Analysis and Implementation of Graph Indexing on Graph Database using Tree Decomposition Based Indexing (TEDI) Method for The Shortest Path Search

1st Saadah S
School of Computing
Telkom University
Bandung, Indonesia
sitisaadah@telkomuniversity.ac.id

2nd Tohari H
School of Computing
Telkom University
Bandung, Indonesia
hamim.tohari@gmail.com

3rd Kemas R.S.W.H
School of Computing
Telkom University
Bandung, Indonesia
bagindokemas@telkomuniversity.ac.id

***Abstract*— Operating and manipulating data in the graph database becomes most interesting topics to study further, especially concern about expansion of using graph database. By the growing number of data, a new problem arises in shortest path search in graph database. Tree decomposition-based indexing (TEDI) is one of the shortest path search methods that do search process by decomposing graph. In this way, the process of finding the shortest path would not be affected by the size of the graph. However, there are two main factor that influence this method to be well implemented, those are degree of reduction and density of the graph. Our research results shown that the performance of index formation in the TEDI method is not influenced by the number of vertices on the graph, but it is influenced by the number of edges. The greater number of edges, the more solid a graph is. The denser a graph is, the lower the index formation performance would. Then, in a case which a system that have limitations in the form of storage media capacity to store indexes, a large degree of reduction can be used, with the square of the degree of reduction having to be smaller than the number of vertices in the graph.**

Keywords—TEDI, main factor, well implemented, desired system, degree of reduction, density.

I. INTRODUCTION

Operations and manipulation graph data be a major focus of research in the field of graph database along with the growing use of graph databases in various fields [2]. The operation that often used in a graph database is the shortest path searches. However, with the growing amount of data, new problem arises in shortest path search in a graph database, the search performance depends on the number of vertices. By the increasing number of vertices in graph, the search performance is decreasing. The method that can be used to overcome this problem is tree decomposition. By using tree decomposition, many intractable problems can be solved within polynomial or even linear time [2]. Besides, graph can be reduced into tree that the size can be adjust with our constraint.

With those excess, Tree decomposition graph-based indexing (TEDI) [1] is the solution to overcome the problem of scalability in a graph database. TEDI is one method of finding the shortest path in a graph database by using the concept of tree decomposition indexing. The most important thing in this method is the degree of reduction, because its effects on the shortest path search performance and index size. By knowing the influence of those factor on TEDI, we can choose the best value that appropriate with our goal and our system constraint.

The whole schema of the paper is organized as follows. Section II presents the basic knowledge of TEDI. Section III justify the correctness about the algorithm. Section IV show the results of the experiment. Section V concludes the paper.

II. BASIC THEORY

A. Tree Decomposition

Tree decomposition of graph $G=(V,E)$ denoted as T_G is pair of $(\{X_i \mid I \in I, T\})$, where $\{X_i \mid i \in I\}$ is collection of subset V and $T = (I,F)$ is a *tree* that meet [1] :

a. $U_{i \in I} X_i = V$.
b. For each $(u,v) \in E$ there is $i \in I$, so that $(u,v) \in X_i$.
c. For all $v \in V$, set $\{i \mid v \in X_i\}$ induced subtree T.

From these definitions, tree decomposition is correctly to be the decomposition of a graph if and only if meet all the following condition:

a. For each vertex in the graph must appear at least once in a node (a.k.a. bag) in the tree decomposition.
b. For each vertex in the graph that are connected, must appear at least once simultaneously in a node in tree decomposition.
c. For each vertex v that connected in a graph, each node in tree decomposition that containing v must be connected.

B. Bag

Bag is a node in the tree decomposition. Each bag in the tree decomposition consists of one to *n* vertices in a graph. n is the reduction degree that used in the reduction process graph. Vertex or pair of vertices in a graph can appear more than once on different bag.

C. Tree Width

Tree width of a tree decomposition is the greatest cardinality of all the bags that exist in the decomposition tree minus one, or formally the width of tree decomposition $(\{X_i \mid I \in I, T\})$ is defined as $max (\{X_i \mid i \in I\}-1)$ [1]. Bag cardinality is the number of vertices that exist in one bag. While the width of tree decomposition of a graph is the minimum width of all possible tree decomposition of the graph.

978-1-5386-9559-3/18/$31.00 ©2018 IEEE

D. Tree Path

There are two components that make up tree path, those are inner path and inter path. Inner edges are vertex that connected each other in the graph and placed in one bag on a tree decomposition. While inter edge is edge in tree decomposition that made by vertices that located on different nodes. *Inner edge* denoted *({u,i},{v,i})*, while *inter edge* denoted *({u,i},{v,j})*, where *u* and *v* are *vertices* in the graph. Tree path is a sequence of vertices in the tree decomposition connected by both inner edge or inter edge [1].

E. Induced Subtree

Based on the third condition of tree decomposition, for every vertex *v* that connected in a graph, each bag containing *v* must be interconnected, this condition is called induced sub tree [2]. So, in the tree decomposition there are interconnected bag because it has the same vertex. Just like the tree in general, induced sub tree also has root. Root of Induced sub tree is the bag that is located closest to the root of the tree than the other bag on induced sub tree.

F. Youngest Common Ancestor

Youngest common ancestor from node *i* and *j* in a tree is the deepest node in a tree that have node *i* and *j* as its child.

III. METHODOLOGY RESEARCH

Tree decomposition-based indexing (TEDI) is shortest path search scheme that using tree decomposition indexing method [1]. There are two main process in this method, those are index construction, and shortest path searches. Index construction process consist of tree process, namely graph reduction, tree decomposition construction, and local shortest path search.

A. Graph Reduction

The main process of graph reduction is removing the vertices in a graph one by one and then insert the removed vertices into a stack. Vertex removal is done based on its degree. In undirected graph, the degree of the vertex is the number of all edges that connected to it. Elimination of vertices in a graph must meet the definition of *SIMPLICIAL*, that is a vertex v can be simplified or removed from a graph, if all neighbors form a clique [1]. *Clique* on the undirected graph *G=(V,E)* is subset $v \in V$ each pair of which is connected by an edge in *E*, or in other words clique is complete sub graph of *G*[2]. After that, the vertex and its neighbors placed into a stack. The vertex and all edges that connected to it is removed from the graph. This process is carried out continuously until the number of iterations reaches the degree of reduction, or until graph becomes empty.

B. Tree Construction

To construct tree decomposition from the information that stored in stack, firstly check the graph, if the graph is empty, then pop the value of the top stack and create a tree node which contains all of vertices in it, otherwise create a tree node which contain all the remaining vertices on graph. This first node is the root of the tree. Furthermore, do the same process as if the graph is empty. After a node created, then search its parent on tree. Let call the new node as X_c and its parent as X_p. Let further X_c have vertices v, v_1, v_2, v_3 find node X_p in tree that have vertices $v_1, v_2, v_{3, \dots} v_n$, and set is as the parent of X_c. This process is carried out until the stack is empty.

C. Local Shortest Path

After the tree is created, the next process is searching the shortest path for every pair of vertices in all nodes of the tree decomposition. A *path P* from *vertex u* to *vertex v* in the graph is called to be *shortest path* if and only if there is no another *path P'* that connect *vertex u* and *v* in the graph that meet *len(P') < len(P)* [3]. *Local shortest path* search is done using BFS algorithm. BFS algorithm is chosen because it is complete and optimal in finding solution [4]. This condition is enough since our goal is not optimization.

The local shortest path process starts by inspecting a node in tree decomposition. Then create all possible combination for all vertices in it. After that, search the shortest path for all formed combination based on the preliminary graph. Finally save all obtained path in hash map to avoid duplication of the path.

D. Shortest Path

Based on the definition of tree path, all the path in a graph can be mapped on the tree decomposition using tree path. The focus on the path formation is how to determine that the visited bag only bag that included in the simple path. Simple path from bag *i* to bag *j* are all bag on a tree that is always visited by all possible paths from bag *i* to bag *j* [1]. By focusing only on the simple path, the formed path is guaranteed as the shortest distance between source to destination vertex on the tree decomposition.

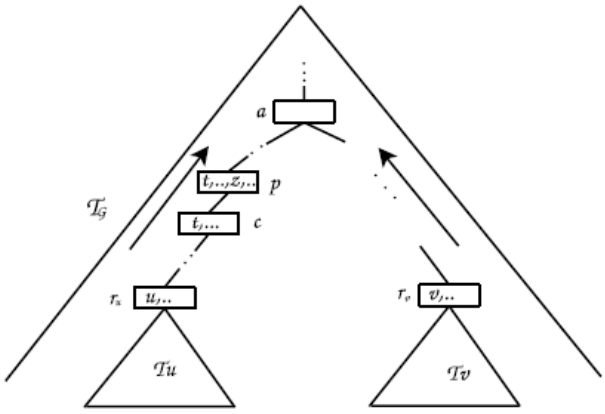

Fig. 1. Shortest Path Search in Bottom Up Manner [1]

Searching and calculating the shortest path distance from source vertex to destination vertex done in bottom-up approach based on tree decomposition structure. The search starts by finding bag that contains source and destination vertex. After the bag is obtained, then conducted search of induced sub tree root for each bag. Suppose the root of the induced sub tree source vertex denoted as r_u and *the root of induced sub tree destination vertex denoted by* r_v. After r_u and r_v obtained, check whether r_u is equal to r_v or not. If this condition is true, then the shortest path is obtained from the index. Otherwise, the system searches the lowest common

ancestor of r_u and r_v. If r_u is the lowest common ancestor, then switch the source and destination vertex, as well as r_u and r_v. This action is aimed to perform a search only on the side of the destination vertex. Furthermore, labeled every vertex x which is in the source bag with the distance from the source vertex to x.

At last, the shortest path search is done in a bottom-up from r_u to its parent until reach the lowest common ancestor. In this process, suppose the current bag is X_c and parent's bag is X_p, then the shortest path search from vertex z in *node* X_c to *vertex* p in *node* X_p carried out by the following conditions:

1. If vertex $z \in X_c$, and $z \in X_p$. In this case, sdist (shortest distance) from p to z or *sdist (p,z)* has been calculated at indexing process, so it does not need to be recalculated.
2. If vertex $z \in X_c$, and $z \notin X_p$, then the shortest path updated with the calculation $sdist(p,z) = min(sdist(u,t) + sdist(t,z))$, $t \in X_c$, and $t \in X_p$.

Once the search has reached the bag which is first child of the lowest common ancestor, then do intersect operation between vertices in bag of the lowest common ancestor and it. This operation was conducted to avoid the complexity of search if the lowest common ancestor is the root of the tree.

The next process is check whether r_v is the *ancestor* of r_u or not. If this condition is meet, then the distance from the destination vertex to the lowest common ancestor is zero, otherwise do the same process to calculate the shortest path in the bottom up manner from r_v to the *lowest common ancestor*.

After that, calculate minimum distance from vertices in the two bag that get in bottom up search process. The calculation is done with $mdist = min(du(x) + dv(y) + sdist(x,y))$, where x is a vertex in the last bag from bottom up by source side and y is the a vertex in the last bag from bottom up by source side. Finally, the shortest path is obtained based on the information in the node x and y.

IV. Experimental Result

This research conducted three experiment. The first experiment is performed to find the shortest path and ensure that our method is valid. The second test is performed to know the influence of density of graph in TEDI. The last test is performed to know the effect of the reduction degree to the performance of index construction, index size, and the shortest path search performance.

The first test conducted by searching the shortest path from five pairs of vertices that taken randomly from a graph. To validate the resulting path is valid or not, then the path is compared with shortest path algorithm derived from the library.

TABLE I. Shortest Path Testing

Source Vertex	Destination Vertex	Shortest Path TEDI	Shortest Path Library
1	10	[1, 0, 10]	[1, 0, 10]
3173	152	[3173, 58, 0, 152]	[3173, 58, 0, 152
2885	582	[2885, 171, 107, 414, 582]	[2885, 58, 107, 414, 582]
3437	3173	[3437, 698, 860, 1684, 58, 3173]	[3437, 567, 428, 1912, 58, 3173]
2649	10	[2649, 136, 67, 10]	[2649, 136, 169, 10]

The results show that both TEDI or shortest path from library has the same distance. This means that the shortest path that generated using TEDI is valid.

TABLE II. Effect of Number of Edge Against Index Construction Performance

Number of Edge	*Tree* Construction (ms)	*Local Shortest Path Search* (ms)	*Indexing Time* (s)
250	2.75	2.096	0.004846
1000	5.392	113.824	0.119216
4000	22.234	984.738	1.006972
16000	57.114	2736207.015	2738.97893
48000	89.324	89765588.18	89765.6775

The second experiment is conducted by varies the number of edge and vertices on graph. All the experiments in Table II is conducted with 800 vertices and reduction degree is 16. While the experiments Table III is conducted with 3043 edges and reduction degree is 9.

Table II is the result of experiment to find out the influence number of edges to TEDI. From the experiment we can conclude that the greater number of edges, the indexing performance gets decrease.

TABLE III. Effect of Number of Vertex Against Index Construction Performance

Number of *Vertex*	Tree Construction (ms)	*Local Shortest Path* Search (ms)	*Indexing* Time (s)
400	2.67	9569.66	9.572
800	5.58	2615.068	2.621
1600	9.67	1990.852	2.001
3200	10.7542	1212.11	1.223
4039	12.794	1167.606	1.180

From the result of experiment to find out the effect of number of vertices on graph in Table III, we can conclude that the greater number of vertices, the performance of indexing gets decrease. But if we observe both Table II and Table III, the decreasing of indexing performance in Table III is not because the increasing number of vertices, nevertheless by the decreasing density of the graph, therefore experiment in both Table II and Table III could be concluded that the indexing performance in TEDI is influenced by the density of the graph, the more dense of the graph, the performance of indexing gets decrease.

Final test was done by selecting some degree of reduction. Furthermore, the process of indexing and shortest path search is conduct using any chosen value of the reduction degree.

TABLE IV. Effect Of Reduction Degree Against Index Construction Performance, Index Size, And Query Answering Performance

Reduction Degree	*Indexing* Time (ms)	*Index* Size (Mb)	*Shortest Path Searching Time* (ms)
1	1786812	21.291	0.5208
5	749298	13.88	0.8516
10	471529	8.49	10.9098
15	316067.7	5.731	13.2836
20	225568.4	4.089	13.6654

25	180396.9	3.39	24.493

In Table IV, the performance of index construction is increased along with increasing the value of the reduction degree. During the development process of tree decomposition, the higher value of reduction degree, the performance of index development gets decreased. This is because, the result of multiplying average number of vertices in the bag with the number of tree nodes is increasing, so it takes a longer time to construct tree.

The next index component is local shortest path. In contrast to the tree construction performance, local shortest path search performance is equal to the value of the degree of reduction. Local shortest path search performance is increasing along with the increasing of reduction degree. This is because local shortest path search performance depends on the number of vertices in the root. The smaller number of vertices in the root, the few numbers of combination local shortest path search that had to be done.

TABLE V. INDEX SIZE

Derajat Reduksi	Tree Size (Mb)	Local Shortesh Path Size (Mb)	Index Size (Mb)
1	0.007	21.284	21.291
5	0.016	13.864	13.88
10	0.028	8.462	8.49
15	0.037	5.694	5.731
20	0.046	4.043	4.089
25	0.05	3.34	3.39

In Table V it is known that the largest component in the index is a local shortest path. The size of the local shortest path depending on the amount of stored path. The smaller reduction degree, the greater number of paths that must be searched and stored. This is because the size of the tree root is the greatest among another bag on the tree.

The final testing is knowing the effect of the degree of reduction on the shortest path search performance. In Table IV shortest path search performance decreases along with the increasing reduction degree. This is because the shortest path search is visiting the tree nodes one by one in a bottom-up, and every time tree node is visited, the path for each vertex in the tree node is updated. Thus, this operation is highly dependent on a tree and the average number of vertices in each bag.

V. CONCLUSION

The performance of index formation in the TEDI method is not influenced by the number of vertices on the graph, but it is influenced by the number of edges. The greater number of edges, the more solid a graph is. Then, the denser a graph, the lower the performance of the index formation.

Systems that have limitations in the form of storage media capacity to store indexes, a large degree of reduction can be used with the square of the degree of reduction must be smaller than the number of vertices on the graph. As a result, a large degree of reduction influences the shortest path search performance to decrease.

Meanwhile, during the focus is the performance of a good system path shortest search with the capacity of storage media is not a problem, then the value of the reduction degree used can be selected as small as possible with the minimum reduction degree limit selected must be greater than zero. however, when the value of the degree of reduction is small, the performance of the indexing will decrease considerably.

REFERENCES

[1] A. Castelltort and A. Laurent, "Representing History in Graph-Oriented NoSQL DataBase: A Versioning System", Eight International Conference on Digital Information Management ICDIM, pp. 228-234, September 2013.

[2] F. Wei, "TEDI: Efficient Shortest Path QueryAnswering on Graphs", ACM SIGMOD, pp. 99-110, January 2010.

[3] M. Grohe, "Definable Tree Decomposition", 23rd Annual IEEE Symposium on Logic in Computer Science, p. 406, June 2008.

[4] A. Kaveh, Introduction to Graph Theory and Algebraic Graph Theory, Wien: Springer-Verlag, 2013.

[5] I. Robinson, J. Webbe and E. Eifrem, in Book Graph Databases, Cambridge, O'Reilly, p. 5, 2013.

[6] T. H. Cormen and C. D. Leiserson, Introduction to Algorithms second edition, Toronto: McGrawHill, 2011.

[7] R. J. Wilson, Introduction to Graph Theory fourth edition: Longman, 1996.