A Method of Subgraphs Extraction in a Large Graph Database in a Distributed System

Ritu Yadav
National Institute of Technology, Warangal
Bangalore, India

Samarth Varshney
National Institute of Technology, Warangal
Bangalore, India

ABSTRACT

Since many real applications such as web connectivity, social networks, and so on, are emerging now-a-days, thus graph databases have been commonly used as significant tools to exemplify and query complex graph data wherein each vertex in a graph usually contains information, which can be modeled 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.

General Terms

centralized systems, graph databases, parallel processing, subgraphs extraction

Keywords

apache spark, distributed systems, graph topology, set similarity

1. 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 commonly used as significant tools to exemplify and query complex graph data wherein each vertex in a graph usually contains information, which can be casted 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.

2. PROBLEM DESCRIPTION

Given a large graph G represented by $\langle V(G),E(G)\rangle$, 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 $\langle V(Q),E(Q)\rangle$. 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 τ

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), iff the following conditions hold:

1) There is a mapping function f, for each ui in V(Q) and vj ∈ V(G) (1≤ i ≤n, 1≤j≤n) it holds that f(ui)=vj;

2) sim(S(ui),S(vj)) ≥ τ where S(ui) and S(vj) are the sets associated with ui and vj, respectively, and sim(S(ui),S(vj)) outputs a set similarity score between S(ui) and S(vj) ≥ τ ;

3) For any edge (ui , uk) ∈ E(Q), there is (f(ui),f(uk))∈ E(G) (1≤ k ≤ 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].

3. PROPOSED MODEL

3.1 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.

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.

3.2 Techniques designed for subgraphs extraction in distributed systems

3.2.1 Set Similarity Pruning

Given: a vertex u in a dominating set of query graph Q

To Find: candidate vertices of u in graph G.

3.2.1.1 Generation of Frequent ItemSets

Let U be the set of distinct elements in V(G). A pattern P is a set of elements in U, i.e., P U. If an element set S(v) contains all the elements of a pattern P, then we say S(v) supports P and S(v) is a supporting element set of P. The support of P, denoted by supp(P), is the number of element sets that support P. If supp(P) is larger than a user-specified threshold minsup, then P is called a frequent pattern.The computation of support of pattern P (i.e., supp(P)) can be referred to [5].Thus, following above technique, frequent patterns are generated.

Further AS (Anti-Monotone Similarity) Upper Bound is applied Using Inculsion Relation in order to prune vertices.

Inculsion Relation : For two element sets S(v) and S(v') of vertex v and v' respectively, if S(v) ⊆ S(v'), the relationship of S(v) being a subset of S(v') is called inclusion relation.

Thus, AS Upper Bound is defined as- Given : a query vertex u’s set S(u) and a data vertex v’s set

S(v),then AS upper bound :

$$\mathrm{UB}(S(u),S(v)) = \frac{\sum_{a \in S(u)} W(a)}{\sum_{a \in S(u) \cup S(v)} W(a)} \geq \mathrm{sim}(S(u),S(v)),$$

where,

W(a) denotes the weight assigned to element.

Since 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)∈ S(v'), if UB(S(u'),S(v)) < τ, then UB(S(u),S(v')) < τ.

3.2.1.2 Pruning Techniques

3.2.1.2.1 Anti-Monotone Pruning

Given a query vertex u, for each accessed frequent pattern P in the inverted pattern lattice, if UB(S(u),P) < τ, all vertices in the inverted list L(P) and L(P') can be safely pruned, where P' is a descendant node of P in the lattice.

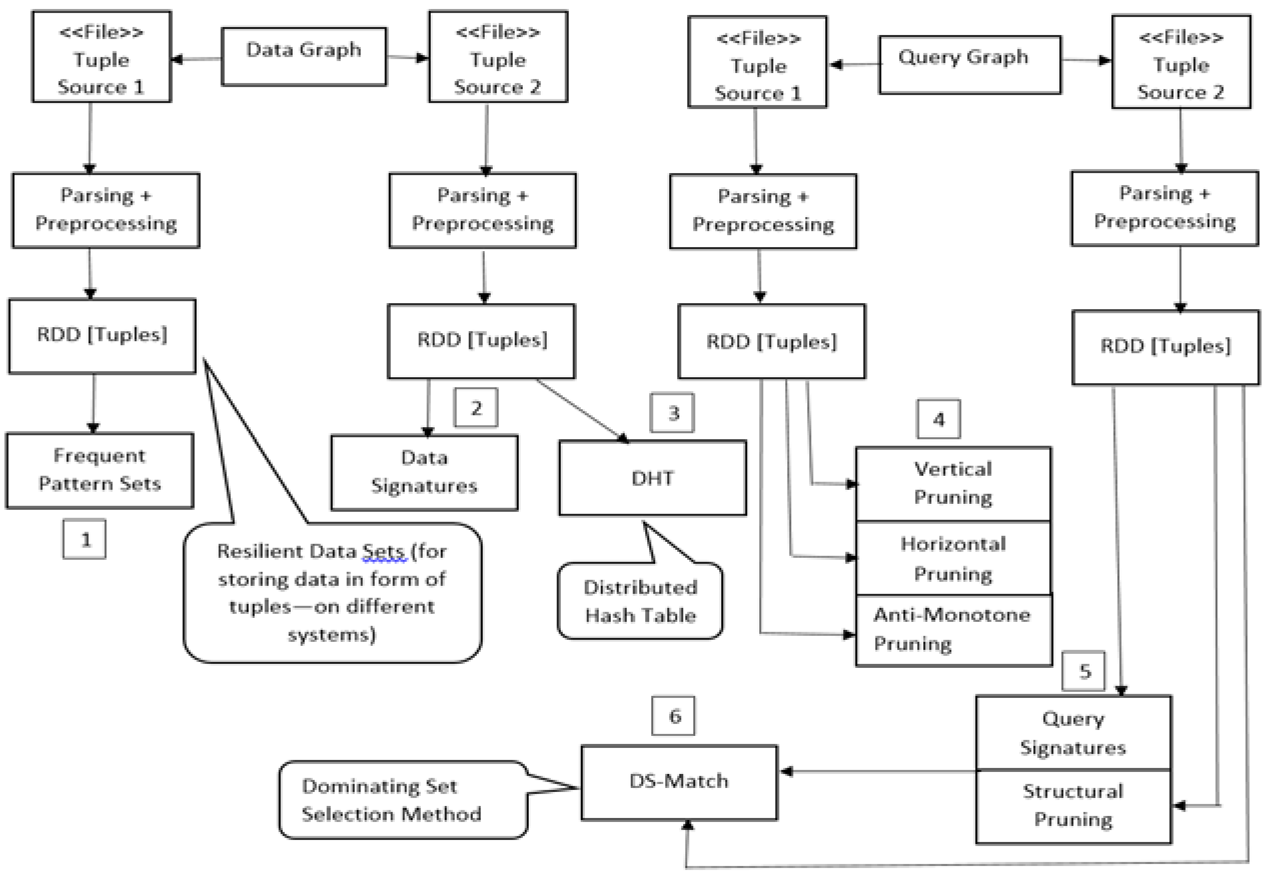

Fig 1: Framework for subgraphs extraction (query processing in a distributed system)

3.2.1.2.2 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.

p-prefix of S(u) is calculated as:

Whenever we remove the element with the largest weight from S(u), we check whether the remaining set S'(u) meets the

similarity threshold with S(u). ‖S(u)‖ = $\sum_{a \in S(u)} W(a)$.

If ‖S'(u)‖ < τ×‖S(u)‖ , the removal stops. The value of p is equal to |S'(u)|-1, where |S'(u)| is the number of elements in S'(u)

3.2.1.2.3 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 sim(S(u),S'(u)) ≥τ holds, we continue to add elements to S'(u). Otherwise, the upper bound LU(u) equals to |S'(u)|-1.

All frequent patterns under Level LU(u) will be pruned.

3.2.2 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.

3.2.2.1 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 sort 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 aj belongs to set S(u), then in bit vector BV(u), we have BV(u)[j]=1, otherwise BV(u)[j]=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 neighbor 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(u_1),...,BV(u_m)}, where BV(u) and BV(u_i) are the bit vectors that encode elements in set S(u) and S(u_i), respectively.

For Data Signature: Suppos 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),$\vee_{i=1}^{n} BV(v_i)$], where v is a bitwise OR operator, BV(v) is the bit vector associated with v, $\vee_{i=1}^{n} BV(v_i)$ is called a union bit vector, which equals to bitwise-OR over all bit vectors of v's one-hop neighbors.

3.2.2.2 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.

3.2.2.3 Structural Pruning

Finding Similarity using Jaccard Similarity-

Given : Bit vectors BV(u) and BV(v)

To Find :similarity between BV(u) and BV(v)

$$\text{sim(BV(u),BV(v))}= \frac{\sum_{a\in BV(u)\wedge BV(v)} W(a)}{\sum_{a\in BV(u)\vee BV(v)} W(a)}$$

where $\wedge$ is a bitwise AND operator and $\vee$ is a bitwise OR operator, a∈ (BV(u) $\wedge$ BV(v)) means the bit corresponding to element a is 1, W(a) is the assigned weight of a.

For each BV(u_i), we need to determine whether there exists a BV(v_j) so that sim(BV(u_i),BV(v_j)) $\geq \tau$ holds.

3.2.3 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.

3.2.3.1 DS-Match Algorithm(for the distributed systems)

It first finds matches of a dominating query graph (DQG) Q_D formed by the vertices in dominating set DS(Q), then verifies whether each match of Q_D 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 Q_D, we first find a dominating set DS(Q) of Q. Then for each pair of vertices u_i,u_j in DS(Q), we determine whether there is an edge (u_i,u_j) between them and the weight of (u_i,u_j).

To find matches of dominating query graph, we propose the distance preservation principle is used.

3.2.3.2 Dominating Set Selection

A query graph may have multiple dominating sets, leading to different performance of SMS2 query processing. Thus, a cost-efficient dominating set is found using a Minimum Dominating Set (MDS) problem[2].

4. 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.

4.1 Datasets

Both synthetic as well as real datasets have been used to evaluate the 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

4.2 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, 100 query graphs have been extracted 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.

5. CONCLUSION

In this paper, a distributed algorithm is proposed 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.

6. REFERENCES

[1] 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.

[2] P. Zhao and J. Han, "On graph query optimization in large networks," Proc. VLDB Endowment, vol. 3, nos. 1/2, pp. 340–351, 2010.

[3] 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.

[4] 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.

[5] H. He and A. K. Singh, “Closure-tree: An index structure for graph queries,” in Proc. 222nd Int. Conf. Data Eng., 2006, p. 38.

[6] W.-S. Han, J. Lee, and J.-H. Lee, “Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2013, pp. 337–348.

[7] 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.

[8] J. R. Ullmann, “An algorithm for subgraph isomorphism,” J. ACM, vol. 23, no. 1, pp. 31–42, 1976.

[9] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, “Capturing topology in graph pattern matching,” Proc. VLDB Endowments, vol. 5, no. 4, pp. 310–321, 2012.

[10] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, “Nema: Fast graph search with label similarity,” in Proc. VLDB Endowment, vol. 6, no. 3, pp. 181–192, 2013.

[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