Index Based Graph Algorithm (IBGA) for Query Optimization in Graph Databases

D. Neelakant[1], E. Pradeep[2], D. Prashanth Kumar[3]
[1, 2,3]Assistant Professor
[1,2,3]Kakatiya Institute of Technology and Science, Huzurabad, Telangana, India.

ABSTRACT
Now a day's collecting the huge amounts of complex information is very common. It is very difficult to represent, store and manipulate complex information, data can be change in time or change in their structure, it is very difficult to represent in a system to store and maintain. To overcome the problems of traditional databases for storing, manipulating and retrieving data we have to use graph structures. It can be used by many applications that manage graph data by implementing queries. In this paper we proposed Index Based Graph Alignment algorithm for GDB. The main characteristics of algorithm provide dynamic structure, highly interconnected data, and ability to efficiently access data relationships in database.

Keywords: RDBMS, Graph DB, Queries, Indexes, Distribution.

1. INTRODUCTION:

Many applications managing data with graph structure such as social networks, service providers, communication networks...etc. These applications are manage trillions of interconnected data, which construct the graph structure. User queries in these applications are more interested on the relationships between data rather than on the nodes of the graph.

Increase the no of these applications and the requirement to access complex data relations , graph databases (GDB) to be the most efficient systems for storing and querying graph data. The implementation of graph databases have been introduced several years ago[1], but the ability of traditional database systems, especially relational databases and XML manage little data graphs. Today, traditional databases are enable to store large graphs and querying their complex relationships, which need complex joins of relational tables. MongoDB databases are efficiently to store large volume of historic data but this efficiency decreases when querying highly interconnected data with dynamic structure.

Current representation and storage systems are not very flexible in dealing with big changes and also they are not concerned with the ability of performing complex data manipulations. On the other hand, data manipulation systems cannot easily work with structural or relational data, but just with flat data representations. We want to bridge the gap between the two, by introducing a new type of database structure, called Graph Databases (GDB), based on a natural graph representation. Our Graph Databases are able to represent as graphs any kind of information, naturally accommodate changes in data, and they also make easier for Machine Learning methods to use the stored information. In figure 1 the retrieval of information and modeling of information is shown. Usually data resides in multiple locations, is autonomously maintained, and may change in time, thus posing even more challenges. Being autonomous, it is possible that the data sources don't have unified schema or we cannot control their schemas, match schemas of different data sources, how to obtain a global schema from multiple schemas, and how to change the global schema according to possible changes in data sources schemas.(RDBMS,OOXML).However, neither the objects nor the trees can naturally represent graphs, which are the most general data structure. Subsequently, they do not support natural queries on graphs. Moreover, when changes in schema of the data sources occur, these approaches may lead to major restructuring (redesigning) of global schema.

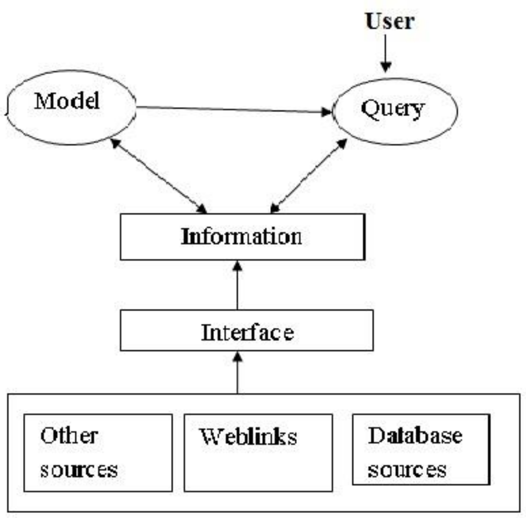


Figure 1: Information retrieval and modelling

2. RELATED WORK

Graph databases are new storage systems that manage highly interconnected data. propose typical solutions for recent applications for graph- data structure, such as social networks, communication networks, etc. It the process of developing standard tools for design models and query languages, which represent the two main elements of any database project.

2.1 Definition and examples of graph databases

All the studied papers, in this work, agree that a graph database is a system to represent, store and manage data, which are naturally interconnected and structured in a form of graph. The represented data are those contained in both nodes (vertices) and edges of the graph. According to[4], the term graph database embodies two main characteristics: First, a graph database is a storage system whose native representation of data is in terms of objects (vertices) and inter object relationships (edges); Second, a graph database supports single-object access via indexed lookup or iteration. That is, in addition to enabling whole-graph analysis, we require that a graph database be able to efficiently answer queries about the attributes and relationships of specific elements. The authors of [9]provide a formal definition of graph database using graph terminology.

A graph database as a finite edge-labeled graph ,i.e ∑ be a finite alphabet and V a infinite set of node ids, then a graph database over ∑ is a pair G=(N,E)where N is the set of nodes and E is the set of edges can be represented as

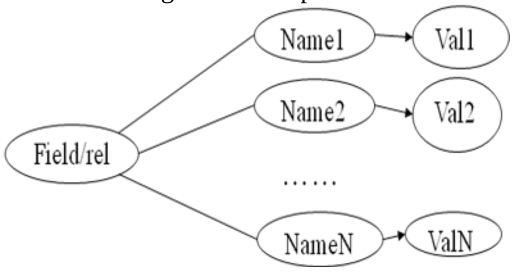


Figure 2: Attribute representation for the relation in the GDB

The generalization of graph database instances is called schema or model for the design and implementation of graph databases. The different models of graph databases [5]. The two main elements that make up all these models are nodes and relationships. The nodes represents entities and can hold any number of attributes. They can be tagged with labels representing their different roles in the studied domain. The relationships provide directed connections between two nodes. In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths. Because of these properties, two nodes can share any number of relationships.

For example we show how a relational database can be transformed into a graph databases using following example of RDBMS. We assume that we have three tables, Sailors, Boats and Reserves, whose schemas are given below[9]

Table 1: Sailors relation

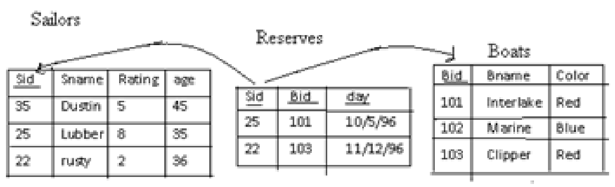
Sid	Sname	Rating	age
35	Dustin	5	45
25	Lubber	8	35
22	rusty	2	36

Table 2: Reserves relation

Sid	Bid	day
25	101	10/5/96
22	103	11/12/96

Table 3: Boat relation

Bid	Bname	Color
101	Interlake	Red
102	Marine	Blue
103	Clipper	Red



Sid	Sname	Rating	age
35	Dustin	5	45
25	Lubber	8	35
22	rusty	2	36

Sid	Bid	day
25	101	10/5/96
22	103	11/12/96

Bid	Bname	Color
101	Interlake	Red
102	Marine	Blue
103	Clipper	Red

Figure 3: Relations and dependencies

- Sailors(sid:integer, sname:char(10), rating: integer, age:integerl)
- Boats(bid:integer, bname:char(10), color:char(10))
- Reserve(sid:integer, bid:integer, day:date)

In figure 4 we represent how a graph can be implemented for the above relations, where IO represents Instance Of the relation, TB denotes table of relation and values and F1,F2,…. represents fields in a relation.

2.2 Query Language

To represent a query by a graph similar to the graph representing an intentional definition, and callit query graph (CG). Actually, queries are graphs to be matched in the big graph containing facts, intentional and extensional definitions. A query can be written formally as: Query :- QG.

Figure 4:Graph Representation for relation

The semantic definition of a query model can be informally viewed in Figure 5. Thus, when a query is asked, it is matched with the big data and knowledge graph.

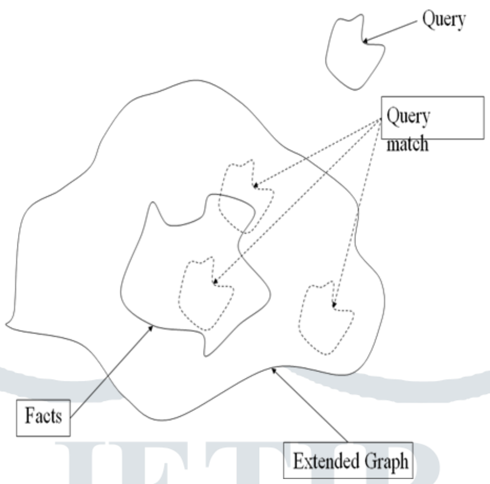


Figure 5:Semantic query model for GDB

3. PROPOSED ALGORITHM (IBGA)

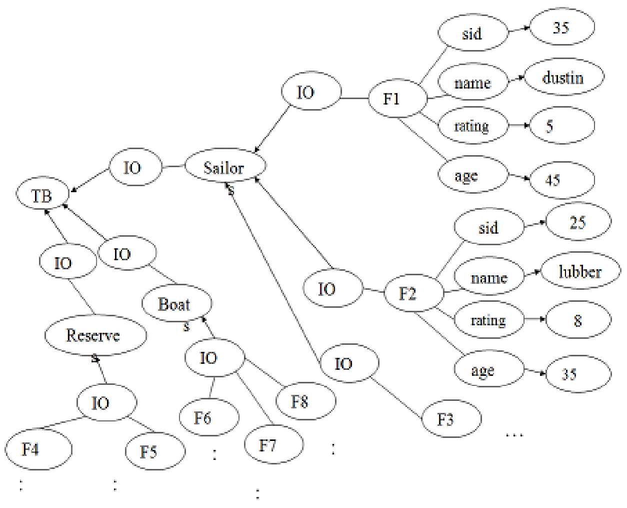
IBGA is a approximate graph matching algorithm. Given a query graph and a database of graphs, we can to find sub graphs in the database that are similar to the query, that allow for node mismatches, node gaps (node insertions or deletions), as well as graph structural differences. Node mis-matches model the behavior that two nodes representing different entities in the graph.

Node gaps represent the situation where a certain node in one graph cannot be mapped to any node in the other graph. Graph structural differences allow for differences in node connectivity relationships. For example, two nodes may be directly connected in one graph, whereas the corresponding matching nodes in the other graphs may be indirectly related through one or more additional nodes.

In this paper the work has focused on to exact sub graph, GIndex are index-based filtering methods .Path Aligner [7],[8] is a tool for aligning pathways. However, it assumes that all path-ways are linear paths. The tools most closely related to our work are Path Blast [6]and the successive Network Blast [8], which are designed for aligning protein interaction networks. Their graph similarity model allows node mismatches and node gaps, but graph structural differences are largely confined to short paths.

Another related method [7] has been proposed for aligning protein interaction networks. However, the match technique used in this method largely focuses on capturing the penalty associated with gene duplication. Finally, PathBlast, Net-work Blast, and the method proposed in [7] can only perform one graph comparison at a time. To match a query against a database of graphs, the matching algorithms must be run for each graph in the database

We present a novel approximate subgraph matching technique called IBGA. At the heart of IBGA is a flexible model for computing graph , which permits node gaps, node mismatches, and graph structural differences. To speed up the execution of queries with this powerful matching model, we employ an indexing method for efficient query evaluation. IBGA allows additional information derived from the relationships between entities in path- ways to be incorporated into comparative analysis.



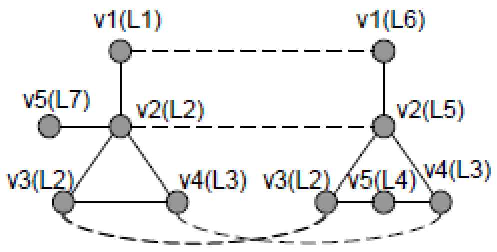


Figure 6:Subgraph matching

3.1 Subgraph Matching

In our model, a graph, G, is a 3-tuple G = (V,E, Ø). V is the set of nodes and E ⊆ V X V is the set of (directed or undirected) edges. Nodes in the graphs have labels specified by the mapping Ø : V→ L, where L is the set of node labels.

Formally, the subgraph matching is defined as follows: Let G1 = (V1;E1; Ø1) and G2 = (V2;E2; Ø2) be two graphs. An approximate matching from G1 (the query) to G2 (the target) is a bijection mapping function λ: V1 <→ V2..

$$SGD_{\lambda}(G1,G2) = w_e \text{ X } StrucDIST_{\lambda} + w_n \text{ X } NodeMissmat_{\lambda} + w_g \text{ X } Nodegaps_{\lambda} \qquad \text{i}$$

$$StrucDIST_{\lambda} = \sum_{u,v\in V1,u<v} d_{G1}(u,v) - d_{G2}(\lambda u, \lambda v) \qquad \text{ii}$$

$$NodeMissmat_{\lambda} = \sum_{u\in V1} mismatch(Ø1(u), Ø2(\lambda u)) \qquad \text{iii}$$

$$Nodegaps_{\lambda} = \sum_{u\in V1} gap_{G1}(u) \qquad \text{iv}$$

The distance model contains three components. The StructDist component measures the structural differences of the match, the NodeMismatches component is the penalty associated with matching two nodes with different labels, and the NodeGapscomponent is used to measure the penalty for the gap nodes. (Gap nodes are nodes in the query that cannot be mapped to any nodes in the target graph.)

$$SGD(G1;G2) = \min SGD_{\lambda}(G1;G2) \qquad \text{v}$$

3.2 The NodeGaps component

The NodeGaps component in Equation iv measures the penalties associated with the gap nodes in the query graph, thereby favoring matches that have fewer gap nodes. In our model, different nodes in the query graph can have different penalty values, and nodes with the same label can have different penalties as well. The model also gives users the freedom to choose between gapped matches (matches that allow gap nodes) and ungapped matches. If gapG(u) is set to 1 for every node,then the model only supports ungapped matches, otherwise it allows gapped matches. For simplicity, for the rest of the discussion, we will assume that all nodes have the same gap penalty value denoted as SingleGapCost.

3.3 Index-Based Matching Algorithm

First, an index is built on small substructures of graphs in the database. This index is then used to match fragments of the query with fragments in the database. Finally, the matching fragments are assembled into larger matches. The actual method is described in detail below.

3.3.1 The index structures

The index on small substructures of graphs in the database is called the FragmentIndex. It is probed by the matching algorithm to produce hits for substructures in the query. The indexing unit is a set of k nodes from the graphs in the database. We call each such set a fragment. Here k is a user specified parameter, and is usually a small number. However, simply enumerating all possible k-node sets is expensive in terms of both time and space. At the same time, if any pair of nodes in a fragment is too far apart by the pair wise distance measure. Therefore, a parameter d_{max} is specified to control whether a fragment is to be indexed.For a given k-node set v1, v2,… vk, if any two nodes vi and vj satisfy d(vi; vj) <= d_{max} we connect the two nodes by a pseudo edge.

For a given k-node set v1, v2,… vk, if any two nodes vi and vj satisfy d(vi; vj) <= d_{max} we connect the two nodes by a pseudo edge.For example nodes v3 and v4 in G1 can be matched to nodes v3 and v4 in G2, respectively. Although v3 and v4 do not form a connected subgraph in G2, they correspond to a fragment that needs to be indexed so that this match can be detected.An entry in the FragmentIndex has the following format: fnodeSeq, groupSeq,distSeq, sumDist gidg, where nodeSeq is the sequence of node IDs for the nodes in

the fragment, groupSeq is the sequence of group labels associated with the nodes,distSeq is the sequence of pairwise distances between the nodes in the fragment,sumDist is the sum of these pairwise distances, and gid is a unique graph ID.

A sample FragmentIndex, with k = 3 and d_{max}=2 for the database shown in fig 7. In this index, the groupSeq's are ordered by the group IDs, and the nodeSeq's are ordered according to the groupSeq's. If u; v;w is the nodeSeq, then the corresponding distSeq is d(u; v), d(u;w), d(v;w).node v8 with the label L8 in G1 belongs to two groups B and D, thus for this node set fv1; v3; v8g, there are two index entries f(B;E;E);G1; (v8; v1; v3); (1; 2; 2); 5g and f(D;E;E);G1; (v8; v1; v3); (1; 2; 2); 5g in the index.

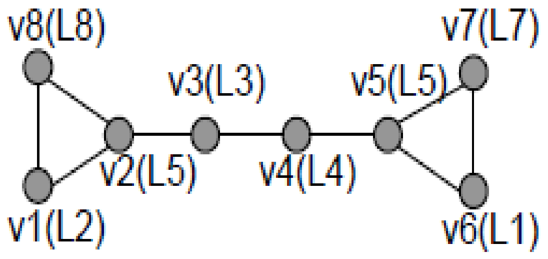
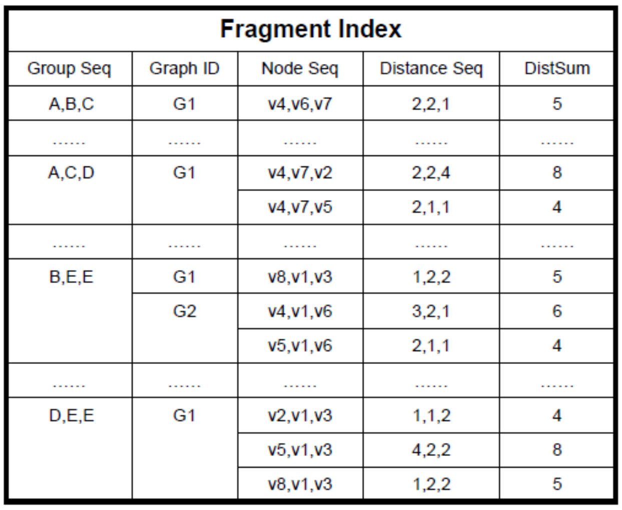


Figure 7:Database subgraph for query

The below table shows the distances between the nodes using IBGA.

Table 4: Fragment Index values



Fragment Index

Group Seq	Graph ID	Node Seq	Distance Seq	DistSum
A,B,C	G1	v4,v6,v7	2,2,1	5
......	......	......	......	......
A,C,D	G1	v4,v7,v2	2,2,4	8
		v4,v7,v5	2,1,1	4
......	......	......	......	......
B,E,E	G1	v8,v1,v3	1,2,2	5
	G2	v4,v1,v6	3,2,1	6
		v5,v1,v6	2,1,1	4
......	......	......	......	......
D,E,E	G1	v2,v1,v3	1,1,2	4
		v5,v1,v3	4,2,2	8
		v8,v1,v3	1,2,2	5

Using those values we can retrieve the information based on matching nodes and distance.

4. CONCLUSION AND FUTURE WORK

Now a day's graph databases represent an important requirement for applications which provide the data(graph data.) The following applications use graph data social networks, telecommunication networks. The concepts and algorithms of graph database area constitute an overlap between traditional domains such as database and graph theory. In this paper, we have presented an overview about graph databases. In particular, we have focused on the optimization technique used to improve query response time , databases and distributed systems such as query is decomposed using sub graph. We identified the specificities of their current use in graph database and their limits. We conclude that the current use of all optimization techniques, studied in this paper, is limited to specific queries, Moreover, there is no standard language to query graph database. In future this paper is extended by using advanced matching algorithms to shown better performance.

5. REFERENCES:

[1] Ahamed, B. B., & Ramkumar, T. (2015). Deduce User Search Progression with Feedback Session. Advances in Systems Science and Applications, 15(4), 366-383.
[2] L. Getoor, Friedman, N., D. Koller, B. Taskar (2001). Probabilistic Models of Relational Structure. International Conference on Machine Learning, Williamstown, MA.
[3] R. Angles, "A Comparison of Current Graph Database Models," Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW '12), pp. 171-177, 2012.
[4] Ali Ben Ammar "Query Optimiztion Techniques In Graph Databases", IJDMS Vol.8, No.4, August 2016
[5 Ahamed, B. B., & Ramkumar, T. (2016). An intelligent web search framework for performing efficient retrieval of data. Computers & Electrical Engineering, 56, 289-299.
[6] H. R. Vyawahare and P. P. Karde, "An Overview on Graph Database Model," International Journal of Innovative Research in Computer and Communication Engineering, Vol. 3, Issue 8, August 2015
[7] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwel, and T. Ideker. Pathblast: a tool for alignment of protein interaction networks, pages W83{W88,.2004
[8] R. Sharan et al. Conserved patterns of protein interaction in multiple species. PNAS, 102:1974{1979, 2005.
[9] Ahamed, B., & Ramkumar, T. (2016). Data integration-challenges, techniques and future directions: a comprehensive study. Indian J. Sci. Technol, 9, 1-9