Implementing and Analysing an Approximate Data Extraction Algorithm from Graph Databases

Computer Science MS Project

Prof. Carlos Riverio.

Gautame Vakkalagadda
ROCHESTER INSTITUTE OF TECHNOLOGY
Computer Science MS Project

CONTENTS

1. INTRODUCTION ................................................................. 3

2. MOTIVATION ................................................................... 3

3. ALGORITHM DETAILS ....................................................... 3

4. INDEX STRUCTURE .......................................................... 5

5. MEASURMENTS USED ..................................................... 9

6. CLASSES AND METHODS ADDED .....................................10

7. DESIGN ...........................................................................11

8. MAP DATA RESULTS ........................................................12

9. YEAST DATA RESULTS ..................................................... 18

10. SUGGESTIONS FOR IMPROVEMENTS ...............................22

11. PERFORMANCE SUMMARY ............................................... 23

12. CONCLUSION .................................................................. 23

13. SYSTEM SPECIFICATIONS ...............................................23

13. REFERENCES ................................................................... 23
LIST OF FIGURES

1. Fig 1 – Structure of Index
2. Fig 2 – example data graph
3. Fig 3 – Index on the data graph
4. Fig 4 – Design flow
5. Fig 5 – Map data
6. Fig 6 – Sample Map data graph
7. Fig 7 – Input Query graph
8. Fig 8 – Output of the query graph
9. Fig 9 – Input query graph
10. Fig 10 – Output of the query graph
11. Fig 11 – Input query graph
12. Fig 12 – Output of the query graph
13. Fig 13 – Input query graph
14. Fig 14 – Input query graph
15. Fig 15 – Input query graph
16. Fig 16 – Input query graph
17. Fig 17 – Input query graph
18. Fig 18 – Input query graph

LIST OF TABLES:

1. Table 1 to Table 5 – secondary indices
2. Table 6 – Map details
3. Table 7 to Table 11 – Results table
4. Table 12 – Performance table
1. INTRODUCTION:

Graph databases are one of the leading drivers in these currently emerging heterogeneous database management systems. Now-a-days many modern applications implement graph databases each of which has thousands of nodes and edges. There is a recent increase in interest towards graph databases because of growing interest in social media analysis. It gives lot of flexibility and can serve real time scenarios efficiently. To fully analyze the information stored in the graphs effective matching tools are necessary.

This Project deals with the implementation of a prototype of an approximate sub graph matching algorithm on Graph Databases. The authors of the algorithm proposed an approach which uses high pruning hybrid Index structure the size of which increases linearly with the database. The technique used helps in identifying the important nodes matching to the query node in the database and then progressively extending the matches until all the approximate matches is found. Finally the implementation is tested with different database sets and sizes to provide analysis of the prototype.

2. MOTIVATION:

In this decade of new millennium several web giants like Google, Facebook and twitter started using graph databases as their business involves around connectivity between different data points. Google has figured out how to process the web documents and capture web graphs which had a really good impacted their business.

A Graph can store a lot of information and the properties of the graph like degree, labeled edges can give some useful analysis on the data by just querying a data node in the graph which might help to improve any business further.

Graph databases can be very useful in the scenarios described below:

- If the data that is being used is connected.
- If there is no fixed schema or structure to the data.
- If there will be lot of changes in the application in future.
- It is easier to provide user recommendations for the applications as analyzing paths and patterns is easier using graph data.

3. ALGORITHM DETAILS

High Level Understanding of the Algorithm:

- Evaluate important Nodes of the query graph.
- Probe the index Structure to find all the matching nodes in the database.
- Grow the Matches in the Database from the data nodes obtained in previous step.
- Output all the resulted approximate matches in the graph database.
Computer Science MS Project

Overview:

Algorithm 1 - Bitmap Probe for Approximate Sub graph Matching [1].

Description:

This algorithm helps in evaluating the nodes that satisfy the query node.

Input:

- \( N_q \) – It is the query node.
- Bitmap is the index to be probed, that has \( n \) nodes in the bitmap index and the size of neighbor array is \( S_{\text{bit}} \).
- \( p \) - it is the percentage of neighbors of a query node that can be missing to a match to a node in the data graph to the query node [1].

Output:

- \( \text{Result}_{\text{le}} \) is the bit vector indicating which nodes satisfy the query

Algorithm 2 - Grow Match:

Description:

This algorithm helps in extending the matches and finding all the resultant matching Nodes to the Query Graph [1].

Input:

- \( G_q \) is the query graph.
- \( G_{\text{db.}} \) is the database graph.
- \( M_{\text{imp}} \) contains the matches for the important nodes in \( G_q \).

Output:

- \( M \) contains the node matches for the resulting graph match.

Algorithm 3 - Examine Nodes Near By:

This algorithm is used to evaluate the priority queue \( M \) in Algorithm2. It basically examines nodes which are one and two hops away from the query node and also the resultant matched database nodes and adds them to \( M \) which is declared in algorithm 2.
Algorithm 4 - Match Nodes:
This Algorithm is called in algorithm 3. It helps in evaluating the nodes which are one and two hops away and return the matches to Algorithm 3. \( (N_q, N_{dB}) \) [1].

Note: Output after implementing the entire main algorithm is stored in the Data Structure M.

Algorithm 5: Evaluating Triangles
This algorithm is used to evaluate neighborhood connections of every node in the graph database. It lists all the triangles formed by every node, which means number of triangles formed by every node is nothing but the number of neighborhood connections it has [2].

3. INDEX STRUCTURE:
The Index Structure has two levels:

High Level Index Structure is a B+ Tree Index structure built on node label, degree and neighborhood connection. The Leaf nodes of the B+ tree points to the secondary level index [1].

Secondary Level tabular index has actual data and has two components:

- List of Database Node ids of the database nodes that are represented by the B+ tree leaf entries [1].
- Bitmap index for the neighbor arrays of these database nodes.
Conditions to probe the index and find the matches in the Database [1]:

- \((N_{db}).\text{label} = (N_{q}).\text{label}\)
- \((N_{db}).\text{degree} \geq (N_{q}).\text{degree} - N_{\text{bmiss}}\)
- \((N_{db}).N_{\text{connection}} \geq (N_{q}).N_{\text{connection}} - N_{\text{bcmiss}}\)

\(N_{\text{connection}}\): number of edges between the neighbors.

\(N_{\text{bcmiss}}\): neighborhood connections that can be missing.

\(N_{\text{bmiss}}\): number of neighbors for the query node that can be missing.
Example Data Graph:

![Graph Diagram]

**Graph Details:**

<table>
<thead>
<tr>
<th>id</th>
<th>Label</th>
<th>Degree</th>
<th>Neighborhood connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>R</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>H</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>J</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>S</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Understanding the Index Structure:**

**Assumptions:**

Number of keys in the inner nodes: 2

Number of Keys in the leaf nodes: 2
Visualization of the B+ Tree on the Graph (Fig 1):

SECONDARY INDEX STRUCTURES (Key-Value pair):

Key: Database id Value: 
BitMap array Sec_Index5:

<table>
<thead>
<tr>
<th>Id</th>
<th>Bit-Map Neighbor Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>M, H</td>
</tr>
<tr>
<td>6</td>
<td>H, R</td>
</tr>
</tbody>
</table>

Table: 1

Sec_Index1:

<table>
<thead>
<tr>
<th>Id</th>
<th>Bit-Map Neighbor Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>S,S</td>
</tr>
</tbody>
</table>

Table: 2
4. MEASUREMENTS USED

Quality of Node Calculations:

\( \text{Nbmiss} \) - actual number of missing neighbor's in the node match

\( \text{Nbcmiss} \) - actual number of missing neighbor connections.

\( \text{Fnb} \) - fraction of missing neighbors in a query node.

\( \text{Fnbc} \) – fraction of missing neighborhood connections in a query node.

\[
\text{Fnb} = \frac{\text{Nbmiss}}{\text{Nq.\ degree}}
\]

\[
\text{Fnbc} = \frac{\text{Nbcmiss}}{\text{Nq.\ nbconnection}}
\]
Computer Science MS Project

Quality (w):

if (Nbmiss == 0) w
    = 2-Fnbc
else
    w = 2 – (Fnb + Fnbc/Nbmiss)

Match Criteria:
Two nodes are considered a match if [1]:

- The labels of the two nodes must be same.
- The degree of the query node should be no more than that of the database node.
- The neighbors of the query node should have corresponding matching nodes in the neighborhood the database node.

Selection of Important Nodes in a query graph:
Fraction of important nodes are selected from the query graph which depends on the graph properties and the end application. The nodes obtained are sorted by their importance, degree of centrality is used as to measure importance of a node in this algorithm [1].

Centrality measure: The importance of the node depends on the degree of the node in the graph. (Different measures can be used depending on the application)

5. METHODS AND CLASSES ADDED:
   1. BplusTree.java:
   2. Algorithm1.java
      • BitProbeAlgorithm()
   3. Algorithm2.java
      • GrowMatch()
      • ExamineNearByNodes()
      • MatchNodes()
      • QualityCalculation()
      • MatchCalculation()
      • BestMap()
   4. DataNode.java.
   5. Graph.java
   6. DataParser.java
   7. Main.java
Main: It is the main controller class to perform the algorithm and communicate between the classes.

Data Parser: It cleans and parses the data from the files and generates data nodes for the graph.

Graph Generator: This class takes care of forming edges, maintaining adjacency list and also evaluates neighborhood connection.

Index: It builds the primary and secondary level indices over the database.

Probe Algorithm: It decides the important nodes and probes the index for each important query node to obtain a list of database nodes matches.

Grow Algorithm: It utilizes all the node matches produced and examines the remaining nodes of the database to produce a final match.
7. MAP DATA RESULTS:

Data Graph:

![Map of the United States](image)

**Fig: 5**

**Data Description:**

The collected data consists of information of all the states in USA.

Node: Every node represents a specific state.

Edge: If two nodes have an edge then the states are connected by a border.
Sample Database Graph with 1 hope and 2 hopes away from AL:

<table>
<thead>
<tr>
<th>Node</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>1</td>
</tr>
<tr>
<td>MS</td>
<td>2</td>
</tr>
<tr>
<td>TN</td>
<td>3</td>
</tr>
<tr>
<td>GA</td>
<td>4</td>
</tr>
<tr>
<td>FL</td>
<td>5</td>
</tr>
<tr>
<td>SC</td>
<td>6</td>
</tr>
<tr>
<td>NC</td>
<td>7</td>
</tr>
</tbody>
</table>
Computer Science MS Project

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>8</td>
</tr>
<tr>
<td>KY</td>
<td>9</td>
</tr>
<tr>
<td>MO</td>
<td>10</td>
</tr>
<tr>
<td>AR</td>
<td>11</td>
</tr>
<tr>
<td>LA</td>
<td>12</td>
</tr>
</tbody>
</table>

Table: 6

Input Query Graph:

```
MS   AL   GA
```

Fig: 7

Output:

Id's nodes satisfying the query: 1, 2, 4

Nodes of the query Graph[MS,1,0, AL,2,0, GA,1,0]
neighbors 1 hope of GA,1,0 are [AL,2,0] neighbors 2
hope of GA,1,0 are [MS,1,0]
networks 1 hope of GA,5,4 are [FL,2,1, NC,4,3, SC,2,1, TN,8,8]
networks 2 hope of GA,5,4 are [VA,6,5, AR,6,6, KY,7,7, MO,8,8]
neighbors 1 hope of AL,2,0 are [MS,1,0, GA,1,0] neighbors 2
hope of AL,2,0 are []
networks 1 hope of AL,4,3 are [FL,2,1, GA,5,4, TN,8,8]
networks 2 hope of AL,4,3 are [NC,4,3, SC,2,1, AR,6,6, LA,3,2, KY,7,7, MO,8,8, VA,6,5] neighbors 1 hope of MS,1,0 are [AL,2,0] neighbors 2 hope of MS,1,0 are [GA,1,0]
networks 1 hope of MS,4,3 are [AL,4,3, AR,6,6, LA,3,2, TN,8,8]
networks 2 hope of MS,4,3 are [FL,2,1, GA,5,4, MO,8,8, OK,6,6, TX,4,3, KY,7,7, NC,4,3, VA,6,5] Nodes Satisfying the Query

ID: Node
4 : GA,5,4
1 : AL,4,3 2
2 : MS,4,3
Highlighting output in the database:

Fig: 8

Input Query Graph:

Output:

Nodes of the query Graph[AL,3,0, MS,2,0, GA,2,0, FL,3,0] neighbors 1 hope ofGA,2,0 are [AL,3,0, FL,3,0] neighbors 2 hope ofGA,2,0 are [MS,2,0] neighbors 1 hope ofGA,5,4 are [FL,2,1, NC,4,3, SC,2,1, TN,8,8] neighbors 2 hope ofGA,5,4 are [MS,4,3, VA,6,5, AR,6,6, KY,7,7, MO,8,8] neighbors 1 hope ofAL,3,0 are [MS,2,0, GA,2,0, FL,3,0] neighbors 2 hope ofAL,3,0 are [] neighbors 1 hope of AL,4,3 are [FL,2,1, GA,5,4, MS,4,3, TN,8,8]
nodes neighbors 2 hope of AL,4,3 are [NC,4,3, SC,2,1, AR,6,6, LA,3,2, KY,7,7, MO,8,8, VA,6,5]

Nodes Satisfying the Query
ID: Node
4 : GA,5,4
1 : AL,4,3
Highlighting output in the database:

Input Query:

Output:

Nodes of the query Graph[AL,3,0, GA,2,0, FL,1,0, TN,2,0] neighbors 1 hope of TN,2,0 are [AL,3,0, GA,2,0] neighbors 2 hope of TN,2,0 are [FL,1,0]
neighbors 1 hope of TN,8,8 are [AR,6,6, KY,7,7, MO,8,8, MS,4,3, NC,4,3, VA,6,5] neighbors 2 hope of TN,8,8 are [LA,3,2, OK,6,6, TX,4,3, SC,2,1, IL,5,4, IN,4,3, OH,5,4, WV,5,5, IA,6,6, KS,4,4, NE,6,6, DC,2,1, MD,5,4] neighbors 1 hope of GA,2,0 are [AL,3,0, TN,2,0] neighbors 2 hope of GA,2,0 are [FL,1,0] neighbors 1 hope of GA,5,4 are [NC,4,3, SC,2,1, TN,8,8]
neighbors 2 hope of GA,5,4 are [MS,4,3, VA,6,5, AR,6,6, KY,7,7, MO,8,8]
neighbors 1 hope of AL,3,0 are [GA,2,0, FL,1,0, TN,2,0] neighbors 2
hope of AL,3,0 are []
neighbors 1 hope of AL,4,3 are [GA,5,4, MS,4,3, TN,8,8]
neighbors 2 hope of AL,4,3 are [NC,4,3, SC,2,1, AR,6,6, LA,3,2, KY,7,7, MO,8,8,
VA,6,5]
neighbors 1 hope of FL,1,0 are [AL,3,0]
neighbors 2 hope of FL,1,0 are [GA,2,0, TN,2,0] neighbors
1 hope of FL,2,1 are [AL,4,3, GA,5,4]
neighbors 2 hope of FL,2,1 are [MS,4,3, TN,8,8, NC,4,3, SC,2,1]

Nodes Satisfying the Query
ID: Node
3 : TN,8,8
4 : GA,5,4
1 : AL,4,3
5 : FL,2,1

Highlighting output in the database:

Input Query:

![Graph](image)

Output:

No Nodes satisfy this query.
8. YEAST DATA RESULTS:

Approach to generate graph for yeast data:

Vertex Representation:

Vertex: v 0 1 2 3 4
0 – id
1,2,3,4 – Labels

Each vertex is divided into different nodes:
- V 0 1
- V 0 2
- V 0 3
- V 0 4

Edge Representation:

Vertices: v 0 1 2, v 1 5 6
Edge: e 0 1

e – edge from vertex with id 0 to vertex with id 1

Each edge is divided to several edges as shown below:
- v 0 1 to v 1 5
- v 0 1 to v 1 6
- v 0 2 to v 1 5
- v 0 2 to v 1 6

Following this approach 3000 vertices in the data is divided to 23429 data nodes.

CONDITIONS:

Condition 1: Considering only immediate neighbors of query node and database node.

Condition 2: condition 1 and considering only neighbors of query node which are two hopes away and immediate neighbors of database node.

Condition 3: condition 2 & considering immediate neighbors of query node and neighbors two hopes away for database node
Input Query Graph:

![Diagram](image)

Fig: 14

Total Size of the data Graph: 23429

Time Taken for building B+ Tree index: 55004ms

Time for probing: 187ms

<table>
<thead>
<tr>
<th>S.No</th>
<th>Conditions</th>
<th>Time taken for Grow Algorithm(ms)</th>
<th>Final number of node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition1</td>
<td>237425</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>Condition2</td>
<td>298982</td>
<td>76</td>
</tr>
<tr>
<td>3</td>
<td>Condition3</td>
<td>303655</td>
<td>76</td>
</tr>
</tbody>
</table>

Table: 7

Input Query:

![Diagram](image)
Fig: 15

Total Size of the data Graph: 23429

Time Taken for building B+ Tree index: 55004ms

Time for probing: 876ms

<table>
<thead>
<tr>
<th>No</th>
<th>Conditions</th>
<th>Time taken for Grow Algorithm(ms)</th>
<th>Final number of node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition1</td>
<td>1787655</td>
<td>205</td>
</tr>
<tr>
<td>2</td>
<td>Condition2</td>
<td>2667886</td>
<td>212</td>
</tr>
<tr>
<td>3</td>
<td>Condition3</td>
<td>3653214</td>
<td>241</td>
</tr>
</tbody>
</table>

Table: 8

Input Query:

Fig: 16

Total Size of the data Graph: 23429

Time Taken for building B+ Tree index: 55004ms

Time for probing: 557ms

<table>
<thead>
<tr>
<th>S.No</th>
<th>Conditions</th>
<th>Time taken for Grow Algorithm(ms)</th>
<th>Final number of node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition1</td>
<td>2787655</td>
<td>198</td>
</tr>
<tr>
<td>2</td>
<td>Condition2</td>
<td>3667886</td>
<td>220</td>
</tr>
</tbody>
</table>
3  Condition3  4368095  255

Table: 9

Input Query:

![Graph Image]

Fig: 17

Total Size of the data Graph: 23429

Time Taken for building B+ Tree index: 55004ms

Time for probing: 1076ms

<table>
<thead>
<tr>
<th>S.No</th>
<th>Conditions</th>
<th>Time taken for Grow Algorithm(ms)</th>
<th>Final number of node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition1</td>
<td>3518253</td>
<td>264</td>
</tr>
<tr>
<td>2</td>
<td>Condition2</td>
<td>5362679</td>
<td>330</td>
</tr>
<tr>
<td>3</td>
<td>Condition3</td>
<td>8217544</td>
<td>255</td>
</tr>
</tbody>
</table>

Table: 10
Fig: 18

Total Size of the data Graph: 23429

Time Taken for building B+ Tree index: 55004ms

Time for probing: 6617ms

<table>
<thead>
<tr>
<th>S.No</th>
<th>Conditions</th>
<th>Time taken for Grow Algorithm(ms)</th>
<th>Final number of node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Condition1</td>
<td>8011780</td>
<td>498</td>
</tr>
<tr>
<td>2</td>
<td>Condition2</td>
<td>9081020</td>
<td>528</td>
</tr>
<tr>
<td>3</td>
<td>Condition3</td>
<td>11001980</td>
<td>568</td>
</tr>
</tbody>
</table>

Total: 11

9. **SUGGESTIONS TO IMPROVE PERFORMANCE**

- The Degree of centrality measure that is used to select important nodes from the query graph might also effect the performance of the algorithm so a different measure can be used.
- The Grow Algorithm can have minor changes depending on the application for better performance.
- The query results can be saved so that if a new query comes in, which has a subgraph whose query result is already computed the existing results can be used and reduce re-computation.
10. PERFORMANCE MEASUREMENT

<table>
<thead>
<tr>
<th></th>
<th>Data Nodes</th>
<th>Data Edges</th>
<th>Query Size</th>
<th>Probing Time(ms)</th>
<th>Growth Time(ms)</th>
<th>Final Node matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yeast Data</td>
<td>23429</td>
<td>994398</td>
<td>4</td>
<td>876</td>
<td>3653214</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1076</td>
<td>8217544</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>6617</td>
<td>11001980</td>
<td>568</td>
</tr>
<tr>
<td>Map Data</td>
<td>50</td>
<td>107</td>
<td>4</td>
<td>0</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>1</td>
<td>43</td>
<td>5</td>
</tr>
</tbody>
</table>

Table: 12

11. SYSTEM SPECIFICATIONS:

Operating System: Windows 8

Coding Language: Java

Tools: Eclipse

12. CONCLUSION

- The algorithm works well with sparse graphs compare to dense graphs.
- The algorithm checks the neighbors which are one and two hopes away from the matching node which effects the performance of the algorithm as it might end up examining the whole graph.
- It is a heuristic algorithm it does not guarantee that it will find all the best matches [1].
- A node which is a match should be have tolerance towards missing unimportant nodes than missing important nodes.
- The pruning power of the algorithm is good because of the index structure and the probe algorithm [1].
- The future work of the algorithm is to work on improving the efficiency of the algorithm, especially the growth algorithm by making few modifications specific to an application.

REFERENCES

1. A Tool for Approximate Subgraph Matching, Yuanyuan Tian, Jignesh M. Patel, EECS Department, University of Michigan, Ann Arbor, Michigan, USA.