A Tool for Fast Indexing and Querying of Graphs

Dipali Pal
University of Missouri-Kansas City
Kansas City, MO 64110, USA
dp244@umkc.edu

Praveen R. Rao
University of Missouri-Kansas City
Kansas City, MO 64110, USA
raopr@umkc.edu

ABSTRACT

We present a tool called GiS for indexing and querying a large database of labeled, undirected graphs. Such graphs can model chemical compounds, represent contact maps constructed from 3D structure of proteins, and so forth. GiS supports exact subgraph matching and approximate graph matching queries. It adopts a suite of new techniques and algorithms for (a) fast construction of disk-based indexes with small index sizes, and (b) efficient query processing with high precision of matching. During the demo, the user can index real graph datasets using a recommendation facility in GiS, pose exact subgraph matching and approximate graph matching queries, and view matching graphs using the Jmol browser.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Design

1. INTRODUCTION

Graphs are widely used to model data in a variety of domains such as biology, chemistry, computer vision, and the World Wide Web. In chemistry, chemical compounds can be represented by labeled, undirected graphs. In proteomics, a field of biology, tertiary (or 3D) structures of proteins are required to understand the biological functions of proteins. Contact maps [4, 17], which can model 3D structures of proteins, can be represented by graphs. In the Semantic Web, the RDF data model represents data as graphs.

Pattern matching over graphs is an important task in a variety of applications. Exact subgraph matching (or isomorphism) has been a problem of interest for several decades and has been widely used in areas such as chemical informatics, circuit design and verification, scene analysis in computer vision, and so forth. One example is in chemical databases: By posing a query based on *exact subgraph matching*, we can identify those molecules in a database that contain a particular functional group (*e.g.*, a phenyl group C_6H_5) [6]. Another kind of matching called *approximate graph matching* [18] is useful in applications where graphs similar to a query graph are desired. Queries based on such matching are extremely useful for tasks such as similarity searching of chemical compounds [20], comparing biopathway graphs, and so forth.

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

We present a tool called GiS for indexing and querying a large database of labeled, undirected graphs. GiS supports exact subgraph matching and approximate graph matching queries. GiS is available online at http://vortex.sce.umkc.edu/gis.

2. MOTIVATIONS

In recent years, many approaches have been developed for graph indexing and query processing. We highlight some of their limitations that motivate our work.

To cope with the NP-completeness of subgraph isomorphism, a common methodology has been to first *filter* and identify candidate matches, and then *verify* these candidates [16] to retain only the true matches. Similarly, graph edit distance computation required for approximate graph matching is NP-complete and hence, a *filter* and *verification* approach seems to be a viable alternative.

Many of the recent approaches for exact subgraph matching, load the index into memory before processing graph queries (*e.g.*, gIndex [22], Tree+Δ [28], GDIndex [21], QuickSI [13], GCoding [29]). As is, these approaches will fail when the graph database becomes very large and the indexes cannot fit in main memory. Another common trend among recent approaches is to leverage frequent pattern mining to extract features from graphs (*e.g.*, trees, subgraphs) for indexing (*e.g.*, gIndex [22], Tree+Δ [28], TreePi [26], QuickSI [13], FG-index [3]). These approaches are well-suited for query patterns that are frequent. However, significant preprocessing effort is required due to pattern mining that increases the cost of index construction (*e.g.*, when datasets contain large number of distinct vertex labels or when graphs are large and dense with few hundred vertices and edges [11]).

A recent non-mining approach called C-tree organizes graphs into a hierarchical index by computing graph closures [7]. The index is built using hierarchical clustering and pseudo isomorphism tests are applied to achieve high precision of matching. C-tree supports similarity searching on graphs. However, C-tree fails to index large, dense graphs [11] such as protein contact maps. Recently, Zeng *et al.* proposed APPFULL for approximate graph matching queries [25]. APPFULL examines every graph in the database and does not exploit an index.

3. OUR PROPOSED SYSTEM

We have developed a tool called GiS that adopts a suite of new techniques for indexing and querying labeled, undirected graphs. GiS is designed for data graphs that contain few hundred vertices and edges (*e.g.*, protein contact maps, chemical compounds) and targets high selectivity queries.

Inspired by holistic XML pattern matching approaches that yield superior performance than non-holistic ones (*e.g.*, TwigStack [2]), we have developed a *holistic graph pattern matching* approach. By

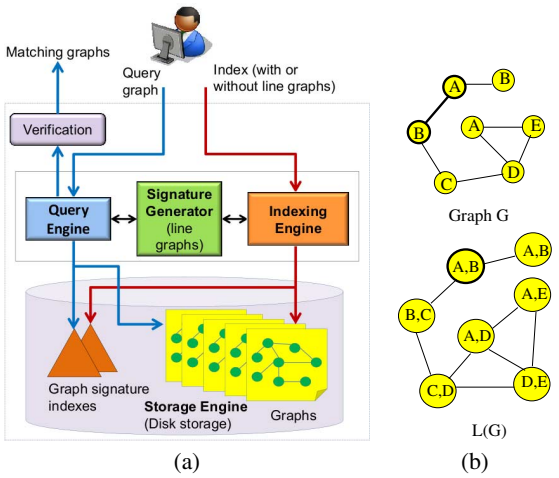


Figure 1: (a) Architecture of GiS (b) Graph and its line graph

this, we mean that the entire query is considered as a single unit to search the index during query processing. In contrast, the state-of-the-art mining-based approach for exact subgraph matching called FG-index [3], is non-holistic: for a class of queries, it breaks a query into smaller units and processes them individually, and finally intersects the partial results.

Our key technical contributions in this work are:

• a new representation of a graph by its signature, which is essentially a multiset and captures the vertex and edge characteristics of the entire graph, to enable *holistic pattern matching*;

• a new method based on the concept of *line graphs* to systematically expose more structural information about a graph, which can be captured by its signature, and therefore improve the precision of exact subgraph matching;

• a new disk-based index over graph signatures for efficient query processing and a bulk loading strategy for fast index construction;

• and new algorithms for processing exact subgraph and approximate graph matching queries by leveraging the properties of graph signatures and the index over graph signatures.

Next, we provide an overview of the key design principles in GiS. Complete details on the design and implementation, including performance comparison of GiS with FG-index and C-tree, is available in an extended version [11]. Compared with FG-index that supports only exact subgraph matching queries, GiS had superior indexing and query performance. Compared with C-tree, GiS had superior indexing performance and smaller index sizes, and faster query processing times for both exact subgraph matching and approximate graph matching queries. Both C-tree and GiS achieved high precision of matching. GiS was able to index and process queries efficiently on a real graph dataset with large, dense graphs (*i.e.*, protein contact maps). However, C-tree and FG-index failed to index this dataset.

Architecture of GiS.

The architecture of GiS is shown in Figure 1(a). The main components of GiS are the *Storage Engine*, the *Signature Generator*, the *Indexing Engine*, and the *Query Engine*. The Storage Engine stores the graphs and indexes on disk. In GiS, data graphs are transformed to their graph signatures and these signatures are indexed on disk. A query is also transformed into its signature. Query processing involves the filtering phase followed by the verification phase. During filtering, the query signature is used to search the index to identify a set of candidate graphs. During verification, candidate graphs are examined to identify true matches.

Signature Generator.

A graph is transformed into a signature that captures the vertex and edge characteristics of the graph. Consider a graph without edge labels. For an edge, we construct an ordered pair (u, v) where u and v are the vertex labels of the edge such that u is less than v in lexicographic order. Then we hash this pair using Rabin's fingerprinting [10]. We compute the hash value for every edge of the graph by traversing it and together the multiset of hash values constitutes a graph signature. The lexicographic ordering ensures that the same signature is generated irrespective of the starting vertex chosen for traversing the graph.

We have developed a novel method to tune the amount of structural information captured by a graph signature based on the concept of line graphs proposed by Whitney (1932) [19]. A line graph of a graph without edge labels is defined as follows.

DEFINITION 1. *Suppose a graph G has a vertex set V and edge set E. A line graph of a graph $G = (V, E)$, denoted by $L(G) = (V_L, E_L)$, is a graph whose vertex set V_L contains one vertex for every edge in G. The edge set E_L is constructed as follows: two vertices in L(G) are adjacent, if and only if, their corresponding edges in G are adjacent* i.e.*, share a vertex.*

We can construct the signature of a line graph using the method discussed earlier. In fact, we can successively apply the line graph computation n times on the input graph. (Line graph computation is similar to a composable function.) We denote the graph resulting from n successive line graph computations by $L^n(G)$, where G is the input graph. (Note that $L^0(G) = G$.) Now the signature of $L^n(G)$ can be constructed and this signature is more precise than that on G. The reason is with successive computation of line graphs, *an edge in $L^n(G)$ captures a (connected) subgraph structure in the original graph G* [11]. Thus, the signature of $L^n(G)$ captures richer structural components in G than the signature of G, resulting in high precision of exact subgraph matching [11]. The tuning parameter n controls the richness of graph signatures. Note that $L^n(G)$ *does not enumerate* all subgraphs in G.

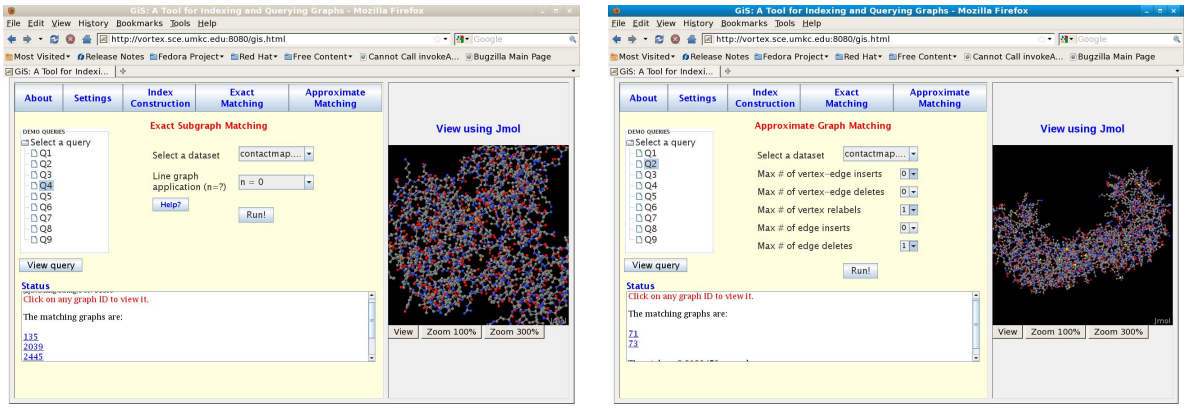
An example of a graph G and its line graph $L(G)$ (one application) is shown in Figure 1(b). We impose a lexicographic ordering on the vertex labels of a line graph as it affects the hash values computed during signature construction.

We can compute the space and time complexity of signatures on line graphs using straightforward combinatorics [11]. (We required at most two line graph computations for high precision of matching on real and synthetic datasets.) Like FG-index, GiS supports graphs with edge labels. Line graphs over such graphs can also be computed. Due to space constraints, we refer the reader to [11] for more details.

One key difference between GCoding [29], a main-memory based approach, and GiS is that the graph code computed by GCoding captures local tree structures around the neighborhood of vertices. But in GiS, we capture subgraph structures in a graph via line graph computations.

Indexing Engine.

GiS constructs a disk-based index over graph signatures using a bulk loading strategy. We call the index *a signature index*. Because signatures are multisets and operations on multisets are performed during query processing, existing set indexing techniques cannot be directly employed. A signature index provides high pruning power



(a) Exact subgraph matching (b) Approximate graph matching

Figure 2: Screenshots of GiS

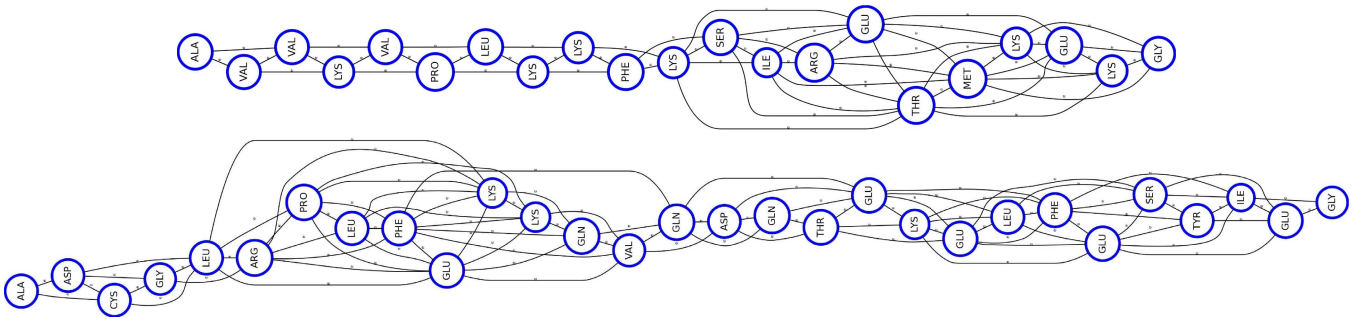


Figure 3: Demo queries for ASTRAL dataset drawn using GraphViz (http://graphviz.org)

over graph signatures during the filtering phase of query processing. The structure of the index is similar to an R-tree in the sense that it hierarchically groups similar signatures together, just like an R-tree that hierarchically groups nearby rectangles together. Each non-leaf node in the index contains (sig, ptr) entries where sig denotes a multiset, and ptr denotes a reference to a child index node. Each leaf node also contains (sig, ptr) entries where sig denotes a graph signature, and ptr denotes a graph id. The sig value in a non-leaf node entry denotes *the (multiset) union of the signatures* in the child node pointed by the entry. This design provides pruning opportunities for exact subgraph matching and approximate graph matching [11].

The bulk loading algorithm in GiS creates groups of similar signatures recursively until each group has enough signatures for the given fanout. The algorithm begins by picking seed signatures and distributing signatures into groups based on their similarity. Similarity of two signatures s_1 and s_2 is given by ratio $\frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$. The index is constructed in a bottom-up fashion starting with the leaf nodes until a root node is created.

Query Engine.

GiS supports two types of queries over a graph database, namely, *exact subgraph matching* and *approximate graph matching* queries. Given a query graph Q, exact subgraph matching finds all graphs in the database that contain a subgraph that is isomorphic to Q. (This is also referred to as a subgraph containment query in prior work.) On the other hand, given a query graph Q and a distance d, approximate graph matching finds all graphs in the database whose edit distance with Q is at most d.

GiS relies on the properties of graph signatures for query processing. A graph signature is implemented as a sorted list so that operations like subset, union, intersection, and difference on multisets can be performed in linear time. We have established necessary conditions for exact subgraph matching (with and without line graph computations) using the subset operation between the query and data signature. (If k line graph computations are applied on the data graphs, then k line graph computations are also applied on the query.) We have proved necessary conditions for approximate graph matching using the set difference operation between the data and query signatures and the difference in their cardinalities [11].

During query processing, appropriate necessary conditions are tested while traversing a signature index. Once the leaf nodes of the index are processed, candidate matches are determined. (The necessary conditions ensure that the recall is always one.) In the verification phase, we apply exact subgraph isomorphism test [16] or compute graph edit distance [9] to discard false matches.

4. DEMONSTRATION SCENARIO

In this demo, we will use 2 real graph datasets, namely, chemical compounds from AIDS Antiviral Screen dataset [1] and protein contact maps [4, 17] constructed from the ASTRAL dataset [2] using a threshold distance of 7 Å between c_α atoms of the protein residues.

There are 3 key activities a user will experience during the demo. The first is the task of indexing graphs for exact subgraph matching. Through the GUI, the user can choose a dataset, number of graphs to index, and number of line graph computations n for that dataset. Once the user completes the indexing process, the index construc-

[1] http://dtp.nci.nih.gov
[2] http://astral.berkeley.edu/pdbstyle-1.71.html

tion time and size are displayed. The user can select parameters such as index fanout and number of buffer pool pages.

GiS has a *Recommender* to assist the user to choose an appropriate value of n for line graph computations. The Recommender samples a user specified fraction of the input dataset and builds test indexes on the sampled dataset for different number of line graph computations. The user also specifies the maximum number of line graph computations m. For each i, $0 \leq i \leq m$, signatures are constructed on $L^i(G)$ on the sampled dataset and a test index is built on these signatures. Queries are generated over the sampled dataset for user specified selectivity. Once the queries are processed using each test index, the average precision of exact subgraph matching and the average query response time (filtering + verification cost) are displayed to the user. GiS recommends a suitable value of n.

The second activity is the process of posing exact subgraph matching queries. For ease of demonstration, we provide a list of different types of queries for each dataset. (A few examples are shown in Figure 3.) The user can also select and view the queries for exact subgraph matching. Figure 1(a) shows a screen shot. Once the true matches are output along with precision and processing time, the user can select a matching graph and view the corresponding protein structure or chemical compound using Jmol [1]. The user can test with different values of n for line graph computation to gauge the effectiveness of the recommendation given by GiS.

The third activity is the process of posing approximate graph matching queries. Note that the index built on the signatures of the original graphs is used. The user can specify an upper bound on each edit operation (*e.g.*, vertex relabel, vertex delete, edge insert) to yield higher precision of matching [11]. Once the query is processed, the processing time and precision are reported and the user can view the corresponding protein structure or chemical compound using Jmol.

5. RELATED WORK

GraphGrep [5], GDIndex [21], and GString [8] are other methods for exact subgraph matching. Both GraphGrep and GDIndex suffer from worst-case index size that is exponential in either the path length or the size of the graphs. GString focuses on chemical compounds and supports approximate subgraph queries. Many approaches have been developed for approximate subgraph matching (*e.g.*, SAGA [14], TALE [15], PIS [24], Grafil [23], GrafD-index [12], SAPPER [27]). TALE and SAPPER support large data graphs. GiS does not support approximate subgraph matching.

6. CONCLUSIONS

We have presented a tool called GiS for scalable and efficient disk-based indexing and query processing over a large database of labeled, undirected graphs. GiS processes queries holistically and supports both exact subgraph matching and approximate graph matching. It adopts a suite of new techniques for fast index construction and query processing and high precision of matching.

7. ACKNOWLEDGMENTS

We are thankful to the authors of FG-index and C-tree for their code and assistance. We thank Vasil Slavov for his help. This work was partly supported by funds from University of Missouri-Kansas City.

8. REFERENCES

[1] Jmol. http://www.jmol.org.

[2] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal XML pattern matching. In *Proc. of the 2002 SIGMOD Conference*.

[3] J. Cheng, Y. Ye, and W. Ng. Efficient Query Processing on Graph Databases. *ACM Transactions on Database Systems*, 34(1):1–48, 2009.

[4] N. V. Dokholyan, L. Li, F. Ding, and E. I. Shakhnovich. Topological Determinants of Protein Folding. *Proceedings of the National Academy of Sciences*, 99(13):8637–8641, 2002.

[5] R. Giugno and D. Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. *Intl. Conference on Pattern Recognition*, 2002.

[6] A. Golovin and K. Henrick. Chemical Substructure Search in SQL. *Journal of Chemical Information and Modeling*, 49(1):22–27, 2009.

[7] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *Proc. of the 22th ICDE Conference*, pages 38–49, 2006.

[8] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. GString: A novel approach for efficient search in graph databases. In *Proc. of the 23th ICDE Conference*, pages 566–575, 2007.

[9] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, August 2006.

[10] M. O. Rabin. Fingerprinting by Random Polynomials. Technical Report TR 15-81, Harvard University, Cambridge, MA 02138, 1981.

[11] P. R. Rao and D. Pal. GiS: Fast Indexing and Querying of Graph Structures. Technical report, University of Missouri-Kansas City, Nov 2009. http://r.faculty.umkc.edu/raopr/TR-DB-2009-01.pdf.

[12] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *Proc. of the 2010 SIGMOD Conference*, pages 903–914, Indianapolis, 2010.

[13] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. In *Proc. of the 34st VLDB Conference*, pages 364–375, New Zealand, 2008.

[14] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinformatics Journal*, 23(2):232–239, 2007.

[15] Y. Tian and J. M. Patel. TALE: A Tool for Approximate Large Graph Matching. In *Proc. of the 24th ICDE Conference (2008)*, pages 963–972.

[16] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of ACM*, 23(1):31–42, 1976.

[17] M. Vendruscolo, E. Kussell, and E. Domany. Recovery of protein structure from contact maps. *Folding and Design*, 2(5):295 – 306, 1997.

[18] J. T. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph matching. *Information Sciences - Informatics and Computer Science*, 82(1-2):45–74, 1995.

[19] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:150–168, 1932.

[20] P. Willett, J. Barnard, and G. Downs. Chemical Similarity Searching. *Journal of Chemical Info. and Comp. Sci.*, 38(6):983–996, 1998.

[21] D. W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *Proc. of the 23th ICDE Conference*, pages 976–985, Istanbul, 2007.

[22] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure based approach. In *Proc. of the 2004 SIGMOD Conference*, France, 2004.

[23] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *Proc. of the 2005 SIGMOD Conference*, Baltimore.

[24] X. Yan, F. Zhu, J. Han, and P. S. Yu. Searching Substructures with Superimposed Distance. In *Proc. of the 22th ICDE Conference*, pages 88–99, Atlanta, 2006.

[25] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing Stars: On Approximating Graph Edit Distance. In *Proc. of the 35st VLDB Conference*, Lyon, France, 2009.

[26] S. Zhang, M. Hu, and J. Yang. TreePi: A Novel Graph Indexing Method. In *Proc. of the 23th ICDE Conference*, pages 966–975, Istanbul, 2007.

[27] S. Zhang, J. Yang, and W. Jin. SAPPER: subgraph indexing and approximate matching in large graphs. In *Proc. of the 36st VLDB Conference*, pages 903–914, Singapore, 2010.

[28] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta >= graph. In *Proc. of the 33rd VLDB Conference*, pages 938–949, 2007.

[29] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A Novel Spectral Coding in a Large Graph Database. In *Proc. of the 11th EDBT Conference*, 2008.