International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 12 (2015) pp. 31613-31628
© Research India Publications
http://www.ripublication.com

Plano Framework For Graph Indexing – A Statistical Analysis

[1]A. Pankaj Moses Monickaraj

Assistant Professor Dept. of Computer Science&Engg Christ University

[2.] J..B.Raj Kumar

Doctoral Scholar Dept. of Computer Science&Engg Bharathiar University

[3.] K.Vivekanandan

Professor BSMED Bharathiar University

[4]E.D. Boopalan

Doctoral Scholar Dept. of Statistics Bharathiar University

Abstract

Graph Mining is becoming one of the most dominant fields of research. There are plenty methods to index, re-index and to search the features throughout the index but still from the literature study there is no specific frame work which can sum up all three so that indexing and updating the index with new feature can be done in consistent intervals according to the arrival of new features. PLANO is the frame work which has the latest algorithms to look into the data and index. In this paper, Time and Memory efficiency of the proposed algorithms in the PLANO framework is tested statistically and compared with the existing algorithms memory and time usage.

Keywords: Graph, Graph Mining, Graph Indexing

1. Introduction

Irrespective of the geographical locations of the users, the birth of World Wide Web in 80's has been rooted in dominant as one of the data generation medium. In the recent days, Internet has become as the once of the cheapest medium to gather required information and to publish information throughout the globe. Web mining can be defined as the discovery and analysis of useful information from the WWW data. It can be categorized into Web Content, Web Structure and Web Usage Mining. Several forms of representing data are available in structured and semi-structured form which can be modeled as graph. Using graphs as a strong method to model complex data, various disciplines have been recognized by various researchers in domains such as chemical, computer vision, image and object retrieval, and machine learning.

If Internet is modeled into a graph, then graph would be in the never ending structure of infinity because it is completely the composition of relationships and connections. Of the various sub domains of graph mining like Graph Classification, Graph Clustering etc, this paper primarily focus on the Graph Indexing.

This paper is organized as follows: we introduce Lieterature Survey in Section II. Then we present the set of related work in section III. In section IV, we first introduce the complete overview of the architecture. Then in section V, we have given the experimental proof for phase III. Finally, we conclude this paper in Section VI.

2. Literature Survey

A. Classic Algorithms of Phase: I

There are bounties of indexing methods for graph databases. There are various types of data and each type has to be more selective in processing the particular kind of data. Algorithms like Day light [4], AnMol [5] are for structured data, where as Data Guides [6], T-index [7], Index Fabric[8], APEX [9] are for semi structured and XML data. These are the few varieties or types of algorithms concentrating on particular types of data. There are hands full of indexing techniques for various graph models. GraphDB [10] and SUBDUE [11] concentrates on query processing from a large graph database while it automatically provides the relevant sub graphs. Of the various methods of indexing one of the dominant is GraphGrep[12] which works on the basis of paths of the graph. Consider if the paths in a graph are huge, the performance of the index will definitely be degraded. To overcome this, a graph based approach named gIndex [13] is reported.

B. Classic Algorithms of Phase: 2

A(k) – index [14] uses the k-bisimilarity to make use of similar patterns in a semi strucured graph database. Every path in a tree is treated as string and it is stored in Patricia trie by Index Fabric[15]. Washio and Matoda [16] introduced graph based data mining to mine the graph databases. The frequent sub graphs have to be analyzed for which Inokuchi et al. [18] and Vanetik et.al [17] have used Apriori-based algorithm. For the generation of sub graphs other than algorithms Yan and Han [19] and Borgelt and Berthold [20] have applied pattern growth approach.

C. Classic Algorithms of Phase: 3

In Frequent subgraph mining, there are various algorithms like AGM [22], FSG [16], and gSpan [23], go after by Path-Join, MoFa, FFSM, SPIN, Gaston but here in this phase a new algorithm is proposed, tested and compared with the classical LEAP algortihm.

3. Related Works [Comparing the Classic and the Latest Contributions]

The algorithms VFG index [Valorous Frequent Graph] [2] and BIGFMA [BON Iterative Graph Feature Mining Algorithm] [3] used in phase: I & II were proposed by the very same authors.

A. Phase I

The most recent and the improved algorithm for graph indexing is FG-index, which is enhanced to VFG-index [2] (Valorous Frequent Graph index) algorithm.

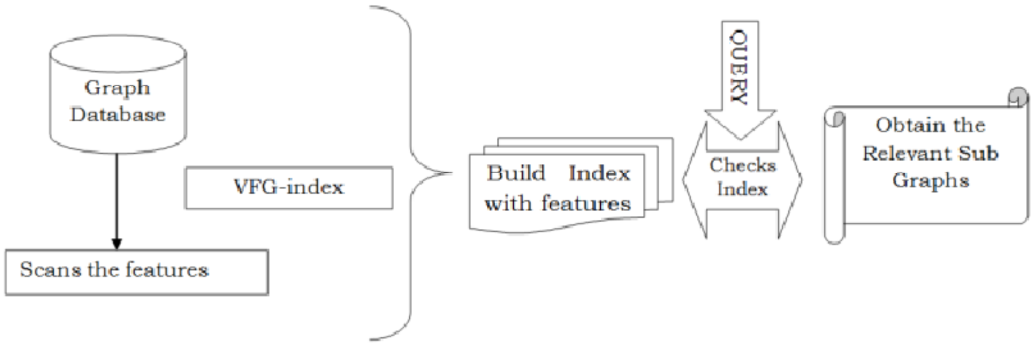

Fig. 1 Working of Phase: I

Fig. 1 gives the complete overview of how the index is constructed by VFG index algorithm initially. Once the query is obtained, it is checked with the index and then irredundant sub graph are retrieved with reference to the previously constructed index.

The phase I of this work is happening by two sub phases.

Index Construction

As a preprocessing step, the algorithm scans throughout the graph database and extracts the selected features in the graph database say G according to the minimum support. These extracted set of features are initially sorted and sequentially arranged with a weighted preferences for indexing through Dijkstra algorithm for shortest path among the extracted features so that graph index are done.

--**Index Construction**
--**Input:**

D - Graph Database
F - The set of Frequent Graphs
δ - The frequency tolerance factor or minimum support.

T - Candidate set
IDA – id –Array
GA – Graph Array
--**Output:** The Edge-index – VFG Index.

// Scan the dataset and extract the features with minimum support so that the shortest path can be found by Dijkstra algorithm//

```
1. Sort F s.t. $\forall g1, g2 \in F$, g1 is ordered before g2 if $g1 \prec g2$;
2. Let T = F and Ti be the set of FGs that consist of i edges;
3.      for each i = 1, 2, . . . do
4.           for each $g \in Ti$ do
5.                for each $g' \in Ti+1$ do
6.                     if $(g \subset g')$
7.                if $(freq(g') \geq (1 - \delta) \cdot freq(g))$
8.                          $T \leftarrow T - \{g\}$;
9.                     break; /* go to Line 10 */
10.     if $(g \in T)$
11.     Store g in the first free entry in the GA;
12.          for each distinct edge e in g do
13.               if (e /$\in$ EA)
14.               Add e to the EA;
15.     Add the ID of g to IDA(count (e, g), size(g), e);
16.     for each $g \in (F - T)$ do//Dijsters algorithm
17.     Find g's closest FG, supergraph, g;
18.     Add g to the nested Index of g';
19.          for each infrequent distinct edge e in D do
20.          Add e and $D_e$ to Edge-index;
```

The entire algorithm can be interpreted by dividing the algorithm into three parts by the lines of codes:

Lines 1-9 =>computation of T (ie) Candidate set generation	
Lines 10-18 =>	working of VFG-index
Lines 19-20=>	creation of Edge-index

Example:
Let G- Graph database, F- set of all feature from G. say for graph feature f€F, G (f) be the set of graphs containing the feature f, then G(f) = {gi/f$\subseteq$ gi, gi€G).
This is an example taken from AIDS antiviral screening database.

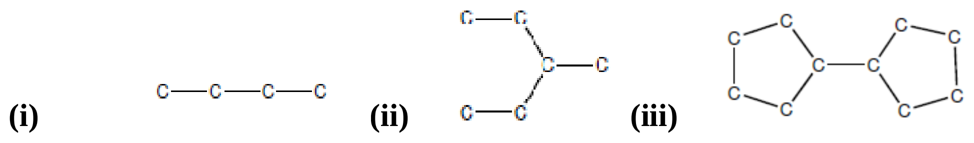

Fig. 2 A Sample Database

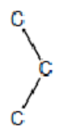

Fig. 3 A Sample Query

Assume a query say fig. 3, it checks out the bonding type c, c-c, c-c-c in the index. We get a subset of all three in Fig. 2 as sub graphs incase if taken with the bonding.

Query Processing

Once the query is obtained, it searches all the features relevant to the given query in the index to generate the candidate set which contains all the extracted features.

Cq – set of candidate set

$C_q = q(f) \cap F(f) \ \forall f \subseteq q$ and $f € F$ --------------------- (2)

Query Processing

Input: The index and a query q.

E- set of all edges

IDA – id Array

GA – Graph Array

Output: Dq.

```
1. Let E be the set of distinct edges in q;
2.      for each i = size(q), size(q) + 1, . . . do
3.           for each e ∈ E do
4.           C(e) ← $U_j \geq count(e,q)$ IDA(j, i, e)_;
5. Sort C(e) in ascending order;
6. Intersect C(e), ∀e ∈ E, until an ID is obtained;
7.      if (g in GA[ID] is a supergraph of q)
8.           if (g = q)
9.            Return Dg;
10.          else
11. Return FG-Query with g's nested IGA and q as input;
12.     else
```

13. Go to Line 6 and continue the intersection;

Line (2-11) => Graphs that have the same size as q until a super graph of q is found

----

If the intersection of the features does not obtain any the subgraph or no super graph of q is found for the current i, FG-Query increments i by 1 (Line 2) and continues a new round of iteration to search a super graph of q.

In [2], the existing FG index is tested with the AIDS dataset and the improvement in memory and time complexity with a newly proposed algorithm VFG index (valorous Frequent Graph Index) is provided as outcome.

B. **Phase II**

The mine-at-once and iterative mining are very good combination of framework that alleviates the low-support frequent subgraph mining bottleneck. In usual, the Mine-at-once algorithms often first obtain all frequent subgraphs and then mine the indexing features out of them. To prevent missing any important feature, frequent features are mined with a very low minimum support, e.g., $0.01|D|$, although most of the selected features have much higher support. The time for frequent subgraph mining dominates the overall mining time. To overcome this bottleneck of feature mining, we first run the mine-at-once algorithm with a higher minimum support ($0.1|D|$), then run the iterative feature mining algorithm to identify the missing features that are important.

Iterative Graph Feature Mining is one of the dominant properties which are focused in this phase. Let us consider a graph database D and a graph index with a feature set P0 of size n − 1, find a new graph feature p, p ∈ P0 such that the expected verification cost would be Tverif (≈ Tresp), which will be proportional to ∑ ∀ unique q |C(q)| • Pr(q), is minimized with the new feature set {p, P0} indexed, where C(q) is the candidate set of the query q.

To distinguish the candidate set generated by using feature set P0 from that by {p, P0}, a new parameter P is added in C(q,P), where P is the graph-feature set that is used to obtain the candidates C(q). The objective function is used as the saving of the number of isomorphism tests (isomorphism-test saving) after bringing the new feature p into the feature set P0:

$$\text{gain}(p, P0) = \forall \sum\nolimits_{\text{unique}} q \ \ (|C(q,P0)| - |C(q, \{p, P0\})|)\ \Pr(q).$$

To minimize the expectation of the verification cost Tverf (≈ response time Tresp), the new feature should be selected as the one maximizing the isomorphism-test saving, p = argmax gain(p, P0).

From the frequentist point of view, |C(q,P0)| = |D| • PrD(maxSub(q,P0)), where PrD(maxSub(q)) is the probability of graphs in the dataset D containing all features in maxSub(q). Pr(q) is the probability of a query graph isomorphic to q. And PrD(maxSub(q)) = Pr(g ∈ D s.t. g ⊃ maxSub(q)). It is hard to estimate the distribution of Pr(q), PrD(maxSub(q,P0)) and PrD(maxSub(q, {p, P0})). Hence a a practical approach is used for calculating the objective function over the graph database D and a training query log Q:

$$\text{gain}(p, P0) = 1/|Q| \sum_{q \in Q} [|C(q, P0)| - |C(q, \{p, P0\})|]$$

Since the objective function of a feature p only relates to the query q = p and queries for which p is a maximal subgraph given that {p, P0} is indexed, the irrelevant queries are not considered while calculating the objective function. The queries that have a maximal subgraph p by the minimal super queries of p are taken into consideration.

MinSup Query is another key term playing its vital role in this phase. Consider a query set Q and a subgraph feature p ∈ P, a graph q ∈ Q is a minimal super query of the feature p if and only if the feature p is a maximal subgraph feature of q.

$$\text{minSup}(p,Q) = \{q \in Q | p \in \text{maxSub}(q,P)\}$$

Therefore,

$$\text{gain}(p, P0) = 1|Q| \sum_{q \in \text{minSup}(p,Q)} |C(q, P0) - C(q, \{p, P0\})| + 1/|Q| \sum_{q \in Q} I(p = q)\, |C(q, P0)|$$

where, I is the indicator function: I(p = q) = 1 iff p is isomorphic to q.

Various iterative algorithms like Branch and Bound algorithm, Query grouping, TK algorithm are discussed of which the latest approach is Iterative Graph Feature Mining Algorithm which extract features from the updated data and they are be indexed.

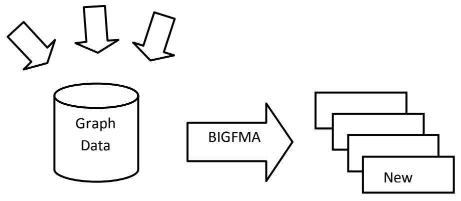

Fig. 4 Working of Phase II

In [1] it is clearly given in the future work that one can consider the development of algorithms that estimate the objective function instead of calculating it precisely, which can further enhance the speed of an iterative mining algorithm. This is concentrated on so that a new algorithm Bon Iterative Graph Feature Mining Algorithm (BIGFMA) [3] is developed. This is how the flow of the algorithm moves on:

BON Iterative Graph Feature Mining

Input: Current featuresP, Queries Q

Output: The optimum feature p∗

```
1: p∗ = null
2: for each feature p in all enumerated subgraphs do
3:     if gain(p∗, P0) is not precisely calculated then
4:         if Upp(gain(p, P)) < Low(p∗, P) then
5:             Continue
6:         else if Low(gain(p, P)) < Upp(p∗, P) then
7:             p∗ = p
8:         else
9:             Calculate gain(p∗, P)
10:        end if
11:    end if
12:    if gain(p∗, P0) is precisely calculated then
13:        if Upp(gain(p, P)) < gain(p∗, P) then
14:            Continue
15:        else if Low(gain(p, P)) > gain(p∗, P) then
16:            p∗ = p
17:        else
18:            Calculate gain(p, P), compare with gain(p∗, P)
19:        end if
20:    end if
21: end
```

Lines (2-9) =>calculation of the gain for the new features.
Lines (12-18) => The feature gain which exceeds the set value of objective function are taken out as the newly updated features from the updated data.

For each feature extracted a weight is given such that the only the features with particular weight are taken into account and added into the index. The two most dominant objectives measure would be G- test and Information gain. Here the objective function gain is calculated for the algorithm. These are the values through which the new features are extracted from the updated data.

In [3], the existing BON iterative algorithm is tested with the AIDS and E-Molecule datasets such that the improvement in memory and time complexity with a newly proposed algorithm BIGFMA (BON Iterative Graph Feature Mining Algorithm) is provided as outcome.

C. Phase: III

In phase I, data was indexed by VFG-index algorithm; on the arrival of updates data gets upgraded. In phase II, updated data are analysed by BIGFMA such that the new features are extracted from the updated data. In this phase, the right features have to be inserted into the right position in the index without performance degradation.

The flow of the sequence Phase III is as follows in fig. 5.

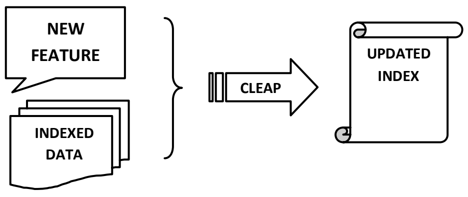

Fig. 5 Working of Phase III

The structure of CLEAP is explored by two mining concepts which is the extension LEAP search algorithm. Structural leap search, and frequency-descending mining, both are related to specific properties in pattern search space. Initially the existing branch-and-bound method only performs “vertical" pruning. In a graph if the upper bound and its descendants is less than the most significant pattern mined, the whole branch below g could be pruned. To improve the efficiency can the pruning be done horizontally? Yes, this is possible. This is called prune by structural proximity: Interestingly, many branches in the pattern search tree exhibit strong similarity not only in pattern composition, but also in their frequencies and their significance.

When a complex objective function is considered, pattern frequency is often put aside and never used in existing solutions.

If all of sub graphs patterns are ranked according to their frequency, significant graph patterns often have a high rank. This is where the phenomenon frequency association into picture. An iterative frequency-descending mining method is used find the right feature. This makes the minimum frequency threshold exponentially, making the proposed algorithm to improve the efficiency of capturing significant graph pattern candidates. The discovered candidates can then be taken as seed patterns to identify the most significant one. An examination of an increasingly important mining problem in graph data and the proposal of a general approach for significant graph pattern mining with objective functions are the core properties which the CLEAP algorithm to perform in an improved format.

These structural proximity and frequency association are used in the existing LEAP study. The very same is improved and demonstrated that the widely adopted branch-and-bound search is not fast enough, thus when the very same is tested with the proposed; it showed improvement in information gain and sketched a new picture on scalable graph pattern discovery.

```
CLEAP search
Input: Graph dataset D,
Output: Optimal graph pattern g¤.
1: S = f1-edge graph g;
2: g* = □; F(g*) = -∞;
3: while S ≠ do
4: choose g from S, S = S \ f{g};
```

```
5: if g was examined then
6: continue;
7: if F(g) > F(g*) then \\ Structural leap
8: g* = g;
9: if  F(g) < F(g*) then
10: continue;
11: S = S Ų { g' / g' = g . e}; \\frequency-descending compared
12: return g* = g**;
--------------------------------------------------------------------
```

The principle of LEAP is not to mine the most significant graph pattern in one shot. Instead, it first iteratively derives significant patterns with increasing objective score. In the second shot, it usually runs branch-and-bound search to discover the most significant one where unpromising branches will be cut quickly. With this new mining framework, the existing algorithm is able to capture the optimal pattern in a faster way but however when it comes to CLEAP it performance faster than the existing Branch and Bound and LEAP. This phase implementations is as follows.

4. Experimental Results For Phase: III

Dataset Information:

AIDS dataset

The dataset is NCI AIDS antiviral, donated Jun Feng , Laura Lurati , Haojun Ouyang , Tracy Robinson , Yuanyuan Wang , Shenglan Yuan , S. Stanley Young, J. Chem. Inf. Comput. Sci., 43 (5),1463 -1470, 2003. 10.1021/ci034032s S0095-2338(03)04032-0. Three classes (B)donated on July 29, 2004 completely consist of 29374 row numbers.

The Methodological overview of the PLANO framework is as follows.

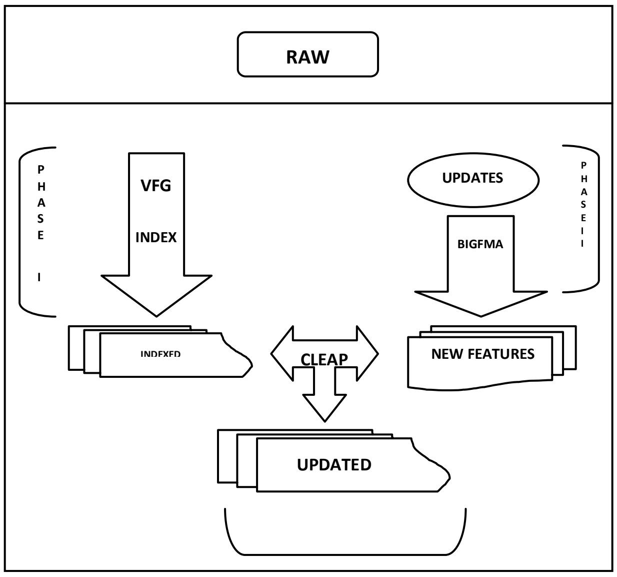

Fig:6 Iterative Indexing Plano Frame Work

4.7 Statistical Analysis

Table: 1 Centralized Memory Management table

		Existing	Proposed
FG/VFG-Index	Index Construction	2.5214	1.8001
	Query Processing	2.101	1.4201
IGFMA/BIGFMA	AIDS dataset	1.995	1.4990
	E-Molecule dataset	2.512	1.6600
LEAP/CLEAP		2.411	1.912

Table: 2 Centralized Time Management table

		Existing	Proposed
FG/VFG-Index	IndexConstruction	9.754	7.120
	Query Processing	0.031	0.021
IGFMA/BIGFMA	AIDS dataset	7.615	6.401
	E-Molecule dataset	7.403	6.019
LEAP/CLEAP		4.421	4.222

To prove the efficiency of PLANO framework, all the existing results both memory and time are taken into a single consideration and the all the proposed algorithms results are considered on the other hand.

So we frame the following tables for existing algorithm results and proposed algorithms results:

Existing Algorithm results

Table: 3 Centralized Memory Scores table

X [Memory]	Y [Time]
2.5214	9.754
2.101	0.031
1.995	7.615
2.512	7.403
2.411	4.421

Table: 4.11 Centralized Time Management table Proposed Algorithm results

X [Memory]	Y [Time]
1.8001	7.120
1.4201	0.021

1.4990	6.401
1.6600	6.019
1.912	4.222

Memory usage scores chart

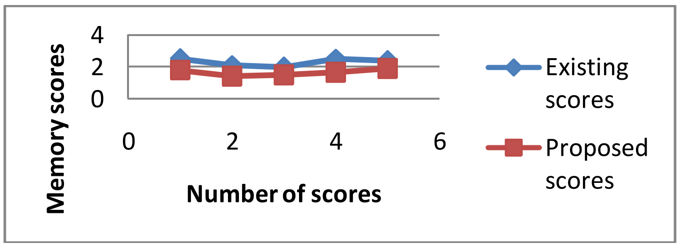

Figure: 7 comparing the memory scores of existing and proposed algorithm

The line chart compares the memory usage of both existing and proposed algorithms. The proposed algorithms consume less memory than all other existing algorithms.

Time Management scores chart

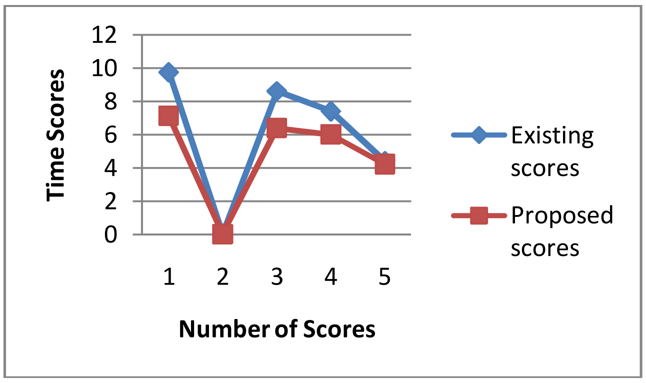

Figure: 8 comparing the time scores with existing and proposed algorithm

The line chart compares the time taken by both existing and proposed algorithms. The proposed algorithms consumes less time than all other existing algorithms.

Regression Definition
A regression is a statistical analysis assessing the association between two variables.

It is used to find the relationship between two variables.
Regression Formula
Regression Equation(y) = a + bx
Slope(b) = $(N\Sigma XY - (\Sigma X)(\Sigma Y)) / (N\Sigma X^2 - (\Sigma X)^2)$
Intercept(a) = $(\Sigma Y - b(\Sigma X)) / N$
where

x and y are the variables.
b = The slope of the regression line
a = The intercept point of the regression line and the y axis.
N = Number of values or elements
X = Memory Score
Y = Time Score
ΣXY = Sum of the product of Memory and Time Scores
ΣX = Sum of Memory Scores
ΣY = Sum of Time Scores
ΣX^2 = Sum of square Memory Scores

For Existing Algorithms
Slope b = 6.93608
Y intercept = -10.16424
Regression Equation => -10.164+6.936x ------- (1)
Substituting x=2.5214, 2.101, 1.995, 2.512, 2.411 in the above equation (1), the values of y would be 7.324, 4.4085, 3.673, 7.259, 6.559
For Proposed Algorithms
Slope b = 6.40926
Y intercept = -5.87149
Regression Equation: => -5.871+6.409x ------- (2)
Substituting x=1.8001, 1.4201, 1.4990, 1.6600, 1.912 in the above equation (2), the values of y would be 5.666, 3.230, 3.636, 4.768, 6.383

Plotting the equations (1) and (2) on the graph:

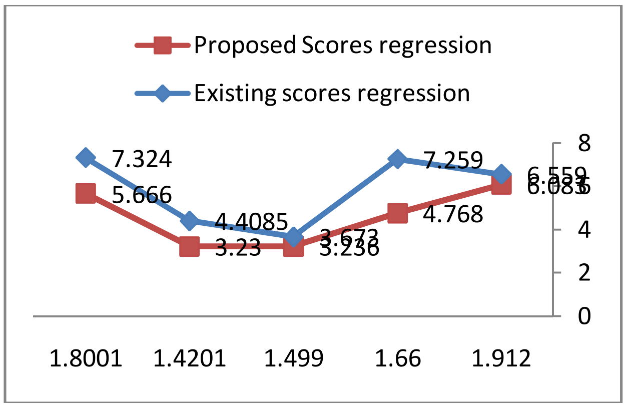

Figure:9 Regression curve for Existing and Proposed algorithms

From the figure 9, we can strongly conclude that the PLANO framework acquire less memory and time than the classical algorithms which were used for indexing and searching throughout the graph.

5. Conclusion

The complete research work comprises of the flow of data into an orderly arranged indexed sequence so that they can be retrieved for the concerned query. Initially, raw data is processed with the VFG index algorithm; their results are compared with the already existing most dominant algorithm FG index. This phase has two process, Index construction and Query processing. In Index construction, data are scanned and the features are extracted from the graph so that they are arranged as index. On arrival of a query, index is checked throughout and the sub graphs with specific features of the query are extracted and provided as output.

The memory and time complexity are tested for this working such that their results are compared with existing. Now there is an index for data, but when data gets updated this data is scanned through the iterative algorithm and the new proposed BIGFM algorithm took less memory and time complexity than the other classic iterative algorithms. This very same algorithm is tested with very two datasets so as to test the scalability and the working efficiency. During both the environment, the proposed algorithm performed better and came up with the updated features from the updated data. So as to insert this data into right place into the index, the search algorithm LEAP is tested. A new improved algorithm CLEAP is proposed and tested such the results stood tall better than the existing search algorithm.

From Figure:9 we can say that the proposed algorithms are giving improved results in both memory usage and time. From the literature survey, it is also said found that there is no existing framework for indexing. Extending on with the results, here we finally make this into a new frame work which can index and alters the index when updates come into data.

Alas, we say that PLANO is the only latest framework with improved latest algorithms which could effectively index data and provide improved results.

Acknowledgement

A Special thanks to Dr. V.Saravanan.

References

[1] Dayu Yuan, Prasenjit Mitra , Huiwen Yu, C. Lee Giles “Iterative Graph Feature Mining for Graph Indexing”, IEEE 28th International Conference on Data Engineering, 2012.

[2] A.Pankaj Moses Monickaraj, K.Vivekanadan, D.Ramyachitra, “VFG – INDEX: A Novel Graph Indexing Method” International Journal Of Computer

Science And Informatics, ISSN (PRINT): 2231 –5292, Volume-3, Issue-2, 2013.

[3] A.Pankaj Moses Monickaraj, K. Vivekanandan, D.Ramya Chitra, "Bon Iterative Graph Feature Mining for Graph Indexing", Journal of Research in Computing Science", ISSN: 1870-4069, volume 66, 2013.

[4] Daylight theory manual - daylight version 4.9. Daylight Chemica Information Systems, Inc. www.daylight.com, 2006.

[5] S. Srinivasa and S. Kumar. A platform based on the multi-dimensionaldata model for analysis of bio-molecular structures. In Proc. of VLDB, pages 975–986, 2003.

[6] R. Goldman and J. Widom. DataGuides: Enabling query formulation andptimization in semistructured databases. In Proc. of VLDB, pages 436–445, 1997.

[7] T. Milo and D. Suciu. Index structures for path expressions. In Proc. Of ICDT, pages 277–295, 1999.

[8] B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A fast index for semistructured data. In Proc. of VLDB, pages 341–350, 2001.

[9] C.-W. Chung, J.-K. Min, and K. Shim. Apex: an adaptive path index for xml data. In Proc. Of SIGMOD, pages 121–132, 2002.

[10] R. H. G¨uting. GraphDB: Modeling and querying graphs in databases. In Proc. of VLDB, pages 297–308, 1994.

[11] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In Proc. of KDD, pages 169–180, 1994.

[12] D. Shasha, J. T. L. Wang, and R. Giugno.Algorithmics and applications of tree and graph searching. In Proc. of PODS, pages 39–52, 2002.

[13] X. Yan, P. S. Yu, and J. Han. Graph indexing basedon discriminative frequent structure analysis. ACM Trans. Database Syst., 30(4):960–993, 2005.

[14] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for e±cient indexing of paths in graph structured data. In Proc. 2000 Int. Conf. Data Engineering (ICDE'00), San Jose, CA, Feb. 2002.

[15] B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A fast index for semistructured data. In Proc. 2001 Int. Conf. Very Large Data Bases (VLDB'01), pages 341{350, 2001.

[16] T. Washio and H. Motoda. State of the art of graph-based data mining. SIGKDD Explorations, 5:59{68, 2003.

[17] N. Vanetik, E. Gudes, and S. E. Shimony. Computingfrequent graph patterns from semistructured data. In Proc. 2002 Int. Conf. on Data Mining (ICDM'02), pages 458{465, Maebashi, Japan, Dec. 2002.

[18] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Proc. 2000 European Symp. Principle of Data Mining and Knowledge Discovery (PKDD'00), pages 13{23, Lyon, France, Sept. 1998.

[19] X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. I Proc. 2003 Int. Conf. Knowledge Discovery and Data Mining (KDD'03), pages 286{295, Washington, D.C., Aug. 2003.

[20] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In Proc. 2002 Int. Conf. on Data Mining (ICDM'02), pages 211{218, Maebashi, Japan, Dec. 2002.

[21] J. Cheng, Y. Ke, W. Ng, and A. Lu. "Fg-index: towards verification-free query processing on graph databases," In Proc. Of the ACM SIGMOD international conference on Management of data, pp. 857-872,2007

[22] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of 2000 European Symp. Principle of Data Mining and Knowledge Discovery*, pages 13{23, 2000)

[23] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of ICDM*, pages 313{320, 2001.