

Analysis and Implementation of Graph Indexing for Graph Database Using GraphGrep Algorithm

Emir Septian Sori Dongoran¹, Kemas Rahmat Saleh W², Alfian Akbar Gozali³

School of Computing
Telkom University
Bandung, Indonesia

¹emirkipang@gmail.com, ²bagindokemas@telkomuniversity.ac.id, ³alfian@tass.telkomuniversity.ac.id

Abstract—Graph database is a database that uses a graph structure to represent and manage the data. Not all types of data are suit for graph database, one that fits is the molecular graph data type. It has characteristic labeled vertices and undirected edges. Graph database also have indexing method. For molecular data type study case, GraphGrep is the most appropriate method because it assume each node in the graph database has a unique number (id-node) and label (label-node). So it is suitable for molecular data type. GraphGrep using a hash table (fingerprint) as an index, comparing the graph database fingerprint with graph query fingerprint to filter the database and use Ullman algorithm to perform subgraph matching. By implementing GraphGrep, we can filter database up to 100% filtering based on length-path we used and get the exact answer set. We also get the most efficient length-path based on the deepest depth in a graph query.

Keywords: graph indexing, GraphGrep, subgraph matching, backtrack, Ullman

I. INTRODUCTION

Enormous growth of data are forcing the use of database almost in every field begin from academic, financial, pharmaceutical, and so on. Most of the databases used are relational databases, for easy use and supports many types of data, such as string, integer, float, and binary.

However, for certain data types such as molecular data type that have characteristic labeled vertices and undirected edges, relational database is less effective because of the type of data used are interrelated independently. To handle this, graph database is the most appropriate solution. But there is a difference in doing search query between relational database and the database graph due to the data contained in the graph database is in graph structure.

To handle this, we need a new search query method for indexing. The graph data type that will be used this time is labeled vertex and undirected edge. From many techniques of indexing such as Gindex [4], FGIndex [2], C-tree [3], and Graphgrep [1], GraphGrep is considered the most appropriate method. Because of its phases commonly used such as using a hash table (fingerprint) for indexing and compare the fingerprint with the fingerprint database

query to filter the database and use Ullman Algorithm to perform subgraph matching[6].

II. THEORETICAL FOUNDATIONS OF THE STUDY

A. Graph Database

Graph database is a database that uses a graph structure to represent and manage the data. Graph is a collection of vertices and edges. Each vertex declare entities (such as people, products, etc.) and each edge declare the relationship between the two vertices. In this project each vertex of the graph is identified by id and label. In addition, the edge we used is undirect.

B. GraphGrep

GraphGrep is one graph indexing techniques that utilizes the path of the graph. GraphGrep assume each node in the graph database has a number (id-nodes) and label (label-node). Edge which used is undirected and unlabeled. For example, in Figure 1, (C, A) are the labels of vertices, while (0,1,2,3) are the id of the vertices.

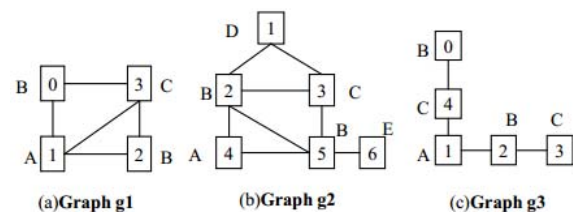


Fig. 1. Graph database consist of three graphs[7]

The steps of the implementation of the GraphGrep algorithm are index construction, database filtering, and subgraph matching.

Index construction. For each graph, it will search for all paths that can be formed, starting with $lp = 1$ to specified lp . Because some path is likely to be on the order of the label (label-paths) are the same, then the id-paths with the same labels will be used as one group. Path-representation of a graph is a set of label-path, and each label has a collection of id-path. Keys in the hash table is the label-path. And values in the hash table is number of id-paths that form the label-path. Hash table in this paper is commonly called fingerprint.

Database filtering. Filtering the database using search query to get the candidate sets by comparing the database fingerprint with graph query fingerprint by removing any graph in the database if its fingerprint's value is greater or equal than the fingerprint's value of graph query.

Subgraph matching. After filtering, we will do the subgraph isomorphism between the query graph with each graph from the candidate set. Then we use Ullman algorithm that uses adjacency matrix and backtrack to do subgraph isomorphism.

C. SMILES

Simplified molecular input line entry system (SMILES) is the notation used to describe the structure of the graph of the data type molecules using ASCII strings. This notation will also be used in the dataset. Writing SMILES has some rules, such as:

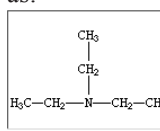
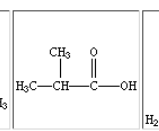
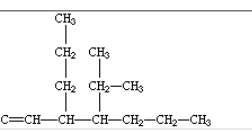
		
<chem>CCN(CC)CC</chem>	<chem>CC(C)C(=O)O</chem>	<chem>C=CC(CCC)C(C(C)C)CCC</chem>
Triethylamine	Isobutyric acid	3-propyl-4-isopropyl-1-heptene

Fig. 2. SMILES branch writing rules in molecular graph

Based on figure 2, each atom string like "C" represent the vertices in a molecular graph, whereas the string "(" represent to make new branch of an atom until it find the string ")".

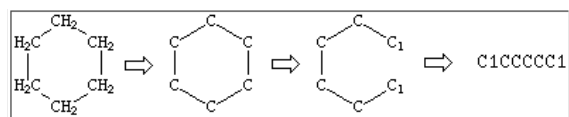


Fig. 3. SMILES cyclic writing rules in molecular graph

Based on the figure 3, the string "1" represent that after an atom string that contained the string "1" has the adjacency relationships resulting in a cyclic graph.

D. Subgraph Matching

The subgraph matching used in this paper to do a subgraph isomorphism is Ullman algorithm.

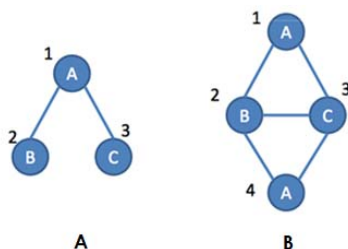


Fig. 4. Subgraph isomorphism. (a) Graph A. Graph B

Suppose we have two graphs such as figure 4 above, the graph A and graph B. Ullman formula

used to determine whether the graph A subgraph isomorphic to the graph B is:

$$MC = M'(M' \bullet MB)^T \quad \forall i \forall j : (MA[i][j] = 1) \\ \Rightarrow (MC[i][j] = 1)$$

Fig. 5. Ullman subgraph isomorphism formula

The first step is we will create a adjacency matrix for graph A and graph B.

	1	2	3
1	0	1	1
2	1	0	0
3	1	0	0

MA

Fig. 6. adjacency matrix for Graph A

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

MB

Fig. 7. adjacency matrix for Graph B

After that we will make the matrix M which means to find label-matching from each vertex in graph A against each vertex in graph B.

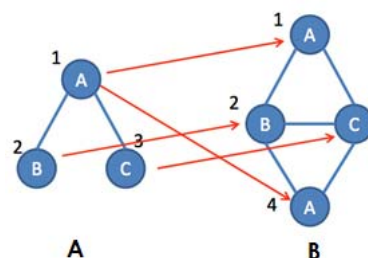


Fig. 8. Label-matching graph A to graph B

Based on the matching of each label vartex from graph A against graph B in figure 8 above, then we have matrix M like below:

	1	2	3	4
1	1	0	0	1
2	0	1	0	0
3	0	0	1	0

Fig. 9. Matrix M

3

The test is performed to determine the time subgraph matching between candidate sets generated from each dataset in the query graph Q1 (look table 4-1) with $lp = 5$ which adjusts the maximum path length / depth in Q1. The subgraph matching is done with Ullman Algorithm. In addition, this test is performed to determine the answer set by the candidate set generated.

C. Analysis of Test Results

After the implementation of the system, which has been designed test scenarios can be fulfilled well. This section will present the results of tests that have been carried out.

1) Analysis the lp influence on index construction time

Based on the testing, the following are the results obtained:

TABLE 2. TABLE OF LP INFLUENCE ON INDEX CONSTRUCTION TIME

	250	500	750	1000	1250	1500
Lp 10	1.25	1.82	2.57	3.20	3.74	4.16
Lp 7	0.77	1.12	1.51	2.15	2.48	2.85
Lp 4	0.48	0.66	0.75	1.18	1.46	1.64

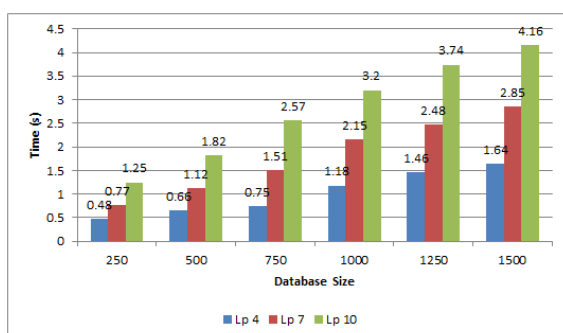


Fig. 12. Chart of lp influence on index construction time

Based on the chart in figure 11 above, it can be concluded that the higher lp is used, the longer the time it takes to build the index on graph database.

2) Analysis the lp influence on filtering results

The testing performed in three different types of queries against different lp also in the different database size. Here are the results obtained (see table 3):

TABLE 3. TABLE OF LP INFLUENCE ON FILTERING RESULT

	250	500	750	1000	1250	1500
Q1 Lp 10	2	9	13	25	45	68
Q1 Lp 7	2	9	13	25	45	68
Q1 Lp 4	6	31	58	93	159	240
Q2 Lp 10	1	5	8	10	11	11
Q2 Lp 7	2	9	17	24	26	27
Q2 Lp 4	20	44	81	115	143	161
Q3 Lp 10	0	2	3	3	3	3
Q3 Lp 7	1	8	16	25	35	56
Q3 Lp 4	14	30	65	92	125	155

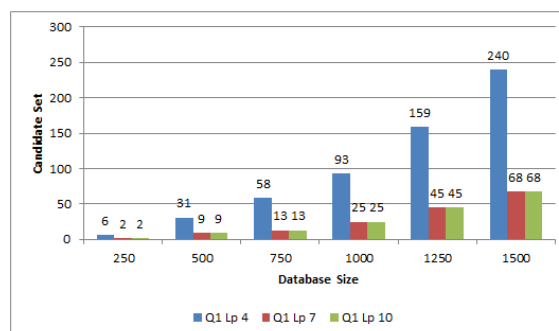


Fig. 13 : Chart of lp influence on filtering result with Q1

Based on the chart in figure 13 above, the higher the lp is used, the less candidate set we get. This has led to a growing number of label-paths generated during the index construction in the graph database and graph query. Then generate more hash table/fingerprint row so the filtering process become more specific. But in $lp = 7$ and $lp = 10$ produced the same number of candidate sets. This is because the maximum path length/depth that can generated by Q1 worth 5, so it can be concluded that the lp is used will depend on the depth of a graph query.

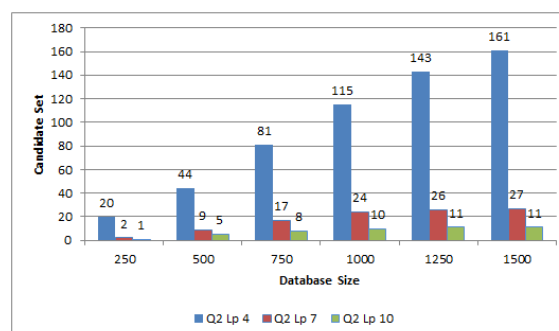


Fig. 14. Chart of lp influence on filtering result with Q2

Based on the chart in figure 14 above, then statements related to figure 13 previously proved true for the higher value of lp , the less candidate sets we get.

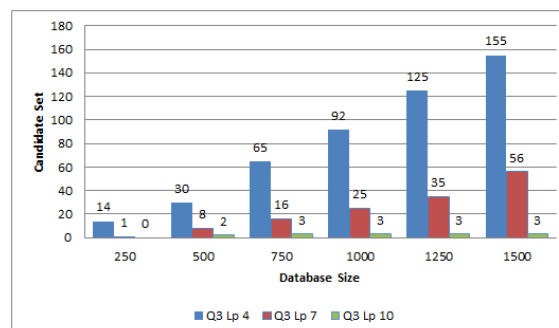


Fig. 15. Chart of lp influence on filtering result with Q3

Based on the chart in figure 15 above, when testing with $lp = 10$ in database with size of 250 graph molecule, it shows that none of the graphs in the dataset is a subgraph isomorphic to Q3. So this shows that the most effective use of lp value in

graph database is when the value is equal to the depth of a graph query.

3) Analysis the result of subgraph matching with Ullman Algorithm

Based on the l_p influence on filtering result with Q1 (see figure 4-5), we get candidate sets those are 2, 9, 13, 25, 45, and 68 totaled molecular graphs in each different database size. Based on the testing of subgraph matching against that current candidate set with graph query Q1 in each different database size using $l_p = 5$, the following are the results obtained:

TABLE 4 .TABLE OF SUBGRAPH MATCHING TIME BETWEEN CANDIDATE SET WITH GRAPH QUERY Q1

	2	9	13	25	45	68
Query	7.66	31.86	97.14	305.08	666.37	1185.34

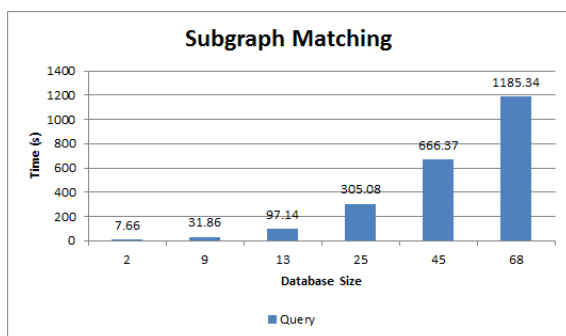


Fig. 16. Chart of subgraph matching time between candidate set with graph query Q1

Based on the table 4 above and table 5 below, the more the candidate set, the more the time required to perform the subgraph matching. In addition, all of the candidate sets can also become the answer sets.

TABLE 5 . TABLE OF COMPARISON BETWEEN CANDIDATE SET AND ANSWER SET ON GRAPH QUERY Q1

No	Database Size	Candidate Set	Answer Set
1	250	2	2
2	500	9	9
3	750	13	13
4	1000	25	17
5	1250	45	24
6	1500	68	44

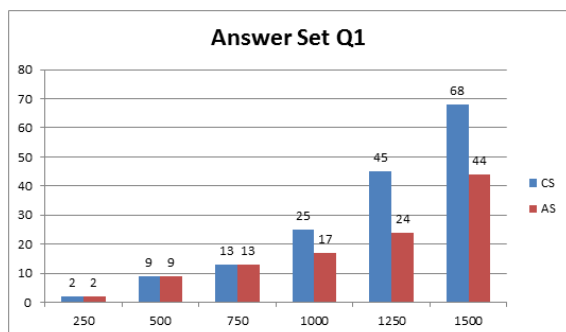


Fig. 17. Chart of comparison between candidate set and answer set on graph query Q1

Based on figure 17 above, subgraph matching with Ullman algorithm successfully performed and showed that at the size of the dataset 250, 500, and 750 the candidate sets generated are also directly become the answer sets. While the size of the 1000, 1250, and 1500 there is a reduction, so the answer sets produced less than the candidate set.

III. CONCLUSION

Based on testing that has been done, it is concluded as follows:

1. The larger the value of l_p is used, the greater the time required for the index construction.
2. The larger the value of l_p is used, the less the candidate sets we get.
3. The value of l_p is used depend on the depth of the graph query we used.
4. The implementation of subgraph matching with Ullman algorithm successfully performed and can produce the same or less answer sets of the candidate set.

IV. FUTURE WORK

Following are some suggestions that can be used as a reference for future research, especially for the scope of graph indexing:

1. Type of data used in this research is the data type of molecule that has a graph structure with labeled vertices and undirected edge and also not weighted, for the future can be done with other data types that have different graph structures.
2. use the datasets with average vertex that is much greater than 50 vertices.
3. It is expected to further research can apply different matching subgraph algorithms such as VF.

REFERENCES

- [1] D. Shasha, J.T-L Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching", In Proc. 21th ACM Symp. Principles of Database Systems(PODS'02), 2002, pp 39-52.
- [2] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: Towards verification-free query processing on graph databases", Accessed from : <http://dl.acm.org/citation.cfm?id=1247574> , Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp 857-872.
- [3] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries", Proceedings of the 22nd International Conference on Data Engineering, 2006.
- [4] X. Yan, P. S. Yu and J. Han, "Graph indexing: A frequent structure-based approach", Accessed from : <http://dl.acm.org/citation.cfm?id=1007607> , Proceedings of the 2004 ACM SIGMOD international conference on Management of data, 2004, pp. 335-346.
- [5] D.W. Williams, J. Huan and W. Wang, "Graph database indexing using structured graph decomposition".

- Proceedings of the International Conference on Data Engineering, 2007.
- [6] D.N. Putra, O. O. Sardjito and C. Lawrence. "Penerapan dan Implementasi Algoritma Backtracking", Published document accessed from: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/Makalah/MakalahStmik26.pdf>, 2005.
 - [7] R. Giugno, and D. Shasha, "GraphGrep: A Fast and Universal Method for Querying Graphs", The proceedings of 16th International Conference on Pattern Recognition, 2002.
 - [8] I. Robinson, J. Webber and E. Eifrem, "Graph databases", CA : O'Reilly Media, Inc, ISBN: 978-1-449-35626-2, 2013.
 - [9] A.T Balaban, "Applications of Graph Theory in Chemistry". Journal of Chem. Inf. Computer Science, 1985, pp 334-343.
 - [10] K. Ruohonen, "Graph Theory", Published Document accessed from http://math.tut.fi/~ruohonen/GT_English.pdf, 2013.
 - [11] A. Silberschatz, H.F. Korth and S. Sudarshan, "Database System Concepts". New York : McGraw-Hill, 2007.