http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (2): 214-219, 2009
ISSN 1812-5638

A Decomposition Based Algorithm of Graph Containment Query

Li Xian-Tong and Li Jian-Zhong
School of Computer Science and Technology,
Harbin Institute of Technology, Harbin 150001, China

Abstract: In this study, an algorithm ESGC is proposed to implement graph containment query problem, both exact and similar. The index of ESGC is built on two parts, the process of graph dataset decomposition and a hash table. The processing of graph dataset decomposition forms a structure which reduces the size of candidate answer set. And the hash table is composed by graph canonical code, through which the algorithm avoids subgraph isomorphism test during picking candidate answers out. The progress of the performance is coming from thus two parts. Experimental results illustrate that ESGC performs an efficient graph containment query and achieves right and entire answer set.

Key words: Graph query, graph mining, containment query, graph containment

INTRODUCTION

Graphs, as a general data structure, provide a powerful and primitive tool to model the data in a variety of applications, e.g., social or information networks, biological networks, 2D/3D objects in pattern recognition, wired or wireless interconnections, chemical compounds or protein networks. Nodes in graphs usually represent real world objects and edges indicate relationships between the objects. For example, in computer vision, graphs are used to represent complex relationships, such as the organization of entities in images. In chemical informatics and bio-informatics, graphs are employed to denote compounds and proteins. In order to accurately describe the characters of the data, the labeled graphs are often applied. The nodes and edges are associated with attributes in a labeled graph. With the tremendous amount of structured or networked data accumulated in large databases, how to efficiently support the scalable graph query processing becomes a challenging research issue in database area.

Given a graph database, $D = \{g_1, g_2, \ldots g_n\}$, generally, the graph queries over the database can be divided into two categories, subgraph query and containment query. The subgraph query retrieves the graphs in the database that are supergraphs of the query graph. For example, in chemistry, the subgraph can be used to retrieve the compounds containing the given special substructure. On the other hand, the graph containment query finds the graphs that are subgraph of the query graph. For example, also in chemistry, a descriptor (model graph) is a set of atoms with designated bonds that has certain attributes in chemical reactions. Given a new molecule, identifying descriptor structures can help researchers to understand its possible properties.

Recently, the subgraph query, both exact and approximate query, has been received much more concern and been well studied (Cheng *et al*., 2007; He and Singh, 2006; Jiang *et al*., 2007; Williams *et al*., 2007; Yan *et al*., 2004; Zhang *et al*., 2007; Zhao *et al*., 2007). At the same time, some subgraph similarity query algorithms are proposed by Yan *et al*. (2005) and Yan *et al*. (2006). However, the research on graph containment query is far from enough. While processing the containment queries, the algorithm can still adopt the filtering-and-verification approach to reduce the size of candidate graphs set, but the inclusion logic which is used in subgraph query processing can not be applied. Here, the exclusion logic is employed instead. Suppose a query graph q is the supergraph of a graph g. Then, all features in g must be included in q. According to the exclusion logic, if a feature f is not a subgraph of q, the graphs that are supergraphs of f can be safely pruned from candidate answer set. Chen *et al*. (2007) proposed an algorithm, cIndex, to process the exact containment query. The primary cost of the algorithm is composed by two parts, including the cost on performing subgraph isomorphism between the query graph and the picked features, performing subgraph isomorphism between the query graph and the candidate graphs. During query processing procedure, cIndex performs subgraph isomorphism tests between the query graph and all indexed frequent subgraphs. This indiscriminate approach greatly reduces the efficiency of cIndex. Moreover, cIndex can not manipulate similarity containment query.

Corresponding Author: Li Xian-Tong, School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

In this study, an algorithm ESGC (Exact/Similarity Graph Containment query) is proposed to solve the problem of graph containment query, both exact and similarity. Graphs are decomposed into DAG (Direct Acyclic Graph) to form index. The main contributions of this study are:

- As the best knowledge, this is the first algorithm to solve graph containment similarity search
- When there is no answer of exact graph containment query, ESGC provides approximate answer
- The efficiency of ESGC is more efficient than cIndex when it takes place in exact situation

PRELIMINARIES

Definition 1: (Labeled Graph) A labeled graph is a 4-tuple, G = {V, E, Σ, l}, where V is the set of vertex, E is the set of edges, Σ is the set of labels and l is a labeling function which mapping a label to a vertex of an edge, or l: V→Σ (l: E→Σ).

As a general data structure, labeled graph is used to describe complicated structure and data. Vertices and edges in labeled graph represent entities and relationships between entities, respectively. The attributes of entities and relationships are labels. XML data is a kind of directed labeled graph and chemical compound is a kind of undirected labeled graph.

Given a labeled graph G, the vertex set of G is signed as V(G) and the edge set of G is E(G). The size of G is the number of vertices it has, or |V(G)|, which is signed as S(G).

Definition 2: (Subgraph Isomorphism) Given two graphs $g_1 = (V_1, E_1, \Sigma_1, l_1)$ and $g_2 = (V_2, E_2, \Sigma_2, l_2)$, g_1 is subgraph isomorphic to g_2, if there is an injective function b: $V_1 \rightarrow V_2$ that satisfied:

- For $\forall v \in V_1$, $b(v) \in V_2$, $l_1(v) = l_2(h(v))$
- For $\forall(u, v) \in E_1$, $\exists(b(u), b(v)) \in E_2$ and $l_1(u, v) = l_2(b(u), b(v))$

Here, the function b is called an embedding of g_1 in g_2 which is signed as $b(g_1, g_2)$.

Given two graphs g_1 and g_2, if g_1 is a subgraph of g_2, there is a subgraph isomorphism between g_1 and g_2, signed as $g_1 \subseteq g_2$. And g_2 is called a supergraph of g_1, or $g_2 \supseteq g_1$. If B is an embedding of g_1 in g_2, the distance from g_1 to g_2 is defined as:

$$d_B(g_1, g_2) = \sum[l(u) \neq l'(B(u))] + \sum[l(u,v) \neq l'(B(u,v))]$$

where, u, v is vertices and (u, v) is an edge. If p is the common subgraph of g_1 and g_2 and B_p is and embedding of g_1 in g_2, $d_B(g_1, g_2)$ is smallest when p is the biggest among all common subgraphs. This smallest distance is called the similarity between g_1 and g_2, or $d(g_1, g_2) = \min(d_B(g_1, g_2))$.

Definition 3: (Similar Graph) Given two labeled graphs g_1, g_2, $S(g_1) = S(g_2)$ and the minimum similarity δ, if g_1 and g_2 satisfy:

$$\frac{|d(g_1, g_2)|}{S(g_1)} \leq \delta$$

they are similar. Here, g_1 is similar contained by g_2, or $g_1 \subseteq g_2$.

In this study, an algorithm is proposed, which named ESGC (Exact/Similar Graph Containment Query), to solve graph containment query. Exact query and similar query can be transformed smoothly. The algorithm in this study is used to query on labeled undirected graph set, but it can also be applied to solve directed graph environments.

GRAPH CONTAINMENT QUERY

The methodology of filter-verification, which filters out unsatisfied graphs to reduce the size of candidate set and gets the answers through verification, is well used in graph query algorithms. But, subgraph isomorphism test is NP-complete problem (Garey and Johnson, 1979), which cannot be avoid in verification stage, the size of candidate set is one of the most important parameters about the efficiency of the algorithm. In this study, the target of ESGC is to reduce the size of candidate set as small as possible to promote the efficiency of query.

Here, canonical code of graph is given first with which is used to translate a graph into a sequence. Then, decomposition of graph is introduced and an index construction algorithm is founded on it. At last, the algorithm ESGC is given.

The canonical code: Graphs can be described as adjacency list or adjacency matrix. Both of these two forms give a chance to translate a graph into sequence. However, the primary problem of such translation is to found a one-one mapping between sequences and graphs. Both maximum sequence and minimum sequence can realize the mapping and form the canonical code. In a word, the following function is needed in translating graphs: f: g→s. If g and g is isomorphism, then, f(g) = f(g). Otherwise, f(g)≠f(g). Here, g means a graph, s means a sequence which represents graph g.

In this study, the minimum adjacency list sequence is picked as the canonical code of a graph.

Graph decomposition: Graph decomposition is a process to generate ALL induced subgraphs. These subgraphs form a DAG (Directed Acyclic Graph). For a graph G, its decomposition DAG records the following messages (p and q are two nodes in DAG).

- Every node is a subgraph of G
- If there is an edge which begins from p and ends at q, the $p \subset q$
- If there is a path which begins from p and ends at q, then $p \subset q$. If there are n node on this path, p, g_1, g_2, …, g_{n-1}, g_n, q, then $p \subset g_1 \subset g_2 \subset \ldots \subset g_n \subset q$
- Every path is from one node which refers to a NULL graph and ends at node which represents G

This DAG is called graph decomposition DAG: Graph decomposition DAG has some special features. If the sizes of two subgraphs of G are same, the distances from root to both nodes are equal. The distances from these two nodes to G are equal, too. And there are filiations between nodes in neighbor layers (Nodes in upper layer are subgraphs of lower layer).

Example 1: Figure 1 is a labeled graph G and its decomposition DAG. For simplicity to describe, the labels of edges are omitted. In decomposition DAG, sequences of vertices are used to represent subgraphs of G. Every longest path in this DAG begins from root (or NULL graph) and ends at G. a and b are equal size. The distance from root to a or b is equal. Obviously, the distance from a or b to aabc is equal, too. At the same time, a at layer 1 and aa at layer 2. a is a subgraph of aa because there is an edge from a to aa.

The decomposition method can be easily extended to the whole graph dataset. If two graphs generate one same subgraph in DAG, it only appears once in the dataset DAG. Each graph in dataset shares the same root node in DAG. In next subsection, an index structure based on graph dataset DAG is introduced.

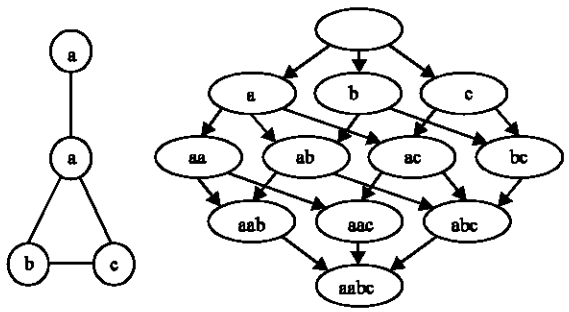

Fig. 1: Labeled graph g and its decomposition DAG

DAG index: There are two parts in DAG index. One part is the decomposition DAG of the whole graph dataset which records all possible induced subgraphs in the dataset. The other part is a hash table. Each entry in hash table is composed by two items, which are hash key and a pointer. The hash key is corresponding to the canonical code of subgraph in DAG and the pointer points to the counterpart node in DAG.

DAG index is constructed by the algorithm in algorithm 1. The algorithm decomposes every graph in graph dataset one bye one. If a graph in dataset has been already decomposed, the algorithm should pick up another graph in dataset and decompose it. Along with decomposition, it adds induced subgraphs of current graph into the whole graph dataset DAG and forms directed edges according to the relationships between them. It also forms an entry points to that subgraph node in hash table.

Algorithm 1: Index Construction

```
Input: Graph Dataset D
Output: Graph Dataset DAG, Canonical Code Hash Table
Construct (D)
   H = Φ /*Canonical code hash table*/
   DAG = Φ /*Graph dataset DAG*/
   for every G in D
   if G has not been decomposed then
   DAG = DAG∪G
   H(G) = G /*add the entry of G into hash table*/
   Decomposition (G, DAG, H)
   end if
 end for
 return (DAG, H)

Decomposition (G, DAG, H)
   for every vertex v in G
      G' = G-v; /*also prunes edges connected v*/
      if G' is not contained by DAG then
        DAG = DAG∪G'
        E_DAG = E_DAG + {G', G} /*An edge from G' to G*/
        H(G') = G'
        Decomposition(G', DAG, H)
      end if
   end for
```

Algorithm 1 Graph Dataset Decomposition

Graph containment query: Graph containment query algorithm ESGC is given in algorithm 2. At the beginning of this algorithm, it initializes two sets. One is candidate set C, the other one is a set of nodes of DAG_q which are visited by the algorithm, checked. There are two signs in this algorithm, h_q and H_D. H_D represents a hash entry in DAG and h_q represents a hash entry in DAG_q. Iteratively,

it checks whether the current hash key h_q is an entry of DAG. If it is, the algorithm checks whether it is a dataset graph (a candidate). Otherwise, it tracks back to parents of q in DAG_q until it reaches root. If $\delta > 0$, ESGC calculates the similarity bound (in algorithm 2, λ). This value is the bound of missing vertices in h_q, which represents a node of DAG here, about similarity candidates. As a candidate of answer, ESGC puts all dataset graphs into C which are offspring in λ layers from current node h_q. If $\delta = 0$, ESGC picks out exact candidates only.

```
Algorithms 2: Graph containment query
Input: Dataset DAG, DAGq, δ, q
Output: Candidate Answer Set
ESGC(DAG, DAGq, δ)
    C = Φ /*candidate set*/
    checked = Φ /*whether a node in DAGq is cheched*/
    if hq is an entry of HD then
        checked = checked∪hq
        if hq represents a dataset graph then
          C = C∪hq
for every father q' in DAGq of q
        VisitUpper (hq', DAG, DAGq, C)
return C
VisitUpper (hq, DAG, DAGq, C)
if hq checked then
        checked = checked hq /*corresponding node in DAGq*/
        if hq is an entry of HD then
            if hq represents a dataset graph then
              C = C∪hq
            if δ > 0 then
λ←S(hq)*δ
Ct ←all dataset graphs in λ steps offspring of hq in DAG
        C = C∪Ct
    for every father q' in DAGq of q
VisitUpper1(hq, DAG, DAGq, C)
```

Algorithm ESGC filters out all candidates which are contained or similar contained by query graph. After the candidate set if formed by ESGC, the verification stage takes place to verify whether a candidate is an answer. This stage takes subgraph isomorphism test between query graph q and every candidate.

EXPERIMENTAL RESULTS

In order to evaluate the effectiveness and efficiency of the proposed query processing algorithm, ESGC is compared with cIndex, the up-to-date approach, when exact query occurs by comprehensive experiments. The datasets are chosen as the workload in the experiments. Experiments are conducted on real datasets which are the source of real demand after all. The experiments are performed on a 3.2 GHz, 512 MB-memory, Intel PC running Windows XP Professional SP3. Both cIndex and ESGC are compiled with gcc/g++.

Dataset: The real data is getting from DTP chemical compound set, which cab be get from http://dtp.nci.nih.gov/docs/aids/aids_screen.html. First, 20,000 graphs are picked out randomly from all DTP dataset which has more than 40,000 entities. Then, these 20,000 graphs are divided into two parts equally. One of them forms the query dataset. The graph database is generated on the other part by graph mining algorithm which min-support is set from 0.5% to 10%. The mining results forms the graph database D_{init}. 10,000 graphs are picked randomly from D_{init} as the experimental graph database. In the process of query, query results are divided into 8 buckets: [0,10), [10,20), [20,30), [30,40), [40,100), [100,200), [200,500), [500,8). Each bucket represents the average number of query results. The query efficiency is compared with cIndex when the query results are exact. If the user proposed a similarity graph containment query, the performance of ESGC is compared with SCAN, the most naïve method of query.

Exact query: In Fig. 2, ESGC is compared with cIndex under the condition of exact graph containment query. It uses logarithm coordinate. In this experiment, ESGC is much more efficient than cIndex. The promotion of efficiency is coming from two parts. The first part is that ESGC locates subgraphs is faster than cIndex who takes subgraph isomorphism tests between q and all index features. The second part is that ESGC forms a smaller candidate set than cIndex. The smaller the candidate set is, the less the subgraph isomorphism takes.

Similarity query: Figure 3 shows the comparison of average query interval between ESGC and SCAN, which is the naïve method that checks every graph in dataset linearly, in graph similarity containment query. It can be drawn that ESGC is outperforming SCAN nearly 5 or 6 times. While the size of answer set grows, this ratio falls to about 3 times.

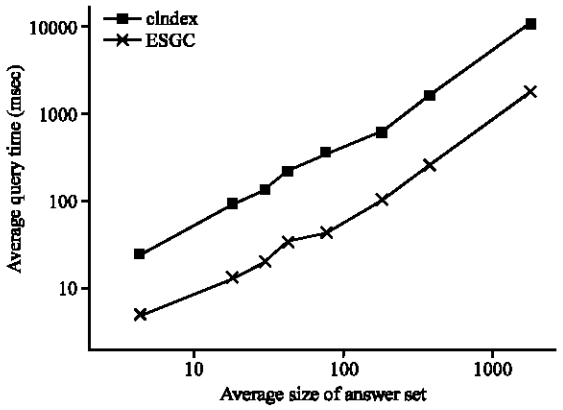

Fig. 2: Average query time

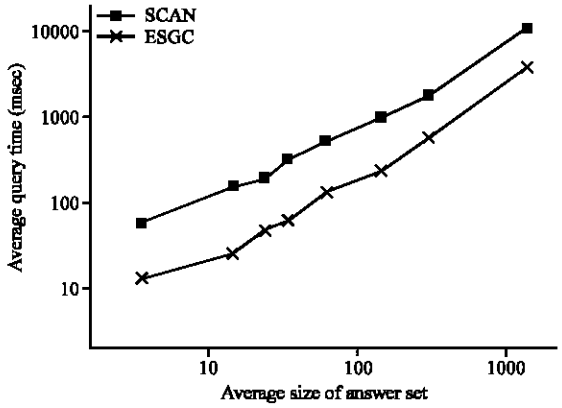

Fig. 3: Average query time

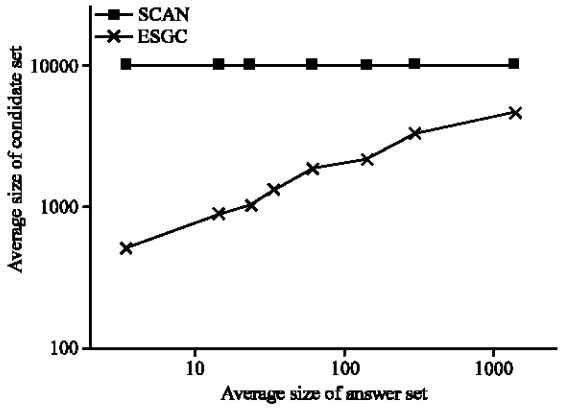

Fig. 4: Average size of candidate set

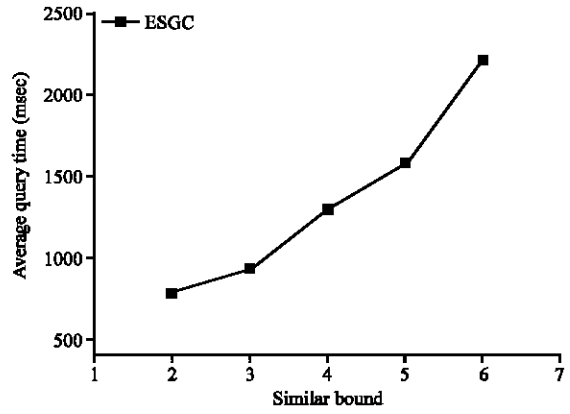

Fig. 5: The influence of similarity bound

Figure 4 shows the comparison of average size of candidate set, which is a primary reference of efficiency. The size ESGC gets is much smaller than SCAN. Referring to the result of Fig. 3, the gap of efficiency between ESGC and SCAN is getting closer while the average size of candidate grows in ESGC. This is the main reason of the reduction that ESGC outperforms SCAN.

The experiment in Fig. 5 shows that ESGC is influenced by δ. While δ increases, the efficiency of ESGC reduces. This result is under the condition that λ is influenced by δ, which determines the search depth in DAG. The performance of ESGC is affected by λ heavily.

CONCLUSION

In this study, an algorithm ESGC is proposed to solve graph containment query problem.

In this problem, there are two sub-problems, exact and similar graph containment query. As the best knowledge, this is the first study about graph similar containment query.

ESGC is built on an index which composed by two parts, graph dataset decomposition DAG and a hash table. Through that index structure, subgraph isomorphism test in finding candidates are avoids and forms a smaller candidate answer set. Experimental results illustrate that the proposed algorithm is more efficient than the up-to-date approaches.

REFERENCES

Chen, C., X. Yan, P. Yu, J. Han, D. Zhang and X. Gu, 2007. Towards graph containment search and indexing. Proceeding of the 33rd International Conference on Very Large Data Bases, Sept. 23-27, Vienna, Austria, pp: 926-937.

Cheng, J., Y. Ke, W. Ng and A. Lu, 2007. FG-index: Towards verification free query processing on graph databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, Jun. 11-14, Beijing, China, pp: 857-872.

Garey, M. and D. Johnson, 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. 11th Edn., W.H. Freeman and Company, New York, ISBN-10: 0716710455.

He, H. and A. Singh, 2006. Closure-tree: An index structure for graph queries. Proceeding of the 22nd International Conference on Data Engineering, Apr. 03-07, Washington, DC, USA., pp: 38-38.

Jiang, H., H. Wang, P. Yu and S. Zhou, 2007. GString: A novel approach for efficient search in graph databases. Proceeding of the 23rd International Conference on Data Engineering, Apr. 15-20, Istanbul, pp: 566-575.

Williams, D., J. Huan and W. Wang, 2007. Graph database indexing using structured graph decomposition. Proceeding of the 23rd International Conference on Data Engineering, Apr. 15-20, Istanbul, pp: 976-985.

Yan, X., P. Yu and J. Han, 2004. Graph indexing: A frequent structure based approach. Proceedings of the Special Interest Group on Management of Data, Jun. 13-18, Paris, France, pp: 335-346.

Yan, X., P. Yu and J. Han, 2005. Substructure similarity search in graph databases. Proceeding of the Special Interest Group on Management of Data, Jun. 14-16, Baltimore, Maryland, pp: 766-777.

Yan, X., F. Zhu, J. Han and P. Yu, 2006. Searching substructures with superimposed distance. Proceeding of the 22nd International Conference on Data Engineering, Apr. 03-07, Washington, DC, USA., pp: 88-88.

Zhang, S., M. Hu and J. Yang, 2007. TreePi: A novel graph indexing method. Proceeding of the 23rd International Conference on Data Engineering, Apr. 15-20, Istanbul, pp: 966-975.

Zhao, P., J. Yu and P. Yu, 2007. Graph indexing: Tree+delta >=graph. Proceeding of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, Sept. 23-27, VLDB Endowment, pp: 938-949.