International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 4, Jul. - Aug. 2013 pp-2356-2359 ISSN: 2249-6645

A Subgraph Pattern Search over Graph Databases

Zareen Nikhat[1], Ch. N.Santhosh Kumar[2], V. Sitha Ramulu[3]
[1]*M. Tech, Swarna Bharathi Institute of Science & Technology, Khammam, A.P., India.*
[2]*Assoc. Professor, Dept.of CSE, Swarna Bharathi Institute of Science & Technology, Khammam, A.P., India.*
[3]*Assoc. Professor, Dept.of IT, Swarna Bharathi Institute of Science & Technology, Khammam, A.P., India.*

***ABSTRACT*:** *Graphs has been used in various real applications such as social network modeling and chemical compound analysis. Due to their wide usages, many interesting graph problems are extensively studied, for example, sub graph search, graph reachability, and keyword search in graphs. Given an example, during a chemical reaction, the structures of the chemical compounds often change along the reaction process. We can model these evolving graphs as graph streams, that is, a sequence of graphs which grow indefinitely over time . However, most of the previous works assume that graph data are rather static, which raises challenges when applying to the graph streams. Compared to the static graphs, graph streams not only inherit the complexity of graphs but also possess their own characteristics. In this paper, we study the problem of continuous sub graph pattern search over graph databases, which can be used in many real applications.*

KEYWORDS: *Graph, NNT, Stream, Trees.*

I. INTRODUCTION

Graph pattern matching is a routine process in a variety of applications, e.g., knowledge discovery, computer vision, biology, chem- informatics, dynamic network traffic, intelligence analysis and social networks. It is often defined in terms of sub- graph isomorphism [1], graph simulation [2] or bounded simulation [3]. Given a pattern graph GP and a data graph G, the graph pattern matching is to find the set M(GP, G) of matches in G for GP . For sub graph isomorphism, M(GP,G) is the set of all the sub graphs of G that are isomorphic to the pattern GP . For bounded simulation, M(GP, G) consists of a unique maximum match, a relation defining edge-to-edge (edge-to-path) mappings. Graph pattern matching is very costly: NP-complete for subgraph isomorphism [4], cubic-time for bounded simulation [3], and quadratic-time for simulation [5]. In practice, a data graph G is typically very large, and moreover, is frequently updated. This is particularly evident in, e.g., social networks [6], Web graphs [7] and also traffic networks [8]. It is often prohibitively expensive to re compute the matches starting from the scratch when G is updated. These highlight the need for incremental algorithms to compute the matches.

Given a pattern graph GP , a data graph G, that matches M(GP, G) in G for GP and changes ΔG to G, the incremental matching problem is to compute changes ΔM to the matches such that $M(GP,G \oplus \Delta G) = M(GP,G) \oplus \Delta M$, where (a) ΔG consists of a set of edges to be inserted into or deleted from G, and (b) operator $\oplus$ applies changes ΔS to S, where S is a data graph G or matching results M. As opposed to batch algorithms that re compute the new output from the scratch, an incremental matching algorithm aims to minimize unnecessary recomputation and improve response time. Indeed, when the changes ΔG to G are very small, the increment ΔM to the matches is often small as well, and is much less costly to find than recompute the entire $M(GP, G \oplus \Delta G)$. While real life graphs are constantly updated, the changes are typically minor; for example, only 5% to 10% of the nodes are updated weekly in a Web graph [7]. We can cope with the dynamic nature of the social networks and Web graphs by computing matches once on the entire graph via a batch algorithm, and then incrementally identifying their changes in response to updates. That is, we find new matches by making maximal use of the previous computation, without paying the price of the high complexity of graph pattern matching.

II. RELATED WORK

A lot of interesting works have been done to address the sub graph search problem. In [9], the authors proposed a closure-tree (C-tree) to organize graphs into a tree-based multi-dimensional index and used the graph closures as bounding boxes. The C-tree can support both exact sub graph queries and the similarity-based sub graph queries. In [10], the authors decomposed a graph into a full set of sub graphs and indexed the hash value of canonical forms of the sub graphs.

In [11], the authors proposed GCoding for graph search, which assigns a signature to each vertex based on its local structures. Then, they produced a spectral graph code by combining all vertex signatures in the graph. Based on the spectral graph codes, a necessary condition for sub graph isomorphism was derived. In [12], the authors proposed gIndex which uses frequent sub graphs as filtering features. Because of anti- monotonicity, once a sub graph pattern is not frequent, any super graph that contains it will not be frequent as well. In [13], the authors used frequent sub graphs as indexing features and constructed a nested inverted-index, thus, a frequent graph query could be answered directly. Only an infrequent graph query needs to be verified for the sub graph isomorphism. In [14], the authors proposed an improved subgraph isomorphism checking method using tree features. They also integrated indexing with the sub graph searching. Thus, not only the sub graph isomorphism verification time was reduced as well.

III. PROPOSED WORK

In this paper, we focus on answering continuous sub graph patterns over graph databases. More specifically, we assume a user has a set of sub graph patterns and starts monitoring graph streams from timestamp zero. Then, as time

evolves, the user wants the system to continuously report the appearances of certain sub graph patterns on the graph streams at each and every timestamp.

A. Node- neighbour tree: Node-neighbor tree or NNT, which captures the local structure around each vertex. An example graph *G* together with the NNTs of all its vertices and edges under *l* = 2 is shown in Figure 1(a). In the example, *G* has 4 vertices with ids from 1 to 4, which have labels *A,A,B,C* respectively. The NNTs of vertices 1 and 2 have the same structure, thus, we use only one tree to represent *T*1 and *T*2 in this example. *T*3 rooted at vertex 3 has only two branches consisting of the same labels *A*, which indicates that node 3 has two distinct neighbors with label *A*. *T*4 has two different branches rooted at the node with label *B*. In node-neighbor trees, each node is identified by the lower case character in the figure. The numbers in the brackets are referring to the node IDs in the original graph "*G"*.

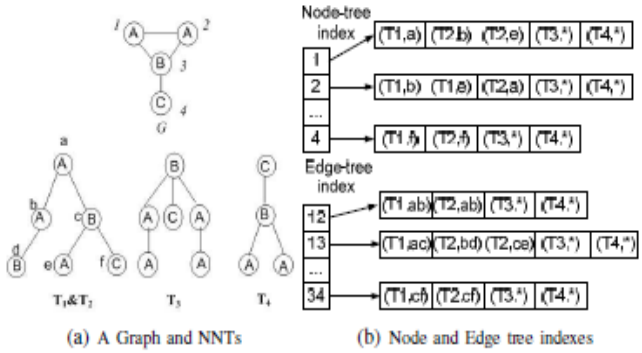

Figure 1: Graph, Node- neighbour trees and index

For example, in *T*1, the node *a* is referring to node 1 in *G* and then nodes *b*,*e* are all referring to 2 in *G*. Figure 1(b) shows an example of node-tree index for the NNTs in Figure 1(a). In the example index, node 2 appears in positions *b* and *e* of tree *T*1 rooted at node 1, thus in the entry 2 of the node tree index, (*T*1, *b*) and (*T*1, *e*) are stored.

B. Projecting to Numerical Vectors: In this section, we propose a novel encoding method to transform a NNT to a set of vectors and approximate sub- tree isomorphism checking by dominant relationship verification between two vector sets. Figure 2 shows an example of dimension derived from the query graph Q (upper left). We show that the NNTs of vertices l = 2. Thus, there are fourteen possible dimensions from the NNTs, < 1,A,A >, < 2,A,A >, . . ., < 2, C,B >. Based on these dimensions, we can apply Procedure 1 to project a NNT into a node projected vector (NPV).

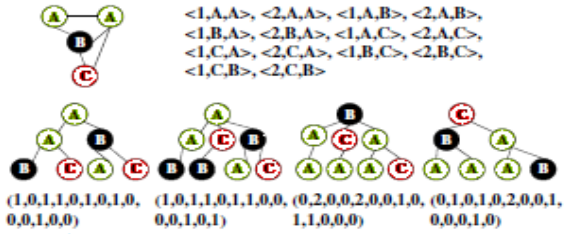

Figure 2: Projecting NNTs to Numerical Vectors

Procedure 1: Tree Projection
{**Input:** A NNT tree up to depth of l
Output: Node projected set
(1) NPV ← 0
(2) for each level li in Input
(3) for each edge (u, u^1) at level li
(4) NPV [(li, label(u), label(u^1))] ← NPV [(li, label(u), label(u^1))] + 1
(5) return NPV}

C. Search in the Vector Space: After projecting NNTs to their NPVs, we can check every possible joinable pair of streams and the query graphs based on the dominant relationship of NPVs using a nested loop algorithm. We set this nested loop algorithm as the baseline and propose two improved search strategies: the first improved method utilizes the idea of checking the dominated vector set as a whole instead of checking the whole dominant relationship pair by pair. When the stream graph of the next timestamp comes, for each dimension, we only need to update the number of dominated vectors of *Q*, when the position of the projected node vector in *G* changes. The detailed steps are listed in Procedure 2. Second, for each

vector of the stream graph, we maintain two counter vectors for it, namely, the position counter vector and the dominant counter vector.

Procedure 2: Dominated Set Cover Join
{**Input:** stream graphs {G1, ...Gk1}, query graphs {Q1, ...Qk2}
Output: Reported positive pairs
(1) for i ← 1 to k1
(2) for u ∈ Gi
(3) for each non-zero dimension of NPV (u) {
(4) update u's position counter NPV (u)pos
(5) update u's dominant counter NPV (u)dom
(6) mark query vectors dominated by Gi based on
NPV(u)dom }
(7) for j ← 1 to k2
(8) if Gi dominates all vectors in Qj
(9) answer ← answer U (i, j)
(10) return answer

D. Search in Uncertain Graph Streams: For sub- graph pattern search over uncertain graph streams, besides using the structural features, we want to utilize the probabilities to reduce the search space. Specifically, other than removing the stream graph G^1 that does not contain the query graph Q, that is, the probability of G^1 containing Q is zero, we also want to filter out a stream graph that contains Q but with a probability less than $\in$. Thus, for sub- graph search over uncertain graph streams, we would like to conduct the pruning in two steps: For structural pruning, we can utilize procedure 2. Compared to certain graph streams, for uncertain graph streams, we have to make some modifications for projecting the node- neighbour trees (NNTs) to numerical vectors, since each of the NNT of an uncertain graph has a probability associated with each of its edges. We call the converted vectors from NNTS of an uncertain graph called as Probability Node Projected Vectors (PNPV). The basic idea of the probability pruning is to derive an upper bound for the probability of a stream uncertain graph G^1 containing the query graph Q, called matching probability upper bound, denoted as MPbound. Based on the global mapping probability upper bounds, we have:
MPbound=max{min{Plocal-mapping(NPV (u), PNPV (v))}

From the above equation, we can observe that we need to compute local mapping probability upper bound before we derive MPbound. Therefore, we listed detailed steps on computing this local mapping bound in Procedure 3.

Procedure 3: Local Mapping Bound
{**Input:** Query vector NPV (u), PNPV (v) of the uncertain graph node
Output: Probability bound
(1) check if PNPV (v) dominates NPV (u)
(2) if no, return 0
(3) bound = 1
(4) for each dimension i in NPV (u)
(5) k ← value of i dimension in NPV (u)
(6) l ← kth largest probability of the probability array pointed by p of i dimension in PNPV (v)
(7) if l < bound bound = l
(8) return bound}

Now we propose one exact solution and one approximate solution. The exact solution will derive the exact MPbound(procedure 4). The approximate solution will get an approximate solution(procedure 5), whose value is less than MPbound, but much faster than the exact solution.

Procedure 4: Exact MP Bound
{**Input:** weighted bipartite graph G
Output: MPbound
(1) low = 0, high = 1
(2) while low + δ < high
(3) mid = (low + high)/2
(4) remove from G all edges having weights less than mid
(5) use Hopcroft-Karp algorithm to check if there is a maximum matching
with regard to the number of nodes in query graph
(6) if yes, low = mid, otherwise high = mid
(7) return low}

Procedure 5: Approximate MP Bound
{**Input:** weighted bipartite graph G
Output: Approximate MPbound
(1) bound = 1
(2) for each node u in query graph
(3) find the edge incident to u with maximum probability p
(4) if p < bound bound = p
(5) return bound}
Finally, the overall query procedure for the sub graph search over uncertain graph stream is presented in Procedure 6.

Procedure 6: Uncertain Join
{**Input:** uncertain stream graphs {G1, ...Gk1}, query graphs {Q1, ...Qk2}, and a probability threshold ε
Output: Joinable query-stream pairs
(1) answer ← ϕ , answer2 ← ϕ
(2) conduct Skyline with Early Stop or
Dominated Set Cover algorithm to obtain structural
filtering results to answer
(5) for each query-stream pair (Qj,Gi) $\in$ answer
(6) bipartite graph G ← ϕ
(7) for each node u in Qj
(8) for each node v in Gi
(9) call Procedure Local Mapping Bound and store return value in p
(10) if p is non-zero, add (u, v) with weight p to G
(11) call Procedure Exact MP Bound or Procedure Approximate MP Bound
with parameter G to derive MPbound
(12) if MPbound $\geq$ ε , then answer2 ← answer2 U (i, j)
(13) return answer2}

IV. CONCLUSION

In this paper, we propose a continuous sub graph patterns over graph databases. We introduce a light-weight yet effective feature structure called Node-Neighbor Tree to filter out false candidate query-stream pairs. Later we propose a novel encoding method to transform a NNT to a set of vectors and approximate sub- tree isomorphism checking by dominant relationship verification between two vector sets. After projecting NNTs to their NPVs, we can check every possible joinable pair of streams and the query graphs based on the dominant relationship of NPVs using a nested loop algorithm. After projecting NNTs to their NPVs, we can check every possible joinable pair of streams and the query graphs based on the dominant relationship of NPVs using a nested loop algorithm. for sub- graph search over uncertain graph streams, we would like to conduct the pruning in two steps: structural and probability pruning. This reduces search space for capturing patterns over uncertain graph streams.

REFERENCES

[1]. Stotz, R. Nagi, and M. Sudit. Incremental graph matching for situation awareness. FUSION, 2009.
[2]. S. Abiteboul, P. Buneman, and D. Suciu. Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufman, 2000.
[3]. W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractability to polynomial time. In PVLDB, 2010.
[4]. M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, 1979.
[5]. M. R. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In FOCS, 1995.
[6]. S. Garg, T. Gupta, N. Carlsson, and A. Mahanti. Evolution of an online social aggregation network: An empirical study. In IMC, 2009.
[7]. Ntoulas, J. Cho, and C. Olston. What's new on the Web? The evolution of the Web from a search engine perspective. In WWW, 2004.
[8]. Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu. Monitoring path nearest neighbor in road networks. In SIGMOD, 2009.
[9]. H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," Proc. 22nd Int'l Conf. Data Eng. (ICDE), 2006.
[10]. D. W. Williams, J. Huan, and W. Wang, "Graph database indexing using structured graph decomposition," Proc. 23rd Int'l Conf. Data Eng. (ICDE), 2007.
[11]. L. Zou, L. Chen, J. X. Yu, and Y. Lu, "A novel spectral coding in a large graph database," Int. Conf. on Extending Database Technology (EDBT), 2008.
[12]. X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structurebased
[13]. approach," Proc. ACM SIGMOD, 2004.
[14]. J. Cheng, Y. Ke, W. Ng, and A. Lu, "FG-index: towards verification-free query processing on graph databases," Proc. ACM SIGMOD, 2007.
[15]. H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," Proc. 34th Int'l Conf. Very Large Data Bases (VLDB), 2008.