, which allows to insert, delete and update data in the database; and the Query Language, which allows to retrieve data by using a query expression. Data operation and manipulation features are summarized in Table II.

In comparison with the traditional approach in databases, where high-level languages for data operation and manipulation are provided, the most common mechanism in graph databases is the use of APIs. It means several advantages: standard vocabulary (for functions and procedures), easy development of applications, and an unlimited power for querying data. However, it also brings serious problems: lower level of abstraction (for the general user), programming language restrictions, implementation-dependent efficiency, and decidability problems.

An important feature, not included in Table I, is the support to import and export data in different data formats. Although there exists some data formats for encoding graphs (e.g, GraphML and TGV) none of them has been selected as the standard one. This issue is particularly relevant for data exchange and sharing.

TABLE I
DATA STORING FEATURES

Graph Database	Main memory	External memory	Backend Storage	Indexes
AllegroGraph	•	•		•
DEX	•	•		•
Filament	•		•	
G-Store		•		
HyperGraphDB	•	•	•	•
InfiniteGraph		•		•
Neo4j	•	•		•
Sones	•			•
vertexDB		•	•	

TABLE II
OPERATION AND MANIPULATION FEATURES

Graph Database	Data Definition Language	Data Manipulat. Language	Query Language	API	GUI
AllegroGraph	•	•	•	•	•
DEX				•	
Filament				•	
G-Store	•		•	•	
HyperGraphDB				•	
InfiniteGraph				•	
Neo4j				•	
Sones	•	•	•	•	•
vertexDB				•	

A. Graph data structures

The data structures refer to the types of entities or objects that can be used to model data. In the case of graph databases, the data structure is naturally defined around the notions of graphs, nodes and edges (see Table III).

We consider four graph data structures: simple graphs, hypergraphs, nested graphs and attributed graphs. The basic structure is a simple flat graph defined as a set of nodes (or vertices) connected by edges (i.e., a binary relation over the set of nodes). An Hypergraph extends this notion by allowing an edge to relate an arbitrary set of nodes (called an hyperedge). A nested graph is a graph whose nodes can be themselves graphs (called hypernodes). Attributed graphs are graphs where nodes and edges can contain attributes for describing their properties [32]. Additionally, over the above types of graphs, we consider directed or undirected edges, labeled or unlabeled nodes/edges, and attributed nodes/edges (i.e., edges between edges are possible).

Note that most graph databases are based on simple graphs or attributed graphs. Only two support hypergraphs and no one nested graphs. We can remark that hypergraphs and attributed graphs can be modeled by nested graphs. In contrast, the multilevel nesting provided by nested graphs cannot be modeled by any of the other structures [2].

In comparison with past graph database models, the inclusion of attributes for nodes and edges is a particular feature in current proposals. The introduction of attributes is oriented to improve the speed of retrieval for the data directly related to a given node. This feature shows the influence of implementation issues in the selection and definition of the data structures (and consequently of the data model).

TABLE III
GRAPH DATA STRUCTURES

	Graphs				Nodes		Edges		
Graph Database	Simple graphs	Hypergraphs	Nested graphs	Attributed graphs	Node labeled	Node attribution	Directed	Edge labeled	Edge attribution
AllegroGraph	•				•		•	•	
DEX				•	•	•	•	•	•
Filament	•				•		•	•	
G-Store	•				•		•	•	
HyperGraphDB		•			•		•	•	
InfiniteGraph				•	•	•	•	•	•
Neo4j				•	•	•	•	•	•
Sones		•		•	•	•	•	•	•
vertexDB	•				•		•	•	

The expressive power for data modeling can be analyzed by comparing the support for representing entities, properties and relations at both instance and schema levels. This evaluation is shown in Table IV.

At the schema level we found that models support the definition of node, attribute and relation types. We also evaluate the support for several nodes and relations at the instance level: an object node, identified by an object-ID, represents an instance of a node type; a value node represents an entity identified by a primitive value (i.e., its name); a complex node can represent an special complex entity, for example a tuple or a

set; an object relation, identified by a relation-ID, is an instance of a relation type; a simple relation represents a node-edge-node instance; a complex relation is a relation with special semantics, for example grouping, derivation, and inheritance.

Value nodes and simple relations are supported by all the models. The reason is that both conform the most basic and simple model for representing graph data. The inclusion of object-oriented concepts (e.g., IDs for objects) for representing entities and relations reflects the use of APIs as the favorite high-level interface for the database. Note that this issue is not new in graph databases. In fact, it was naturally introduced by the so called graph object-oriented data models [2].

Finally, the use of objects (for both nodes and relations) is different of using values. For example, an object node represents an entity identified by an object-ID, but it does not represent the value-name of the entity. In this case, it is necessary to introduce an explicit property or relation "name" in order to specify the name of the entity. The same applies for relations. This issue generates an unnatural form of modeling graph data.

TABLE IV
REPRESENTATION OF ENTITIES AND RELATIONS

	Schema			Instance					
Graph Database	Node types	Property types	Relation types	Object nodes	Value nodes	Complex nodes	Object relations	Simple relations	Complex relations
AllegroGraph					●			●	
DEX	●		●	●	●		●	●	
Filament					●			●	
G-Store					●			●	
HyperGraphDB	●		●		●			●	●
InfiniteGraph	●		●	●	●		●	●	
Neo4j				●	●		●	●	
Sones					●			●	●
vertexDB					●			●	

B. Query languages

A query language is a collection of operators or inference rules that can be applied to any valid instance of the database, this with the objective of manipulating and querying data in any combination desired [2]. As is shown in Table III, query languages are not frequent in current graph databases. In fact, there is not proposal for a standard one.

AllegroGraph supports SPARQL, the standard query language for RDF. SPARQL is based on graph pattern matching but is not oriented to querying the graph structure of RDF data. Neo4j is developing Cypher, a query language for property graphs. G-Store and Sones include SQL-based query languages with special instructions for querying graphs. To the best of our knowledge, there is not a formal definition of the semantics for the above query languages, making a systematic study of their complexity and expressive power difficult.

Data retrieval is the main objective in current graph databases. AllegroGraph supports reasoning via its Prolog implementation. Data analysis is supported in terms of special functions (e.g., shortest path) for querying graph properties.

The lack of a standard query language is a disadvantage of current graph databases. Recall that in mature databases the operation of the database is performed via standard and well-defined database languages. Instead, the focus in current graph databases is to provide APIs for popular programming languages. Hence, the selection is hardly determined by the programmer skills or by application requirements.

TABLE V
COMPARISON OF QUERY FACILITIES (● INDICATES SUPPORT, AND ○ PARTIAL SUPPORT)

	Type			Use		
Graph Database	Query Lang.	API	Graphical Q. L.	Retrieval	Reasoning	Analysis
AllegroGraph	○	●	●	●	●	●
DEX		●		●		●
Filament		●		●		
G-Store	●			●		
HyperGraphDB		●		●		
InfiniteGraph		●		●		
Neo4j	○	●		●		
Sones	●		●	●		●
vertexDB		●		●		

C. Integrity constraints

Integrity constraints are general statements and rules that define the set of consistent database states, or changes of state, or both [2]. Table VI shows that integrity constraints are poorly studied in graph databases. In fact, there are not important variations of the notions studied in the past.

We consider several integrity constraints: types checking, to test the consistency of an instance with respect to the definitions in the schema; node/edge identity, to verify that an entity or relation can be identified by either a value (e.g., name or ID) or the values of its attributes (e.g., neighborhood identification); referential integrity, to test that only existing entities are referenced; cardinality checking, to verify uniqueness of properties or relations; functional dependency, to test that an element in the graph determines the value of another; and graph pattern constraints, to verify an structural restriction (e.g., path constraints).

The support for evolving schemas is a characteristic of graph databases that is commonly used to justify the lack of integrity constraints. We aim that is not a valid argument assuming that data consistency in a database is equal or even more important than a flexible schema. Moreover, an evolving schema can be supported by allowing flexible structures in the schema (as in semi-structure data models). For example, the definition of a

relation type as optional, enables the user to decide either the inclusion or the absence of such relation for a given entity.

TABLE VI
COMPARISON OF INTEGRITY CONSTRAINTS

Graph Database	Types checking	Node/edge identity	Referential integrity	Cardinality checking	Functional dependency	Graph pattern constrains
DEX	•	•	•			
HyperGraphDB	•	•				
InfiniteGraph	•	•				
Sones		•		•		

IV. CURRENT GRAPH DATABASES AND THEIR SUPPORT FOR QUERYING GRAPHS

In the database literature we can found several works studying problems related to storing and querying graphs [33], [34]. Such problems play an important role for applications that use graphs as their data model [4]. In this section, we compare graph databases in terms of their facilities to solve several queries which can be considered essential in graphs. We grouped then in adjacency, reachability, pattern matching and summarization queries.[1]

1) Adjacency queries: The primary notion in this type of queries is node/edge *adjacency*. Two nodes are adjacent (or neighbors) when there is and edge between them. Similarly, two edges are adjacent when they share a common node.

Some typical queries in this group are: *basic node/edge adjacency*, to test whether two nodes (edges) are adjacent [36]; and k-neighborhood of a node, to list all the neighbors of the node [37]. Adjacency queries are useful in several contexts, for example in molecular biology [38], information retrieval[39] and the semantic Web [40].

2) Reachability queries: This type of queries is characterized by path or traversal problems. The problem of *reachability* tests whether two given nodes are connected by a path.

In this context, we can consider two types of paths: *fixed-length paths*, which contain a fixed number of nodes and edges; and *regular simple paths*, which allow some node and edge restrictions (e.g., regular expressions). A related but more complicated problem is to find the *shortest path*, that is to compute the quickest/shortest route between two nodes.

Reachability queries are studied and required in several application domains. For example in database theory [41], [42] (where recursive queries are in practice graph traversals), spatial databases [43], biological databases [44], and the semantic Web [45]. One of the challenges to incorporate reachability queries into a query language is their computational complexity. For example, finding simple paths with desired properties in direct graphs is an NP-complete problem [34].

3) Pattern matching queries: Graph pattern matching consists in to find all sub-graphs of a data graph that are isomorphic to a pattern graph. Particularly, it deals with two problems: the graph isomorphism problem that has a unknown computational complexity; and the sub-graph isomorphism problem which is an NP-complete problem [33].

Pattern matching has attracted a great deal of attention in database theory [33], [34], [46], [47], data mining [48], bioinformatics [49], and semantic Web [50].

4) Summarization queries: This type of queries are not related to consult the graph structure. Instead they are based on special functions that allow to summarize or operate on the query results, normally returning a single value. Aggregate functions (e.g., average, Count, maximum, etc.) are included in this group.

Additionally, we consider functions to compute some properties of a graph and its elements. For example: the order of the graph (i.e., the number of vertices), the degree of a node (i.e., the number of neighbors of the node), the minimum, maximum and average degree in the graph, the length of a path (i.e., the number of edges in the path), the distance between nodes (i.e., the length of a shortest path between the nodes), the diameter of the graph (i.e., the greatest distance between any two nodes), etc.

In Table VII, we summarized the support that current graph databases provide for answering the queries defined above. Recall that most graph databases implements an API instead of a query language. Hence, we evaluate whether an API can answer an essential query by using a combination of basic functions, more than the facilities to implement an algorithm that solve the query (feature clearly supported by a programming language).

Additionally, Table VIII presents the results of a previous study [35] about (past) graph query languages and their support for querying essential graph queries. Such study provides a positive conclusion about the feasibility of developing a well-designed graph query language. It is important to note that these languages were well-studied from a theoretical point of view. Hence, they provide a formal background for the definition of a standard query language for graph databases.

V. CONCLUSIONS

In this paper we surveyed current graph databases and compare them according to their data modeling features. We shown that most graph database models provide an innate support for different graph structures, query facilities in the form of APIs (most of the models) and query languages (a few of them), and basic notions of integrity constraints. Additionally, we defined a set of essential graph queries and evaluated the query facilities provided by graph databases in order to answer such queries.

The review shown that some aspects of current graph database models deserve more development. In particular, the definition of standard graph database languages (for defining,

[1] A similar classification was proposed and used in a previous evaluation of graph query languages [35].

TABLE VII

CURRENT GRAPH DATABASES AND THEIR SUPPORT FOR ESSENTIAL GRAPH QUERIES

	Adjacency		Reachability				
Graph Database	Node/edge adjacency	k-neighborhood	Fixed-length paths	Regular simple paths	Shortest path	Pattern matching	Summarization
Allegro	●		●			●	
DEX	●		●	●	●	●	
Filament	●		●			●	
G-Store	●		●	●	●	●	
HyperGraph	●					●	
Infinite	●		●	●	●	●	
Neo4j	●		●	●	●	●	
Sones	●					●	
vertexDB	●		●	●		●	

TABLE VIII

PAST GRAPH QUERY LANGUAGES AND THEIR SUPPORT FOR ESSENTIAL GRAPH QUERIES (● INDICATES SUPPORT, AND ○ PARTIAL SUPPORT)

Graph Database	Node/edge adjacency	Fixed-length paths	Regular simple paths	Degree of a node	Distance between nodes	Diameter
G	○	●	●			
G+	●	●	●	●	●	●
GraphLog	●	●	●	●	●	●
Gram	●	●	●			
GraphDB	○	●	●			
Lorel	●	●	●			
F-G	○	●	●			

manipulating and querying the data) and notions of integrity constraints (for preserving the consistency of the database).

As future work we plan to develop an empirical evaluation of current graph databases; this oriented to make a quantitative and qualitative analysis of their support for storing and querying graph data.

ACKNOWLEDGMENT

This work was supported by the Chilean Fondecyt Project No. 11100364.

REFERENCES

[1] "NOSQL Databases," http://nosql-database.org/.

[2] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, pp. 1–39, 2008.

[3] A. Nayak. and I. Stojmenovic, *Handbook of Applied Algorithms: Solving Scientific, Engineering, and Practical Problems*. Wiley-IEEE Press, 2008, ch. Graph Theoretic Models in Chemistry and Molecular Biology, pp. 85–113.

[4] B. A. Eckman and P. G. Brown, "Graph data management for molecular and cell biology," *IBM Journal of Research and Development*, vol. 50, no. 6, pp. 545–560, 2006.

[5] A. Schenker, H. Bunke, M. Last, and A. Kandel, *Graph-Theoretic Techniques for Web Content Mining*, ser. Series in Machine Perception and Artificial Intelligence. World Scientific, 2005, vol. 62.

[6] J. Hayes and C. Gutierrez, "Bipartite Graphs as Intermediate Model for RDF," in *Proceedings of the 3th International Semantic Web Conference (ISWC)*, ser. LNCS, no. 3298. Springer-Verlag, Nov 2004, pp. 47–61.

[7] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Data Models," *ACM Computing Surveys*, vol. 28, no. 1, pp. 105–108, 1996.

[8] E. F. Codd, "Data Models in Database Management," in *Proceedings of the 1980 Workshop on Data abstraction, Databases and Conceptual Modeling*. ACM Press, 1980, pp. 112–114.

[9] P. Urbón, "Nosql graph database matrix," http://nosql.mypopescu.com/post/619181345/nosql-graph-database-matrix, May 2010.

[10] "Short overview on the emerging world of graph databases," http://www.graph-database.org/overview.html.

[11] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazán, and J. L. Larriba-Pey, "Survey of graph database performance on the hpc scalable graph analysis benchmark," in *Proc. of the 2010 international conference on Web-age information management (WAIM)*. Springer-Verlag, 2010, pp. 37–48.

[12] "AllegroGraph," http://www.franz.com/agraph/allegrograph/.

[13] N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey, "DEX: High-Performance Exploration on Large Graphs for Information Retrieval," in *Proceedings of the 16th Conference on Information and Knowledge Management (CIKM)*. ACM, 2007, pp. 573–582.

[14] "DEX," http://www.sparsity-technologies.com/dex.

[15] "HyperGraphDB," http://www.hypergraphdb.org/.

[16] B. Iordanov, "Hypergraphdb: a generalized graph database," in *Proceedings of the 2010 international conference on Web-age information management (WAIM)*. Springer-Verlag, 2010, pp. 25–36.

[17] "Infinitegraph," http://infinitegraph.com/.

[18] "Neo4j," http://neo4j.org/.

[19] "Sones GraphDB," http://www.sones.com/.

[20] "Filament," http://filament.sourceforge.net/.

[21] "G-Store," http://g-store.sourceforge.net/.

[22] "redis graph: Graph database for python," https://github.com/amix/redis_graph.

[23] "vertexdb," http://www.dekorte.com/projects/opensource/vertexdb/.

[24] "Cloudgraph," http://www.cloudgraph.com/.

[25] "Horton," http://research.microsoft.com/en-us/projects/ldg.

[26] "Trinity," http://research.microsoft.com/en-us/projects/trinity/.

[27] "Orientdb," http://www.orientechnologies.com/.

[28] "Giraph," https://github.com/aching/Giraph.

[29] "Angrapa - the graph package," http://wiki.apache.org/hama/GraphPackage.

[30] "Goldenorb," http://www.goldenorbos.org/.

[31] "Phoebus," https://github.com/xslogic/phoebus.

[32] H. Ehrig, U. Prange, and G. Taentzer, "Fundamental theory for typed attributed graph transformation," in *Proc. of the 2nd Int. Conference on Graph Transformation (ICGT)*, ser. LNCS, no. 3256. Springer, 2004, pp. 162–177.

[33] M. Yannakakis, "Graph-Theoretic Methods in Database Theory," in *Proceedings of the 9th Symposium on Principles of Database Systems (PODS)*. ACM Press, 1990, pp. 230–242.

[34] D. Shasha, J. T. L. Wang, and R. Giugno, "Algorithmics and Applications of Tree and Graph Searching," in *Proceedings of the 21th Symposium on Principles of Database Systems (PODS)*. ACM Press, 2002, pp. 39–52.

[35] R. Angles and C. Gutierrez, "Querying RDF Data from a Graph Database Perspective," in *Proceedings of the 2nd European Semantic Web Conference (ESWC)*, ser. LNCS, no. 3532, 2005, pp. 346–360.

[36] L. Kowalik, "Adjacency queries in dynamic sparse graphs," *Information Processing Letters*, vol. 102, pp. 191–195, May 2007.

[37] A. N. Papadopoulos and Y. Manolopoulos, *Nearest Neighbor Search - A Database Perspective*, ser. Series in Computer Science. Springer, 2005.

[38] T. Seidl and H. peter Kriegel, "A 3d molecular surface representation supporting neighborhood queries," in *Proc. of the 3rd Conference on Intelligent Systems for Molecular Biology (ISMB)*. Springer, 1995, pp. 240–258.

[39] C.-S. Chang and A. L. P. Chen, "Supporting conceptual and neighborhood queries on the world wide web," *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, vol. 28, no. 2, pp. 300–308, 1998.

[40] R. Guha, R. McCool, and E. Miller, "Semantic Search," in *Proceedings of the 12th International Conference on World Wide Web (WWW)*. ACM Press, 2003, pp. 700–709.

[41] P. Barceló, C. Hurtado, L. Libkin, and P. Wood, "Expressive Languages for Path Queries over Graph-structured Data," in *Proc. of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2010, pp. 3–14.

[42] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu, "Adding regular expressions to graph reachability and pattern queries," in *Proc. of the IEEE 27th International Conference on Data Engineering (ICDE)*, 2011, pp. 39–50.

[43] J. Paredaens and B. Kuijpers, "Data Models and Query Languages for Spatial Databases," *Data & Knowledge Engineering (DKE)*, vol. 25, no. 1-2, pp. 29–53, 1998.

[44] A. Matono, T. Amagasa, M. Yoshikawa, and S. Uemura, "An Eficient Pathway Search Using an Indexing Scheme for RDF," *Genome Informatics*, no. 14, pp. 374–375, 2003.

[45] A. Gubichev and T. Neumann, "Path query processing on very large rdf graphs," in *Proc. of the 14th International Workshop on the Web and Databases (WebDB)*, 2011.

[46] P. Barcelo, L. Libkin, and J. Reutter, "Querying graph patterns," in *Proc. of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2011, pp. 199–210.

[47] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang, "Fast graph pattern matching," in *Proc. of the 2008 IEEE 24th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2008, pp. 913–922.

[48] T. Washio and H. Motoda, "State of the Art of Graph-based Data Mining," *SIGKDD Explorer Newsletter*, vol. 5, no. 1, pp. 59–68, 2003.

[49] X. Wang, "Finding patterns on protein surfaces: Algorithms and applications to protein classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 1065–1078, 2005.

[50] J. Carroll, "Matching RDF Graphs," in *Proceedings of the International Semantic Web Conference (ISWC)*, 2002.