BasisGraph: Combining Storage and Structure Index for Similarity Search in Graph Databases

Mistry Harjinder Singh, Srinath Srinivasa

International Institute of Information Technology,
26/C, Electronics City,
Hosur Road, Bangalore
India 560100
{mistry.harjinder,sri}@iiitb.ac.in

Abstract

Graph databases supporting retrieval based on structural similarity utilize *structure indexes*. A structure index is a data structure that indexes different structural features of member graphs. These features are typically extracted at insertion time. However, as the extraction of structural features may suffer from combinatorial explosion, the structure index takes long insertion time and large disk space. The member graphs cannot be modified, since even a simple structural modification may require re-extraction of structural features. The depth to which structural features were extracted during insertion, forms a hard limit on query precision, which cannot be changed at query time. In order to address these issues, a new model is introduced that combines the database graph storage and the structure index into one data structure. There is no need to extract structural features at insertion time, no hard limit on query refinement and graph modifications are supported naturally.

1 Introduction

Several application areas like image retrieval, CAD, cartography and bioinformatics require management of a collection of graphs. Graphs depict *relationship structures* that exist in the application domain.

In this work, we are concerned with databases of *labeled* graphs. A labeled graph is of the form $G = (V, E, V_L, E_L, \delta, \gamma)$. Here V is the set of nodes or vertices, E is the set of edges across the nodes, V_L and E_L are sets of node labels and edge labels respectively, and $\delta : V \rightarrow V_L$, $\gamma : E \rightarrow E_L$ represent node and edge labeling functions. Usually, the following inequalities hold: $|V_L| \ll |V|$ and $|E_L| \ll |E|$.

Often it is required to retrieve graphs based on their *structural* properties. User may want to find out all member graphs which are structurally similar to a query graph. Two graphs G_1 and G_2 are said to be *structurally similar* if one can be rewritten as the other with a small number of edge (and node) additions or deletions. In this paper, we are concentrating only on the labeled structural similarity. In *labeled structural similarity* computation, node and edge labels are taken into account, while in *topological* similarity search, nodes and edges are treated as typeless or unlabeled entities.

Answering structural queries is a hard problem since it involves one or more (sub)graph isomorphism checks. In graph databases, the problem is compounded by the fact that the query graph has to be matched against a *set* of member graphs.

In order to answer structural queries in interactive response times, *structure indexes* have been developed in existing literature [1, 2, 3, 4, 6, 7]. The common theme behind these indexes is to extract structural features of member graphs, typically at insertion time, and store them in a global index. When a query graph is presented, its own structural features are extracted and compared with the features stored in the index. This helps in pruning irrelevant member graphs from the result set.

However, the structure indexes in the literature today have the following problems: Extracting structural features is prone to combinatorial explosion as the graph size and/or complexity increase. This has two-fold disadvantages: long insertion time and large disk space. Since even a single structural modification may require re-extraction of structural features, member graphs cannot be modified. To reduce the time of feature extraction a limit is set on the *length* of structural features being indexed. But this limits the accuracy of the result also.

To address these shortcomings, a new model called *basis graph* is introduced in this paper, where the structure index and the graph storage are combined into one data structure. Basis graph has the following

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

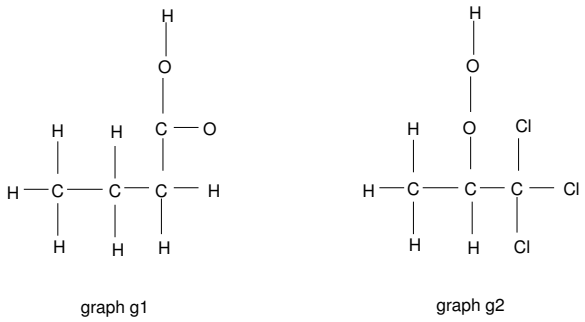

Figure 1: Two example member graphs $g1$ and $g2$

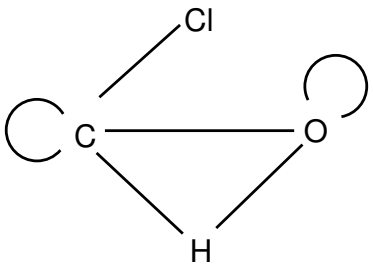

Figure 2: Basis graph for the graphs $g1$ and $g2$

advantages: Structural features need not be extracted at insertion time. Modifying structural properties of member graphs is straightforward. Query results can be refined to any depth till isomorphism.

2 The Basis Graph

In the proposed approach, the entire database is logically in the form of a single graph called the "basis graph." A node in the basis graph denotes a node label that is seen in at least one graph in the database. An edge labeled p connecting nodes l_i and l_k denotes the occurrence of at least one edge in at least one graph in the database, that has a label p and connects nodes having labels l_i and l_j respectively. Conceptually, it is similar to DataGuides [8]. However, the objectives of basis graph are quite different, resulting in an implementation that is optimized to answer structural queries.

Figure 2 shows the basis graph formed after inserting graphs $g1$ and $g2$ from fig. 1. Between the two graphs, there are four node labels: C, H, O and Cl, which are shown in the basis graph.

The basis graph structure is implemented by a set of B^+ trees. There is one B^+ tree for each node of the basis graph and two B^+ trees for every edge that connects distinct node labels. Self-loops have only one B^+ tree associated with them. Figure 3 expands a part of the basis graph of fig. 2 to show the underlying data structures. In that figure, a triangle represents a B^+ tree.

Insertion of graphs: When a graph is added to the database, a unique identifier *gid* is assigned to the graph. Then, its nodes and edges are indexed in the basis graph.

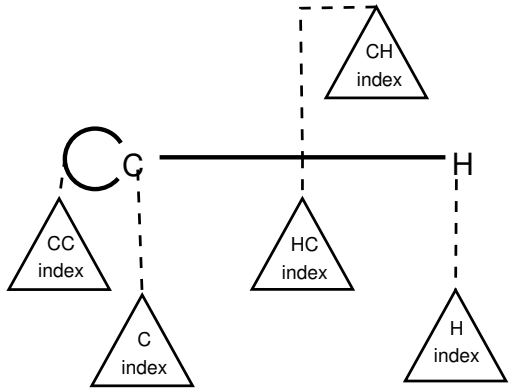

Figure 3: Detailed structure of a part of basis graph of Fig. 2

The B^+ tree for a node depicting label L in the basis graph indexes records of the following form: [(gid, name, rid)] Here, *gid* is the graph id that is being added and *name* is the instance name of the L-labeled node that is being added. [1] The term *rid* is a pointer that points to a disk record containing attributes associated with this node.

For each edge in the basis graph of the form `L-M`, there are two B^+ trees. One indexes the edge label `L-M`, while the other indexes `M-L`. For undirected graphs, both `L-M` and `M-L` will be added, while in directed graphs, only one of them will be added depending on the edge direction. Even though there is duplication of data when storing undirected graphs, having two B^+ trees helps during query answering.

A B^+ tree for an edge of the form `L-M` of the basis graph holds records in the following form: [(gid,$name_1$,$name_2$,rid)] Here $name_1$ and $name_2$ indicate an `L-M` edge in the graph *gid* between nodes with names $name_1$ and $name_2$. A record of the form $(gid, name_1, name_2, rid)$ in the `L-M` B^+ tree will have a corresponding record of the form $(gid, name_2, name_1, rid)$ in the `M-L` B^+ tree if the graph is undirected.

For B^+ trees representing self-loops of the form `L-L`, an undirected edge is added twice. Suppose there is an undirected `L-L` edge between nodes with names a and b respectively. Then two records of the form (gid, a, b, rid) and (gid, b, a, rid) are added to the tree.

Structural modifications of graphs: Addition or deletion of nodes and edges are translated into addition and deletion of records in corresponding B^+ trees. A change in the attributes of nodes and edges does not affect the basis graph.

Attribute Information: A hash file organization is used to maintain attributes and their values, pointed

[1] Every node in a graph is identified by a unique combination of label and name.

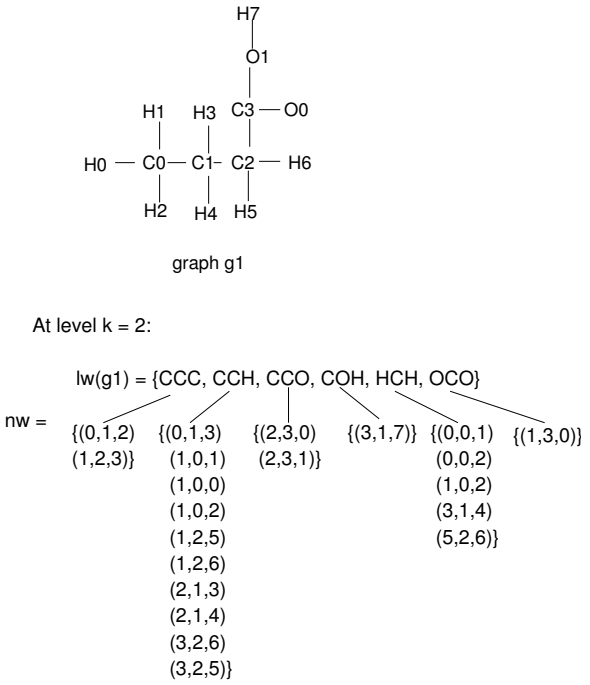

Figure 4: lw and nw sets for the graph $g1$

by the *rid* fields. In the implementation, each hash bucket is a separate file. The attributes are hashed to buckets based on the *gid* of their member graphs.

3 Processing Structural Queries

Basis Graph is an improvement over the structure index: Label Walk Index (LWI) [6]. LWI stores the counts of label walks in member graphs. A label-walk forms a dimension in a vector space. The number of name-walks associated with the given label-walk, becomes its projection. Then, structural queries are answered by comparing distance between the query graph's walk-vectors and the vectors of member graphs. The walk-vectors of member graphs are computed by lateral traversal of the LWI tree. However, in the proposed approach, no seperate index is maintained. Instead, the required walk-vectors are computed by querying various B^+ trees on the fly.

Figure 4 shows the graph $g1$ from Figure 1, with nodes being given names. The figure also shows the set of label-walks $lw(g1)$ and the set of name-walks nw for each label-walk of length $k = 2$.

Next, we describe the process of retrieving graphs from a database $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ that have a *labeled structural similarity* to the given query graph Q. The overall algorithm is as follows:

Algorithm: Similarity search
Input: graph database $\mathcal{G}$, query graph Q, similarity threshold d, max length l
Output: result set R

1. $R \leftarrow \mathcal{G}$
2. $k \leftarrow 0$
3. do
4. For each $G_i \in \mathcal{G}$ create label-walk $lw(G_i)$ of length k and for each $lw_j \in lw(G_i)$, create name walk $nw(lw_j)$
5. Create label walk $lw(Q)$ of length k and for each $lw_j \in lw(Q)$, create name walk $nw(lw_j)$
6. Compare walk vectors for every $G_i \in R$ and Q and discard all G_i whose L_1 distance from Q is greater than d
7. $k \leftarrow k + 1$
8. while $k < l$
9. return R

For computing walk-vectors of member graphs, we need to find the label walks of different length and corresponding name-walks. The walks of length-0 and length-1 are found by querying the B^+ tress corresponding to the basis graph nodes and edges, respectively.

For lengths $k > 1$, a set of wild-card queries are issued on corresponding B^+ trees to obtain walks. For every name-walk $n_1 n_2 \ldots n_k$ corresponding to a label-walk of the form $l_1 l_2 \ldots l_k$ in graph with gid G_i, B^+ trees of the basis graph at the edges incident on nodes l_1 and l_k (at either ends of the walk) are consulted. In other words, the walk $l_1 l_2 \ldots l_k$ is tried to be extended from either ends.

Then, the following wild-card query is given on the corresponding B^+ tree: $(G_i, n_1, *)$ and $(G_i, n_k, *)$. This returns the set of all node names that are connected to n_1 and n_k. A set of $k + 1$ walks are then generated from this result set.

4 Performance Studies

Performance studies for the basis graph approach was conducted on an Intel Celeron 1.2 Ghz machine with 512MB of RAM. For input data, both synthetic graphs and graphs of covalent structures of protein molecules (obtained from PDB[2]) were used. The B^+ trees that made up the basis graph used page sizes of 8192 bytes with fanout of approximately 300.

Figure 5 compares the insertion performance of Basis Graph (GRACE3), LWI and GraphGrep for different graph sizes in terms of the number of edges, and *enr* values. The factor *edge-to-node ratio* or *enr* was used to model the complexity of enumerating walks of greater lengths. For LWI and GraphGrep, the value of max. length of walks to be enumerated (l_p) was set to 4 in these experiments. Since walks are not enumerated at insertion time, basis graph is unaffected by different walk lengths and *enr*.

We also compared the pre-processing time of gIndex (fig. 6), which needs to generate a DFS tree with the smallest signature. Since DFS is not limited by a

[2]http://www.pdb.org/

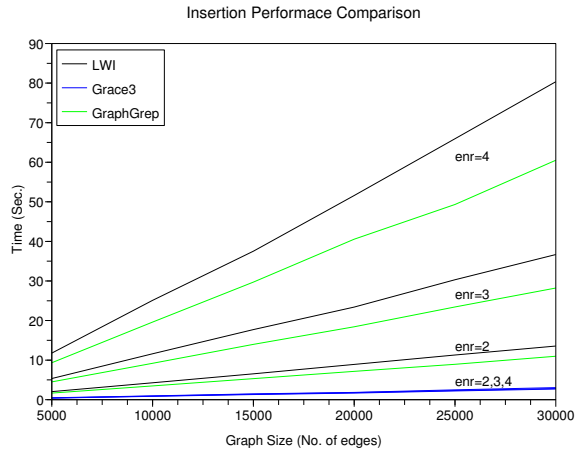

Figure 5: Insertion time against graph size and *enr*

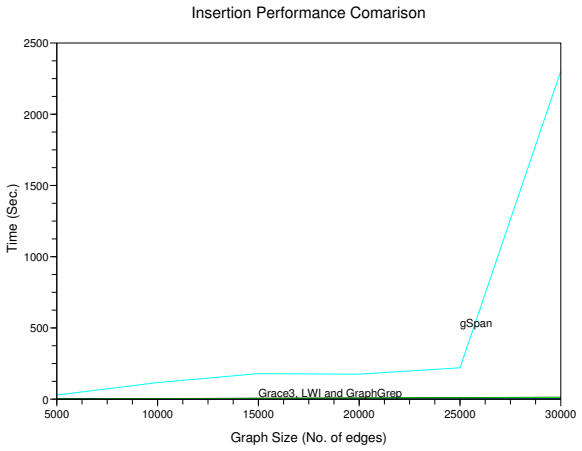

Figure 6: gIndex preprocessing time for $enr = 2$

parameter like l_p, the pre-processing time for gIndex is much more than that of basis graph or GraphGrep.

Labeled similarity search was carried out using basis graph and LWI. GraphGrep crashed when the size of the query graph exceeded 1000 and so was not included in this evaluation. gIndex was not able to insert all the graphs for the evaluation since it took a very long time. It was also excluded from this evaluation.

Figure 7 plots the cumulative search time taken by Basis graph and LWI for different graph sizes. The maximum *enr* of the query graph was 2 in this experiment. As is apparent, Basis graph incurs extra overhead for walk generation at query time. However, this is balanced by the advantages it provides over structure indexes.

Since query refinements in basis graph can be extended to any length, basis graph is also suitable for answering topological similarity queries. [5] contains more details about basis graph, and evaluation experiments on topological similarity search.

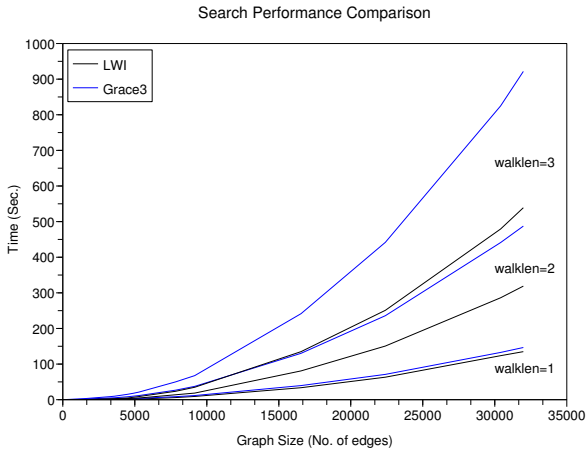

Figure 7: Similarity searching in Grace3 and LWI

5 Conclusions

Walk based indexing is a practical structural index for graph databases in terms of result quality, which had some limiting shortcomings. The proposed basis graph model demonstrates a significant step forward in making graph databases using walk-based indexing practical.

References

[1] J. Abello, Y. Kotidis. Hierarchical Graph Indexing. Proc. of Int'l Conf. on Information and Knowledge Management (CIKM), Nov 2003.

[2] Rosalba Giugno, Dennis Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. Proc. of the 16th Int'l Conf. on Pattern Recognition (ICPR'02), Volume 2, 2002.

[3] L. Holder, D. Cook, J. Gonzalez, I. Jonyer. Structural Pattern Recognition in Graphs. In *Pattern Recognition and String Matching*, Kluwer Academic Publishers, 2002.

[4] J.W. Raymond, E.J. Gardiner, P. Willet. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs, *Oxford Computer Journal*, Vol 45, pages 631–644, 2002.

[5] M.H. Singh, S. Srinivasa. Combining Storage and Structure Index in Graph Databases. *Technical Report OSL-IIITB-0601*, Bangalore, India, 2006.

[6] S. Srinivasa, M. Maier, M.R. Mutalikdesai, Gopinath P.S., Gowrishankar K.A. LWI and Safari: A New Index Structure and Query Model for Graph Databases. Proc. of COMAD 2005, Jan 2005.

[7] X. Yan, P.S. Yu, J. Han. Graph Indexing: A Frequent-structure Based Approach. *Proc. of SIGMOD 2004*.

[8] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proc. of Int. Conf. on Very Large Data Bases, Athens, Greece, August 1997.