An Improved Method of Subgraphs Extraction in a Large Graph Database in a Distributed System by considering both
Graph Topology and Set Similarity

Ritu Yadav, Samarth Varshney
Computer Science and Engineering, National Institute of Technology, Warangal, Telangana, India

DOI: 10.23956/ijarcsse/V716/0233

Abstract—Since many real applications such as web connectivity, social networks, and so on, are emerging now-a-days, thus graph databases have been widely used as important tools to model and query complex graph data wherein each vertex(node) in a graph usually contains information, which can be modelled by a set of tokens or elements. The method for subgraphs extraction by considering set similarity query over a large graph database has already been proposed, which retrieves subgraphs that are structurally isomorphic to the query graph, and meanwhile satisfy the condition of vertex pair matching with the (dynamic/fixed) weighted set similarity in a centralized system. This paper explains the efficient implementation of subgraphs extraction in a large graph database in a distributed environment by considering both vertex set similarity and graph topology which offers a better price/performance ratio and increases availability using redundancy when parts of a system fail than centralized systems in case of a large dataset (i.e., a graph with millions/billions of nodes wherein each node contains some information) by performing parallel processing.

Keywords—centralized systems, distributed systems, graph databases, graph topology, parallel processing, set similarity, subgraphs extraction

I. INTRODUCTION

With the increasing growth in the generation of information now-a-days from a wide range of sources, graphs have been the simplest and the compact way to represent the complex as well as simple information. Thus, graph databases have been widely used as important tools to model and query complex graph data wherein each vertex in a graph usually contains information, which can be modelled by a set of tokens or elements. The method for subgraphs extraction by considering both graph topology and set similarity over a large graph database retrieves subgraphs of the same topology (structurally isomorphic) as that of the query graph, and simultaneously satisfy the condition of vertex pair matching with the (dynamic/fixed) weighted set similarity. In order to achieve better query performance, all the techniques are designed for the distributed systems using Apache Spark since distributed systems possess many advantages over the centralized ones:

- distributed systems offer a better price/performance ratio than centralized systems
- redundancy increases availability when parts of a system fail
- applications that can easily be run simultaneously also offer benefits in terms of faster performance vis-à-vis centralized solutions
- distributed systems can be extended through the addition of components, thereby providing better scalability compared to centralized systems.

A number of solutions were formulated and proposed for subgraphs extraction over a large graph database [1], [2], [3], [4],[6]. Suppose a query graph Q and a large graph G is given, then a typical subgraph matching query retrieves those subgraphs in G that exactly match with Q in terms of both graph structure and vertex labels[4]. However, in some real graph applications, each vertex often contains a rich set of tokens or elements representing features of the vertex, and the exact matching of vertex labels is sometimes not feasible or practical.

The motivation examples show that subgraphs extraction by considering both graph topology and set similarity queries are very useful in many real-world applications. To the best of our knowledge, no prior work studied the subgraphs extraction problem under the semantic of structural isomorphism and set similarity with dynamic element weights (called dynamic weighted set similarity). However a traditional weighted set similarity [11] that focuses on fixed element weight has already been proposed which is actually a special case of dynamic weighted set similarity.

For the methods considering exact subgraph matching query for subgraphs extraction requires that all the vertices and edges are matched exactly. The Ullmann’s subgraph isomorphism method [8] and VF2 [7] algorithm are costly for large graphs since they do not utilize any index structure. A distance-based multi-way join algorithm has also been proposed by Zou et al. [3] for answering pattern match queries over a large graph.

Each of the various methods formulated for subgraphs extraction[9],[10] are inefficient in one or the other way when the query is performed over a large graph database. Thus, we proposed the method for subgraphs extraction in a large graph database in a distributed system by taking into account both one-to-one structural isomorphism and dynamic set similarity of matching vertices.
Section 2 states the problem description in brief followed by section 3 in which proposed system design is explained, including the basic framework and all design modules and their constraints including the flow diagram in distributed environment. Section 4 presents the implementation details of all modules, results, analysis and comparison of the results with the existing method. Section 5 concludes the paper.

II. PROBLEM DESCRIPTION

Given a large graph $G$ represented by $(V(G),E(G))$, where $V(G)$ is a set of vertices, $E(G)$ is a set of edges is considered. Each vertex $v \in V(G)$ is associated with a set, $S(v)$, of elements. Query graph $Q$ is represented by $(V(Q),E(Q))$. The set of element domain is denoted by $U$, in which each element $a$ has a weight $W(a)$ to indicate the importance of ‘$a$’. Weights can change dynamically in different queries due to varying requirements or evolving data in real applications.

Given:
- A Query Graph, $Q$, with $n$ vertices ($u_1,......,u_n$)
- A Data Graph $G$
- A user-specified similarity threshold $\tau$

To Find: A Subgraph match $X$ of $Q$ in $G$ in a distributed system

A subgraph match of $Q$ is a subgraph $X$ of $G$ containing $n$ vertices of $V(G)$, if the following conditions hold:
1) There is a mapping function $f$, for each $u_i \in V(Q)$ and $v_j \in V(G)$ ($1 \leq i \leq n$, $1 \leq j \leq n$) it holds that $f(u_i)=v_j$;
2) $\text{sim}(S(u_i),S(v_j)) \geq \tau$ where $S(u_i)$ and $S(v_j)$ are the sets associated with $u_i$ and $v_j$, respectively, and $\text{sim}(S(u_i),S(v_j))$ outputs a set similarity score between $S(u_i)$ and $S(v_j)$ $\geq \tau$;
3) For any edge $(u_i, u_k) \in E(Q)$, there is $(f(u_i),f(u_k)) \in E(G)$ ($1 \leq k \leq n$)

The query retrieves all subgraph matches of $Q$ in graph $G$ under the semantic of the set similarity. Any similarity method can be used, we have used weighted Jaccard Similarity [1].

III. PROPOSED MODEL

A. Framework

All the phases of the framework are designed for the distributed systems using Apache Spark. In the filtering phase, we find frequent patterns of element sets of vertices in data graph $G$. Then, data vertices are encoded into signatures, and organized into Distributed Hash Tables. In order to reduce the search space, an efficient two-phase pruning strategy is proposed based on the frequent patterns found in the filtering phase and signature. In the refinement phase, a dominating set (DS)-based subgraph matching algorithm to find subgraph matches with set similarity. A dominating set selection method is proposed to select a cost-efficient dominating set of the query graph.

Fig 3: Framework For Subgraphs extraction(query processing in a distributed system)
In brief, the following contributions are made:
- Frequent patterns of element sets of vertices and a structural signature-based Distributed Hash Table (DHT) are
  first constructed. A set of pruning techniques (vertical and horizontal pruning) are performed and integrated
together to greatly reduce the search space of subgraphs extraction queries.
- Set similarity pruning techniques (vertical, horizontal, anti-monotone) are proposed that utilize the frequent
  patterns over the element sets of data vertices to evaluate dynamic weighted set similarity. Anti-monotone
  principle is also applied to achieve high pruning power.
- Structure-based pruning techniques are proposed based on the signature buckets.
- Subgraphs extraction is done based on the dominating set of query graph. When filling up the non-dominating
  vertices of the graph, a distance preservation principle is devised to prune candidate vertices that do not preserve
  the distance to dominating vertices.
- Thus, our approach can effectively and efficiently extract the subgraphs along with set similarity and structural
  similarity in a large graph database.

Steps in Flow Chart:
1. Frequent patterns of element sets of vertices in data graph G. (Filtering Phase)
2. Data vertices are encoded into signatures.
3. After encoding data vertices into signatures, they are organized into Distributed Hash Tables.
4. In order to reduce subgraph matching with set similarity search space, different pruning strategies are proposed.
5. Query Signatures are found and are put in signature buckets to reduce the subgraph matching with set similarity
   search space.

B. Techniques designed for subgraphs extraction in distributed systems

A: Set Similarity Pruning

Given : a query vertex u”s set S(u) and a data vertex v”s set S(v), then AS upper bound :
UB(S(u),S(v)) = \frac{\sum_{a \in S(u)} W(a)}{\sum_{a \in S(u) \cup S(v)} W(a)} \geq \text{sim}(S(u),S(v)) ,
where W(a) denotes the weight assigned to element .
Since \sum_{a \in S(u)} W(a) does not change once the query is given, AS upper bound is anti-monotone with regard to S(v).
That is, for any set S(v)\subseteq S(v'), if UB(S(u'),S(v)) < \tau, then UB(S(u),S(v')) < \tau.

B: Pruning Techniques

A: Anti-Monotone Pruning

Given : a query vertex u’s set S(u) and a data vertex v’s set S(v), then AS upper bound :
UB(S(u),S(v)) = \frac{\sum_{a \in S(u)} W(a)}{\sum_{a \in S(u) \cup S(v)} W(a)} \geq \text{sim}(S(u),S(v)) ,
where W(a) denotes the weight assigned to element .
Since \sum_{a \in S(u)} W(a) does not change once the query is given, AS upper bound is anti-monotone with regard to S(v).
That is, for any set S(v)\subseteq S(v'), if UB(S(u'),S(v)) < \tau, then UB(S(u),S(v')) < \tau.

B: Vertical Pruning

Vertical pruning is based on the prefix filtering principle [18]. Given a query set S(u) and a frequent pattern P in
the lattice, if P is not a one-frequent pattern (or its descendant) in S(u)’s p-prefix, all vertices in the inverted list L(P)
can be safely pruned, where P is a descendant node of P in the lattice.

C: Horizontal Pruning

We find LU(u) (length upper bound) by adding elements in (U-S(u)) to S(u) in an increasing order of their weights.
Each time an element is added, a new set S(u) is formed. We calculate the similarity value between S(u) and S'(u). If
\text{sim}(S(u),S'(u)) \geq \tau, then we continue to add elements to S(u). Otherwise, the upper bound LU(u) equals to |S(u)|-1.
B: Structure Based Pruning

A matching subgraph should not only have its vertices (element sets) similar to corresponding query vertices, but also preserve the same structure as Q. We design lightweight signatures for both query vertices and data vertices to further filter the candidates after set similarity pruning by structural constraints.

A: Structural signatures

Two structural signatures are defined - query signature Sig(u), data signature Sig(v), for each query vertex u and data vertex v, respectively.

To encode structural information – Sig(u)=Sig(v) should contain the element information of both u=v and its surrounding vertices.

Procedure:

- First set elements in element sets S(u) and S(v) according to a predefined order
- Based on the sorted sets, we encode the element set S(u) by a bit vector, denoted by BV(u), for the former part of Sig(u).
- Each position BV(u)[i] in the vector corresponds to one element aᵢ, where 1 ≤ i ≤ |U| and |U| is the total number of elements in the universe U. If an element aᵢ belongs to set S(u), then in bit vector BV(u), we have BV(u)[i]=1, otherwise BV(u)[i]=0 holds.
- Similarly, S(v) is also encoded using the above technique.
- For the latter part of Sig(u) and Sig(v) (i.e., encoding surrounding vertices), we propose two different encoding techniques for Sig(u) and Sig(v), respectively.

For Query Signature: Suppose a vertex u is given with m adjacent neighbour vertices uᵢ ( i=1,...,m) in a query graph Q, the query signature Sig(u) of vertex u is given by a set of bit vectors, that is, Sig(u)={BV(u),BV(u₁),...,BV(uₘ)}, where BV(u) and BV(uᵢ) are the bit vectors that encode elements in set S(u) and S(uᵢ), respectively.

For Data Signature: Suppose a vertex v is given with n adjacent neighbor vertices vᵢ (i =1,...,n) in a data graph G, the data signature, Sig(v), of vertex v is given by: Sig(v)= [BV(v),BV(v₁),...,BV(vₙ)], where BV(v) is the bit vector associated with v, BV(vᵢ) is called a union bit vector, which equals to bitwise-OR over all bit vectors of vᵢ’s one-hop neighbors.

B: Signature Based DHT(Distributed Hash Table)

To enable efficient pruning based on structural information, we use Distributed Hash Table to hash each data signature Sig(v) into a signature bucket.

C: Structural Pruning

Finding Similarity using Jaccard Similarity:

Given : Bit vectors BV(u) and BV(v)
To Find :similarity between BV(u) and BV(v)

\[ \text{sim}(BV(u), BV(v)) = \frac{BV(u) \land BV(v)}{BV(u) \lor BV(v)} \]

where \( \land \) is a bitwise AND operator and \( \lor \) is a bitwise OR operator, \( a \in (BV(u) \land BV(v)) \) means the bit corresponding to element a is 1, W(a) is the assigned weight of a.

For each BV(uᵢ), we need to determine whether there exists a BV(vᵢ) so that \( \text{sim}(BV(uᵢ),BV(vᵢ)) \geq \tau \) holds.

C: Dominating Set-based Subgraphs Matching

An efficient dominating-set-based subgraph matching algorithm (denoted by DS-Match) facilitated by a dominating set selection method is proposed.

A: DS-Match Algorithm (for the distributed systems)

It first finds matches of a dominating query graph (DQG) Qᵩ formed by the vertices in dominating set DS(Q), then verifies whether each match of Qᵩ can be extended as a match of Q.

By using DS-match algorithm, we can save filtering cost by only finding candidate vertices for dominating vertices rather than all vertices in Q and we can speed up subgraph matching by only finding matches of dominating query vertices.

Dominating Set :

Let Q = (V,E) be a undirected, simple graph without loops, where V is the set of vertices and E is the set of edges. A set DS(Q) ⊆ V is called a dominating set for Q if every vertex of Q is either in DS(Q), or adjacent to some vertex in DS(Q).

In order to transform a query graph Q to a dominating query graph Qᵩ, we first find a dominating set DS(Q) of Q. Then for each pair of vertices uᵢ, uⱼ in DS(Q), we determine whether there is an edge (uᵢ, uⱼ) between them and the weight of (uᵢ, uⱼ). To find matches of dominating query graph, we propose the distance preservation principle is used.
B: Dominating Set Selection

A query graph may have multiple dominating sets, leading to different performance of subgraphs extraction query processing. Thus, a cost-efficient dominating set is found using a Minimum Dominating Set (MDS) problem[2].

IV. EXPERIMENTAL RESULTS

The modules which were discussed in the proposed paper design model are implemented in the distributed system. The focus of this implementation is on efficiency and time.

A. Datasets

We have used both synthetic as well as real datasets to evaluate our framework. The real dataset used is flight dataset flight data of approximately 4,00,000 nodes wherein average number of elements of each node is 5(source station, destination station, take_off time, landing_time, duration etc.) and min threshold is set to 0.8

B. Experimental Setup

All the experiments are carried out on the operating system: Ubuntu 14.04 using Scala Language and the development environment used is Apache Spark 1.6.0.

Observations:

For the data graph of approximately 4,00,000 nodes, each node having average number of elements of each node is 5 and min threshold is set to 0.8, we extract 100 query graphs by starting from any random vertex.

The query response time for subgraph extraction in a large graph database implemented with the centralized approach is approx. 200 sec, while the query response time for the distributed system is less than 170 sec.

V. CONCLUSION

In this paper, we have proposed a distributed algorithm for subgraphs extraction in a large graph database in a distributed system by considering both vertex set similarity and graph topology using Apache Spark which produces the same result as the centralized system but speeds up the process by a considerable amount (since the application can run simultaneously) and also redundancy increases availability when parts of a system fail. As followed from our experiments, our distributed approach performs the same processes in a lesser time with a greater flexibility and scalability (through addition of components). The solution devised can be combined easily with any type of application such as social networks, Semantic Web, biological networks and it will undoubtedly boost the performance since the methodology carries out parallel processing. One of the applications can be a chatting application wherein a query is asked and the relevant response is desired. It can also be used in any search engine that can use formulas more sophisticated than words and for security purposes like in fingerprints scanners, facial scanners, retina scanners and so on. Also, any system that performs clustering would benefit from this fast algorithms.

REFERENCES

[11] Liang Hong, Member, IEEE, Lei Zou, Member, IEEE, Xiang Lian, Member, IEEE, and Philip S. Yu, Fellow, IEEE, “Subgraph Matching with Set Similarity in a Large Graph Database,” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 27, NO. 9, SEPTEMBER 2015