Graph indexing and querying: a review

Sherif Sakr and Ghazi Al-Naymat
School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia

Received 2 September 2009
Revised 16 March 2010
Accepted 18 March 2010

Abstract

Purpose – The purpose of this paper is to provide a detailed discussion for different types of graph queries and a different mechanism for indexing and querying graph databases.

Design/methodology/approach – The paper reviews the existing approaches and techniques for indexing and querying graph databases. For each approach, the strengths and weaknesses are discussed with particular emphasis on the target application domain. Based on an analysis of the state-of-the-art of research literature, the paper provides insights for future research directions and untouched challenging research aspects.

Findings – Several graph indexing and querying techniques have been proposed in the literature. However, there is still a clear room for improvement and further research issues in that domain.

Research limitations/implications – The paper identifies the advantages and disadvantages of the different graph indexing mechanisms and their suitability for different practical applications. The paper provides some guidelines and recommendations which are useful for future research in the area of graph databases.

Practical implications – The paper has practical implications for social networks, protein networks, chemical compounds, multimedia database, and semantic web.

Originality/value – The paper contributes to the implementation of an efficient indexing and querying mechanism for graph databases in different application domains.

Keywords Graphical user interface, Indexing, Databases

Paper type Research paper

Introduction

The field of graph databases and graph query processing has received a lot of attention due to the constantly increasing usage of graph data structure for representing data in different domains such as: XML (Sayed, 2009), chemical compounds (National Cancer Institute (NCI), 2010; Klinger and Austin, 2005), multimedia databases (Lee *et al.*, 2005), social networks (Cai *et al.*, 2005), biological pathways (Kyoto Encyclopedia of Genes and Genomes (KEGG), 2010; The Gene Ontology, 2010), protein networks (Huan *et al.*, 2004; Westbrook *et al.*, 2003), semantic web (Manola and Miller, 2004) and business process models (Sakr and Awad, 2010). To effectively understand and utilize any collection of graphs, a graph database that efficiently supports elementary querying mechanisms is crucially required. Hence, determining graph database members which constitute the answer set of a graph query *q* from a large graph database is a key performance issue in all graph-based applications. A primary challenge in computing the answers of graph queries is that pair-wise comparisons of graphs are usually really hard problems. For example, subgraph isomorphism is known to be NP-complete (Garey and Johnson, 1979). Even a relatively simple comparison, graph isomorphism defies a polynomial bound for the general case (Fortin, 1996).

International Journal of Web Information Systems
Vol. 6 No. 2, 2010
pp. 101-120
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/17440081011053104

A naive approach to compute the answer set of a graph query q is to perform a sequential scan on the graph database and to check whether each graph database member satisfies the conditions of q or not. However, the graph database can be very large which makes the sequential scan over the database impractical. Thus, finding an efficient search technique is immensely important due to the combined costs of pair-wise comparisons and the increasing size of modern graph databases. It is apparent that the success of any graph database application is directly dependent on the efficiency of the graph indexing and query processing mechanisms. Recently, there are many techniques that have been proposed to tackle these problems. This survey summarizes the work done in the literature by presenting different techniques of indexing and querying graph databases. It compares the strengths and weaknesses of these techniques and classifies them according to their target graph query types and their indexing strategy. We believe that this kind of survey and comparison is important to derive further research and contribution in these related areas. Other surveys (Angles and Gutiérrez, 2008; Gao *et al.*, 2010) which focused on other aspects such as graph data models, graph query languages and graph similarity metrics complement the work presented in this paper to understand different aspects of graph database indexing and querying techniques.

The rest of the paper is organized as follows. The next section introduces preliminaries of graph databases and graph query processing. In the third section, a classification of the approaches of subgraph query processing problem and their index structures is given while the fourth section focuses on the approaches for resolving the supergraph query processing problem. The fifth section discusses the approach of approximate graph matching queries. Finally, the last section concludes the paper and provides some suggestions for possible future research directions on the subject.

Preliminaries

In this section, we introduce the basic terminologies used in this paper and give the formal definition of graph querying problems which are considered in this paper.

Graph data structure

Graphs are used to model complicated structures and schemaless data. In graph data structures, vertices and edges represent the entities and the relationships between them, respectively. The attributes associated with these entities and relationships are called labels.

A graph database D is defined as a collection of member graphs $D = \{g_1, g_2, \dots, g_n\}$ where each member graph database member g_i is denoted as $(V, E, L_v, L_e, F_v, F_e)$ where V is the set of vertices; $E \subseteq V \times V$ is the set of edges joining two distinct vertices; L_v is the set of vertex labels; L_e is the set of edge labels; F_V is a function $V \rightarrow L_v$ that assigns labels to vertices and F_e is a function $E \rightarrow L_e$ that assigns labels to edges.

In general, graph data structures can be classified according to the direction of their edges into two main classes:

(1) *Directed-labeled graphs*. Such as XML, resource description framework (RDF), and traffic networks.

(2) *Undirected-labeled graphs*. Such as social networks and chemical compounds.

Graph isomorphism
Let $g_1 = (V_1, E_1, L_{v1}, L_{e1}, F_{v1}, F_{e1})$ and $g_1 = (V_2, E_2, L_{v2}, L_{e2}, F_{v2}, F_{e2})$ be two graphs, g_1 is defined as a graph isomorphism to g_2, if and only if there exists at least one bijective function $f:V_1 \to V_2$ such that:

- For any edge $uv \in E_1$, there is an edge $f(u)f(v) \in E_2$.
- $F_{v1}(u) = F_{v2}(f(u))$ and $F_{v1}(v) = F_{v2}(f(v))$.
- $F_{e1}(uv) = F_{e2}(f(u)f(v))$.

Figure 1 shows an example of two isomorphic graphs g_1 and g_2.

Graph queries
In principle, queries in graph databases can be broadly classified into the following main categories:

- *Subgraph queries.* This category searches for a specific pattern in the graph database. The pattern can be either a small graph or a graph where some parts of it are uncertain, e.g. vertices with wildcard labels. Therefore, given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ and a subgraph query q, the query answer set $A = \{g_i | q \subseteq g_i, g_i \in D\}$. A graph q is described as a subgraph of another graph database member g_i if the set of vertices and edges of q form subset of the vertices and edges of g_i. Figure 2(a) shows an example of a graph database. Figure 2(b) shows examples of graph queries (q_1 and q_2). Let us assume that these queries are subgraph queries. If we evaluate these queries over the sample graph database (Figure 2(a)) then the answer set of q_1 will consist of the graph database members g_1 and g_2 while the answer set of q_2 will be empty.
- *Supergraph queries.* This category searches for the graph database members of which their whole structures are contained in the input query. Therefore, given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ and a supergraph query q, the query answer set $A = \{g_i | q \supseteq g_i, g_i \in D\}$. Let us assume that the graph queries of Figure 2(b) are supergraph queries. If we evaluate these queries over the sample graph database (Figure 2(a)) then the answer set of q_1 will be empty while the answer set of q_2 will contain the graph database member g_3.
- *Similarity (approximate matching) queries.* This category finds graphs which are similar, but not necessarily isomorphic to a given query graph. Given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ and a query graph q, the goal of a similarity search is to discover all graphs that are approximately similar to the graph query q. This category of queries can be further classified to the following two kinds of queries:

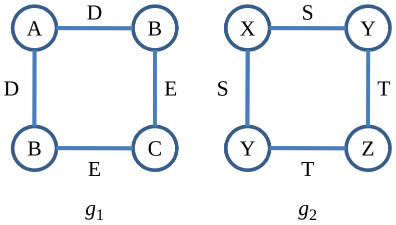
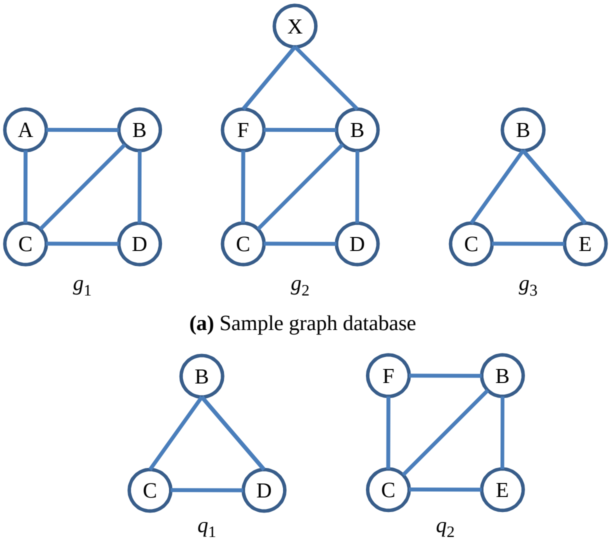


Figure 1.
Graph isomorphism



(a) Sample graph database

(b) Graph queries

Figure 2.
An example graph database and graph queries

- *Substructure similarity search.* These queries are used to discover all graphs that approximately contain the query graph.
- *Reverse similarity search.* These queries are used to discover all graphs that are approximately contained by the query graph.

Two main approaches are used to determine the answer set of similarity queries:

(1) *K-NN queries.* This approach looks for the K nearest graphs to the query graph.

(2) *Range queries.* This approach looks for the graphs whose similarity scorer are within a user-specified threshold.

A key question in graph similarity queries is on how to measure the similarity between a target graph member of the database and the query graph. In fact, it is difficult to give a precise definition of graph similarity. Different approaches have proposed different similarity metrics for graph data structures (Bunke and Shearer, 1998; Fernández and Valiente, 2001; Raymond *et al.*, 2002). Discussing these different similarity metrics is out of the scope of this paper. However, we would refer the interested reader to a detailed survey reported by Gao *et al.* (2010).

Filtering and verification framework

Owing to the very expensive cost of pair-wise comparison of graph data structures, recently proposed graph query processing techniques rely on a strategy which consists of two steps: filtering and verification. For a given graph database D and a graph query q, the main objective of the filtering and verification methodology is to avoid comparing the query graph q with each graph database member g_i that belongs to D to check whether g_i satisfies the conditions of q or not. Hence, the processing of the framework goes through the following three main steps:

(1) *Off-line index construction*. Generates and maybe selects a feature set *F* from the graph database *D*. An inverted index is then built to map each feature to its associated list of related graphs.

(2) *Filtering*. Probes the constructed index of (step 1) to eliminate, as much as possible, members that are identified as being of the false results. The indexed features are used to produce a set of candidate graphs which is related to the feature of the query *q*.

(3) *Verification*. Checks each candidate graph *g* from the filtering phase to verify whether it indeed satisfies the conditions of the query *q*.

Figure 3 shows the steps of the filtering-and-verification framework of graph query processing techniques. Since the candidate answer set is in general smaller than the entire graph database, query processing using the indexes is significantly more efficient than the sequential scan approach. However, candidate verification is still very expensive since the size of the candidate answer set is at least equal to that of the exact answer set in the optimal case but it is usually larger for most of the queries. Therefore, reducing the size of the candidate answer set by removing as much as possible of the false positive graphs is the main criteria to evaluate the effectiveness of the filtering techniques.

Subgraph query processing

There are many graph indexing techniques that have been recently proposed to deal with the problem of subgraph query processing (Cheng *et al.*, 2007; Giugno and Shasha, 2002; He and Singh, 2006; Jiang *et al.*, 2007; Sakr, 2009; Tian and Patel, 2008; Williams *et al.*, 2007; Yan *et al.*, 2004; Zhang *et al.*, 2007; Zhao *et al.*, 2007; Zou *et al.*, 2008). According to (Zou *et al.*, 2008), subgraph query processing techniques can be divided into the following two approaches:

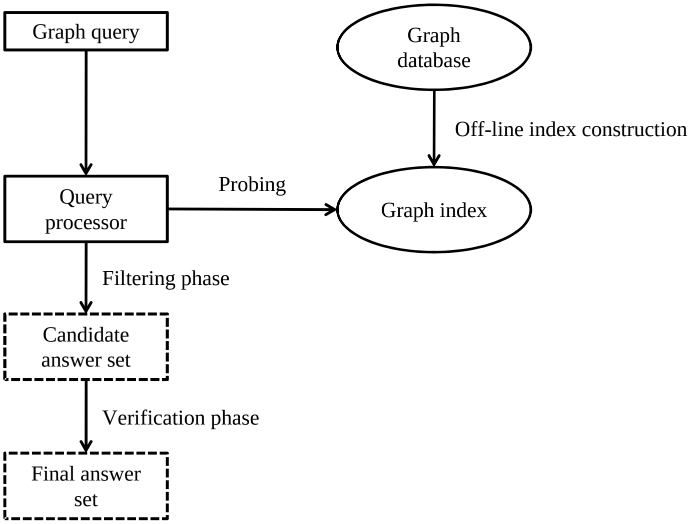


Figure 3.
Filtering and verification framework

(1) *Non mining-based graph indexing techniques.* The techniques of this approach focus on indexing the whole constructs of the graph database instead of indexing only some selected features (Giugno and Shasha, 2002; He and Singh, 2006; Jiang *et al.*, 2007; Sakr, 2009; Tian and Patel, 2008; Williams *et al.*, 2007). The main criticisms of these approaches are that:
   - they can be less effective in their pruning power; and
   - they may need to conduct expensive structure comparisons in the filtering process and thus degrades the filtering efficiency.

Therefore, these techniques need to employ efficient filtering and pruning mechanisms to overcome these limitations. However, the techniques of this approach have a clear advantage in that they can handle the graph updates with less cost as they do not rely on the effectiveness of the selected features and they do not need to rebuild their whole indexes.

(2) *Mining-based graph indexing techniques*. The techniques of this approach apply graph mining methods (Kuramochi and Karypis, 2001, 2004; Wang *et al.*, 2004; Washio and Motoda, 2003; Yan and Han, 2002, 2003) to extract some features (sub-structures) from the graph database members (Cheng *et al.*, 2007; Yan *et al.*, 2004; Zhang *et al.*, 2007; Zhao *et al.*, 2007). An inverted index is created for each feature. Answering a subgraph query q is achieved through two steps:
   - Identifying the set of features of the subgraph query.
   - Using the inverted index to retrieve all graphs that contain the same features of q.

The rationale behind this type of query processing techniques is that if some features of graph q do not exist in a data graph G, then G cannot contain q as its subgraph. Clearly, the effectiveness of these filtering methods depends on the quality of mining techniques to effectively identify the set of features. Therefore, important decisions need to be made about: the indexing feature, the number and the size of indexing features. These decisions crucially affect the cost of the mining process and the pruning power of the indexing mechanism. On the one hand, the more indexed features the higher the pruning power of the filtering phases. However, it also increases the space cost. On the other hand, the less indexed features lead to poor pruning power in the filtering process and consequently resulting in a performance hit in the query processing time. A main limitation of these approaches is that the quality of the selected features may degrade over time after lots of insertions and deletions. In this case, the set of features in the whole updated graph database need to be re-identified and the index need to be re-built from scratch. It should be noted that, achieving these tasks is quite time consuming.

Non-mining-based graph indexing techniques

GraphGrep. The GraphGrep (Giugno and Shasha, 2002) index structure uses enumerated paths as index features to filter unmatched graphs. For each graph database member, it enumerates all paths up to a certain maximum length and records the number of occurrences of each path. Hence, in this index table, each row stands for a path and each column stands for a graph. Each entry in the table is the number of occurrences of the path in the graph. In the query processing, the path indexes is used to find a set of candidate graphs which contains the paths in the query structure and to check if the counts of such paths are beyond the threshold specified in the query. In the verification step, each candidate graph is examined by subgraph isomorphism to obtain the final results. A main advantage of this approach is that paths are easier to manipulate than other data structures such as trees or graphs. However, the size of the indexed paths could drastically increase in line with the size of graph database.

GCoding. In the GCoding approach (Zou *et al.*, 2008), a tree data structure is used to represent the local structure (set of surrounding nodes) associated with each graph node. Each graph node is then assigned a signature based on its local structures. A spectral graph code is generated by combining all nodes signatures in each graph. The same vertex signature can be shared in different graphs. Therefore, a vertex signature dictionary is built to store all distinct vertex signatures. Based on the derived spectral graph codes and node signatures, an algorithm is defined to check the conditions for the subgraph isomorphism tests. The filtering phase uses both of the graph codes and local node signatures to avoid introducing many false negatives candidates. In GCoding, there are two required processing steps: offline processing and online processing. In the offline processing step, the node and graph codes are computed. In the online processing step, the subgraph search is then conducted over these computed graph code databases.

GDIndex. Williams *et al.* (2007) have presented an approach for graph database indexing using a structured graph decomposition named GDIndex. In this approach, all connected and induced subgraphs of a given graph are enumerated. Therefore, a graph of size n is decomposed into at most 2^n subgraphs when each of the vertices has a unique label. However, due to isomorphism between enumerated graphs, a complete graph with multiple occurrences of the same label may decompose into fewer subgraphs. If all labels are identical, a complete graph of size n is decomposed into just $n+1$ subgraphs. A directed acyclic graph (DAG) is constructed to model the decomposed graphs and the contained relationships between them. In this DAG, there is always one node that represents the whole graph G, and one node that represents the null graph. The children of a node P are all graphs Q where there is a directed link in the DAG between P and Q. Moreover, the descendants of a node P are all nodes that are reachable from P in the DAG. Figure 4 shows a sample of graph decomposition using the GDIndex approach. A hash table is used to index the subgraphs enumerated during the decomposition process. The hash key of each subgraph is determined from the string value given by the canonical code of the subgraph. This canonical code is

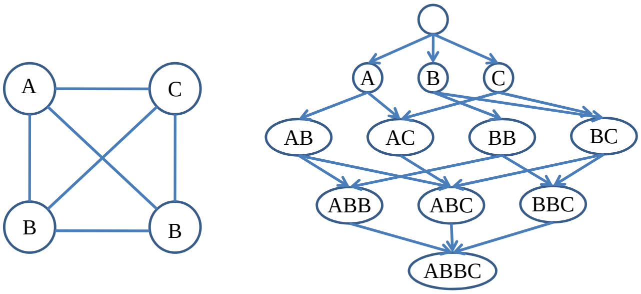


Figure 4.
Sample graph decomposition

computed from its adjacency matrix. In this way, all isomorphic graphs produce the same hash key. Since all entries in the hash table are in canonical form, only one entry is made for each unique canonical code. This hash table enables the search function to quickly locate any node in the decomposition DAG which is isomorphic to a query graph, if it exists. Therefore, in the query processing step, the hash key of the graph query *q* is computed from the query's canonical code. This computed hash value of the graph query is then used to identify and verify the set of queries that matches the canonical codes of the graph query. A clear advantage of the GDIndex approach is that no candidate verification is required. However, the index is designed for databases that consist of relatively smaller graphs and do not have a large number of distinct graphs.

GString. The GString approach (Jiang *et al.*, 2007) considers the semantics of the graph structures in the database. It focuses on modeling graph objects in the context of organic chemistry using basic structures (line, star, and cycle) that have semantic meaning and use them as indexing features. Line structure denotes a structure consisting of a series of vertices connected end to end, cycle structure denotes a structure consisting of a series of vertices that form a close loop and star structure denotes a structure where a core vertex directly connects to several vertexes. For a graph *g*, GString first extracts all cycle structures, then it extracts all star structures, and finally, it identifies the remaining structures as line structures. Figure 5 shows a sample graph representation using the GString basic structures. GString represents both graphs and queries on graphs as string sequences and transforms the subgraph search problem to the subsequent string matching domain.

A suffix tree-based index structure for the string representations is then created to support an efficient string matching process. Given a basic structure, its GString has three components: type, size, and a set of annotations (edits). For line or cycle, the size is the number of vertices in the structure. For star, the size indicates the fan out of the central vertex. For a query graph *q*, GString derives its summary string representation which is then matched against the suffix-tree of the graph database. An element of a summary string matches a node in the suffix-tree if their types match, sizes are equal or the size in the query is no more than the size in the node and the counts of corresponding types of edits in the query are no larger than those in the node. A key disadvantage of the GString approach is that converting subgraph search queries into a string matching problem could be an inefficient approach especially if the size of the graph database or the subgraph query is large. Additionally, GString focuses on decomposing chemical compounds into basic structures that have semantic meaning in the context of organic chemistry and it is not trivial to extend this approach to other domain of applications.

GraphREL. Sakr (2009) has presented a purely relational framework for processing graph queries named GraphREL. In this approach, the graph data set is encoded using

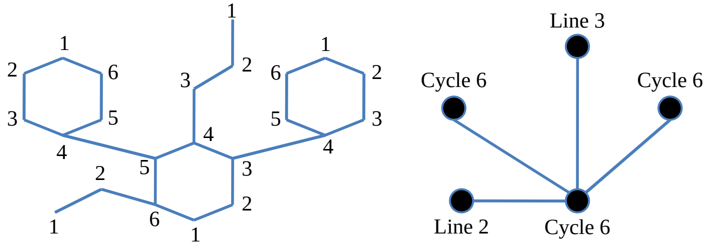
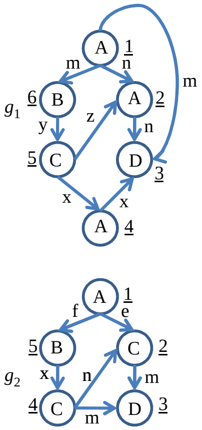


Figure 5.
Sample graph representation using GString basic structures

an intuitive vertex-edge relational mapping scheme (Figure 6) and the graph query is translated into a sequence of structured query language (SQL) evaluation steps over the defined storage scheme. An obvious problem in the relational-based evaluation approach of graph queries is the huge cost which may result from the large number of join operations which are required to be performed between the encoding relations. Several relational query optimization techniques have been exploited to speed up the search efficiency in the context of graph queries. The main optimization technique of GraphREL is based on the observation that the size of the intermediate results dramatically affects the overall evaluation performance of SQL scripts (Teubner *et al.*, 2008). Therefore, GraphREL keeps statistical information about the less frequently existing nodes and edges in the graph database in the form of simple Markov tables.

For a graph query q, the maintained statistical information is used to identify the highest pruning point on its structure (nodes or edges with very low frequency) to firstly filter out, as many as possible, the false positives graphs that are guaranteed to not exist in the final results first before passing the candidate result set to an optional verification process. This statistical information is also used to influence the decision of relational query optimizers by selectivity annotations of the translated query predicates to make the accurate decision regarding the selection of most efficient join order and the cheapest execution plan (Bruno *et al.*, 2009).

Based on the fact that the number of distinct vertices and edges labels are usually far less than the number of vertices and edges in graph databases, GraphREL utilizes the powerful partitioned B-trees indexing mechanism of the relational databases to reduce the access costs of the secondary storage to a minimum. GraphREL applies an optional verification process only if more than one vertex of the set of query vertices have the same label. For large graph queries, GraphREL applies a decomposition mechanism to divide the large and complex SQL translation query into a sequence of intermediate queries (using temporary tables) before evaluating the final results. This decomposition mechanism reuses the statistical summary information to achieve an effective selectivity-aware decomposition process and reduces the size of the intermediate result of each step.



graphID	vertexID	vLabel
1	1	A
1	2	A
1	3	D
1	4	A
1	5	C
1	6	B
2	1	A
2	2	C
2	3	D
2	4	C
2	5	B

Vertices table

graphID	sVertex	dVertex	eLabel
1	1	2	n
1	1	3	m
1	2	3	n
1	4	3	x
1	5	4	x
1	6	5	y
1	5	2	z
1	1	6	m
2	1	2	e
2	2	3	m
2	4	3	m
2	4	2	n
2	5	4	x
2	1	5	f

Edges table

Figure 6. Sample GraphREL encoding of graph databases

Mining-based graph indexing techniques

Graph-based mining. The GIndex technique (Yan *et al.*, 2004) makes use of frequent subgraphs as the basic indexing unit. The main observation of this approach is that graph-based index can significantly improve query performance over a path-based one. However, the limitation of using subgraphs as the indexing unit is that the number of graph structures is usually much larger than the number of paths in a graph database. To deal with this problem, GIndex considers only frequent subgraphs. Therefore, in order to avoid the exponential growth of the number of frequent subgraphs, the support threshold is progressively increased when the subgraphs grow large. Any subgraph is considered as being frequent if its support is greater than a minimum support threshold (Yan *et al.*, 2005a). GIndex uses low support for small subgraphs and high support for large subgraphs. Given a query graph *q*, if *q* is a frequent subgraph, the exact set of query answers containing *q* can be retrieved directly without performing any candidate verification since *q* is indexed. Otherwise, *q* probably has a frequent subgraph *f* whose support is close to the support threshold. A candidate answer set of query *q* is then retrieved and verified. If the query graph *q* is infrequent that means this subgraph is only contained in a small number of graphs in the database. Therefore, the number of subgraph isomorphism tests is going to be small. Among similar fragments with the same support, GIndex only index the smallest common fragment since more query graphs may contain the smallest fragment. Therefore, the subgraph indexes can be more compact and more effective.

Cheng *et al.* (2007) have extended the ideas of GIndex (Yan *et al.*, 2004) by using nested inverted-index in a new graph index structure named FG-Index. In this index structure, a memory-resident inverted-index is built using the set of frequent subgraphs. Adisk-resident inverted-index is built on the closure of the frequent graphs. If the closure is too large, a local set of closure frequent subgraphs can be computed from the set of frequent graphs and a further nested inverted-index can be constructed. Another edge-index can be also built on the set of infrequent distinct edges in the graph database. Cheng *et al.* (2009) have extended their work further where they introduced the *FG*-index* which consists of three components: the FG-index, feature-index and the FAQ-index. The FG-index allows the set of queries that represent frequent graphs to be answered without candidate verification. The feature-index is used to reduce the index probing cost by using the features to filter false results that are matched in the FG-index. The FAQ-index is dynamically constructed from the set of frequently asked non-frequent subgraphs. Hence, verification is not required for processing frequently asked queries and only a small number of candidates need to be verified for processing non-frequent graph queries that are not frequently asked.

Tree-based mining. Zhang *et al.* (2007) have presented an approach for using frequent subtrees as the indexing unit for graph structures named TreePI. The main idea of this approach is based on two main observations:

1. Tree data structures are more complex patterns than paths and trees as it can preserve almost equivalent amount of structural information as arbitrary subgraph patterns.
2. The frequent subtree mining process is relatively easier than the general frequent subgraph mining process.

Therefore, TreePI starts by mining the frequent tree on the graph database and then selecting a set of frequent trees as index patterns. In the query processing, for a query graph *q*, the frequent subtrees in *q* are identified and then matched with the set of indexing features to obtain a candidate set. In the verification phase, the advantage of the location information partially stored with the feature trees is utilized for devising efficient subgraph isomorphism tests. As the canonical form of any tree can be calculated in polynomial time, the indexing and searching operations can be effectively improved. Moreover, operations on trees, such as isomorphism and normalization are asymptotically simpler than graphs, which are usually NP-complete (Fortin, 1996).

Zhao *et al.* (2007) have extended the ideas of TreePi (Zhang *et al.*, 2007) to achieve better pruning ability by adding a small number of discriminative graphs (Δ) to the frequent tree-features in the index structure. They propose a new graph indexing mechanism, named (Tree + Δ), which first selects frequent tree-features as the basis of a graph index, and then, on-demand, selects a small number of discriminative graph-features that can prune graphs more effectively than the selected tree-features, without conducting costly graph mining beforehand.

Super-graph query processing

The supergraph query processing problem is important in practice. However, it has not been extensively considered in the research literature. Only two approaches have been presented to deal with this problem. Chen *et al.* (2007) have presented an approach named cIndex (contrast index). The indexing unit of this approach is the subgraphs which are extracted from graph databases based on the rarity of their occurrence in historical query graphs. Sometimes, the extracted subgraphs are very similar to each other. Therefore, cIndex tries to find a set of contrast subgraphs that collectively perform well. It uses a redundancy-aware selection mechanism to sort out the most significant and distinctive contrast subgraphs. The distinctive subgraph is stored in a hierarchical fashion using bottom-up and top-down proposed approaches. During query processing, cIndex reduces the number of subgraph isomorphism testings by using the filtering and verification methodology. An advantage of the cIndex approach is that the size of the feature index is small. On the other hand, the query logs may frequently change over time so that the feature index could become outdated quite often and thus need to be recomputed to stay effective.

Zhang *et al.* (2009) have investigated the supergraph query processing problem from another angle. They proposed an approach for compact organization of graph database members named GPTree. In this approach, all of the graph database members are stored into one graph where the common subgraphs are stored only once. An algorithm for extracting the key features from graph databases is used to construct the feature indexes. Based on the containment relationship between the sets of the extracted features, a mathematical approach is used to determine the ordering of the feature set which can reduce the number of subgraph isomorphism tests during query processing.

Graph similarity queries

The problem of similarity (approximate) subgraph queries has been addressed by different research efforts in the literature. Given a query graph and a database of graphs, these approaches try to find subgraphs in the database that are similar to the query. Therefore, these approaches can allow for node mismatches, node gaps (gap node is a node in the query that cannot be mapped to any node in the target graph) as well as graph structural differences. Approximate graph matching techniques are used in some cases when the graph databases are noisy or incomplete. In these cases, using approximate graph matching query processing techniques can be more useful and effective than exact matching. In this section we give an overview of the main approaches which address this problem.

Grafil

Yan *et al.* (2005b) have proposed a feature-based structural filtering algorithm, named Grafil (graph similarity filtering) to perform substructure similarity search in graph databases. Grafil models each query graph as a set of features and transforms the edge deletions into the feature misses in the query graph. With an upper bound on the maximum allowed feature misses, Grafil can filter many graphs directly without performing pair-wise similarity computation. It uses two data structures: feature-graph matrix and edge-feature matrix.

The feature-graph matrix is used to compute the difference in the number of features between a query graph and graphs in the database. In this matrix, each column corresponds to a target graph in the graph database, each row corresponds to a feature being indexed and each entry records the number of the embeddings of a specific feature in a target graph. The edge-feature matrix is used to compute a bound on the maximum allowed feature misses based on a query relaxation ratio. In this matrix, each row represents an edge while each column represents an embedding of a feature. Grafil uses a multi-filter composition strategy where each filter uses a distinct and complimentary subset of the features. The filters are constructed by a hierarchical, one dimensional clustering algorithm that groups features with similar selectivity into a feature set.

During the query processing, the feature-graph matrix is used to calculate the difference in the number of features between each graph database member g_i and the query q. If the difference is greater than a user-defined parameter d_{max} then it is discarded while the remaining graphs constitute a candidate answer set. The substructure similarity is then calculated for each candidate to prune the false positives candidates. A loop of query relaxation steps can be applied if the user needs more matches than those returned from the current value of d_{max}.

PIS

Yan *et al.* (2006) have extended their original work (Yan *et al.*, 2005b) where they proposed another indexing mechanism named partition-based graph index and search (PIS). PIS builds a fragment-based index on the graph database. In this index, graph database members are decomposed into overlapping fragments where fragments with the same topology are indexed using R-tree data structure. It uses a superimposed distance measure between a query graph and a target graph which is lower-bounded by the sum of distances between their corresponding non-overlapping fragments. This lower bound leads to efficient pruning of most invalid graphs in the database.

A query graph is partitioned into fragments according to the index structure. Each query graph q is partitioned into a set of indexed non-overlapping fragments $(g_1, g_2, \ldots, g_n)$. For each fragment g_i, its equivalence class in the index is determined and a range query $d(g_i, g_*) \leq \sigma$ is submitted to find all fragments that meet the superimposed distance threshold. Each g_i with lower bound distance greater than σ is dropped from the candidate answer set. The real superimposed distance between q and the candidate graphs is calculated and then graphs that do not satisfy the distance threshold are removed. Each query graph can be partitioned in multiple ways. However, PIS tries to choose the optimal one that achieves the best pruning performance by identifying the criterion of an optimal partition that should give a set of non-overlapping fragments with the highest selectivity.

Closure tree

He and Singh (2006) have proposed a tree-based index structure named CTree (closure-tree). CTree index is very similar to the R-tree indexing mechanism (Guttman, 1984) but extended to support graph matching queries. In this index structure, each node in the tree contains discriminative information about its descendants in order to facilitate effective pruning. The closure of a set of vertices is defined as a generalized vertex whose attribute is the union of the attribute values of the vertices. Similarly, the closure of a set of edges is defined as a generalized edge whose attribute is the union of the attribute values of the edges. The closure of two graphs g_1 and g_2 under a mapping M is defined as a generalized graph (V, E) where V is the set of vertex closures of the corresponding vertices and E is the set of edge closures of the corresponding edges (Figure 7). Hence, a graph closure has the same characteristics of a graph. However, the only difference is that the graph database member has singleton labels on vertices and edges while the graph closure can have multiple labels.

In a closure tree, each node is a graph closure of its children where the children of an internal node are nodes and the children of a leaf node are database graphs. A subgraph query is processed in two phases. The first phase traverses the CTree and the nodes are prunned based on a pseudo subgraph isomorphism where a candidate answer set is returned. The second phase verifies each candidate answer for exact subgraph isomorphism and returns the answers. In addition to pruning based on pseudo subgraph isomorphism, a lightweight histogram-based pruning operation is also employed. The histogram of a graph is a vector that counts the number of each distinct attribute of the vertices and edges. For similarity queries, CTree defines graph similarity based on edit distance, and computes it using heuristic graph mapping methods. It conceptually approximate subgraph isomorphism by sub-isomorphism using adjacent subgraphs after which it approximates sub-isomorphism by using adjacent subtrees.

SAGA

Tian *et al.* (2007) have presented an approach of approximate graph query matching named substructure index-based approximate graph alignment (SAGA). SAGA measures graph similarity by a distance value such that graphs that are more similar have a smaller distance. The distance model contains three components:

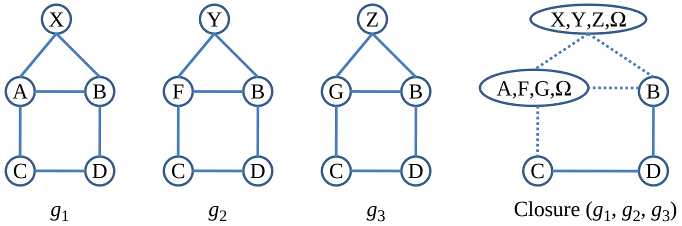


Figure 7.
Graph closure

(1) The StructDist component measures the structural differences for the matching node pairs in the two graphs.

(2) The NodeMismatches component is the penalty associated with matching two nodes with different labels.

(3) The NodeGaps component is used to measure the penalty for the gap nodes in the query graph.

SAGA index is built on small substructures of graphs in the database (fragment index). Each fragment is a set of k nodes from the graphs in the database where K is a user-defined parameter. The index does not enumerate all possible k-node sets. It uses another user-defined parameter $dist_{max}$ to avoid indexing any pair of nodes in a fragment if its distance measure is greater than $dist_{max}$. The fragments in SAGA do not always correspond to connected subgraphs. The reason behind this is to allow node gaps in the matching process. To efficiently evaluate the subgraph distance between a query graph and a database graph, an additional index called DistanceIndex is also maintained. This index is used to look up the precomputed distance between any pair of nodes in a graph. The graph matching process goes through the following three steps:

(1) The query is broken into small fragments and the fragment index is probed to find database fragments that are similar to the query fragments.

(2) The hits from the index probes are combined to produce larger candidate matches. A hit-compatible graph is built for each matching graph. Each node in the hit-compatible graph corresponds to a pair of matching query and database fragments. An edge is drawn between two nodes in the hit-compatible graph if and only if two query fragments share zero or more nodes, and the corresponding database fragments in the hit-compatible graph also share the same corresponding nodes. An edge between two nodes tells us that the corresponding two hits can be merged to form a larger match which is then a candidate match.

(3) Each candidate is examined to produce the query results. For each candidate, the percentage of the gap nodes is checked. If it exceeds a user-defined threshold P_g, then the candidate match is ignored otherwise the DistanceIndex is probed to calculate the real subgraph matching distance. If two matches have the same matching distance and one is a submatch of the other, only the supermatch is considered.

TALE – periscope

Tian and Patel (2008) and Tian *et al.* (2008) have presented an approach to deal with the problem of approximate subgraph matching of large query graphs named a tool for approximate subgraph matching of large queries efficiently (TALE). Specifically, given a large query graph, with hundreds to thousands of nodes and edges, and a database of large graphs, the approach tries to find the subgraphs in the database that are similar to the query. TALE employs a graph indexing method, called neighborhood index (NH-Index). The indexing unit of NH-Index is the neighborhood of each database node. The neighborhood concept captures the local graph structure around each node.

Three main properties are used to characterize the neighborhood of a node: the number of neighbors (node degree), how the neighbors connect to each other and the labels of the actual neighbors. The number of indexing units is equal to the number of nodes in the database. The NH-Index is a disk-based index, which allows handling graph databases that do not fit in memory. The NH-Index is used by the matching algorithm to match the important nodes in the query graph. These initial matches for the important nodes are then extended to produce the final matching results. A database node matches the query node, only if the two nodes match and their neighborhoods also match. Specifically, in order to match a query node to a database node, the two nodes must have the same label. The degree of the query node should be no more than that of the database node. The same condition holds for neighbor connections. In addition, the neighbors of the query node should have corresponding matching nodes in the neighborhood of the database node.

Discussion and conclusions

In this study, we have investigated the problem of graph indexing and querying techniques, a problem raised by the continued emergence and increase of massive and complex structural data. Graph querying is a critical problem in graph database management. Several techniques have been proposed to tackle the performance challenges of different graph query types in a general mode. However, due to the diversity of graph models as well as the complexity of graph processing, some of the existing graph indexing and query processing techniques are designed solely for specific application domains (Jiang *et al.*, 2007; Tian *et al.*, 2007).

Most of the graph indexing strategies shift the high online graph query processing cost to the off-line index construction phase. Index construction is thus, always computationally expensive because it requires the use of high quality indexing features with great pruning power from the graph database. However, the number of indexing features should be as small as possible to keep the whole index structure compact so that it is possible to be held in the main memory for efficient access and retrieval. Hence, a high quality graph indexing mechanism should be time-efficient in index construction and indexing features should be compact and powerful for pruning purposes. In this context, a clear metric for evaluating the pruning strategy is the size of the candidate answer set in comparison to the actual answer set. The closer the size of the candidate answer set to that of the actual answer set, the better the evaluation of the strategy would be. Figure 8 shows a timeline representation of the graph indexing techniques. Table I provides a comparison between the different graph indexing techniques in terms of their supported query types, indexing unit and indexing strategy.

A clear gap in the research efforts in the domain of graph database is the absence of a standard graph query language which plays the same role as that of SQL for the relational data model or XPath and XQuery for the XML hierarchical model. Although there are a number of query languages that have been proposed in the literature (Consens and Mendelzon, 1990; Gyssens *et al.*, 1994; Prud'hommeaux and Seaborne, 2008; Sheng *et al.*, 1999; Srinivasa *et al.*, 2005), none of them has been universally accepted as they are designed to deal with different representations of the graph data model. For example, GOQL (Sheng *et al.*, 1999) is designed as an extension to OQL and uses an object-oriented graph data model. SPARQL query language (Prud'hommeaux and Seaborne, 2008) is a W3C recommendation for querying RDF graph data. It describes a directed labeled graph by a set of triples, each of which describes a (attribute and value) pair or an interconnection between two nodes. The SPARQL query language works primarily through a primitive triple pattern matching techniques with simple constraints on query nodes and edges. A standard definition of a general purpose

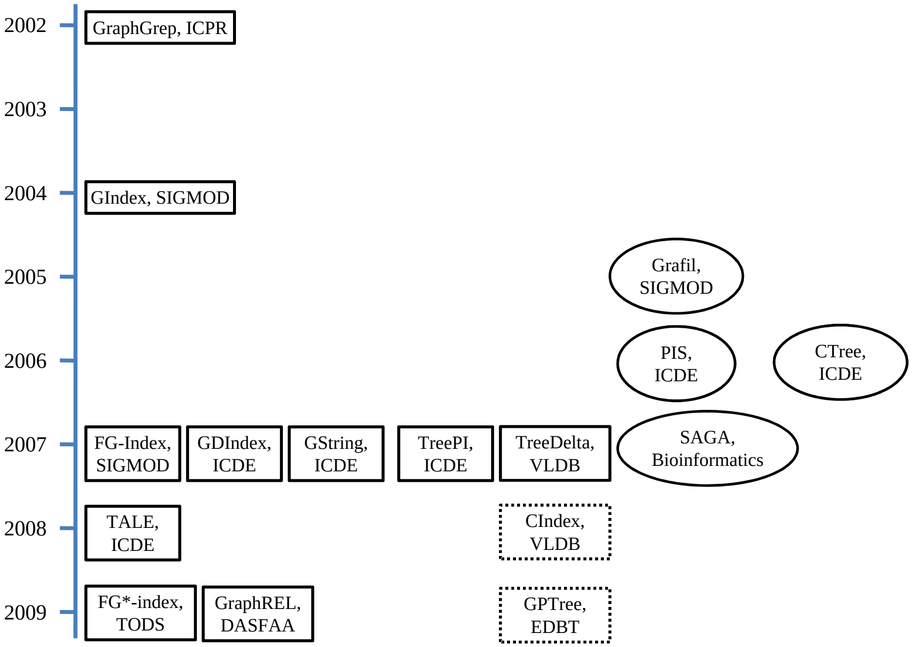


Figure 8.
Time line representation of graph indexing techniques

Notes: Solid-lined rectangles denote techniques which are supporting subgraph queries, dashed-lined rectangles denote techniques which are supporting supergraph queries, and ovals denote techniques which are supporting similarity queries

graph query language with more powerful and flexible constructs is essentially required. A concrete algebra behind this expected query language is also quite important.

By analyzing the proposed graph query processing techniques, we observed that all of them concentrate on the retrieval speed of their indexing structures in addition to their compact size. However, further management of these indexes are rarely taken into account. Although efficient query processing is an important objective, efficient update maintenance is also an important concern. In the case of dynamic graph databases, it is quite important that indexing techniques avoid the costly recomputation of the whole index and provide more efficient mechanisms to update the underneath index structures with minimum effort. Therefore, efficient mechanisms to handle dynamic graph databases are necessary.

Finally, query processing usually involves a cost-based optimization phase in which query optimizers rely on cost models to attempt on choosing an optimal query plan from amongst several alternatives. A key issue of any cost model is the cardinality estimation of the intermediate and final query results. Although there is an initial effort has been proposed by (Stocker *et al.*, 2008) for estimating the selectivity estimation of basic graph patterns, there is still a clear need for summarization and estimation frameworks for graph databases. These frameworks need to provide accurate selectivity estimations of more complex graph patterns which can be utilized in accelerating the processing of different types of graph queries.

Table I. Summary of the graph indexing and querying techniques

Indexing technique	Supported query types	Indexing unit	Indexing mechanism
CIndex (Chen *et al.*, 2007)	Supergraph queries	Subgraph structure	Rarely occurring features
Closure-tree (He and Singh, 2006)	Subgraph and similarity	Closure tree	Enumeration of graph closures
FG-Index (Cheng *et al.*, 2007)	Subgraph queries	Subgraph structure	Frequent features
GDIndex (Williams *et al.*, 2007)	Subgraph queries	Decomposed subgraph	Full enumeration
GIndex (Yan *et al.*, 2004)	Subgraph queries	Subgraph structure	Frequent features
GPTree (Zhang *et al.*, 2009)	Supergraph queries	Subgraph structure	Full enumeration
Grafil (Yan *et al.*, 2005b)	Similarity queries	Any	Full enumeration
GraphGrep (Giugno and Shasha, 2002)	Subgraph queries	Path structure	Full enumeration
GraphREL (Sakr, 2009)	Subgraph queries	Nodes and edges	Full enumeration
GString (Jiang *et al.*, 2007)	Subgraph queries	Subgraph structure	Full enumeration
PIS (Yan *et al.*, 2006)	Similarity queries	Subgraph structure	Full enumeration
SAGA (Tian *et al.*, 2007)	Similarity queries	Subgraph structure	Full enumeration
TALE (Tian and Patel, 2008)	Subgraph and similarity	Neighborhood structure	Full enumeration
TreeDelta (Zhao *et al.*, 2007)	Subgraph queries	Tree structure	Frequent features
TreePi (Zhang *et al.*, 2007)	Subgraph queries	Tree structure	Frequent features

References

Angles, R. and Gutiérrez, C. (2008), "Survey of graph database models", *ACM Computing Surveys*, Vol. 40 No. 1, pp. 1-39.

Bruno, N., Chaudhuri, S. and Ramamurthy, R. (2009), "Power hints for query optimization", *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, March 29-April 2*, pp. 469-80.

Bunke, H. and Shearer, K. (1998), "A graph distance metric based on the maximal common subgraph", *Pattern Recognition Letters*, Vol. 19 Nos 3-4, pp. 255-9.

Cai, D., Shao, Z., He, X., Yan, X. and Han, J. (2005), "Community mining from multi-relational networks", *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Porto, October 3-7*, pp. 445-52.

Chen, C., Yan, X., Yu, P.S., Han, J., Zhang, D.-Q. and Gu, X. (2007), "Towards graph containment search and indexing", *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), University of Vienna, September 23-27*, pp. 926-37.

Cheng, J., Ke, Y. and Ng, W. (2009), "Efficient query processing on graph databases", *ACM Transactions of Database Systems (TODS)*, Vol. 34 No. 1.

Cheng, J., Ke, Y., Ng, W. and Lu, A. (2007), "FG-index: towards verification-free query processing on graph databases", *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, June 11-14*, pp. 857-72.

Consens, M.P. and Mendelzon, A.O. (1990), "GraphLog: a visual formalism for real life recursion", *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Nashville, TN, April 2-4*, pp. 404-16.

Fernández, M.-L. and Valiente, G. (2001), "A graph distance metric combining maximum common subgraph and minimum common supergraph", *Pattern Recognition Letters*, Vol. 22 Nos 6/7, pp. 753-8.

Fortin, S. (1996), *The Graph Isomorphism Problem, Tech. Rep.*, Department of Computing Science, University of Alberta, Edmonton.

Gao, X., Xiao, B., Tao, D. and Li, X. (2010), "A survey of graph edit distance", *Pattern Analysis and Applications*, Vol. 13 No. 1, pp. 113-29.

Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, CA.

Giugno, R. and Shasha, D. (2002), "GraphGrep: a fast and universal method for querying graphs", *IEEE International Conference in Pattern Recognition (ICPR)*, pp. 112-15.

Guttman, A. (1984), "R-trees : a dynamic index structure for spatial searching", *Proceedings of the ACM SIGMOD International Conference on Management of Data, Boston, MA*, pp. 47-57.

Gyssens, M., Paredaens, J., den Bussche, J.V. and Gucht, D.V. (1994), "A graph-oriented object database model", *IEEE Transactions of Knowledge and Data Engineering (TKDE)*, Vol. 6 No. 4, pp. 572-86.

He, H. and Singh, A.K. (2006), "Closure-tree: an index structure for graph queries", *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA, April 3-7*, pp. 38-52.

Huan, J., Wang, W., Bandyopadhyay, D., Snoeyink, J., Prins, J. and Tropsha, A. (2004), "Mining protein family specific residue packing patterns from protein structure graphs", *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology (RECOMB), San Diego, CA, March 27-31*, pp. 308-15.

Jiang, H., Wang, H., Yu, P.S. and Zhou, S. (2007), "GString: a novel approach for efficient search in graph databases", *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, April 15-20*, pp. 566-75.

Klinger, S. and Austin, J. (2005), “Chemical similarity searching using a neural graph matcher”, *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN), Bruges, April 27-29*, pp. 479-84.

Kuramochi, M. and Karypis, G. (2001), “Frequent subgraph discovery”, *Proceedings of the IEEE International Conference on Data Mining (ICDM), San Jose, CA, November 29-December 2*, pp. 313-20.

Kuramochi, M. and Karypis, G. (2004), “GREW-A scalable frequent subgraph discovery algorithm”, *Proceedings of the IEEE International Conference on Data Mining (ICDM), Brighton, November 1-4*, pp. 439-42.

Kyoto Encyclopedia of Genes and Genomes (KEGG) (2010), available at: www.genome.jp/kegg/

Lee, J., Oh, J.-H. and Hwang, S. (2005), “STRG-Index: spatio-temporal region graph indexing for large video databases”, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, MD, June 14-16*, pp. 718-29.

Manola, F. and Miller, E. (2004), “RDF primer: world wide web consortium proposed recommendation”, available at: www.w3.org/TR/rdf-primer/

National Cancer Institute (NCI) (2010), available at: http://dtp.nci.nih.gov/

Prud’hommeaux, E. and Seaborne, A. (2008), “SPARQL query language for RDF: world wide web consortium proposed recommendation”, available at: www.w3.org/TR/rdf-sparql-query/

Raymond, J.W., Gardiner, E.J. and Willett, P. (2002), “RASCAL: calculation of graph similarity using maximum common edge subgraphs”, *Computer Journal*, Vol. 45 No. 6, pp. 631-44.

Sakr, S. (2009), “GraphREL: a decomposition-based and selectivity-aware relational framework for processing sub-graph queries”, *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA), Brisbane, April 21-23*, pp. 123-37.

Sakr, S. and Awad, A. (2010), “A framework for querying graph-based business process models”, *Proceedings of the 19th International World Wide Web Conference (www), Raleigh, NC.*

Sayed, A. (2009), “Fast and efficient computation of reachability queries over linked XML documents’ graphs”, *International Journal of Web Information Systems (IJWIS)*, Vol. 5 No. 1, pp. 56-76.

Sheng, L., Özsoyoglu, Z.M. and Özsoyoglu, G. (1999), “A graph query language and its query processing”, *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, IEEE Computer Society, Washington, DC, pp. 572-81.

Srinivasa, S., Maier, M., Mutalikdesai, M.R., Gowrishankar, K.A. and Gopinath, P.S. (2005), “LWI and safari: a new index structure and query model for graph databases”, *Proceedings of the Eleventh International Conference on Management of Data (COMAD), Goa, January*, pp. 138-47.

Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C. and Reynolds, D. (2008), “SPARQL basic graph pattern optimization using selectivity estimation”, *Proceedings of the 17th International Conference on World Wide Web (www), Beijing*, pp. 595-604.

Teubner, J., Grust, T., Maneth, S. and Sakr, S. (2008), “Dependable cardinality forecasts for XQuery”, *Proceedings of the VLDB Endowment (PVLDB), Auckland, August 23-28*, Vol. 1 (1) pp. 463-77.

The Gene Ontology (2010), available at: www.geneontology.org/

Tian, Y. and Patel, J.M. (2008), “TALE : a tool for approximate large graph matching”, *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, April 7-12*, pp. 963-72.

Tian, Y., McEachin, R.C., Santos, C., States, D.J. and Patel, J.M. (2007), “SAGA: a subgraph matching tool for biological graphs”, *Bioinformatics*, Vol. 23 No. 2, pp. 232-9.

Tian, Y., Patel, J.M., Nair, V., Martini, S. and Kretzler, M. (2008), "Periscope/GQ: a graph querying toolkit", *Proceedings of the VLDB Endowment (PVLDB)*, Vol. 1 No. 2, pp. 1404-7.

Wang, C., Wang, W., Pei, J., Zhu, Y. and Shi, B. (2004), "Scalable mining of large disk-based graph databases", *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Seattle, Washington, DC, August 22-25*, pp. 316-25.

Washio, T. and Motoda, H. (2003), "State of the art of graph-based data mining", *SIGKDD Explorations*, Vol. 5 No. 1, pp. 59-68.

Westbrook, J.D., Feng, Z., Chen, L., Yang, H. and Berman, H.M. (2003), "The protein data bank and structural genomics", *Nucleic Acids Research*, Vol. 31 No. 1, pp. 489-91.

Williams, D.W., Huan, J. and Wang, W. (2007), "Graph database indexing using structured graph decomposition", *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul*, pp. 976-85.

Yan, X. and Han, J. (2002), "gSpan: graph-based substructure pattern mining", *Proceedings of the IEEE International Conference on Data Mining (ICDM), Maebashi City, December 9-12*, pp. 721-4.

Yan, X. and Han, J. (2003), "CloseGraph: mining closed frequent graph patterns", *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), New York, NY*, pp. 286-95.

Yan, X., Yu, P.S. and Han, J. (2004), "Graph indexing: a frequent structure-based approach", *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, June 13-18*, pp. 335-46.

Yan, X., Yu, P.S. and Han, J. (2005a), "Graph indexing based on discriminative frequent structure analysis", *ACM Transactions on Database Systems (TODS)*, Vol. 30 No. 4, pp. 960-93.

Yan, X., Yu, P.S. and Han, J. (2005b), "Substructure similarity search in graph databases", *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, MD, June 14-16*, pp. 766-77.

Yan, X., Zhu, F., Han, J. and Yu, P.S. (2006), "Searching substructures with superimposed distance", *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Washington, DC*, pp. 88-98.

Zhang, S., Hu, M. and Yang, J. (2007), "TreePi: a novel graph indexing method", *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, 15-20 April*, pp. 966-75.

Zhang, S., Li, J., Gao, H. and Zou, Z. (2009), "A novel approach for efficient supergraph query processing on graph databases", *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint Petersburg, March 24-26*, pp. 204-15.

Zhao, P., Yu, J.X. and Yu, P.S. (2007), "Graph indexing: tree+delta ≥ graph", *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), Vienna*, pp. 938-49.

Zou, L., Chen, L., Yu, J.X. and Lu, Y. (2008), "A novel spectral coding in a large graph database", *Proceedings of the 11th International Conference on Extending Database Technology (EDBT), Nantes, March 25-29*, pp. 181-92.

Corresponding author
Sherif Sakr can be contacted at: ssakr@cse.unsw.edu.au