iGraph in Action: Performance Analysis of Disk-Based Graph Indexing Techniques

Wook-Shin Han
Kyungpook National University, Korea
wshan@knu.ac.kr

Minh-Duc Pham
Kyungpook National University, Korea
duc@www-db.knu.ac.kr

Jinsoo Lee
Kyungpook National University, Korea
jslee@www-db.knu.ac.kr

Romans Kasperovics
Kyungpook National University, Korea
romans@www-db.knu.ac.kr

Jeffrey Xu Yu
Chinese University of Hong Kong, Hong Kong
yu@se.cuhk.edu.hk

ABSTRACT

Graphs provide a powerful way to model complex structures such as chemical compounds, protein interactions, images, and program dependencies. The previous practice for experiments in graph indexing techniques is that the author of a newly proposed technique does not implement existing indexes on his own code base, but uses the original author's binary executables and reports only the wall clock time. We observed that this practice may result in several problems [6]. In order to address these problems, we implemented all representative graph indexing techniques in a common framework, called iGraph [6]. In this demonstration we showcase iGraph and its visual tools using several real datasets and their workloads. For selected queries from the workloads, we show several unique features including in-depth performance analysis.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Experimentation, Performance

Keywords

Graph indexing techniques, Visualization, Performance analysis

1. INTRODUCTION

The number of graphs in a database can grow up to tens of millions. For instance, PubChem database [4] contains more than 30 million chemical compounds. The graph containment is one of the most important graph queries. That is, given a query graph Q and a database D of graphs, the graph containment finds all graphs in D that contain Q. Recently, there have been many research efforts to solve the graph containment problem by utilizing graph indexes.

The current practice for experiments in graph indexing is to compare a newly proposed technique against the existing techniques using the original author's binary executables and to report only the wall clock time. In essence, this practice compares the final numbers by dealing with black boxes without knowing exactly why or what makes the performance different. In our previous work [6] we identified several problems with this practice, such as the brittleness of the performance measure used.

iGraph provides a fair comparison of disk-based graph indexing techniques by using a common code base and full disk-based implementations rather than (full or partial) in-memory based implementations. As a first step towards a next real graph database system to handle a large number of graphs, the iGraph framework can provide developers new insights on analysis of performance of each graph indexing algorithm.

The visual tools of the iGraph framework are a convenient GUI for editing graph datasets and query graphs, executing queries, and navigating through the answers. The analysis tool offers a unique in-depth analysis functions, such as display of the best and the worst queries, comparison of any two algorithms, display of the involved indexing features their influence on the index-based filtering.

2. SYSTEM ARCHITECTURE

Figure 1 shows the system architecture of iGraph and its visual tools. Currently, iGraph supports all representative state-of-the-art graph indexing algorithms: gIndex [9], FG-Index [5], Tree+Δ [10], SwiftIndex [8], gCode [11], and C-Tree [7].

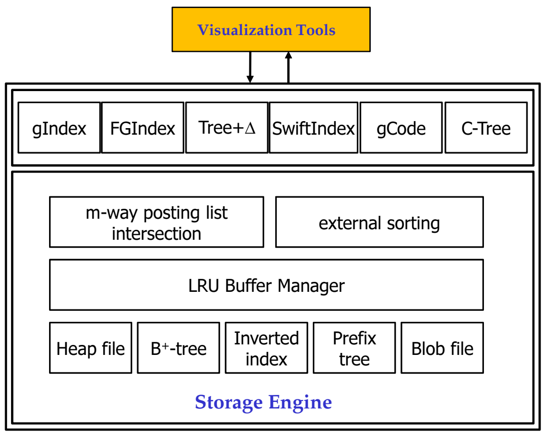
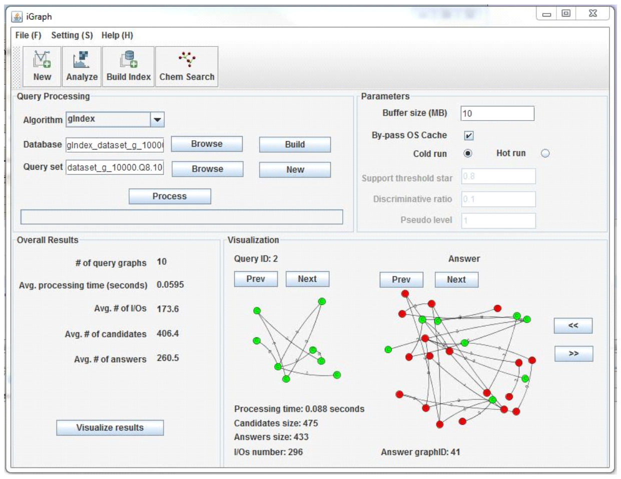


Figure 1: The system architecture of iGraph.

iGraph is built on a storage engine which supports standard disk-based structures and algorithms including heap files, B+-trees, inverted indexes, disk-based prefix trees, binary large object (BLOB) files, an LRU buffer manager, m-way posting list intersection, and external sorting. By utilizing iGraph, the effects of using different environments can be minimized. In addition, we can easily see the effect of the buffer size. The visual tools are implemented using the JUNG [2] library for visualizing graphs.

The visual tools provide the basic operations for editing query and data graphs, building necessary index structures, as well as visualizing the results of the query processing. These operations can be called from the main control panel, shown in the Figure 2.



Query Processing
Algorithm: gIndex
Database: gIndex_dataset_g_1000(
Query set: dataset_g_10000.Q8.10

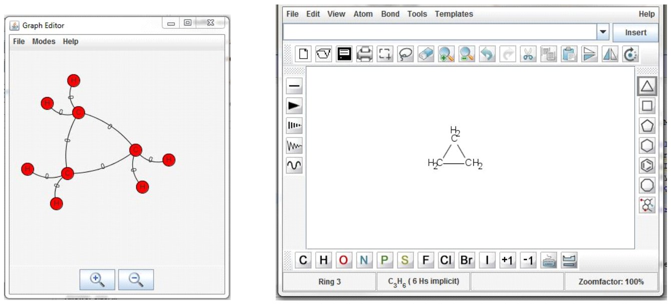
Parameters
Buffer size (MB): 10
By-pass OS Cache: ✓
Cold run [✓] Hot run
Support threshold star: 0.8
Discriminative ratio: 0.1
Pseudo level: 1

Overall Results
# of query graphs: 10
Avg. processing time (seconds): 0.0595
Avg. # of I/Os: 173.6
Avg. # of candidates: 406.4
Avg. # of answers: 260.5

Visualization
Query ID: 2
Answer
Processing time: 0.088 seconds
Candidates size: 475
Answers size: 433
I/Os number: 296
Answer graphID: 41

Figure 2: The main control panel of iGraph visualization tool.

- **Creating query and data graphs:** A user can intuitively create and edit query and data graphs by using a visual graph editor as shown in Figure 3(a). In addition, we integrate the JChempaint editor and viewer [3] for chemical structures in order to support search for chemical compounds from a chemical dataset such as PubChem (See Figure 3(b)).


Graph Editor
File Modes Help
File Edit View Atom Bond Tools Templates Help
Insert
C H O N P S F Cl Br I +1 -1
Ring 3
C_3H_6 (6 Hs implicit)
Zoomfactor: 100%

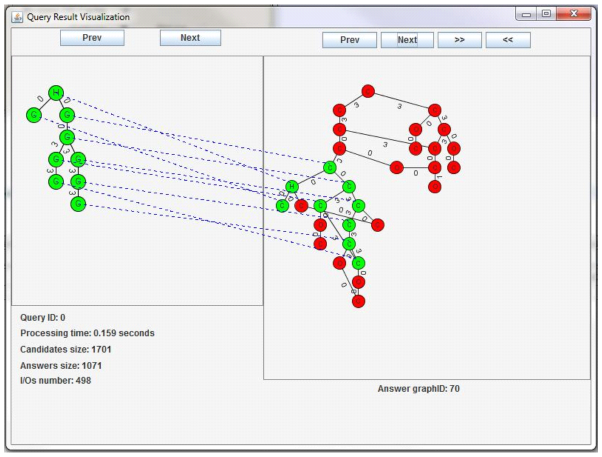
(a) Visual graph editor.   (b) JChemPaint Editor.

Figure 3: Editing tools for graphs and chemical structures.

- **Index building:** The index building tool provides a necessary interface for importing graph datasets into the iGraph framework and building necessary index structures for the selected indexing algorithm using various parameters, e.g., discriminative ratio for gIndex. Due to space limit, we omit the screenshot of this tool.

- **Query processing:** Using the main panel shown in Figure 2, a user can select any indexing algorithm in order to process a selected query set over a specific dataset and then see the statistical information of the performance results.

- **Visualizing results:** The user can visualize the query results as shown in Figure 4 and navigate through the query graphs and the corresponding answer graphs. The user also can view the matching vertices between a query graph and an answer graph.



Query ID: 0
Processing time: 0.159 seconds
Candidates size: 1701
Answers size: 1071
I/Os number: 498
Answer graphID: 70

Figure 4: A screenshot of query result visualization.

3. DEMO SCENARIOS

We show how the iGraph visual tools support the basic operations allowing a user to 1) import a real AIDS antiviral dataset [1], 2) select any of the six indexing algorithms, 3) build necessary index structures, 4) specify a query workload, 5) execute the specified query workload, and 6) navigate through the query results.

We next demonstrate the performance analysis for two indexing algorithms, gIndex and Tree+Δ. However, we would be glad to demonstrate any two algorithms that the audience is interested in.

4. ACKNOWLEDGMENTS

This paper is based on research supported by the R&D program of MKE/KEIT (KI10033545). This paper is also supported in part by MEST/KOSEF (R11-2008-007-03003-0).

5. REFERENCES

[1] Aids antiviral dataset used for gindex. http://www.xifengyan.net/software.htm.

[2] Java universal network/graph framework. http://jung.sourceforge.net.

[3] Jchempaint. http://http://sourceforge.net/projects/cdk.

[4] The pubchem project. pubchem.ncbi.nlm.nih.gov.

[5] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD*, 2007.

[6] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu. igraph: A framework for comparisons of disk-based graph indexing techniques. *PVLDB*, 3(1), 2010.

[7] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.

[8] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *PVLDB*, 1(1), 2008.

[9] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.

[10] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, 2007.

[11] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT*, 2008.