Efficient Subgraph Query Algorithm in Large Graph

Yun HUANG\(^1,2\) and Jia-ming HONG\(^3,*\)

\(^1\)School of Data Science and Computer, Sun Yat-sen University, Guangzhou, Guangdong 510006, China
\(^2\)School of Software and Service Outsourcing, Jishou University, Zhangjiajie, Hunan 427000, China
\(^3\)School of Medical Information Engineering, Guangzhou University of Chinese Medicine, Guangzhou, Guangdong 510006, China

*Corresponding author

Keywords: Subgraph query, Large graph, Symmetric relation, Core decomposition.

Abstract. Subgraph query is an important problem in the research and application of large graph mining. For large graphs with symmetry relation substructures, the existing decomposition-join strategy always leads to low searching efficiency. To solve this problem, we proposed a new decomposition-Detection-join strategy, in which we detect symmetric relations of each sub part of the decomposition, and then determine the sequence of queries based on the detection results. The experimental results show that the algorithm has much improvement in query efficiency.

Introduction

In the past ten years, with the popularity of micro-blog, WeChat, navigation and positioning service applications, the world wide web, social networks, traffic networks and other related fields become the hottest research directions, and the problem of subgraph query in the large single graph has become a key issue in all these fields.

Subgraph isomorphism is the primary problem of subgraph query algorithms, in which Ullmann[1], QuickSI[2] and TurboISO[3] algorithm are subgraph isomorphism test methods based on the idea of backtracking. The TurboISO algorithm could identify the equivalence vertices in the query graph, and find the neighborhood equivalence classes of each vertex (NEC), and then use the query graph to rewrite the NEC tree, which can reduce the repeated queries on the equivalent vertices. A number of algorithms are based on the extended strategy \[4,5\] for subgraph query, and the essential idea is to generate candidate point set for each vertex, and then search for the required subgraph through the extension of the edge. In the paper[6], the authors study the problem of equivalent vertex optimization query in data graph, and improve the query efficiency by the transformation of the data graph.

There are some algorithms proposed for subgraph query based on decomposition-join strategy. SPath[7] will divide the query graph into many shortest paths and respectively match each shortest path, and connect the path matching result into the subgraph matching result. Lin et al [8] proposed a subgraph query method named NTree, based on neighbor tree decomposition. NTree first generates neighbor tree and finds the candidate set for each vertex, and then selects the minimum connection cost vertices to join together, until all vertices in the query graph are accessed. Unlike NTree, the SGMatch method [9] finds the Hub cover set with the minimum edge coverage in the query graph at first, and then matches each Hub vertex and its adjacent points, and connects the query results. In paper [10], the query graph is decomposed into stars and the matching results of each star are obtained. Bi [11] et al found a lot of unnecessary intermediate results by Cartesian products from dissimilar vertices and developed a new framework in order to postpone Cartesian products. By the core-forest-leaf decomposition strategy, the framework deals with all leaf vertices of a query in the last step.

Literature [12] introduces one of the best subgraph query algorithms. However, when the core of the query graph has a large number of vertices with same labels or a certain symmetric substructure,
the query efficiency is low. As shown in Figure 1, the core of the query graph $Q$ is four vertices with the same label ‘a’, and many vertices in the data graph $G$ is also labeled ‘a’. Because the four vertices have the same label, if we first process the core query based on adjacency relation, we would find too much vertex candidates in the graph (see Figure 2); on the other hand, even if it is to find a set of matching, such as \{\textit{u}_2, \textit{u}_3, \textit{u}_6, \textit{u}_7\}, however, due to the symmetry of the query core, there are 24 different enumerations, making it too feasible to judge the matching order.

![Figure 1. Example query graph and data graph.](image1)

![Figure 2. Results of core matching in Fig1. (part).](image2)

From the analysis above, we can see that the existing methods are designed to produce the corresponding solutions by analysing structural similarity and symmetry in the graph. However, when the structures of query graphs are diverse, such methods are not suitable due to its low efficiency.

To solve this problem, we propose a new decomposition-Detection-join framework of subgraph query. In the offline phase, we first analyze the data graph and establish the label-label join index. After given query graph, we will process core-Forest decomposition, and then test the symmetry or duplicate substructure of the core and forest, selecting appropriate starting point and matching path, and connect them to generate the query results at the final step.

The rest of this paper is organized as follows. We give the preliminaries in Section 2. Section 3 gives an overview of our decomposition-Detection-join framework. In Section 4, we introduce the subgraph query algorithm. Experimental results are reported in Section 5. We give a conclusion in Section 6.

**Preliminaries**

In this paper, we mainly study the undirected simple labeled graph. The method is also applicable to other types of graphs. If there is no special explanation, all the graphs in the following text are undirected simple labeled graph. The definition of the label graph is as follows.
Definition 1 (labeled graph) a labeled graph $G$ is a five element tuple $G=(V, E, L_V, L_E, l)$ in which $V$ represents the set of vertices in the graph, $E \subseteq V \times V$ is a edge set, $L_V$ is the vertex labels set, $L_E$ is the edge labels set. The labeling function $l$ defines the mappings $l: V \rightarrow L_V$ and $l: E \rightarrow L_E$.

Graph isomorphism and subgraph isomorphism are the core operations in the query graph, the following we give the definitions of graph isomorphism and subgraph isomorphism.

Definition 2 (graph isomorphism) graph $G=(V, E, L_V, L_E, l)$ and $G'=(V', E', L_V', L_E', l')$ are isomorphic, if there is a bijective function $f: V(G) \rightarrow V(G')$, such that:

1. $\forall u \in V, l(u) = l'(f(u))$;
2. $\forall u, v \in V, ((u,v) \in E) \iff ((f(u), f(v)) \in E')$;
3. $\forall (u,v) \in E, l(u,v) = l'(f(u), f(v))$.

$G$ and $G'$ are denoted as $G \cong G'$ they are isomorphic.

Definition 3 (subgraph isomorphism) if there exists graph $G'$ which is the subgraph of $G$, graph $G'$ and Graph $H$ are isomorphic, we called graph $H$ subgraph isomorphic to graph $G$, denoted as $H \subseteq G$. $G$ is the supergraph of $H$. $G'$ is a embedding of $H$ in $G$, denoted as $G' \equiv \text{Em}(H, G)$.

Now we will give the problem statement in this paper.

Problem statement: Given a query graph $Q$ and a data graph $G$, we will find all distinct embeddings of $Q$ in $G$.

A New Framework

In our new framework, the subgraph query approach consists of two phases: offline index construction phase and online query processing phase. Before query graph is given for processing, the label-label index structures are constructed offline, as shown in Table 1.

<table>
<thead>
<tr>
<th>(a,a)</th>
<th>(u_1, u_5)(u_1, u_3)(u_1, u_7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a,b)</td>
<td>(u_1, u_9)(u_2, u_10)(u_1, u_7)(u_3, u_12)</td>
</tr>
<tr>
<td>(a,d)</td>
<td>(u_2, u_11)(u_3, u_11)</td>
</tr>
<tr>
<td>(b,c)</td>
<td>(u_9, u_13)(u_12, u_14)</td>
</tr>
</tbody>
</table>

Table 1. The Label-Label Index of Data Graph G in fig.1.

When a query graph is given, we process core-forest decomposition and forest-leaf decomposition for query graph. And the vertex in the core is the vertex of some circle in the query graph. The forest is composed with the trees whose root is the vertex in core. The leaf is the vertex whose degree is equal to 1.

After decomposing the graph, we will detect the symmetric substructure in the core and forest, and count the number of vertices with the same label. If we find some symmetric substructures, we will rewrite the query graph and sort the query vertex.

In the last step, we will match the every parts of the query graph according to the computed subsequence. For the data graph and query graph in fig 1, we would get the spanning tree matching result showing in fig 3. Obviously, our approach reduce many unnecessary intermediate matchings.

Figure 3. Spanning Tree Matching in fig 1.
The Symmetric Substructure Detection and Query Algorithm

First, we find the core structure from the query graph:

1. Copy query graph $Q$ to $Q_c$;
2. Record and remove the leaf vertex (and the edge with the leaf vertex) from $Q_c$;
3. If exist vertex whose degree is 1 in $Q_c$, goto(2), otherwise return $Q_c$.

Then, we search the symmetric vertices set $SymSetGr$ from $Q_c$, and we could see the method in Algorithm 1.

Algorithm 1: Discovery Symmetric Relation // algorithm to find the vertex symmetry

```
Input: core $Q_c$, search level threshold $ln$, vertex number threshold $vn$
Output: equivalent vertex set group SymSetGr
1 SymSetGr$\leftarrow \phi$;
2 $v$ $\leftarrow$ selectFirstVertex ($Q_c$); //select the first searching vertex
3 while($v$)
4    foreach $L_a$ $\in \ell(N_1(v))$
5        search the quasi symmetric vertices //with same label, their neighbors’ label are same too;
6        $SymS$ $\leftarrow$ search symmetric vertices set from the quasi symmetric vertices;  
(no more than $ln$ level, no more than $vn$ vertices
7        $SymSetGr$ $\leftarrow$ $SymSetGr$ $\cup \{SymS\}$;
8        $v$ $\leftarrow$ selectnextVertex($Q_c$)
9 return $SymSetGr$;
```

In Algorithm 1, $N_1(v)$ is the set of the neighbors of $v$, and $\ell(N_1(v))$ is the label set of $N_1(v)$. Next, we will use the symmetric vertices sets to improve the query efficiency. Specifically, we will compute the vertices number num($Lb, Q_c$) with the same label. If $SymSetGr$ is empty, and $\Sigma$num($Lb, Q_c$/|V($Q_c$)|) is small than the give value, we will search the graph using the method mentioned in Literature [12]. Otherwise, we will merge the equivalent vertices (select from $SymSetGr$) into one single vertex and reform the query graph, and we will sort the query vertices as Formulation 1.

\[ \text{Score}(v, Q_c, G) = |\text{cand}(v, G)| \times \text{num}(l(v), Q_c) \]  

(1)

If The vertex matching is proceeded according to the sorted scores, i.e., the vertex $v$ with smaller Score($v, Q_c, G$) would be matched earlier than the vertexes with larger scores.

As the last step, we process the query with the backtracking methods, such as VF2, QuickSI, TurboISO and so on.

Experiment

The experiment was carried out on a I7 4.0G basic frequency CPU computer with 16G memory, and the experiment was carried out in Java language.

We used 3 real datasets in our experiments: Human, HPRD, Yeast. Human contains 4674 vertices, 86282 edges, 44 distinct labels. There are 9460 vertices, 37081 edges and 307 vertex labels in HPRD dataset. Yeast dataset contains 12519 edges, 3112 vertices with an average degree 8.1, 71 distinct vertex labels.

We comparing the performances of the state-of-the-art algorithms with DDJMatch. The algorithms considered are as follows: (1) QuickSI [3], (2) TurboISO [4], (3) CFL-Match[12]. There are three type of query graph in our experiments: none symmetric substructure in the query graph, symmetric substructure in the forest part ($Q_{tree}$) of the query graph, and symmetric substructure in the Core part ($Q_{core}$) of the query graph. The query graph size could vary with 4,8,12,16,20,24 vertices, and the average vertex degree is 6. In every experiment we provide 100 query graphs and compute the average processing time.

The experiments results are plotted in Fig 4. and Fig 5. The CFL-Match method has similar query efficiency with our algorithm when the core has no symmetric substructure, and their performances are much better than that of QuickSI and TurboISO methods. When there are some symmetric
substructures available in core part of the query graphs, the performance of CFL-Match decreased explicitly, while our method still preserved high efficiency.

Figure 4. Averaged processing time in human dataset.

Figure 5. Averaged processing time in HPRD dataset and Yeast dataset.

Summary
In this paper, we propose a new framework for subgraph query in large graph, and experiments demonstrated that it is effective and efficient.

Acknowledgement
This research was financially supported by Natural Science Foundation of Guangdong Province, China (No. 2015A030310312), Medical Scientific Research Foundation of Guangdong Province, China (No. B2014174), and a special grant from Department of Finance of Guangdong Province, China (No. [2013] 170)

References


