Journal of Theoretical and Applied Information Technology
10th January 2016. Vol.83. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

GRAPH PATTERN MATCHING IN YEAST DATASET

[1]DION TANJUNG, [2]KEMAS RAHMAT SALEH W, ST., M.ENG., [3]SHINTA YULIA P, ST.
[1]School of Computing, Telkom University, Bandung, Indonesia
[2]School of Computing, Telkom University, Bandung, Indonesia
[3]School of Computing, Telkom University, Bandung, Indonesia
E-mail: [1]dae.one11@gmail.com, [2]bagindokemas@telkomuniversity.ac.id,
[3]shintayulia@telkomuniversity.ac.id

ABSTRACT

Graph representation has been widely used in scientific study to analyze a pattern. Geographic maps, computer networks, chemical structures, and databases are examples of information that can be applied to graph representation. This is related to graph representation method that makes manipulation data easier.The needs and growth of information make graph representation larger. Pattern analysis need more time while searching in larger graph. Therefore we need new method to searching graph pattern in a large graph. In thisresearch , graph query language(GraphQL) algorithm is used for searching graph pattern in graph database using yeast dataset. This algorithm is used because it can search graph pattern in graph databases by reducing search space. The result obtain combination of local pruning and global pruning can reduce search space with the result of reduction ratio and running time being lower. Smaller search space make process of searching graph pattern in graph faster.

Keywords: *GraphQLAlgorithm, Reduction Ratio, Running Time, Graph, Graph Pattern*

1 INTRODUCTION

Nowadays, pattern analysis development can not be separated from the role of the graph. Data in multiple domains can be modeled as graphs for ease of data manipulation processes [5], like chemical compounds, geographic maps, computer networks, and databases. Graph is a collection of nodes and edges. Node is described as a point or an object, and the edge is a relation between nodes.

From the point of view of the graph pattern, the most important problem is matching graphs or subgraphs for comparing them[2]. The growing heterogeneity and size of the data that increase over time,make the existing data models, query languages and database systems do not support for the modeling and management of this data[5]. The growing of the data need to be saved in semi structured information. To deal with this problem, database using graph representation. Various studies are being done to reduce the computational cost when matching a large graph with the number of nodes above 100[2]. Computational complexity of the algorithms need to be considered when matching large graphs. Matching on large graphs requires low computational complexity to speed up running time of the process matching. One technique used to reduce computational cost is reducing search space. Result of the graph matching are not limited to the node and edge information. Structural information and value of the entire object of interest is needed, so there is no information loss[5]. This means the ability of graph matching to return a graph or set of graphs.

Graph Query Language (GraphQL) is one method that can be used to searching graph pattern in a graph. The GraphQL algorithm uses profile neighborhood signature based pruning and the pseudo subgraph information test based pruning to reduce the size of the candidate sets[7]. Results obtained from the GraphQL algorithm process is a graph or set of graphs. In this paper we implemented the GraphQL algorithm in graph database using yeast dataset. The graphQL algorithmalso tested usingmany different size of graph pattern from this dataset. The purpose is to find out how GraphQL algorithm deal with different size of graph pattern. The other purpose isto find out the effectiveness of reduction ratio and running time in local pruningand global pruning process that should be carried out in sequence, find outthe influence of search space size to the running time in matching process, and find out the influence of the optimization phase in the running time.The limitation of this research are as follow: simple graph, undirected graph, using adjajency list to implements graph, and choose graph with more than 100 of nodes.

2 BACKGROUND

In this section we describe graph query language, graph, graph database, and GraphQL algorithm.

2.1 Graph Query Language

Graphquerylanguagehasvarious definitions. Herearesomeexamples ofdefinitions ofgraphquery language[5]:

1. Data Model

   Data model in GraphQL algorithm can be described as a graph motif that has an attribute on the node or edge.

   Example :

   Graph G{

   Node v1 <year=2006>;

   Node v2 <name="A">;

   }
2. Graph Pattern

   Graph pattern is a graph motif plus a predicate on attributes. Graph pattern is also called a graph query that used to select graphs of interest.Graphpatternis representedas follows :

   P = (M,F)

   M is graph motif and F is a predicate.

   Example :

   Graph P {

   Node v1 where name ="A";

   Node v2 where year>2000;

   };
3. Graph Pattern Matching

   P (M, F) same as the graph *G* if there is injective mapping 1 to 1, *$\emptyset$: V (M)$\rightarrow$V (G) for $\forall$ e (u, v) $\in$ E (M), ($\emptyset$ (u), $\emptyset$ (v))* is an edge of *G* and predicate *F$\emptyset$ (G).*Example of mapping graph pattern *P* on the graph *G*

   Mapping*$\emptyset$:*

   $\emptyset$(P.v1)$\rightarrow$G.v2

   $\emptyset$(P.v2)$\rightarrow$G.v1

2.2 Graph

Graph is a collection of nodes and edges. Node is described as a point or an object and the edge is described as a relation between nodes. Based on the orientation, the graph can be divided into two types: directed graph and undirected graph. Directed graph is a graph which has direction so that connectivity between nodes only in one direction. Connectivity between nodesin undirected graph can be two-way. Example: *E = {{x, y}}* in a directed graph indicates that the relation of the node only from node*x* to *y*. *X* is the predecessor and *y* is the successor. If the graph is undirected, the two nodes will be interconnected. Node*x* is related to node*y* and node*y* is related to *x*. Here are some graph definitions:

1. Thesimplegraphisa graphthatdoes nothave a loop.
2. Subgraph is a part of graph that has same nodes or edges on the graph.
3. Path represent a way from a node as start point to a destination nodewhichdoes not pass throughthe nodeoredgetwice.
4. Clique in a graph is a subgraph complete which every node linked each other.
5. Matching in a graph is a set of edges which connecting nodes only one to one.
6. Graph to be perfectmatching when allnodesin a graph hasmatchingedge
7. Semi-perfect matchingis aperfectgraphmatchingthatleaves onenodewithoutedgedue totheoddnumber of nodes
8. The bipartite graph is a graph that can be divided into two subsets, such that no nodes are interconnected in the same set. If all nodes in a set connected to another set of nodes that called graph bipartite complete.

2.3 Graph Database

Graph database is one of the categories of NoSQL who model a database in the form of a graph. Just like the graph, node is a data or entity and edge is the relationship between nodes. Modeling graph database using functions Create, Read, Update, and Delete to manipulating data. Graph database can be represented using adjacency list or implementations to array 2D. Graph database consists of [8]:

1. Graph Storage

   Graph storage is a storage area graph database. Graph database used to store the graph native storage, data serialization graph into relational databases, object-based database, and others.

2. Graph Processing Engine

   Used to form a graph database and set of data processing.

2.4 GraphQL Algorithm

GraphQL algorithm is one of graph pattern matching algorithm that is able to handle small to large-sized of graph. The core of GraphQL algorithm is graph algebra for the selection process and composition. Selection for graph pattern matching and composition to generate a new graph from matched graph. Figure 1 is an example of a graph *G*and figure 2 is a graph pattern*P*. Both graph *G* and graph pattern *P*have label*A, B, C*, and *D*.

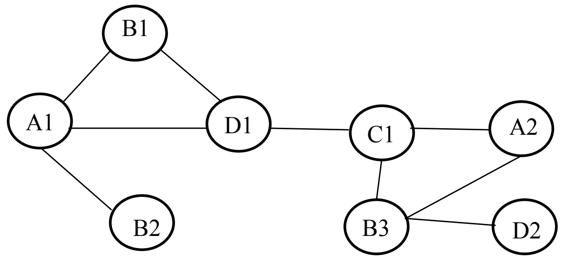


Figure 1: Graph G

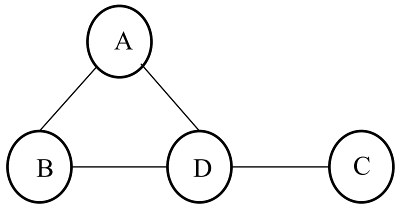


Figure 2 : Graph Pattern P

2.4.1 Retrieve of feasible mates

Feasible mates *Φ (u)* of node *u* is a set of nodes of in graph *G* that satisfies the predicate fu in the graph pattern*Φ(u) = {v | v∈V (G), fu(v) = true}*. From the figure 1, feasible mates each nodes is *Φ(A) = A1, A2, Φ(B) = B1, B2, B3, Φ(C) = C1,* and *Φ(D) = D1, D2*. The resulting product of *(A) x Φ (B) x Φ (C) x Φ (D)* is search space.

2.4.2 Local pruning

Search space need to be reduced for fast matched process, because matched process is checking all nodes that contained in the search space. Local pruning use light weight profile to prune the search space. One can define the profile is node labels. This process is checking profile each neighborhood from feasible mates in radius 1. From figure 1, neighborhood of node label *A* in graph pattern is nodes *B* and *D*, neighborhood of node *A2* with label *A* is nodes *B3* with label *B* and *C1* with label *C*. Node *A* in graph pattern and node *A2* in graph have same label *B* from neighborhood, but node *A* has label *D* and node *A2* not has label *D*. The conclusion is node *A* and node *A2* is not sub-isomorphic, so node *A2* is pruned from the search space. The search space after the pruning process is *Φ (A) =A1,Φ (B) =B1,B3,,Φ (C) =C1,* and *Φ (D) =D1.*

2.4.3 Global pruning

This pruning is the process of join reduction of search space. This pruning technique reduce the overall search space iteratively using the concept of pseudo isomorphism test. Node *u* and the feasible mates of node node *u* need to be checked whether sub tree of node *u* in *Tu* sub-isomorphic with the node of *v* in *Tv*. From the algorithm in figure 2, line 3-18 checking the sub-isomorphic of *Tu* and *Tv* using Breadth First Search (BFS) algorithm until the depth of adjacent sub trees. Each sub-isomorphic sub tree is checked with radius d=1. First process is marking node u and node v from the search space to checked for semi-perfect matching. line 5-9 is construct bipartite graph $B_{u,v}$ from the sub tree of marking nodes. Bipartite graph $B_{u,v}$ is checked for graph semi-perfect matching to pruning nodes. Graph is semi-perfect matching if graph $B_{u,v}$ is complete bipartite graph and each nodes in $B_{u,v}$ is connected 1 to 1. If bipartite graph $B_{u,v}$ not semi-perfect matching, node *v* is pruned from search space and marking node *u'* and *v'* for each neighborhood of node *u* and *v*. This process can be done if there is no marked node *u* and *v*.

```
Algorithm 2: Refine Search Space
 Input: Graph Pattern $\mathcal{P}$, Graph $G$, Search space
        $\Phi(u_1) \times .. \times \Phi(u_k)$, level $l$
 Output: Reduced search space $\Phi'(u_1) \times .. \times \Phi'(u_k)$
1  begin
2   foreach $u \in \mathcal{P}, v \in \Phi(u)$ do Mark $\langle u, v \rangle$;
3   for $i \leftarrow 1$ to $l$ do
4      foreach $u \in \mathcal{P}, v \in \Phi(u)$, $\langle u, v \rangle$ is marked do
5         //Construct bipartite graph $\mathcal{B}_{u,v}$
6         $N_{\mathcal{P}}(u), N_G(v)$: neighbors of $u, v$;
7         foreach $u' \in N_{\mathcal{P}}(u), v' \in N_G(v)$ do
8            $\mathcal{B}_{u,v}(u', v') \leftarrow \begin{cases} 1 & \text{if } v' \in \Phi(u'); \\ 0 & \text{otherwise.} \end{cases}$
9         end
10        if $\mathcal{B}_{u,v}$ has a semi-perfect matching then
11           Unmark $\langle u, v \rangle$;
12        else
13           Remove $v$ from $\Phi(u)$;
14           foreach $u' \in N_{\mathcal{P}}(u), v' \in N_G(v), v' \in \Phi(u')$
             do Mark $\langle u', v' \rangle$;
15        end
16     end
17     if there is no marked $\langle u, v \rangle$ then break;
18  end
19 end
```

Figure 3 : Global Pruning Algorithm[5]

From figure 1, neighborhood of node *A2* is *B3* and *C1*, neighborhood of node *A* is *B* and *D*. Each sub tree of node *A2* and *A* is used to construct bipartite graph $B_{a,a2}$ with left sets are *A* sub tree and right sets are *A2* sub tree. After that, bipartite graph $B_{a,a2}$ is checked for semi-perfect matching. The result of graph $B_{a,a2}$ is not semi-perfect matching because node *A2* not has *D* label of neighborhood so neighborhood *D* from node *A* is not connected to any nodes then node *A2* is pruned from search space. Next iteration, we explained to checked node *B* and node *B3*. Neighborhood of node *B* is *A* and *D,* neighborhood of node *B3* has label *A,C,*and *D*. At the previous iteration, node *A2* is pruned from the search space so node *A* in graph $B_{b,b3}$ can not connected node *A2*. Graph $B_{b,b3}$ is not complete graph bipartite and not semi-perfect matching, so node *B3* is pruned from search space. The result of search space after global pruning process is *Φ(A) =A1,Φ(B)= B1,Φ(C)=C1,* and *Φ(D)=D1.*

2.4.4 Optimization

The goal in this optimization is finding the good search order for better running time the process matching. This optimization is modeled each *Φ (U)* as nodein binary root tree andcalculate the size and the cost of the search order. Size of the tree model can be estimated by :

$$size(i) = size(i.left) x\ size\ (i.right)\ x\ \Upsilon(i)(1)$$

i.left is child node in left i and *i.right*is child node in right i. size left and size right is be obtained from amount of nodes in *Φ (U)*. ϒis the reduction factor that can use constant reduction values [5].

Cost of the tree model can be estimated by :

$$cost(i) = size(i.left) x\ size\ (i.right)(2)$$

The cost total of search order Γ can be estimated by

$$Cost(\Gamma) = \sum_{i\in\Gamma} \mathrm{Cost(i)}(3)$$

For example,search space from local pruning is *Φ (A) =A1,Φ (B) =B1,B3, ,Φ (C) =C1,* and *Φ (D) =D1* with the size of *Φ (B)* is 2 because *Φ (B)* have 2 nodes and other size is 1.

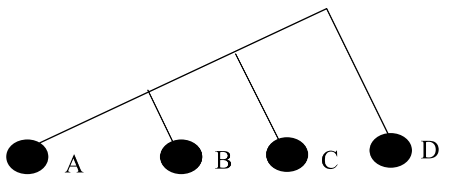


Figure 4 : Search Order A ⋈ B ⋈ C ⋈ D

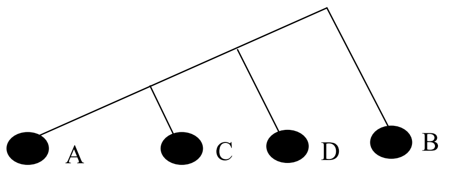


Figure 5 : Search Order A ⋈ C ⋈ D ⋈ B

From figure 4, the search order is $(((A\bowtie B)\bowtie C)\bowtie D)$. *Size* $(A\bowtie B) = 1\ x\ 2 = 2$ and cost $(A\bowtie B) = 1\ x\ 2\ x\ \Upsilon = 2\Upsilon$. *Size* $((A\bowtie B)\bowtie C = 2\Upsilon\ x\ 1 = 2\Upsilon$with cost is $2\Upsilon x1\ x\ \Upsilon = 2\Upsilon^2$. cost $(((A\bowtie B)\bowtie C)\bowtie D) = 2\Upsilon^2 x1 = 2\Upsilon^2$so the total cost if $\Upsilon = 1$ is $2+2\Upsilon + 2\Upsilon^2 = 2+2(1)+2(1)^2 = 2+2+2 = 6$.Similary, the total cost of $(((A\bowtie C)\bowtie D)\bowtie B)$in figure 5is 4. Thus, $(((A\bowtie C)\bowtie D)\bowtie B)$is better than $(((A\bowtie B)\bowtie C)\bowtie D)$.

2.4.5 Graph pattern matching

After optimization process, all of nodes in *Φ (A)Φ,(V),Φ (C),* and*Φ (D)*is combined to make a combination of node that matched with the graph pattern.

```
1.   void Search(i)
2.   begin
3.         foreach v ∈ Φ(ui), v is free do
4.         if not Check(ui, v) then continue;
5.         φ(ui) ← v;
6.         if i<|V (P)| then Search(i + 1);
7.         else if Fφ(G) then
8.                   Report φ ;
9.                      if not exhaustive then stop;
10.        end
11.  end
12.
13.  boolean Check(ui, v)
14.  begin
15.        foreachedge e(ui,uj) ∈ E(P), j<i do
16.        ifedge e (v,φ(uj)) ∉ E(G) or not Fe(e ) then
17.        return false;
18.        end
19.        return true;
20.  end
```

Figure 6: Matching Algorithm[5]

From figure 6, procedure boolean check(*ui,v*) is checking node *ui* whether that node can be mapped *v*. Line 15 examine if node ui is neighborhood of *uj* and examine if node *v* is an edge of node *uj* then return false. Procedure search(1) is examine node from first index or from *Φ (A)*if the first searh order is *fromΦ (A)* . For each node v in *Φ (ui)*is checked in procedure check, if the result is false then the iteration is examine the next *v* (line 4). If return true thennode *v* is replace node *ui*. Line 6 is checking if index of i is less then the size of graph pattern, if true then add I with 1 to search next index in recursive phase. If i is the size of graph pattern then check the combination result. The result of this matching algorithm is not necessarily same as the graph pattern and the results of the new graph can be duplicated. To avoid that problem, we comparing the size of the new graph and the graph pattern. For example, node *YNL*in the new graph has neighborhood*YAL*, *YOL*, and*YDR*, Node*YNL*in*graph pattern*has neighborhood*YAL* dan *YDR*. The new graph and the graph pattern in that case is not sub-isomorphic so the new graph is not the answer of the graph pattern. To avoid duplicated new graph, we checked the previous answer of the graph pattern. The answer of the graph pattern is a set of new matched graph, if new graph pattern found is similar as other result so this new graph is not counted as the answer. For example, combination of nodes $\{\{YAL, YNL, YDR\}, \{YNL, YDR, YAL\}\}$ create similar graph so we choose the first combination as the answer.

3 EXPERIMENT

3.1 System design

In this research, will be built a system that is used for the analysis of algorithms GraphQL. General overview of the system to be built is as follows:

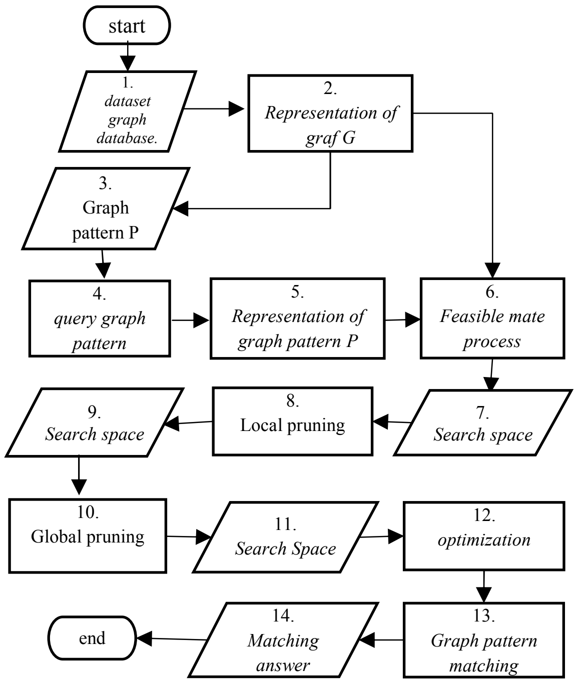
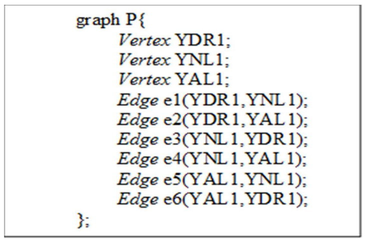


Figure 7: System Design

From figure 7, we evaluate the performance of GraphQL algorithm on yeast dataset that consist 2361 nodes, 7182 edges, and 536 loops. This dataset was taken from http://vlado.fmf.uni-lj.si/in pajek format and we change this pajek format into excel format so our system can read this format. This dataset is represented into simple graph G to produce more varied graph pattern. Id example of this dataset is *YBR236C*. Because this dataset has no label, we make label from Yeast mark(letter 1), number of kromosom(letter 2), and left-right mark(letter 3) for each id. From the example, id *YBR236C*has label *YBR*.Using this technique, this dataset contains more than one label so we have a lot of similar pattern from this label. This idea is to know the effectiveness of pruning process when deal with similar pattern because pruning process check from the label. Although this process is checking the node label, but the result of pattern must matching with the node id.

Graph G on system consist 2361 nodes, 13292 edges, and has no loops because we changed into simple graph. From this graph G, we searching path and cliques to determine amount of graph pattern to be tested. We choose the longest path and the largest clique to make a graph pattern. To searching all of paths contained in graph G, we using dijkstra algorithm. This algorithm is used to searching the shortest path, so we replace and added some function so this algorithm can searching all path contained in graphs. The longest path found in graph G is 22, because path length is 11 to 22 only a few are found, so we choose the longest path uses is 10. We break that path to make path with length 1 to 10. Same as path, we use Bron-Kerbosch algorithm to search all cliques contained in graph G. The largest cliques size is 9 so we break this cliques into cliques size of 1 to 9.

Graph pattern used for testing need to change into query form because system read the input of graph pattern in query form. Figure 8 is the example of query used. This query is to make graph pattern for cliques which has 3 nodes:*YDR1, YNL1*,and *YAL1*interconnected.



```
graph P{
        Vertex YDR1;
        Vertex YNL1;
        Vertex YAL1;
        Edge e1(YDR1,YNL1);
        Edge e2(YDR1,YAL1);
        Edge e3(YNL1,YDR1);
        Edge e4(YNL1,YAL1);
        Edge e5(YAL1,YNL1);
        Edge e6(YAL1,YDR1);
};
```

Figure 8: Clique Query

Graph pattern*P* and graph *G* is executed together to determine search space based on feasible mate rules. This search space size is reduced by local pruning technique and then be reduced again by global pruning. Search space be optimized by sorting the sequence of the nodes. After that, all nodes in search space combined to make the answer of graph pattern in graph using matching algorithm.

3.2 Experiment result

Here are the result of the experiment. This experiment using two kinds of graphs: paths and cliques. The sizes of Paths used from 1 to 10 because they have to represent the entire length of the paths contained in the graph. Cliques used from sizes 2 to 9 because the largest clique size is 9 and sizes greater than 9 have no answer.

For this experiment, we use a PC with Intel I5 3210M 2.5 GHz, 4GB memory and Hard disk 750GB serial ATA 5400RPM. GraphQL algorithm is implemented with java language in netBeans IDE 8.0.2. Library used in this experiment is jxl from http://jexcelapi.sourceforge.net/ to read dataset that have been stored in .xls format. This dataset is represented as a graph using adjacency list because more efficient when searching the neighborhood than using matrix.

3.2.1 Pruning analysis

This testing is used to analyze reduction ratio and running time performance of local pruning and global pruning. This process need search space for input data. We make 3 scenario in this experiment. First, search space from feasible mate process is reduce by local pruning process (F-P1). Second, search space from feasible mate process is reduce by global pruning process (F-P2) without running local pruning. Third, all processes running sequential. Search space from feasible mate process reduced by local pruning then from local pruning process reduced again by global pruning (F-P1-P2).The parameter that used in this testing is time and size of search space to calculate the reduction ratio. Running time and reduction ratio is the parameter to compare the performance of local pruning and global pruning individually.

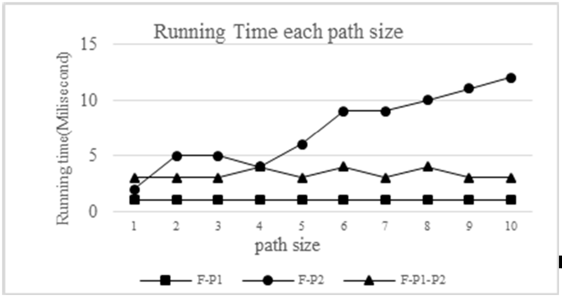


Figure 9: Running Time each Path Size

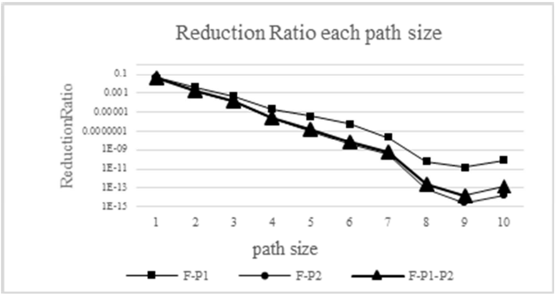


Figure 10: Reduction Ratio each Path Size

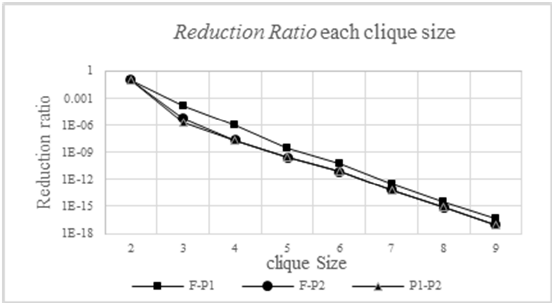


Figure 11: Running Time each Cliques Size

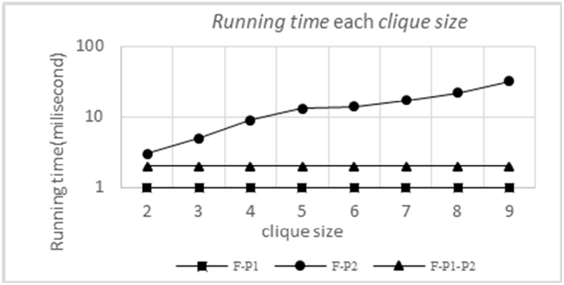


Figure 12: Reduction Ratio each Cliques Size

Figure 9 to 12 shows the reduction ratio and running time both path and clique queries. Local pruning has a running time that is better than the global pruning, but the reduction ratio of global pruning is better than local pruning. This is due to local pruning process that just check the profile of neighborhood sub graph of radius 1 so the running time is fast. Meanwhile, global pruning reduce the overall search space iteratively by checking for each node u in graph pattern P and feasible mate of node u in graph G sub isomorphic. This process take a longer time because this checkwill be done iteratively until the depth of the adjacent subtree. Although this process takes a longer time, this process of global pruning cuts better than local pruning. If local pruning and global pruning are combined, the result of reduction ratio and running time give high pruning power and the running time is faster than using global pruning only.From this result, we can not separate local pruning and global pruning if we want to have the best pruning result.

3.2.2 Search space size analysis

This testing using search space from global pruning process as input data. This search space executed in matching process to getthe answer of graph pattern. Running time of this process is used to analyze the effect of search space size in graph matching process.

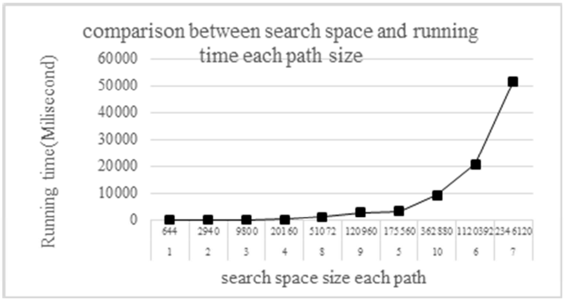


Figure 13 : Search Space and Running Time each Path Size

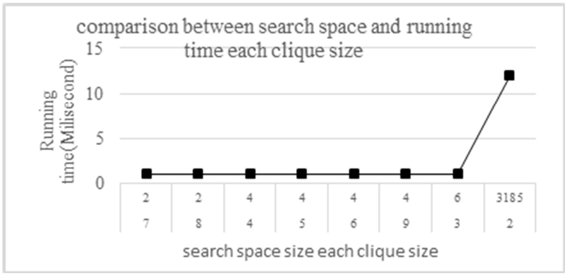


Figure 14 : Search Space and Running Time each Cliques Size

Figure 13 and 14 show that the larger of search space make running time takes longer time. This is due to the matching process algorithm that combined all nodes in the search space to make a new graph. The size of the search space is influenced by the complex of patterns and the number of labels contained in a graph. Less number of label contained in the graph can make the size of the search space larger because local pruning reducing search space by checking the label.On the other hand, global pruning reducing search space by checking the sub tree pattern for each nodes. If the pattern is complex and less label, the size of search space can be longer.From this result, GraphQL algorithm is have better running time when in the graph have a lot of different pattern and a little bit of label.

3.2.3 Optimation analysis

This testing is analyze matching process using search space with optimization and without optimization as input data. Local pruning and global pruning is used to reduce search space. This reduced search space be optimized then be executed in matching process to get time when search space optimized. On the other hand, the reduce search space are not optimized then executed to get the time without optimization. The results obtained are the effectiveness of the optimization phase based on running time process.

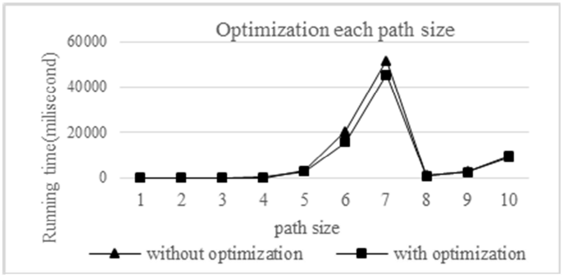


Figure 15: Optimization each path size

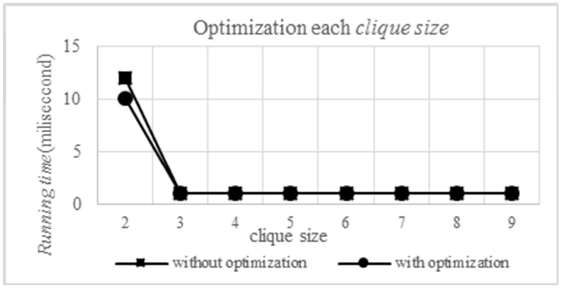


Figure 16: Optimization each clique size

Figure 15 and 16show that optimization process improves the result of running time process. Search space with optimization show faster running time better than search space without optimization. This result due to optimization process are sorting vertexes in the search space based on cost model calculation. Before optimization, the search order was random and after optimization the search order has been ordered for better execution process.

3.2.4 Individual time Analysis

This testing is analyze individual time each process of GraphQL algorithm. GraphQL algorithm is executed without optimization and with optimization.

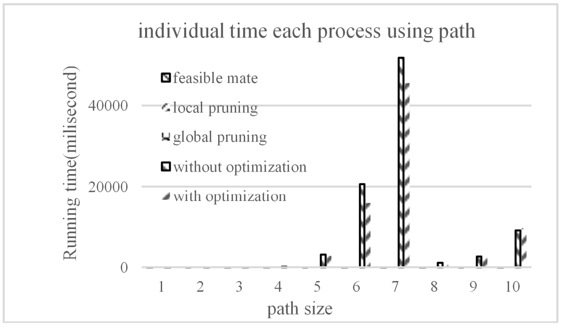


Figure 17: Individual Time each Process using Path

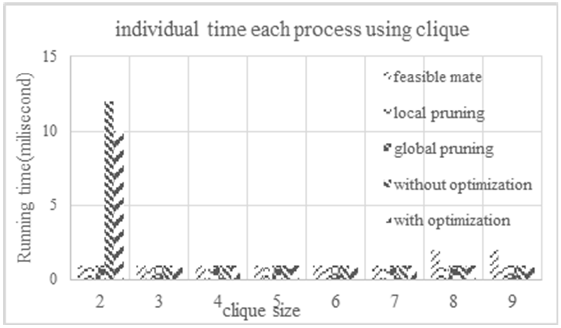


Figure 18: Individual Time each Process using Clique

Figure 17 show running time on searching process for path queries need longer time because searching process combines all the nodes in search space. Search space on path queries has more larger than clique queries because path queries not complex as clique queries so the pattern found in the graph have more matching. Figure 18 show that clique size 8 and 9 have longer running time on feasible mate process because the search space size after pruning process is less than the size of graph that to be checked before pruning.

4 CONCLUTIONS

GraphQL algorithm can be used to find graph pattern in a large graph database because it can read the query language. This algorithm speed up the running time process by reducing search space with local pruning and global pruning. The combination of local pruning and global pruning global give lower reduction ratio with high pruning power and give running time faster than pruning individually. The Optimization of the search space also makes the searching process of graph pattern becomes faster.

In our experiment, we add algorithm to avoid duplicate result and algorithm to matching the result with the graph pattern.Because of that, our experiment has lower running time than other research.

5 FUTURE WORK

Suggestions that can be used for further research is use matrix representation because of the complexity of matrix for the graph representation is lower than adjacency list. Use only dataset with one type of label to test the effectiveness of the reduction ratio of local pruning that reducing with checking label information only. Indexing techniques can also be added to accelerate the process of searching the feasible mate and the overall process of searching.

REFERENCES:

[1] Conte, D., Foggia, P., Sansone, C. & Vento, M., 2004. Thirty years of graph matching in pattern recognition.
[2] Cordela, L. P., Foggia, P., Sansone, C. & Vento, M., 2004. A (sub) Graph Isomorphism Algoritm for Matching Large Graphs. *IEEE PAMI,* pp. 1-5.
[3] Cordella, L. P., Foggia, P., Sanspne, C. & vento, M., 1999. Performance Evaluation of the VF graph Matching Algorithm.
[4] Fort, S., 1996. The graph Isomorphism Problem.
[5] He, Huahai. & Singh, A. K., 2008. Graph-at-a-time : Query Language and Access Methods for graph Databases. *SIGMOD.*
[6] John, S., 2011. *Social network Analysis Theory and Application.* s.l.:s.n.
[7] Lee, J., Kasperovics, R., Han, W.-S. &Lee, J.-H., 2012. An In-depth Comparison of Subgraph Isomorphism Algorithm in Graph Databases.
[8] Ruohonen, k., 2012. Graph theory.
[9] Saltz, M. W., 2013. A fast Algorithm for Subgraph Pattern Matching on Large Labeled Graph.
[10] Santo, M. d., Foggia, P., sansone, C. & vento, M., 2002. A large Database of Graphs and Its Use for Benchmarking Graph Isomorphism Algorithms.
[11] Scot, j., 2011. Graph Theory.
[12] Sutrisna, I Gusti Bagus Ady, W, Kemas Rahmat Saleh, Alfian Akbar Gozali., 2015. Implementation of GRAC

Algorithm (Graph Algorithm Clustering) in Graph Database Compression.

[13] Tian, Y. & Patel, J. M., 2008. Tale : A tool for Approximate Large Graph Matching.

[14] Ullmann, J. R., 1976. an Algorithm for Subgraf Isomorphism.

[15] Zhang, S., Li, S. & Yang, J., 2009. GADDI : Distance Index based Subgraph Matching in Biolgical Network. *ACM.*

[16] Zhao, P. & Han, J., 2010. On Graph Query Optimization in Large Network. *VLDB Endowment.*