A Survey on Subgraph Matching Algorithm for Graph Database

Maninder Kaur Rajput[1] Dr. Prof. Snehal Kamalapur[2]
[1]P.G. Student [2]Assistant Professor
[1,2]Department of Computer Engineering
[1,2]K.K.W.I.E.E.R., Nashik, Maharashtra, India

Abstract— Subgraph matching is a technique to retrieve subgraphs which are isomorphic to query graph. A graph S (V_S, E_S) is subgraph of graph G (V_G, E_G) if $V_S \subseteq V_G$ and $E_S \subseteq E_G$. Subgraph matching is a process to fetch all subgraphs S (V_S, E_S) from graph G (V_G, E_G) which are structurally isomorphic to input graph I(V_I, E_I) using subgraph matching algorithm. Subgraph matching is a NP-hard. In real-world graphs such as semantic web, social networks, protein interaction and biological networks, subgraph matching will retrieve subgraphs which are isomorphic to query graph from data graph. Many subgraph matching algorithms have proposed in recent years, these algorithms aims to find desired results in less time for real datasets using information like join orders, pruning techniques and vertex neighbour information. Subgraph matching algorithms can be classified into two clases: exact and approximate matching algorithms. These two approaches are being described in survey.

Key words: Graph Database, Isomorphism, Offline Phase, Online Phase, Subgraph Matching

I. INTRODUCTION

Graph is an attractive tool to represent and model a data since it allows simple and flexible representation of complex objects. Day by day increasing data in graphs requires new techniques to extract results for graph queries in shorter time. Real world graphs are very large in size i.e., having millions number of nodes and edges. Web graphs, bio informatics, protein interaction, social networks are some examples. Subgraph matching is a technique to retrieve subgraphs which are isomorphic to query graph. Subgraph matching is also called subgraph isomorphism.

A. Subgraph

A graph[1] H = (v,e), is said to be a subgraph of graph G = (V,E) if all the vertices and all the edges of H are in G, and each edge of H has the same end vertices in H as in G.

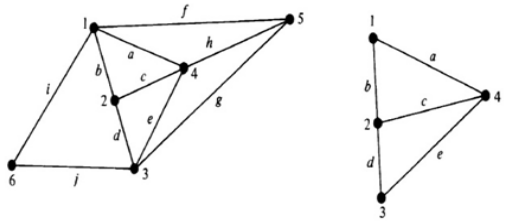

Fig. 1: A graph and its subgraph.

B. Isomorphism

Two graphs[1] G and H are said to be isomorphic (G ≈ H) if there exist a bijective function f, for vertex set of G and H such that,

$$f : V(G) \rightarrow V(H).$$

If u and v are two vertices of G and H respectively, then G and H are isomorphic iff f(u)f(v) ϵ E(H). 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.

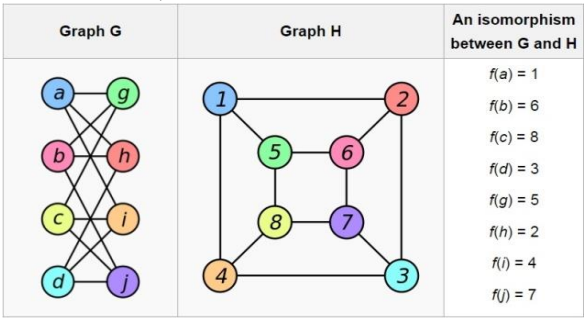

Fig. 2: Graph Isomorphism

In fig.2 vertex (a) of graph G has matching vertex (1) in graph H, represented by f(a) = 1 and vertex (b) has matching vertex (6) in graph H, represented by f(b) = 6, similarly rest of the matching's are shown in figure 2.

Given a query graph Q and a data graph G, subgraph matching algorithm will retrieve all those subgraphs from G, which are isomorphic to Q. Subgraph matching approaches are generally classified into two categories:

- Exact subgraph matching approaches
- Approximate subgraph matching approaches.

Exact subgraph matching approaches aims to find out if an exact mapping between the vertices and the edges of the compared query graphs or data graphs is possible. Approximate subgraph matching approaches aims to compute a distance between vertices of graphs by converting vertices into points in vector space using embedding techniques. Approximate subgraph matching approaches converts pattern match queries into distance based queries. Approximate matching is useful for rank based applications where the distance between the objects to be compared is needed. Several subgraph matching approaches have been proposed in the literature. The aim of this paper is to provide a survey of recent and current subgraph matching approaches on large graphs.

Work describe in detail the existing approaches and can categorize them into two classes i.e., exact and approximate subgraph matching approaches. The advantages, disadvantages and the differences between these approaches are also highlighted here.

II. PROBLEM DEFINITION

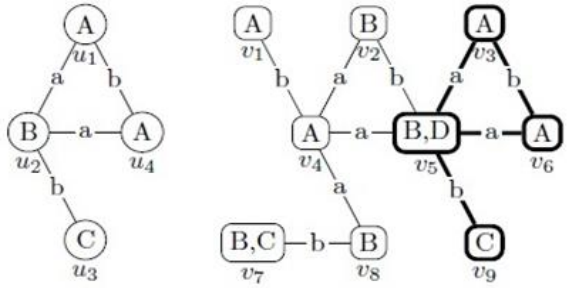

Fig. 3(a) query graph Fig. 3(b): data graph Fig. 3(c) solid line (resulting isomorphic subgraph).

Subgraph matching is a process of extracting subgraphs from large graph database which are isomorphic to input graph. For a given query graph $G_q(u,e)$ and a data graph $G_d(V,E)$ a subgraph matching algorithm will find all isomorphic occurrences of G_q in G_d.

Where matching of query graph in data graph are as follow:

Matching 1 (m1) = {(u_1, v_3), (u_2, v_5), (u_3, v_9), (u_4, v_6)}

Matching 2 (m2) = {(u_1, v_6), (u_2, v_5), (u_3, v_9), (u_4, v_3)}

Which means u_1 vertex of query graph has similar vertex v_3 of data graph and similarly rest of query graph vertices matching's to vertices of data graph have been listed.

III. Literature Survey

Many subgraph matching algorithms have been introduced in recent years. These subgraph matching algorithms can be classified into two classes i.e. exact subgraph matching and approximate subgraph matching.

Exact subgraph matching can be further divided into two subclasses based on their manipulation of indexing techniques. First subclass algorithms are based on filter-and-refine strategy in which filtering manipulates graph index to minimize number of candidate graphs and refine step examines whether there exists an isomorphic subgraph for each candidate. Grep, gIndex, gCode, C-Tree, FG-Index are few examples of this subclass. Second subclass algorithms detect all possible sets for query graph and data graph. Ullmann[2], GADDI[6], VF2[3], QuickSI[4] are few examples of this subclass.

Approximate subgraph matching algorithms searches for approximate sets by using their own similarity techniques. TALE, Ness and SIGMA are few examples of this class.

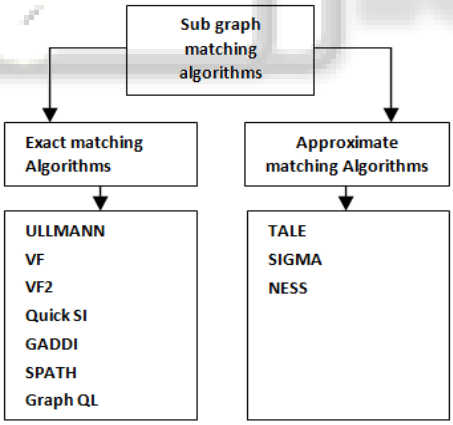

Fig. 4: Subgraph matching algorithms classification

A. Ullmann Algorithm

Ullmann algorithm[2] is a backtracking algorithm. It detects subgraph isomorphism using brute force tree search and enumeration procedure. Enumeration algorithm was designed to generate an adjacency matrix of input graph H and large graph G and then using this adjacency matrix, isomorphism was detected. Formula for isomorphism is:

$$AH = M'(AG)M'^{-1} = AG$$

Where AH is adjacency matrix of input graph H, AG is adjacency matrix of data graph G, M' is adjacency matrix of mappings of input graph vertices to vertices of data graph or a permutation matrix and M'^{-1} is its inverse. Its refinement phase was used to prune undesired matches from adjacency matrix of possible future matched node pairs. Refinement greatly reduces search space by pruning all candidate vertices having degree smaller than the query vertex.

Memory requirement for Ullmann algorithm is O (N^3) for N number of nodes, which is very high. This memory requirement has been reduced to O (N) in VF2 algorithm. It was more efficient for graph isomorphism than a subgraph isomorphism and it blindly iterates along all adjacent vertices of query vertex while searching its matches in data graph. This algorithm works efficiently for smaller graphs i.e. the maximum size of data graph can be 700 nodes only and it is not able to find isomorphism for graph size beyond this limit. This algorithm do not uses any index by pre-processing the dataset graphs. Fig.5 shows improved versions of Ullmann algorithm which overcomes its limitations.

B. VF2 Algorithm

In [3] has introduced a matching algorithm for both graph isomorphism and subgraph isomorphism. It was a deterministic matching algorithm. It started with first vertex, selects sequential vertices, search for a subgraph match, and backtracks if not. It has used state space representation for matching process. For graphs G1 and G2, a query graph and data graph respectively, a state *s* of the matching process can be co-related to a partial mapping solution M(s), where M(s) have isomorphic subgraphs from G1 and G2 and each vertex of G1 in M(s) is mapped to a vertex of G2 in M(s). A set of feasibility rules were introduced to insure consistency of partial solutions and for pruning a search tree.

It is an improved version of VF algorithm which was proposed by the same author. It has reduced the memory requirement to O(N)[VF2] from $O(N^2)$[VF] for N nodes. It's been observed that for graphs of two hundred nodes or more VF2 algorithm is one thousand times faster than VF and Ullmann algorithm.

Here are some limitations for above proposed algorithm, search in this method is not based on index, so it is costly as compared to the new methods. It was designed for graphs with thousands of nodes only.

C. QuickSI Algorithm

Haichuan Shang Ying Zhang Xuemin Lin et al. [4] introduced an efficient algorithm for subgraph isomorphism. It computed the frequencies of labels of vertices of data graph in advance that is before the search procedure started. These labels, source vertex, edge labels and target vertex labels are combined called frequencies of triple. B+ tree were used for storing frequencies of all vertex labels. Weight of each query vertex was computed and assigned using pre-computed edge label frequencies and a minimum spanning tree was formed using modified Prim's algorithm. As algorithm selected its first edge (u_1, u_2), then u_1 acted as a parent to u_2 in minimum spanning tree and frequency of u_1 should be more than frequency of u_2 to act as its parent. It iterated only along adjacent and previously matched query vertices.

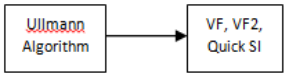

Fig. 5: Subgraph Matching Algorithm Comparison

D. TALE (Tool for Subgraph Matching)

Yuanyuan Tian et al. [5] proposed a tool [TALE] for approximate matching in large graphs. It is based on assumption that approximate matching could generate more and nearby results to the input query as compared to exact matching. For approximate matching some similarity measures must be needed. It was index based search and its index size scales linearly to its database size. The index was hybrid in nature that support efficient search for matching database neighbourhoods. It distinguished nodes on basis of their relative importance (importance here was decided on basis of degree centrality of node in database) in the structure of graph. Firstly it matched only important nodes of input graph and then these nodes lead the remaining matching procedure by considering adjacent nodes of previously matched nodes in second step.

It has better effectiveness as compared to Gramelin. Its index size scales linearly to dataset size and index construction time grows steadily.

E. GADDI Algorithm

Shijie Zhang et al. [6] put forward a technique which relied on Neighbouring Discriminating Structure distance. It indexes neighbourhood discriminating structure distance, between pairs of neighbouring vertices of data graph. NDS is a distance between two vertices v_1 and v_2 of a data graph using a subgraph P, denoted by $\Delta_{NDS}(v_1, v_2)$. It first selects a query vertex appeared first in input graph and then performs DFS (Depth First Search) to find next query vertex for comparison. To refine candidate set for query vertices, vertices of data graph were pruned on basis of NDS. Binary search was used to locate distances between any given pair of vertices.

But it works efficiently only for biological networks. For 30mb of data time taken by algorithms TALE, GADDI and SAPPER[7] to construct index is listed in table 1:

Algorithm	Data Size	Time Taken(min.)
TALE	30 MB	20
GADDI	30 MB	10.5
SAPPER	30 MB	12.5

Table 1: Index construction time

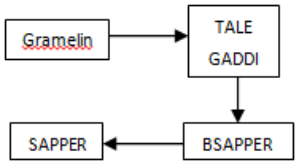

Fig. 6: Subgraph Matching Algorithm Comparison

In fig.6 arrows points to improved versions of previous subgraph matching algorithms.

F. Join Algorithm

Lei Zou et al. [8] proposed a pattern match query in large graph database based on distance and joins. It is a join algorithm. Firstly vertices were transformed into points in vector space using LLR embedding because it is cheap to calculate distance between two vertices then to find nearest vertex of any existing vertex. These embeddings are able to filter out 90% of search space. Index was built by clustering blocks into a flat file. A cost model was also proposed to guide a join order selection that is this model generated a cheap input query from the query entered by a user. During each comparison, an edge query was performed for a newly introduced edge to find its match in data graph. Finally by choosing appropriate set from candidate set, answer set was generated.

It has used costly join operations.

G. SPath Algorithm

Peixiang Zhao et al. [9] introduced a new graph indexing technique. Graph matching was performed in a manner, searching for a query path rather than searching for a query vertex. It was a first method to search in this fashion. It was an index based search. Its index was built on basis of neighbourhood signature, which was a compact indexing structure comprised nearest two hop neighbours information for each vertex. Its main aim was to reduce N, where N is number of vertices of query graph. An algorithm GraphQL [10] was introduced earlier to SPath algorithm and SPath has better performance as compared to GraphQL. Both performs neighbourhood signature based pruning before starting actual subgraph matching procedure. There is a difference in indexing technique of these two algorithms i.e. GraphQL indexes nodes of data graph while SPath indexes nodes of datagraph using their neighbour information.

As the neighbourhood scope increases, performance of Spath algorithm decreases and filtering time increases. It has used costly join operations. Though SPath has better performance but its average cost for recursive calls is more than GraphQL.

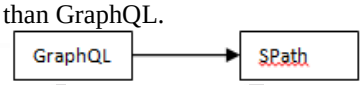

Fig. 7: Subgraph Matching Algorithm Comparison

H. Subgraph Matching using Trinity Cloud Memory

Zhao Sun et al. [11] put forward a subgraph matching technique based on assumption that if data resides on RAM the searching operation performs better as compared to the case where data resides on hard disk, so they stored whole data on trinity memory cloud. Trinity is a memory cloud whose RAM is equal to hundreds of machines. Firstly a query graph was decomposed into small graphs using STwig algorithm. Then binding information for each vertex of decomposed graph was found using index. Then a match for each decomposed graph was found from data graph and later all matching's were merged using pipeline joins to get final result.

I. Subgraph Matching in Parallel Manner

Yingxia Shao et al. [12] proposed a parallel subgraph matching technique to find all occurrences of query graph in data graph. It followed divide and conquer and parallel approaches to solve subgraph matching problem. It decomposed a query graph into small graphs in its first step and then finds similar small graphs from data graph and later using expensive join operations it merged all results to form final result. Three independent mechanisms were introduced to reduce intermediate results: automorphism breaking of the pattern graph, initial pattern vertex selection based on a cost model, and a pruning method based on a light-weight index. A cost model was used to generate an initial pattern vertex.

Several optimization techniques have been introduced for reducing size of intermediate results but still expensive join operations degrades its performance.

Yingxia Shao et al. [13] has introduced one more method for parallel subgraphs detection. It was an improved version of L.Quick's algorithm. It covered various limitations of previously existing method that is: First, the algorithm still enlarges the synchronized frequency by three times. Second, the repeated triangle counting brings in massive redundant computations and communications. For example, the triangles reserved in the final found k-truss will be repeatedly computed as much as the number of iterations. Third, the restriction of the vertex-centric model in Pregel only supports to implement the classic solution for the fundamental operation. In improved version it first constructed a triangle complete subgraph for every node to be computed. Then it finds local k number of triangles in parallel fashion for query node.

IV. CONCLUSION

Work here aims to study and compare the existing subgraph matching algorithms. It's been observed that query time increases as data graph size increases. All algorithms have worked to reduce size of intermediate results using various pruning techniques and small query processing time. There is drastic difference in performance of index based search algorithms as compared to non-index based search algorithms. Last but not least is if the nodes of query graph can be reduced then it affects performance of subgraph matching algorithm greatly.

REFERENCES

[1] Nar Singh Deo, "Graph theory with applications to engineering and computer science".
[2] J. R. Ullmann, "An algorithm for subgraph isomorphism," J. ACM, vol. 23, no. 1, pp. 31–42, 1976.
[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 10, pp. 1367–1372, Oct. 2004.
[4] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: An efficient algorithm for testing subgraph isomorphism," Proc. VLDB Endowment, vol. 1, no. 1, pp. 364–375, 2008.
[5] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in Proc. 24th Int. Conf. Data Eng., 2008, pp. 963–972.
[6] S. Zhang, S. Li, and J. Yang, "Gaddi: Distance index based subgraph matching in biological networks," in Proc. 12th Int. Conf. Extending Database Technol.: Adv. Database Technol., 2009, pp. 192–203.
[7] Shijie Zhang, Jiong Yang, Wei Jin, "SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs", Proceedings of the VLDB Endowment, Vol. 3, No. 1 in September 13-17, 2010, Singapore.
[8] L. Zou, L. Chen, and M. T. Ozsu, "Distance-join: Pattern match query in a large graph database," Proc. VLDB Endowment, vol. 2, no. 1, pp. 886–897, 2009.
[9] P. Zhao and J. Han, "On graph query optimization in large networks," Proc. VLDB Endowment, vol. 3, nos. 1/2, pp. 340–351, 2010.
[10] H. He and A. K. Singh, "Graphs-at-a-time: Query language and access methods for graph databases," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2008, pp. 405–418.
[11] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," Proc. VLDB Endowment, vol. 5, no. 9, pp. 788–799, 2012.
[12] Y. Shao, B. Cui, L. Chen, L. Ma, J. Yao, and N. Xu, "Parallel subgraph listing in a large-scale graph," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2014, pp. 625–636.
[13] Y. Shao, L. Chen, and B. Cui, "Efficient cohesive subgraphs detection in parallel," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2014, pp. 613–624.

