Survey On Efficient Authenticated Sub Graph Query Service In Outsourced Graph Databases

Abstract
Today security is very important in the database system. Advanced database systems face a great challenge raised by the emergence of massive, complex structural data in bioinformatics, chem-informatics, and many other applications. Since exact matching is often too restrictive, similarity search of complex structures becomes a vital operation that must be supported efficiently. The Subgraph similarity search is used in graph databases to retrieve graphs whose subgraphs are similar to a given query graph. It has been proven successful in a wide range of applications including bioinformatics and chem-informatics, etc. Due to the cost of providing efficient similarity search services on ever-increasing graph data, database outsourcing is apparently an appealing solution to database owners. In this paper, we are studying on authentication techniques that follow the popular filtering-and-verification framework. An authentication-friendly metric index called GMTree. Specifically, we transform the similarity search into a search in a graph metric space and derive small verification objects (VOs) to-be-transmitted to query clients. To further optimize GMTree, we are studying on a sampling-based pivot selection method and an authenticated version of MCS computation.
1. Introduction

Database research has been facing a new challenge raised by the emergence of massive, complex structural data, in the form of sequences, trees, and graphs. Among all the complex structured data, graph is the most sophisticated and general form of structure. Graphs have broad applications and have become the first-class citizens in widely used datasets bioinformatics. Similarity search is known to be an NP-hard problem. The owners of graph databases may lack the IT resources and expertise to provide efficient searches of their databases. For example, we issued a small query for a benzene structure to the prototype of a recent chemical database and the query took 7.8 minutes. Such a performance may not be ideal for many applications. Further, graph data is growing explosively in volume. It would be inefficient to process such a large amount of data with a commodity machine.

![Figure 1: Example of Outsourced graph database](image)

For the reasons mentioned above, graph database outsourcing is appealing to database owners. Specifically, voluminous data is delegated to a powerful third-party service provider (SP). The client may submit queries to the SP as if he/she is accessing a utility and the SP provides query processing services on the data owner’s behalf. Data outsourcing has been adopted in many industry sectors. For instance, in drug engineering, many commercial SPs support outsourcing of pharm databases. Drug laboratories may then focus on the duration of their data.

The majority of works on sub graph similarity search adopts a filtering-and-verification framework which consists of two key phases. First, in the filtering phase, indexes are proposed to prune (or filter) the data graphs that are certainly not the answer. The remaining graphs form a candidate set (a superset of answers). Second, in the verification phase, each candidate is checked by computing its distance from the query to verify if it is an answer. Despite the popularity of the framework, to the best of our knowledge, its authentication has not yet been studied and this paper takes the first step toward an authenticated framework for the search.

To facilitate the technical discussions, we briefly list the main steps of query authentication the data owner publishes its database, index and signature to an SP. The SP processes queries from a client and returns to the client both the query result and a verification object (VO) which often encodes query processing traces such as index traversals. Using the query result and the VO, the client constructs the digest of the database/index and compares it with the signature of the data owner to authenticate the query result. As the filtering-and-verification framework is not specially designed for query authentication, first, no previous index specifically considered whether the candidate graphs were located together in the graph database, which directly affects the VO needed. For instance, candidate and non-candidate graphs may be alternately stored in the database; and in this
scenario, each candidate graph needs an item in the VO to authenticate that no candidate has been missed. As the number of candidate graphs for similarity search can be large, the VO for authenticating them can also be large. Second, one performance bottleneck at the client side is the distance computation on large candidate graphs, since the distance computation time is exponential to the graph size. Unfortunately, most existing approaches index similarity search by features or sub graphs. The larger the graph is, the more features/sub graphs are there for indexing. Thus, large graphs are often included in candidate graphs. Third, clients are required to perform the costly sub graph similarity computation numerous times in order to authenticate the processing traces at the SP. Since such computation has already been done once at the SP, it is inefficient for the client to redo it from scratch.

2. Survey Review

i.) Horst Bunke propose a new graph distance measure that is based on the maximal common subgraph of two graphs. The main contribution of the paper is the formal proof that the new distance measure is a metric. An advantage of the new distance measure over graph edit distance is the fact that it does not depend on edit costs. It is well known that any edit distance measure critically depends on the costs of the underlying edit operations. But the problem how these edit costs are obtained is still unsolved. Using the new distance measure, this problem can be avoided.

ii.) Yuanyuan Zhu discovers the problem of finding top-k graphs in a graph database that are most similar to a query graph. This problem has many applications, such as image retrieval and chemical compound structure search. Regarding the similarity measure, feature based and kernel based similarity measures have been used in the literature. But such measures are rough and may lose the connectivity information among substructures.

iii.) Dennis Shasha present five distance-mapping algorithms and conduct experiments to compare their performance in data clustering applications. These include two algorithms called FastMap and MetricMap, and three hybrid heuristics that combine the two algorithms in different ways. Experimental results on both synthetic and RNA data show the superiority of the hybrid algorithms. The results imply that FastMap and MetricMap capture complementary information about distance metrics and therefore can be used together to great benefit. The net effect is that multi-day computations may be done in minutes.

iv.) Xifeng Yan investigates the issues of substructure similarity search using indexed features in graph databases. By transforming the edge relaxation ratio of a query graph into the maximum allowed missing features, our structural filtering algorithm, called Grafil, can filter many graphs without performing pairwise similarity computations. It is further shown that using either too few or too many features can result in poor filtering performance. Thus the challenge is to design an effective feature set selection strategy for filtering. By examining the effect of different feature selection mechanisms, we develop a multi-filter composition strategy, where each filter uses a distinct and complementary subset of the features.

3. The Frequent K-N-Match Problem

The k-n-match query can help us find out similar objects through partial similarity when an appropriate value of n is selected. However, it is not obvious how such a value of n can be determined. Instead of trying to find such a value of n directly, we will instead vary n within a certain range (say, 1 to d) and try to compute some statistics on the set of matches that are returned.
for each n. Specifically, we first find out the k-n-match answer sets for a range \([n_0, n_1]\) of n values. Then we choose the k points that appear most frequently in the k-n-match answer sets for all the n values. Henceforth, we will say that the similar points generated from the k-n-match problem are based on partial similarity (only one value of n) while those generated from the frequent k-n-match problem are based on full similarity (all possible values of n). We use an example to illustrate the intuition behind such a definition. Suppose we are looking for objects similar to an orange. The objects are all represented by its features including color (described by 1 attribute), shape (described by 2 attributes) and other characteristics. When we issue a k-1-match query, we may get a fire and a sun in the answer set. When we issue a k-2-match query, we may get volleyball and a sun in the answer set. The sun appears in both answer sets while none of the volleyball or the fire does, because the sun is more similar to the orange than the others, in both color and shape.

4. The AD Algorithm For Frequent K-N-Match Search

For frequent k-n-match search, the AD algorithm works in a similar fashion as for k-n-match search. The difference is that, instead of monitoring point ID’s that appear n times, we need to monitor point ID’s whose number of appearances are in the range \([n_0, n_1]\). The AD algorithm for frequent k-n-match search, namely “FK-NMatchAD”, is illustrated in Figure 6. Line 1 initializes some structures used in the algorithm. appear[ ], h[ ] and S[ ] have the same meanings as in algorithm K-NMatchAD except that h and S are arrays, each has d elements. After initialization, we locate the query’s attributes in each dimension and put the 2d attributes with smallest differences to the query into the array g[ ]. Next we retrieve (pid, pd, dif) triples from g[ ] in ascending order of dif and update h[ ] and S[ ] accordingly. We keep doing this until there are k points that have appeared n1 times. Before k points appear at least n1 times, they must have already appeared n0 times, ..., n1−1 times. Therefore, when algorithm FK-NMatchAD stops, that is, when it finds the kn1-match answer set, it must have found all the k-i-match answer sets, where i = n0, ..., n1. Then we simply need to scan the k-i-match answer sets for i = n0, ..., n1 to get the k points that appear most frequently. This shows the correctness of the algorithm. At the same time, we can see that algorithm FKNMatchAD retrieves the same number of attributes as if we are performing a k-n1-match search by algorithm KNMatchAD. Since we have to at least retrieve the attributes necessary for answering the k-n1-match query, and we only need to retrieve this many to answer the frequent k-n-match query.

Algorithm FK-N MatchAD
1 Initialize appear[ ], h[ ] and S[ ].
2 for every dimension i
3 Locate qi in dimension i.
4 Calculate the differences between qi and its closest attributes in dimension i along both directions. Form a triple (pid, pd, dif) for each direction. Put this triple to g[pd].
5 do
6 (pid, pd, dif) = smallest(g);
7 appear[pid]++;
8 if n0 ≤ appear[pid] ≤ n1
5. Framework

Given a graph database and a query graph, the substructure similarity search can be performed in the following four steps.

i.) **Index construction:** Select small structures as features in the graph database, and build the feature graph matrix between the features and the graphs in the database.

ii.) **Feature miss estimation:** Determine the indexed features belonging to the query graph, select a feature set (i.e., a subset of the features), calculate the number of selected features contained in the query graph and then compute the upper bound of feature misses if the query graph is relaxed with one edge deletion or relabeling. This upper bound is written as $d_{max}$. Some portion of the query graph can be specified as not to be altered, e.g., key functional structures.

iii.) **Query processing:** Use the feature-graph matrix to calculate the difference in the number of features between each graph $G$ in the database and query $Q$. If the difference is greater than $d_{max}$, discard graph $G$. The remaining graphs constitute a candidate answer set, written as $CQ$. We then calculate substructure similarity using the existing algorithms and prune the false positives in $CQ$.

iv.) **Query relaxation:** Relax the query further if the user needs more matches than those returned from the previous step; iterate Steps 2 to 4.

The feature-graph matrix in Step 1 is built beforehand and can be used by any query. The similarity search for a query graph takes place in Step 2 and Step 3. The filtering algorithm proposed should return the candidate answer set as small as possible since the cost of the accurate similarity computation is proportional to the size of the candidate set. Quite a lot of work has been done at calculating the pairwise substructure similarity.

In the step of feature miss estimation, we calculate the number of features in the query graph. One feature may have multiple occurrences in a graph; thus, we use the number of embedding of a feature as a more precise term. In this paper, these two terms are used interchangeably for convenience.
6. **Graph Metric Tree**

In this section, we propose an index called Graph Metric Tree, which forms the basis of our authentication algorithm. This section focuses on similarity search and the details for authentication

- **GMTree Structure** - GMTree is designed based on a variant of vp-tree, where a metric space is partitioned into a collection of non-overlapping “circular” subspaces with the same centre.

- **Authentication Algorithm** - To authenticate the query results, clients are required to rerun the traversal on GMTree and synthesize the digest of GMTree’s root from the VO. Specifically, clients decide if (i) the graphs in VO.M and VO.N are similar and dissimilar, respectively, to the query; (ii) no boundary index node overlaps with the query; and (iii) the synthesized digest hroot of GMTrees root agrees with the signature sroot provided by the DO.

- **Pivot Selection** - To optimize the performances of GMTree, a sampling approach to select pivots for a GMTree that produces a small candidate set. Existing works on pivot selection often adopt to minimize some cost function in a heuristic manner. In this study paper, we adopt the min max covering radius heuristic as it produces the fewest candidates as reported.

A unique property of our graph data is that they can be decomposed into subgraphs, whereas the data objects in the conventional metric space are assumed atomic. In our preliminary experiments, we observe that using subgraphs as pivots could produce smaller costs than using the data graphs. Therefore, in this paper, we are going to consider the subgraphs in our pivot selection.

7. **EFFECTIVENESS OF OPTIMIZATIONS ON GMTREE**

In this Review paper, we focus on the AIDS data sets, as other data sets exhibit similar performance characteristics, Pivot selection Method and Authenticated MCS computation. To clearly illustrate the performance of our authenticated MCS computation between q and pivots, the query time in this experiment does not include the time to handle candidate graphs, and the VO is just that caused by pivots.

8. **Conclusion**

This paper studies the authenticated sub graph similarity search in outsourced graph databases. It transforms the sub graph similarity search into a search in a graph metric space and propose GMTree. Our novelties and technicalities reside in the authentication techniques. First, candidate graphs determined by GMTree are often localized. Second, we propose a pivot selection which allows using data sub graphs as pivots. Third, we studied an authenticated MCS computation to reduce the computation at clients.

9. **References**


Author’s Biography

Prof. S. R. Durugkar is presently working as Assistant Professors & Head, Department of Computer engineering SNDCOE&RC-Maharashtra. He has attended & presented the papers in several Workshops & Seminars. He has published papers in various National / International Journals. His research interest includes Computer engineering concepts, DS, LD, SE, CO, DBMS, PCD,OR, simulation, HPDB, OSD, MFIT.