Data based elegant models for Super Graph query processing

Mohamed Saber, Mostafa Aref, Tarek F. Gharib
Ain-Shams University
Cairo, Egypt
mohamed.saber@cis.asu.edu.eg, Aref_99@yahoo.com, tgharib@eun.eg

ABSTRACT: *Graphs particularly the data models based ones, have proved to make significant impact on many applications. Efficient query processing over graph databases serves these applications. Having a graph query* ***q****, super-graph query processing finds all the graphs* ***g*** *in a database of graphs* ***D*** *where* ***g*** *is contained in* ***q*** *($g \subseteq q$). Because graph databases contain a lot of graphs and because sub-graph isomorphic tests are NP-complete, an indexingbased technique should be adopted. Now through this study we advocate a system for an optimal supergraph query making. The technique consists of an index called eIndex and a query processing algorithm. Given a query graph, the database is filtered to generate candidate graphs. The elegance of the proposed querying lies in consideration the full structure of the database graphs besides considering the frequent fragments. Through polynomial time algorithms, the proposed technique reduces the subgraph isomorphism tests required for query processing and hence the total processing time is reduced. We have shown evidences for building eIndex which consumes comparatively less time and space than other methods.*

Keywords: Query processing, Super graphs, Data models, eIndex, Query processing algorithm

Received: 13 October 2010, Revised 30 November 2010, Accepted 3 December 2010

1. Introduction

Data management is the process of efficient storage, processing and fast retrieval of data. Data management took great attention from computer scientists since the early days of computers and information systems. Due to the advances in computers and information technologies, new complex data types were presented which also require efficient data management. Examples of these data are images, videos, unstructured textual documents, biological data, graphical information systems data …etc. These new complex data types led to new requirements and problems. Most of these problems are related to processing queries efficiently. Efficient querying for data took great research attention which led to innovative creation of indexing methods. These indexing methods support efficient retrieval of specific data from huge stored data. Most of these research efforts are carried on the relational data model. The relational data management model is a general model for modeling any type of data. This one-for-all modeling may not be the optimal model for some data types such as graph data.

Recently, a lot of data modeled by graphs were emerged from various domains. These resulted in having many graph databases. Efficiently processing queries on these graph databases is critical to various domains. Graph data describes both attributes of data entities as well as the structure and the relationships among these entities. Efficient graph data management is currently a hot research topic due to its wide applicability in many applications domains besides being a performance bottleneck in those applications. These applications include bioinformatics applications, social networking applications, CAD designing, the semantic web, GIS, etc...

There are two important types of graph structural queries that are widely used in many applications. There are sub-graph queries and super-graph queries. Given a graph query **q** and a database of graphs **D** = $\{g_1, g_2,\ldots, g_n\}$, a sub-graph query retrieves the graphs from **D** where **q** is a sub-graph. The bottleneck part of processing sub-graph queries is the sub-graph isomorphism testing which is NPcomplete [24]. So, it is intractable to check all the database graphs with isomorphism tests. Sub-graph queries took a great research attention and many indexes and query processing algorithms were proposed ([12], [13], [14], [15], [16], [18]). The main idea of these researches is to filer the database graphs so that only a subset of graphs are verified using isomorphic tests against the query. This solution framework is known as filtering-verification framework.

The other important type of structural graph queries is the super-graph queries. Given a graph database **D** and a query graph **q**, the super-graph query finds all the graphs $\mathbf{g} \in \mathbf{D}$ where **g** is M contained in **q** i.e. the graphs which are sub-graphs of the query graph ($\mathbf{g} \subseteq \mathbf{q}$). Super-graph queries are important in many applications. For example, when we have a database of chemical descriptors (graph database) and we have a new compound (query). Knowing the descriptors contained in the new compound could help in understanding its properties. This could be done by querying the graph database to retrieve the graphs (descriptors) that are contained in the query graph (new compound).

Although super-graph queries are important in practice, it has not been considered extensively like the sub-graph queries. To our knowledge, only two indexes were proposed for processing super-graph queries. These are cIndex [19] and GPTree [22]. GPTree has the disadvantage of requiring the graph database to be stored in a specific compact form. This may not be practical in some applications where the graph database cannot be changed and also a compact copy cannot be created due to data integrity issues.

cIndex requires a query log (set of graphs used as super-graph queries). It uses this query log and the database graphs to extract features that are contained by many database graphs but are unlikely to be contained by query graphs. These features are called contrast features. The features are initially extracted using gIndex [12]. Then these features are filtered through a selection strategy to produce those contrast features. The contrast features are used by cIndex while doing query processing. cIndex uses matrices to model the relationship among the contrast features, database graphs and the query log graphs.
When processing a super-graph query **q**, it is tested among the contrast features. If it doesn't contain a specific feature, the database graphs associated with this feature are pruned. cIndex has a cascading effect which doesn't require checking all the contrast features against the query graph. If a features is not contained in the query graph, then this hint is used to prune other features which doesn't add any pruning power if checked. cIndex has three design forms. The one used in this paper is cIndex- Basic and we will refer to it simply as cIndex. It has the following disadvantages:

• It uses a lot of isomorphism tests in the index scanning phase which increase the query processing time. Testing whether a feature is included in the query or not is done through an isomorphic test.

• The filtering process is based only on the graph database features. Database graph features does not capture the whole structure of the database graphs. This leads to a relative increase in the number of the graphs in the candidate set.

• It requires the calculation of matrices which become huge in size when dealing with large graphs. This may affect scalability.
This paper proposes a technique for processing super-graph queries. The proposed technique consists of an index called eIndex and a query processing algorithm that make use of it. eIndex is built to reduce the number of sub-graph isomorphism tests to answer a super-graph query. It uses approximate algorithms which run in polynomial time. One of the approximate algorithms is based on subsequence matching and the other one is the pseudo sub-graph isomorphism proposed by Huahai He [14]. The main contribution of this paper is presenting a technique which has the following advantages:

1. It uses polynomial time algorithms when possible to filter the database.

2. It takes into consideration the whole structure of the database graphs which increases the pruning power of the filtering phase.

3. It doesn't require neither query logs nor the preparing/processing of huge data structures.

4. It doesn't require a special form of the underlying graph database. The rest of this paper is organized as follows. Section 2 presents preliminary concepts used along the paper. Section 3 presents a general framework for processing super-graph queries. Section 4 presents the design of the proposed technique. Section 5 presents the experimental results. The paper is concluded in section 6.

The rest of this paper is organized as follows. Section 2 presents preliminary concepts used along the paper. Section 3

presents a general framework for processing super-graph queries. Section 4 presents the design of the proposed technique. Section 5 presents the experimental results. The paper is concluded in section 6.

2. Preliminaries

This paper considers undirected, labeled and connected simple graphs. The work in this paper could be extended to support other types of graphs.

Definition 1**. A graph** is a mathematical structure used to model pair wise relations between objects from a certain collection. A graph G = (V, E) is a set of nodes (vertices) denoted by V and set of edges denoted by E$\subseteq$ V x V. A graph represents a "relationship structure" among different data elements.

Definition 2. **A graph database** is a collection of different graphs representing different relationship structures.

Definition 3. **Graph isomorphism** between two graphs **G** and **H** is a bijection relationship between the vertex sets of **G** and **H:** f: V(G)$\longrightarrow$ V(H)**,** such that any two vertices **u** and **v** of G are adjacent in G if and only if ***f*(u)** and ***f*(v)** are adjacent in **H**. This kind of bijection is commonly called "edgepreserving bijection", in accordance with the general notion of isomorphism being a structure-preserving bijection.

Definition 4. **Level-n Adjacent Subtree**: Given a graph G and a vertex u ™ G, a level-n adjacent subtree of u is a breadth-first tree on G starting at u and consisting of paths of length d™ n.

Definition 5. **Level-n Pseudo Compatible**: Vertex u is called level-n pseudo compatible to v if the level-n adjacent subtree of u is sub-isomorphic to that of v.

Definition 6. **Level-n Pseudo Sub-Isomorphism**: Given two graphs G1 and G2, define a bipartite graph B as follows: the vertex sets of B are the vertex sets of G1 and G2; for any u $\in$ G1, v $\in$ G2, if u is level-n pseudo compatible to v, then (u, v) is an edge in B. G1 is called level-n pseudo subisomorphic to G2 if B has a semi-perfect matching.

Super-graph Query Problem Definition

Input: A query graph **q** and a database of graphs **D** = $\{g_1, g_2, \ldots, g_n\}$

Output: Answer Set of **q** = $\{\mathbf{g_i} \mid \mathbf{g_i} \subseteq \mathbf{q} \text{ and } \mathbf{g_i} \in \mathbf{D}\}$

3. General Solution Framework

Unlike the solution of sub-graph queries, the solution of super-graph queries is based on exclusion logic. If a set of graphs has a common substructure which does not exist in the query graph, then this set of graphs could not be part of the super-graph query. Hence, this set of graphs should be excluded from the candidate graphs of the final result. This solution framework is based on the filtering-verification solution framework.

So, the general solution framework is to flag all the graphs in the database to be candidate graphs. Then, assuming that we have some graphs substructures which are part of the existing graphs in the database. Each of these substructures (features) is associated with a list of graph identifiers which are the identifiers of the database graphs containing this substructure. For every substructure, it is checked if it is not contained in the query graph (sub-graph isomorphism test). If the substructure is not a subgraph of the super-graph query, then its associated database graphs are excluded from the candidate graphs. After excluding the non-candidate graphs, we will have a set of candidate graphs which count much less than the graphs existing in the database. This is the filtering phase. In the verification phase, each of the candidate graphs is tested to determine if it is a sub-graph of the super-graph query or not. Graphs that pass this isomorphic test are the only ones to be included in the final result set.

The main design goal of the proposed technique is to minimize the number of the sub-graph isomorphic tests. It is known that sub-graph isomorphic testing is an NP-complete problem. So, reducing the number of the isomorphic tests is crucial for an efficient technique.

4. The Proposed Technique

We propose a filtering-verification technique for processing super-graph queries. The technique depends on building an index (eIndex) which is discussed in section 4.1. The query processing algorithm that uses eIndex is discussed in section 4.2. The complexity analysis of the technique is presented in section 4.3

4.1 Index Construction

eIndex is used by the proposed technique for processing super-graph queries. eIndex construction is based on extracting the frequent and discriminative features from the graph database. These features are sub-graphs of the database graphs. gIndex [12] is used to extract such features. These extracted features are used in the filtering stage. Each feature is associated with the following attributes:

• String sequence representing the feature graph

• List of database graph IDs containing this feature as a sub-graph

• List of features IDs containing this feature as a sub-graph

The following subsections present the meaning and the usage of each feature attribute:

Graph IDs List: Each graph database has a unique identifier. Each feature has a list of graph identifiers of the graph database containing this feature. These graphs are excluded from the candidate set if and only if their associated feature is not a sub-graph of the super-graph query.

Feature IDs List: Features are represented by graphs. Some features may be sub-features of others. Suppose that feature F1 is a sub-feature of feature F2. Then the database graphs associated with F2 is a subset of the database graphs associated with F1. If feature F1 is not a sub-graph of a super-graph query q ($F1 \not\subseteq q$), then the list of database graphs associated with F1 should be excluded and it is not necessary to check feature F2 as its associated database graphs are already excluded after excluding F1 associated graphs.

String Sequence: Each feature is associated with a string sequence representing the feature edges explicitly and the feature vertexes implicitly in a specific order. This sequence represents the edges of the feature graph sorted by the labels of the vertexes connected by each edge. For example, the following feature graph in **Figure 4-1**(a) will have the following sequence "AAABBABCDADB". The edges are sorted by the label of the first vertex; if they are equal then they are sorted by the labels of the second vertex.

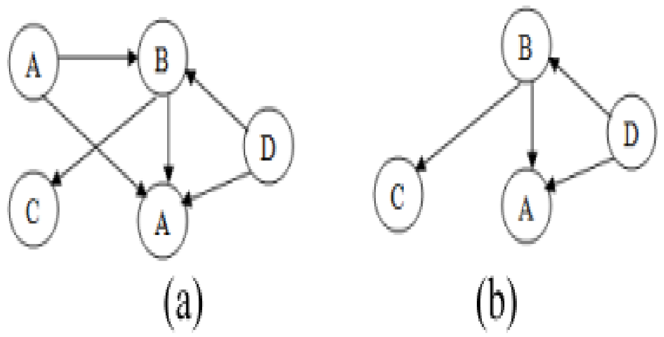

Figure 4-1. Sample Graphs

A string sequence of a graph contains the vertexes labels and the edges of the graph. If a graph **x** is a sub-graph of graph **y** (**x** $\subseteq$ **y**), then the string sequence of graph **x** should be a subsequence string of the sequence representing graph **y**.

The same steps yield the string "BABCDADB" for the graph in **Figure 4-1**(b). And since the samplegraph 1 is a super-graph of the sample graph 2, and then the string "BABCDADB" should be a substring of "AAABBABCDADB" which is true.

eIndex could be represented by a list of size M containing the extracted features. Each entry contains the graph representing the feature, the list of graph IDs containing this feature as a sub-graph, the list of feature IDs containing this feature as a sub graph and the string sequence representing the graph of the feature.
The following diagram shown in Figure 4-2 shows the process of constructing eIndex. The database graphs containing N graphs are processed by gIndex to extract the frequent and discriminative sub-graph features of the database graphs. This

yields a set of features of size M. Then, each feature has its associated subsequence string calculated. The features are sorted by the count of graph IDs associated with them in descending order. Then a feature is associated with the list of features containing that feature as a sub-graph. Finally eIndex is generated which could be represented as a list of these ordered and annotated features.

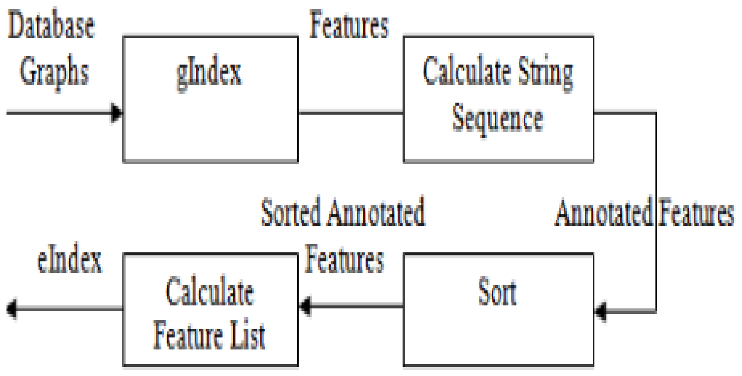

Figure 4-2. eIndex Construction

4.2 Query Processing

Given a query **q**, the proposed technique scans eIndex to exclude the graphs that are not sub-graphs of **q**. eIndex scanning yields a set of candidate graphs that may contain false positives which are verified by isomorphism tests to be excluded from the final result set.

Initially all the graphs contained in the database are candidates and all the features in eIndex are marked as not examined yet. After constructing eIndex, each untested feature of eIndex is tested to know whether it is a sub-graph of the query graph or not. If the feature is not a sub-graph of the query graph, then we can exclude the list of graphs associated with the feature from the candidate set. This containment test could be done using a subgraph isomorphism testing algorithm like Ullman's algorithm [20]. Sub-graph Isomorphism testing algorithms are known to be NP-complete algorithms which may take exponential running time. To reduce the running time without losing validity, approximate tests are tried first to decide the exclusion of graphs from the candidate set. These approximate tests are performed through polynomial time algorithms and hence the running time is reduced. These approximate tests should increase the number of candidate graphs but this increase is not so large. Also, to reduce the number of candidate graphs, a post-filtering is performed after scanning eIndex.

The approximate sub-graph isomorphism tests are based on the pseudo sub-graph isomorphism [13] and the longest common subsequence algorithms. The longest common subsequence algorithm is faster but it is not accurate like the sub-graph isomorphism test which has the highest pruning power.

Given a graph query **q** and a feature **f**, the string sequence of **f** is tested to see if it is a subsequence of that of the query **q** or not. If it is not a subsequence, then the feature cannot be a sub-graph of **q** and the graphs associated with the feature **f** are excluded because they cannot be sub-graphs of **q**. If the subsequence test indicated that the string sequence of **f** is a subsequence of that of q, then the more accurate sub-graph isomorphism test is run. If the sub-graph isomorphism test yields that the feature is not a sub-graph of **q**, then the list of graphs associated with that feature are excluded from the candidate set.

All the previously stated tests were done on substructures (not on the whole database graphs). To reduce the number of candidate graphs, a postfiltering process is performed on the candidate graphs generated from scanning eIndex. This post filtering process is done through testing approximately whether the candidate graphs are subgraphs of **q** or not. This test is performed using the pseudo sub-graph isomorphism algorithm. This test acts on the whole structures of the candidate graphs and hence adds a significant pruning power to the filtering stage.

Figure 4-3 shows the query processing phase.

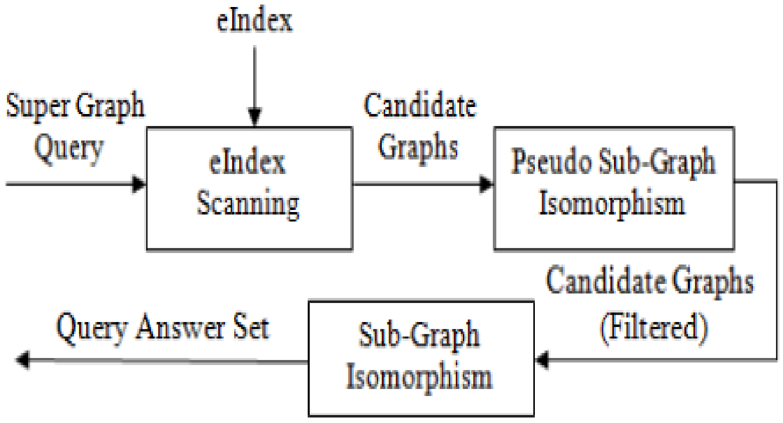

Figure 4-3. The Proposed Super-Graph Query Processing Technique

Figure 4-4 lists the query processing algorithm that uses eIndex.

```
Algorithm 1 The Proposed Technique Algorithm

Input: eIndex = {F_1, F_2,… F_m}
       Graph Database D = {g_1, g_2…, g_N}
       Graphy Query q
Output: Graphy Result Set R

1: R ⇐ D
2: for each feature F in eIndex
3:     F.IsChecked ⇐ False
4:     compute q.StrSeq
5: for each feature F in eIndex
6:   if (F.IsChecked = False)
7:     if (F.StrSeq is not subsequence of q.StrSeq)
8:        for each graph g in F.Graph
9:            exclude g from R
10:       F.IsChecked = True
11:       for each feature F_i in F.Features
12:           Fi.IsChecked ⇐ True
13:    else If (F is not a sub-graph isomorphic of q)
14:       for each graph g in F. Graphs
15:           exclude g from Rxx1x`
16:       F.IsChecked = True
17:       for each feature Fi in F.Features
18:           Fi.IsChecked ⇐ True
19: for each candidate graph g in R
20:   If (g is not a pseudo sub-graph isomorhic of q)
21:     Esclude g from R
22: for each candidate graph g in R
23:   If (g is not sub-graph isomorphic of q)
24:     Exclude g from R
25: return R
```

Figure 4-4. The Proposed Super-Graph Query Processing Algorithm

Line 1 marks all the database graphs as candidate graphs. Line 2-3 marks all the eIndex features as untested. Line 4 computes the string sequence of the super-graph query **q**. Lines 5-18 scans eIndex to exclude non-candidate graphs, the feature is checked whether it could exclude graphs or not before doing the graph checking (Line 6). Lines 7-12 exclude the graphs associated with features which have a string sequence not being a subsequence of the query string sequence. If the string subsequence test is positive, a sub-isomorphic test is performed (Lines 13-18). If the graphs of a feature are excluded, then the associated features with that feature have their associated graphs also excluded (Lines 11-12 and Lines 17-18). Lines 19 21 do the post-filtering step. Line 21 is the end of the filtering phase. Lines 22-24 perform the verification phase. Line 25 returns the result set **R** where only the actual graphs which are sub-graphs of **q** are marked.

4.3 Performance Analysis

Let Tconstruct be the time taken to construct an eIndex. Tconstruct could be calculated as follows:

$$T_{construct} = T_{gIndex} + T_{seq} + T_{sort} + T_{feature}$$

T_{gIndex} is the time taken by gIndex to get the features used by eIndex. T_{seq} is the time taken to compute the string sequence of the features. It takes O(n lg n) to compute the string sequence of a graph g of n edges (string sequence requires sorting the edges). T_{sort} is the time taken to sort the features according to the number of database graphs associated with them. Sorting could be done in time O(m lg m) to sort **m** features. $T_{feature}$ is the time taken to calculate for each feature, the list of features which are super-graphs of it. This requires $O(m^2)$ isomorphic tests where m is the number of features. Tfeature is only useful in saving time in the phase of query processing but it doesn't affect the number of candidate graphs.

The time taken by the query processing algorithm (denoted by T_q) is the one that should be minimized as it is taken for processing every query. String subsequence test could be done in O(n) time where n is the number of edges of the bigger graph. Due to the fact that sub-graph isomorphism takes more time than the string subsequence test, we ignore the time taken by the subsequence test. T_q in the worst case could be calculated as follows:

$$T_q = |M| * T_i + |C_1| * T_p + |C| * T_i$$

where |M| is the number of features. Ti is the time taken by sub-graph isomorphism testing. Tp is the time taken by pseudo sub-graph isomorphism testing. The worst case time complexity of the pseudo subgraph isomorphism test (T_p) between graphs G_1 and G_2 is $O(L\ n_1\ n_2(d_1\ d_2 + M(d_1, d_2)) + M(n_1, n_2))$ [13] where L is the pseudo compatibility level, n_1 and n_2 are numbers of vertices in G_1 and G_2, d_1 and d_2 are the maximum degrees of G_1 and G_2, M() is the time complexity of maximum cardinality matching for bipartite graphs. Hopcroft and Karp's algorithm [21] finds a maximum cardinality matching in $O(n^{2.5})$ time. $|C_1|$ is the number of candidate graphs on which the post-filtering step is performed. |C| denotes the final number of candidate graphs resulting from the filtering phase (after post-filtering).

5. Experimental Results

The experimental results are presented in this section to show the efficiency of the proposed technique. Our technique eIndex is compared to cIndex; unlike GPTree which requires storing the graph database in a compact format, cIndex and eIndex don't require storing the database in a specific format. Storing a version of a database in a compact form for query processing only may not be practical in some applications. So, GPTree is not compared in our experimental results. The results are drawn using two kinds of datasets. The chemical compounds dataset (AIDS) that was used in evaluating cIndex and a synthetic dataset. The experiments were done on an Intel Core2Duo2-2.26 GHz laptop with 3 GB of Ram running Windows Vista. Both eIndex and cIndex were implemented in C++ and compiled using MS C++ compiler version 8.

5.1 AIDS Antiviral Screen Dataset

The experiments described in this section are performed on the AIDS antiviral screen dataset (it can be downloaded from http://dtp.nci.nih.gov/docs/aids/aids_data.html). This dataset contains more than 43000 chemical compounds. The pseudo level of eIndex was set to the multiplication of the number of vertexes of the two graphs being operated on. The pseudo level is used in the pseudo sub-graph isomorphic tests. The parameters in cIndex were set as follows:

- 10000 graphs are randomly chosen from the AIDS antiviral dataset to make a dataset W.
- The W dataset is divided into five sets; each of which contains 2000 graphs. One of the five sets is taken as the testing set

Q which contains 2000 graph. The other four sets are combined to make a query log set L which contains 8000 graph. This set is used only by cIndex for comparison purposes. The set L is not used by eIndex.

In order to build a graph database that contains the chemical descriptors, we applied frequent subgraph mining on W to extract the set D which contains 5000 graphs. We applied frequent sub-graph mining on W and we retained the discovered graphs that have frequency between 5% and 10%.

Figure 5-1 shows the number of candidates generated by the filtering phase of cIndex and eIndex. The X-axis represents the average size of the query answer set generated from the verification phase while the Y-axis represents the average size of the candidate set generated from the filtering phase. The "Actual" line shows the average size of the answer set which represents the minimum number of isomorphism tests that should be done by any query processing technique. The "cIndex" curve shows the average size of the candidate sets generated from the filtering phase of cIndex. The "eIndex" curve shows the average size of the candidate sets generated from the filtering phase of eIndex. eIndex shows better results regarding the number of candidate graphs due to its post filtering phase which takes into consideration the full structure of the database graphs that was not excluded by the index scanning phase.

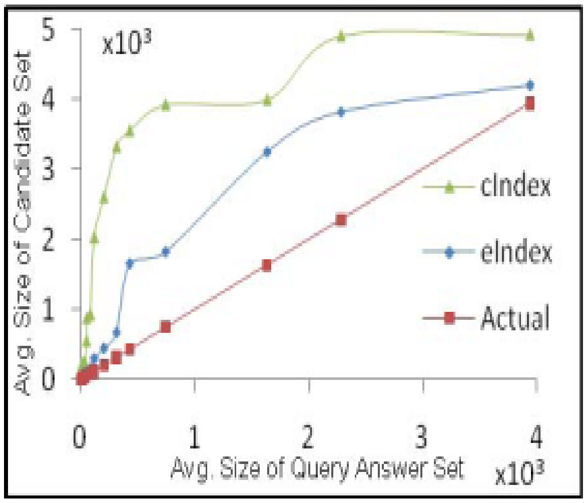

Figure 5-1. Candidate Sets Comparison

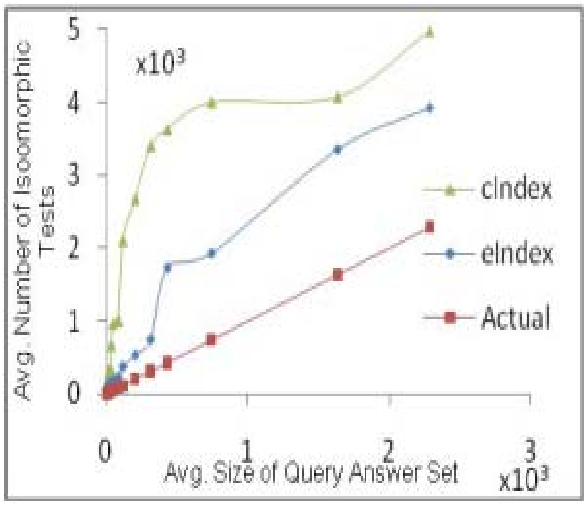

Figure 5-2. Isomorphic Tests Comparison

Figure 5-3 compares the time taken by eIndex and that taken by cIndex during the overall process of querying. The processing time is reduced by eIndex nearly by half for queries having small result sets and by an order of magnitude for queries with large result sets.

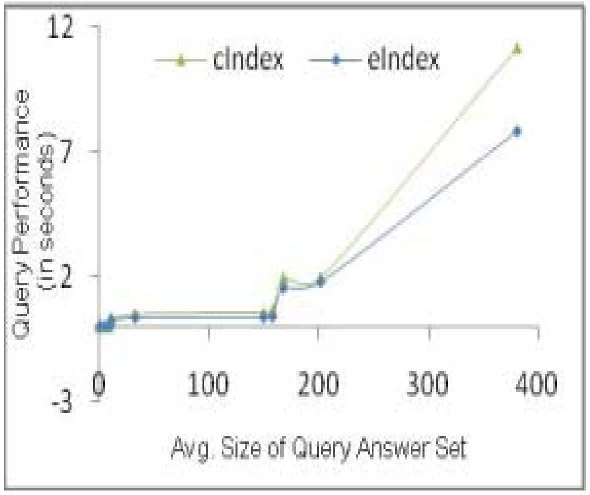

Figure 5-3. Processing Time Comparison

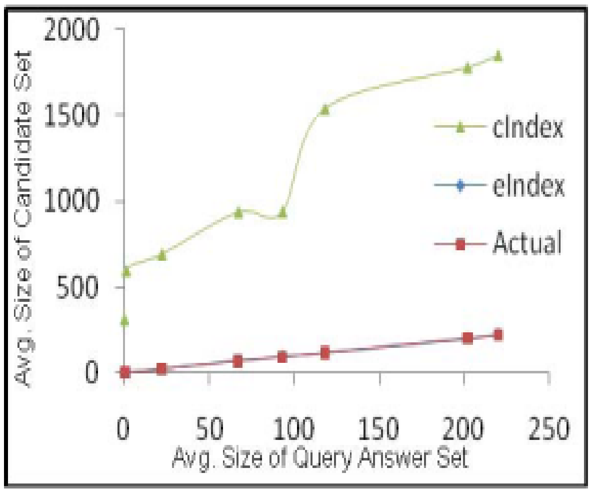

Figure 5-4. Candidate Sets Comparison

Table **5.1** compares between the time taken to build eIndex and that taken to build cIndex. eIndex building time is consumed by sorting the features and assigning to each feature its super features. The time taken by cIndex is consumed in building the contrast matrixes and selecting the features that have more pruning power based on the query logs.

eIndex		cIndex	
Time(Seconds)	|F|	Time (Seconds)	|F|
0.242716	105	408.355	74
0.305592	118	448.816	80
0.422937	139	528.474	90
0.679026	178	623.093	98
1.27463	247	766.134	101
1.73344	283	827.698	101

Table 5.1 Pre-processing performance for AIDS

5.2 Synthetic Dataset

In this section, the experiments were performed on a synthetic dataset. The GraphGen graphs generator was used [23]. The parameters were set to generate 10000 graphs, the average size of the graphs was set to 15 (number of edges); the number of unique vertexes and edges labels was set to 5. The parameters in cIndex were set as follows:

• The 10000 graphs generated by[23] are used to make a dataset W.

• The W dataset is divided into five sets; each of which contains 2000 graphs. One of the five sets is taken as the testing set Q which contains 2000 graph. The other four sets are combined to make a query log set L which contains 8000 graph.

In order to build a graph database that contains the chemical descriptors, we applied frequent subgraph mining on W to extract the set D which contains 5000 graphs; we applied frequent sub-graph mining on W and we retained the discovered graphs that have frequency greater than 3%.

Figure 5-4 shows the number of candidates generated by the filtering phase of cIndex and eIndex.

Figure 5-5 shows the total number of isomorphic tests performed during the whole graph query processing process. The isomorphic tests done by eIndex are near optimal.

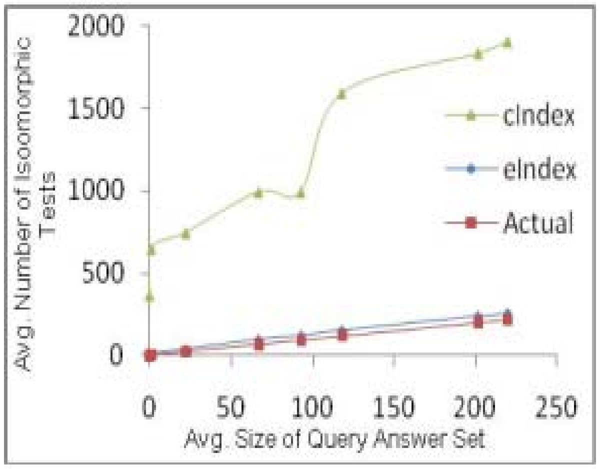

Figure 5-5. Isomorphic Tests Numbers Comparison

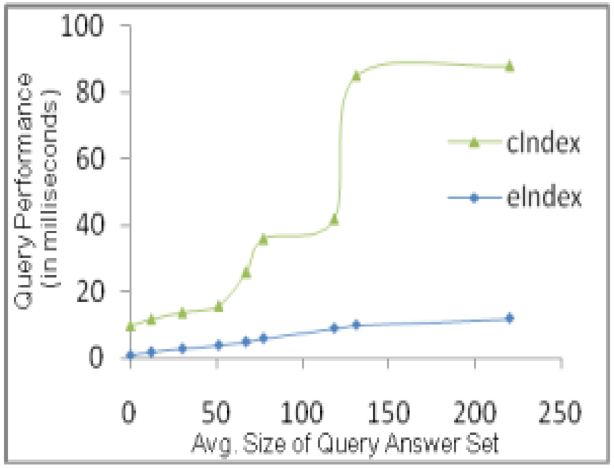

Figure 5-6. Processing Time Comparison

6. Conclusions

In this paper, a query processing technique for answering super-graph queries is presented. Solving this problem requires isomorphic tests which are known to be NP-complete. In order to have an efficient solution for the super-graph query problem, the number of the sub-graph isomorphic tests in the filtering phase was reduced by using approximate algorithms. A polynomial time post-filtering phase is added. This post filtering phase takes into consideration the full structure of the database graphs generated from eIndex scanning. Therefore, a higher pruning power is achieved. Generating few candidate graphs before the verification phase proves its efficiency for processing queries over large graph databases.

7. Acknowledgements

We thank Huahai He for proving the executables of CTree and Chen Chen for providing the executables of cIndex plus their clarifications for some details regarding their work.

8. References

[1] Fry, James., Sibley, Edgar. (1976). Evolution of Data-Base Management Systems, *ACM Computing Surveys* 8 (1) 7-42.
[2] Codd, E. F (1970). A relational model of data for large shared data banks, *ACM Computing Surveys* 13 (6) 377-387.
[3] Angles, Renzo., Gutierrez, Claudio (2008). Survey of graph database models, *ACM Computing Surveys*.
[4] Stonebraker, Mike et al (2005). C-Store: A Column Oriented DBMS. VLDB, p.553-564.
[5] http://www.vertica.com (has been accessed on 1/2/2009).
[6] Boncz, Peter (2002). Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications, PHD thesis, University of Amsterdam.
[7] http://www.luciddb.org (has been accessed on 1/3/2009).
[8] http://www.equi4.com/metakit (has been accessed on 1/3/2009).
[9] http://sdm.lbl.gov/fastbit (has been accessed on 1/3/2009).
[10] Jin, Ruoming et al (2008). Efficiently Answering Reachability Queries on Very Large Directed Graphs. *In:* SIGMOD, p. 595-607.
[11] Cook (1971). The complexity of theorem-proving procedures. *In:* Proceedings of the 3rd ACM Symposium on Theory of Computing.
[12] Yan, Xifeng, Yu, Philips S., Han, Jiawei (2005). Graph Indexing Based on Discriminative Frequent Structure Analysis. ACM Transactions on Database Systems, 30 (4) 960–993.
[13] Yan, Xifeng, Yu, Philips S., Han, Jiawei (2005). Substructure Similarity Search in Graph Databases. *In* SIGMOD, p. 766-777.
[14] He, Huahai., Singh, Ambuj K (2006). Closure-Tree: An Index Structure for Graph Queries, *In*: Proceedings of the 22nd International Conference on Data Engineering, IEEE.
[15] Giugno, Rosalba., Shasha, Dennis (2002). GraphGrep: A Fast and Universal Method for Querying Graphs, *In*: Proceedings of 16th International Conference on Pattern Recognition.
[16] Zhang, Shijie., Li, Shirong., Yang, Jiong (2009). GADDI: Distance Index based Sub-graph Matching in Biological Networks. Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, 360, p.192-203.
[17] Jeffrey, Haichuan (2008). Taming Verification Hardness: An Efficient Algorithm for Testing Sub-graph Isomorphism, *In*: Proceedings of the VLDB Endowment, 1 (1) p. 364-375.
[18] Cheng, J. et al (2009). Efficient query processing on graph databases, ACM Transactions on Database Systems (TODS), 34 (1). Article No. 2.
[19] Chen Chen et al (2007). Towards Graph Containment Search and Indexing,*In*: VLDB Conference, Vienna, Austria.
[20] Ullmann, J. R. (1976). An algorithm for sub-graph isomorphism, *J. ACM*, 23 (1) 31–42.
[21] Hopcroft, J., Karp, R (1973). An n5/2 algorithm for maximum matchings in bipartite graphs. *SIAM J. Computing*.
[22] Zhang, Shuo et al (2009). A Novel Approach for Efficient Super-graph Query Processing on Graph Databases, *In*: EDBT Conference, Saint Petersburg, Russia.
[23] Cheng, James., Ke, Yiping., Ng, Wilfred (2006). GraphGen: A graph synthetic generator. http://www.cse.ust.hk/graphgen/