Journal of Computers Vol. 31 No. 6, 2020, pp. 182-195
doi:10.3966/199115992020123106015

HLMA: An Efficient Subgraph Matching Algorithm

Gang Dai[1,2], Baomin Xu[1*], Hongfeng Yin[3]

[1] School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
{gdai, bmxu}@bjtu.edu.cn

[2] China University of Petroleum-Beijing at KARAMAY, Xinjiang 834099, China

[3] Department of Computer Science, Beijing Jiaotong University Haibin College, Huanghua 061199, China
hfyin@bjtuhbxy.edu.cn

Received 13 October 2019; Revised 22 March 2020; Accepted 13 April 2020

Abstract. Graph mining is of great significance for social network analysis, biological research and other information applications. One interesting but challenging problem of graph mining is subgraph matching. Most of the existing subgraph matching algorithms have not considered both accuracy and efficiency. In this paper, we propose an approximation algorithm for subgraph matching in a large undirected graph. The basic idea is to convert the vertices of the graph into a data structure h-list based on label propagation. According to h-list, we can find a candidate matching set for each query vertex by searching on the target graph. To obtain optimal matching results, we present a scoring metrics to measure the similarity between a query vertex and each vertex of its candidate matching set. The whole algorithm is called HLMA (H-List Matching Algorithm). The experimental results show that HLMA has higher efficiency and matching accuracy, while computational processing of complex subgraph isomorphism can be avoided.

Keywords: graph query, label propagation, subgraph matching, vertex alignment

1 Introduction

Recently, the field of graph mining has grown rapidly, not only because the number and the size of graphs has been growing exponentially, but also because we want to extract much more complicated information from graphs. Due to the advantage on complex relationship expression ability, graph has been increasingly used as a powerful tool to store and express data in various fields.

Subgraph matching, namely how to find a simple query graph in a target graph, is the most critical and most basic problem in graph mining. It has great significance in many areas, such as RDF query based on graph matching techniques is effectively applied in semantic web [1]. Graph alignment technique is important for studying protein function on the protein-protein interaction network [2-3]. With the rise of social networks, more and more scholars are attracted by efficiently subgraph search on big graph.

Subgraph matching is a NP-complete problem. Therefore, the method of developing efficient and flexible subgraph matching arises in a number of applications [4]. For example, exact matching and approximate matching based on strict structure have achieved a lot of valuable results in the field of biology and chemistry [2-5]. Among them, IsoRank [5] considered that if two vertices from different graph are similar, then their adjacent vertices should be similar. Based on this principle, they proposed a scoring function to evaluate the quality of matches. Unfortunately, IsoRank needs to label all possible combinations of the vertices. Owing to the oversized candidate set, it makes the algorithm unable to efficiently perform subgraph similar queries on large graph.

* Corresponding Author

For social networks, accurate structure information of the target graph is usually unknown. Thus, people cannot offer completely correct query graph. Therefore, it is more appealing to find the inexact matches [6-9]. Tong et al [10] proposed that the graph matching should be based on vertex labels rather than strict graph structure. Khan et al [11] proposed an algorithm named Ness to efficiently identify the top-k graph matches. The algorithm inherited Tong's point of view and applies label (information) propagate model to inexact subgraph matching. It firstly converts each vertex into a multidimensional vector based on neighbors' label. Then it presents a cost function to measure the quality of every match. To find top-k matches, the algorithm will perform a propagate-matching iterative process on target graph. However, the time cost of label propagation is much higher than that of the matching process.

The algorithms based on neighborhood have also been used in exact matching problem, such as GraghQL [12]. Compared with Ness, exact matching algorithm has stricter pruning rule. The reason is that inexact and exact matching has different similarity measurement. Inexact algorithms can achieve high efficiency, while exact matching has a low accuracy. The similarity measurement of exact matching is subgraph isomorphism. It is well known that subgraph isomorphism is a NP-complete problem. For some query or target set, exact algorithm like GraghQL often shows exponential behavior [13]. Index algorithm such as gIndex [14], SwiftIndex [16], and FG-Index [16] has the same problem. These algorithms have some successful cases in small graph data set. Usually, they will divide these graphs into fragments and built an index for each fragment. The index can help to prune candidate matching set. But indexing algorithm must eventually return to subgraph isomorphism. Thus, the efficiency of these algorithms is not high when the target graph is a larger graph. With the help of label, HLMA can find out the exact match vertices for every query vertex, while maintaining high accuracy.

Many studies indicate that the exact algorithms often fail to achieve high efficiency. Namely, when the size of target graph increased, the accuracy of exact matching algorithms may decline. To solve this problem, we propose a novel approximation algorithm named HLMA for approximate large graph matching. The strategy is to combine the label of neighborhood vertices and the topological information. Thus, we can avoid costly subgraph isomorphism testing on a large target graph.

In this paper, we tackle the approximation algorithm for subgraph matching in a large undirected graph. To summarize, we make the following contributions: (1) We develop a novel approximate subgraph matching algorithm HLMA (H-List Matching Algorithm) based on neighborhood information. (2) We proposed a search algorithm to improve the search efficiency of finding every query or target pair. Unlike Ness, this algorithm needs not to repeat label propagation on the big target graph [17]. (3) We proposed a scoring metrics to measure the quality of graph matching, so that we could find out the optimal matching results for each query vertex. Compared with IsoRank, we can offer a better candidate set for the scoring process. Specifically, our work can be used to build index for graph matching task on large size graphs, to which the index techniques based on Frequent [18] or discriminative [19] pattern isn't competent.

2 Related Work

These days graph research has attracted great attention, related works include subgraph isomorphism algorithms, graph indexing and subgraph indexing, approximate subgraph matching and graph similarity search.

The first category of related research lies in subgraph isomorphism algorithms [20-21]. These algorithms do not utilize any index structure. Many index-based graph matchings and searching schemes have been proposed to find where the query graph occurs in the graph databases, which can be further divided into the graph indexing and subgraph indexing.

Recently, a number of algorithms are proposed which support approximate graph matching or similarity search through different means [22-23]. In TALE [24], important nodes are matched first and then the match is progressively extended. The method is very effective and fast in approximately finding matches in a large graph. Another possible approach is to build a declarative framework for approximate graph matching where one can design various constraints on the matching. These methods imply the potential approximation had to satisfy constraints such as mandatory and optional nodes and edges. Obviously, the drawback of this method is that many times, we are searching for subgraphs without any prior knowledge of the pattern to be found.

Another category of research related to the subgraph matching is graph alignment. In response to existing graph matching methods being too restrictive, [8] developed a tool called Substructure Index-based Approximate Graph Alignment (SAGA). It allows for node gaps, node mismatches and graph structural differences and does not require any constraints to be designed in advance. The disadvantages are that one has to maintain a database of small structures and that it is query based. In the problem studied in this paper, the size of the query graph may be much larger than that of the database graph. Thus, the graph alignment method cannot be directly applicable in our methods.

In this paper, we will take a similar idea to SAGA. A novel approximate subgraph matching algorithm HLMA based on neighborhood information is provided. The basic idea is to convert the vertices of the graph into a data structure h-list based on label propagation. According to h-list, we can find a matching set of candidates for each query vertex by searching on the target graph; The method can keep the time complexity of HLMA is, roughly, a linear relationship with the number of vertices of a target graph, and the accuracy of HLMA is not sensitive to changes in scale of graph, but the value of label density.

3 Preliminaries

Now we give a few definitions to describe the problem we need to solve.

Degree: $d(v)$ is the number of edges linked to vertex v.

Graph order: $p(g)$ is the number of vertices on graph g.

Distance: the distance between two vertices on a graph is the number of edges on the shortest path between them.

Graph invariant: properties of graphs that are invariant under graph isomorphism.

Graph isomorphism: isomorphism of graphs G and H is a bijection f between the vertex sets of G and H, such that 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.

Query graph: Query graph is an undirected labeled graph $Q=(v_q, e_q, l_q)$, with a vertex set v_q, an edge set e_q, a label function l_q which assigns a label to each vertex in v_q.

Target graph: Target graph is an undirected labeled graph $G=(v_g, e_g, l_g)$, with a vertex set v_g, an edge set e_g, a label function l_g which assigns a label to each vertex in v_g.

A vertex in Q or G denotes an entity in network (maybe social network or PPI network, and so on), an edge represents the relationship between two entities, label of a vertex means attribute of an entity.

Subgraph similarity search: Given a query graph $Q=(v_q, e_q, l_q)$, a target $G=(v_g, e_g, l_g)$, and a similarity measurement S. Subgraph similarity search is to find a subgraph G_{sub} of G, which is similar with Q under S. If S is defined as isomorphism, the subgraph similarity is equivalent to exact subgraph matching.

Label propagation: Label propagation has been deeply studied in semi-supervised learning. Its main idea is to propagate the label of each vertex to its neighborhoods, and generate a multidimensional vector containing the label information for each vertex. Label propagation is often used to define similarity measurement for inexact matching.

H-neighborhood: The h-neighborhood of a vertex v is a set of vertexes that their distance from v is less than or equal to h step.

4 Label Propagation and H-list

In this section, we will discuss how to transform a vertex into a novel data structure h-list. For the convenience of the following description, the following statements are given first.

The number of paths from vertex v_i to vertex v_j with different lengths can be expressed as pathVector(v_i, v_j) = (p_1, p_2, ..., p_h). Where p_k is the number of paths from v_i to v_j, with length of k.

The labelVector of the vertex v with label a is expressed as labelVector(v, a)=(a, pathVector(v, u_1), pathVector(v, u_2), ..., pathVector(v, u_n)), where u_1, u_2, ..., u_n are v's h-nighborhoods having the same label a.

The h-list of the vertex v is represented as v.h-list = (labelVector(v, l_1), labelVector(v, l_2), ..., labelVector(v, l_m)), $\bigcup_{1}^{m} l_i = L$. Where L is the set of all the labels in the graph G.

Here is an example of constructing h-list by label propagation. Consider the graph showed in Fig. 1 and take h=2. $v2$ is a neighbor of $v1$. Its label, a, is propagated to $v1$ through two paths: $v2$-$v1$ and $v2$-$v3$-

*v*1. So, the relationship between *v*1 and *v*2 can be represented by a vector: (a, pathVector(*v*1, *v*2)). The vector's length is 2. The i-th element represents the number of the path with length i. As shown in Fig. 1, there is a one-step and a two-step path between *v*1 and *v*2, so pathVector(*v*1, *v*2)=(1,1). We construct all pathVectors for *v1's* two-step neighborhood, and combine the pathVector of same label to construct a labelVector. For *v*1, labelVector of label a is (*a*, (1,1), (0,1)), where (1,1) comes from *v*2, and (0,1) comes from v4. Furthermore, all labelVector from *v*1's h-step neighborhood composed the h-list of v1: {(*a*, (1,1), (0,1)), (*b*, (1,1), (1,0))}, where $h = 2$. This h-list conveys the relationship between *v*1 and all its 2-step neighbors, so it can be viewed an approximate representation of the subgraph around *v*1. As shown in Fig. 2, we constructed a data structure with three different layers: the most basic part is pathVector, the middle layer labelVector is composed of a number of pathVector, and all labelVector converging to the h-list of a vertex.

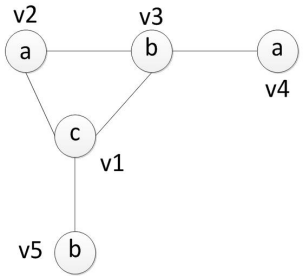

Fig. 1. The path between v1 and v2

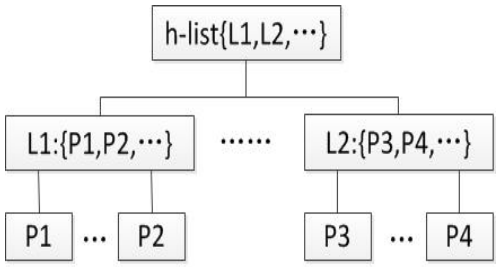

Fig. 2. The h-list structure for v1

Through label propagation, we can transform the vertex set of a graph into an h-list set. Obviously, the h-list set is a graph invariant. It is a classic idea to research similarity matching using graph invariant. Path layer matrix is one of them. It has been proved that the identical path layer matrix is equivalent to isomorphism when the number of vertices is less than 11 [25]. Theorem 1 shows the relationship between path layer matrix and h-list set.

Theorem 1: the h-step path layer matrix of graph *G* can be calculated by all the h-lists of vertexes in *G*.

Proof: Since the path of label propagation has no duplicate edges. The basic element of h-list represents the number of paths between two vertexes within h step. So just to do vector addition on all path vectors in h-list, we can get the number of paths with different length (range from 1 to *h*). The h-step path layer matrix of *G* can be calculated by doing the above operation on all vertexes in *G*.

According to Theorem 1, one h-list set has just one corresponding path layer matrix, but one path layer matrix may have many different corresponding h-list sets. It means h-list set is closer to isomorphism than path layer matrix. Although the identical h-list set is not equivalent to isomorphism when graph become large, we can find similarity matching of query graph with the help of h-list.

Theorem 2: For graphs with equal or less than 11 vertices, identical h-list is equivalent to isomorphism. For graphs with more than 14 vertices, h-list is a better isomorphism filtering condition than path layer matrix (PLM).

Proof:

(1) h-list is obviously a graph invariant, so the isomorphic graphs have identical h-list. It has been proved that when graph order is equal to or less than 11, identical PLM is a sufficient condition for isomorphism. By Theorem 1, a h-list has only one corresponding PLM, so identical h-list is also a sufficient isomorphism condition for such graphs.

(2) When the graph order is large than 11. one PLM or h-list may correspond to many non-isomorphic graphs. The Theorem 1 shows a PLM may also correspond to many different h-lists, but not vice versa. Suppose a PLM corresponds to m graphs and n h-list, and given two graphs share this same PLM, the possibility of the two graphs are isomorphic is *P*1=(1/m)^2. If the two graphs also have same h-list, the possibility is $P2=(n/m)^2$. *P*2 is n^2 times higher than *P*1, so h-list is a better isomorphism filtering condition than PLM.

Theorem 2 means that the h-list of a vertex can show partial similarity between graphs. So, through the h-list matching, we can get some candidate matches of a query vertex. In fact, we have no need for h=10, which is too large. The above theorems just show the topological information in h-list. It also contains label information of h-neighborhoods, which is why we can use h-list to find candidate matching set quickly and precisely.

When the graph order is 14 or more, the non-isomorphic graphs with the same path layer matrix has already been found. In the latter experiment, we will show that HLMA will distinguish the two pairs of graphs described above. This means the h-list has much more graph structure information than the path layer matrix. That is why we propose h-list to be the graph similarity measures for subgraph search.

5 Hierarchical H-list Matching

In this section, a vertex matching function vMatch(v, u) is introduced. vMatch(v, u) is based on the h-list of v and u. v is a query vertex, u is a target vertex. As shown in Fig. 3, hierarchical h-list can be decomposed into a number of labelVector. The labelVector can be decomposed into a number of path Vectors. According to the hierarchical nature of h-list, the h-list matching function vMatch (v, u) can be divided into three levels: The basic is pMatch (P_1, P_2). The path vector P_1 and P_2 are from h-list (v) and h-list (u), respectively. The upper layer is lMatch (L_1, L_2). The label vector L_1 and L_2 are from h-list (v) and h-list (u), respectively. Top layer is vMatch (v, u). If vMatch (v, u) = 1, u is candidate match of v.

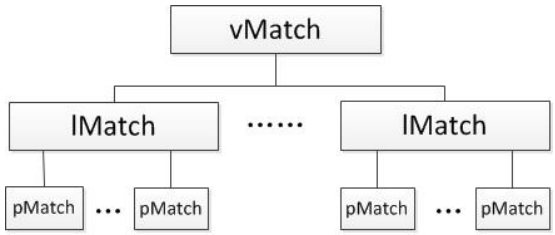

Fig. 3. Hierarchical h-list

pMatch(P_1, P_2), lMatvh (L_1, L_2), and vMatch (v, u) are defined as follows:

$$pMatch(p_1, p_2) = \begin{cases} 1, \exists \text{i } s.t.\ p_i^{(1)} \le p_i^{(2)}, p_i^{(1)} \in P_1, p_i^{(2)} \in P_2 \\ 0, \text{otherwise} \end{cases} \quad \textbf{(1)}$$

$$\text{l}Match(L_1, L_2) = \begin{cases} 1, \quad L_1.label = L_2.label, \text{ and} \\ \qquad \forall P_i \in L_1\ \exists R_i \in L_2, \text{ s.t. } \text{p}Match\ (P_i, R_i) = 1 \wedge (R_i != R_j, i\ != j) \\ 0, \text{otherwise} \end{cases} \quad \textbf{(2)}$$

$$\text{v}Match(v_1, v_2) = \begin{cases} 1, \text{for any } L_i^{(1)} \in v_1.h-list,\ \exists L_i^{(2)} \in v_2.h-list, \\ \qquad \text{s.t. l}Match\ (L_i^{(1)}, L_i^{(2)}) = 1 \\ 0, \text{otherwise} \end{cases} \quad \textbf{(3)}$$

In Eq.(1), $p_i^{(1)}$ and $p_i^{(2)}$ are the i-th element of pathVector P_1 and P_2. In Eq.(2), $p_i^{(1)}$ is a pathVector in labelVector L_1, $p_i^{(2)}$ is a pathVector in labelVector L_2. In Eq. (3), $L_i^{(1)}$ is a labelVector in v_1.h-list, $L_i^{(2)}$ is a labelVector in v_2.h-list.

5.1 Path Vector Matching Function

In the h-list of vertex v, the most basic component is the path vector from v to u which is an h-neighbor of v. The length of the path vector is equal to the range of information propagation, i.e., h. The i-th element of the path vector represents the number of the paths with i-step between v and u.

If we consider two isomorphic graphs G_1 and G_2. Vertices v_1 and v_2 are from G_1, Vertices $u1$ and $u2$ belong to G_2. u_1 and v_1 are similar, u_2 and v_2 are also similar. We should let the pathVector (v_1, v_2) equal to pathVector (u_1, u_2). However, for subgraph similarity matching, the target graph will have more vertexes and edges than the query graph. So, we should allow the i-th element of pathVector (u_1, u_2) larger than the i-th element of pathVector (v_1, v_2).

The corresponding pseudo-code to calculate the pMatch function between path vector P_1 and P_2 is shown in Algorithm 1.

Algorithm 1. path vector matching

```
Input: pathVector P1, pathVector P2
Output: pMatch(P1, P2)
1. While i<pathVector.size // compare the two vectors by formula (1)
2.    If P 1.get(i)>P2.get(i)
3.       return 0;
4.    End if
5.    i++;
6. End while
7. return 1;
```

5.2 Label Matching

The upper layer of path vector is label vector. All path vectors, which have the same label among neighbors within h step, of a vertex is composed of a label vector.

In Eq.(2), the two label vectors L_1 and L_2 have the same label. For an element P_i, which is a path vector, in L_1, if there is an element R_i, which is a path vector, in L_2, satisfy pMatch (P_i, R_i) = 1, and $R_i! = R_j$, i! = j. Thus we say L_1 and L_2 are successful matched, let lMatch(L_1, L_2)=1. For example: L_1: (a, (1,1), (1,2)), L_2: (a, (1,2), (0,1), (2,2)), so lMatch (L_1, L_2) = 1, and lMatch (L_2, L_1) = 0.

Clearly label matching can come down to a bipartite graph perfect matching problem, which can be solved in polynomial time. We adopt the Hungary algorithm to solve it. In the worst case, the Hungary algorithm can find out bipartite graph perfect match with the time complexity of $O(n^3)$. Therefore, when the number of query graph's vertices is larger and the number of distinct labels is far less than the former, we can directly adopt the Hungary algorithm to solve the problem. However, we do not need to find out the perfect match, but only need to prove its existence. According to Hall's marriage theorem [26], we can simply search the bipartite graph adjacency matrix to verify the existence of a complete match.

Hall's Marriage Theorem: Let G be a bipartite graph with bipartition X and Y. Then there is a perfect matching from X to Y if and only if Hall's condition is satisfied: $|\Gamma(A)| \geq |A|$ for all subsets A of X. Here $\Gamma(A)$ denotes the set of neighbors of the vertices in A.

5.3 Vertex Matching Function

On the basis of the definition of label matching, now we give the function about vertex matching.

First, the h-list of vertex v can be simplified as a vertex vector: v (L_1, L_2 ... L_n), and L is a label vector, namely a set of path vectors which have the same label. When given two vertex vectors: V_1 from the query graph Q, and V_2 from the target graph G. Every label vector L_a in v_1.h-list should be compared with every label vector L_b, which has the same label with L_a, in v_2.h-list. If all of the L in v_1 has their matches, the vertex v_1 and v_2 are successfully matched. We can put v_2 into the candidateList, the candidate match list, of v_1. The corresponding pseudo-code is shown in Algorithm 2. The time efficiency is $O(n^3/k^2)$, n is the total number of neighbors of a given query vertex and target vertex, and k is the number of common labels shared by the two vertices' neighbors.

Algorithm 2. vertex matching

```
Input: vertex v, vertex u
Output: vMatch(v, u)
1. For L1∈v.h-list
2.   For L2∈u.h-list
3.     If lmatch(L1, L2)=0
4.        Return 0;
```

```
5.      End if
6.   End for
7. End for
8. v.candidateList.add(u);
9. Return 1;
```

6 Graph Searches

Based on the vertex matching function vMatch(v, u), we can transform subgraph matching problem into h-list matching problem.

H-list matching problem: Given query graph Q and target graph G. For each query vertex v_q, find out all target vertex u_g satisfied vMatch(v_q, u_g)=1.

In this section, we will present a graph search algorithm HLMA to solve h-list matching problem. The algorithm consists of three components: information propagation, the candidate set search, and similarity calculation. First, we let each vertex of the query graph Q pass out their own label along each path within h step. After the label propagation, each vertex generates his own h-list based on the label information obtained from their h-neighbors. Similarly, the target graph G is also doing the same work. The time complexity of label propagation is exponential correlated with the number of steps. So, we hope h as small as possible. In the latter experiment, we set h=1 or 2, which has achieved better results. After generating the h-list, we compare each vertex from query graph Q and the one from target graph G, resulting in a similarity-matching candidate set for every vertex in Q. Since the h-list contains rich information about the graph, we only need to compare Q and G for one time, thus avoiding repeated propagating label on the target graph and iterative calculation. Finally, for the candidate set of a vertex, we propose a scoring function to get the best match for each vertex. In the latter experiment, the best match which get the highest score and the query vertex usually are similar if the query graph has subgraph isomorphic relationship with the target graph.

The corresponding pseudo-code is shown in Algorithm 3.

Algorithm 3. HLMA

```
Input: query graph: Q, target graph: G, Label propagation steps: h
Output: candidate set for each query vertex, best-match vertices for
        each vertex
1. Do label propagation on Q and G, get h-list for every vertex
2. Choose a start query vertex v0,
     do candidateSetSearch(Q, G, v0),
     get the candidate set for each query vertex.
3. Do similarityScoreCalculation for Q.
      get the best matches for each query vertex
```

6.1 Candidate Set Generation

Some algorithms based on label propagation often use an iterative method to calculate the matching vertex [11]. This often requires multiple scans on target graph G and repeated propagating information. Because the number of vertexes and edges in G more large, iterative algorithms often lead to an increase in time complexity. We hope that the running time of the search algorithm can rely more on the query graph Q, rather than the target graph G. Because of the powerful pruning ability of h-list, HLMA only needs a single label propagation and a single scan on the graph G. In addition, blindly enumerating all possible query-target vertices will bring a mass of unnecessary calculation. So we adopt a kind of breadth-first search strategy for calculating vMatch(v, u) among the neighborhoods of matched query-target vertices. When finished candidate set searching, every query vertex v_i will get a candidate set U_i={all u satisfied vMatch(v, u)=1}.

The corresponding pseudo-code is shown in Algorithm 4. The basic idea is given a query vertex v and target vertex u, u is a candidate match of v, when the following condition is satisfied:

(1) vMatch(v, u)=1.

(2) Each v_n in v's neighbor set has a candidate match in u's neighbor set.

Algorithm 4: candidateSetSearch

```
Input: query graph Q, target graph G, first query vertex v0
Output: candidate set for all query vertices
1. List 1.add(v0);
2. List 2.add(all query vertexes);
3. While (list1 is not empty)
4.         For all v in list1，all u in list2
5.         If vMatch(v, u)=1
6.          u.neighbors→temlist2;
7.         End if
8.         If v.neighbors have not been matched
9.            v.neighbors→temlist1;
10.        End if
11.            Remove v from List1
12.        End For
13.        List1=temlist1;
14.   List2=temist2;
15. End While
```

6.2 Similarity Calculation

Through a scan of the target graph, each vertex v_i in the query graph Q gets a candidate match set U. When there are many vertexes in U, it is very important to evaluate these candidate matches. Intuitively, we propose two criteria to help finding better matches:

Criteria 1: if we match v with u, v_i with u_i. v_i is one of v neighbors, u_i is one of u neighbors. When the similarity between v and u is increased, the similarity between v_i and u_i should be increased too.

Criteria 2: if we match v with u. The more neighbors of v match the neighbors of u, the more similar v and u should be.

Thus, we propose a formula to calculate the similarity score between query vertex and its candidate match:

$$\text{similarity}(v,u)=\frac{1}{\varepsilon_{vu}}\sum similarity(v_i,u_i) \quad v_i \in \text{neblist}(v),\ u_i \in \text{nebList}(u)\cap \text{candidate}List(v_i) \tag{4}$$

As shown in Algorithm 5, the similarity score between v and u is depends on the sum of the similarity score between their neighbors. This sum divided by a factor ε_{vu}, which is usually depends on v or u, will be the similarity score we want. Obviously, ε_{vu} and initial similarity value should be assigned to appropriate values. Since the result score is deeply influenced by them. Suppose u is the exact match of v. If the initial similarity score is 1 and let ε_{vu} equal to the degree of v. After the calculation is performed once, the result similarity score of (v, u) will be 1.

It is noteworthy that we do not find isomorphic subgraphs in polynomial time. There are two reasons: First, similarity (v, u)=1 does not mean that u is the exact match of v. Secondly, 1 is not the upper bound of similarity score. We have no intention to define the score as probability. In order to obtain better results, in later experiments, we will set ε_{vu} as the number of u's matched neighbors, and calculate similarity (v, u) with two iterations of Eq. (4).

Algorithm 5. similarityScoreCalculation

Input: query graph Q, candidate sets of every query vertex
Output: similarity score between every query vertex and its candidate matched vertices

```
1. For all v in Q
2.         For all u in candidateList of v, all vi in nebList of v
3.           If ui is a neighbor of u and a candidate match of vi
4.             similarity(v, u)+=similarity(vi, ui);
5          End if
6.         End for
7.         similarity(v, u)=similarity(v, u)/εvu;
8. End for
```

7 Experiment

Our code was implemented in Java (JDK8). The experiments are performed on 1.7 GHz Intel Core i5 machine with 4GB memory running macOS X.

7.1 Search Quality

The test data we used are from Ca-HepPh [27] and ego-Facebook [28]. Table 1 shows the parameters of the two graphs. We randomly assigned labels to the two graphs and adjust the size of label set. We take them as target graph G_1 and G_2. Then, we randomly select 100 trees, with $|V|$=8, from G_1 and G_2, respectively, as query graphs. Every vertex in these query graphs retains its own ID (a unique attribute for each vertex) in the target graphs. Then we search the 100 trees in G_1, where they were extracted from, and another 100 trees in G_2.

In Table 1, AD is average degree. ND denotes network diameter. ACC stands for average clustering coefficient. ASPL represents average shortest path length.

Table 1. The parameters of target graphs

Graph	Vertices	Edges	AD	ND	ACC	ASPL
ego-Facebook	4039	88234	43.691	8	0.618	3.693
Ca-HepPh	12008	118489	19.738	13	0.698	4.673

If a query vertex v and its best match u have the same ID, obviously u is the exact match of v. In HLMA, the best matching of a query vertex is the match vertex with the highest score. We use N_A denotes the number of the best matching vertices which have the same ID with their query vertices, N_R denotes the sum of the best matching vertices which have different ID with their query vertices, N_q represents the number of query vertices.

We use two indicators to evaluate the results of the algorithm. One is the accuracy rate, namely N_A/N_q. The other one is redundancy rate, namely N_R/N_q.

Fig. 4 and Fig. 5 show that the experimental results under different size of label set of G_1 and G_2. In addition, let h=1.

7.2 Label Density

Now let us consider a problem. How the increasing number of vertexes or edges will impact the accuracy rate and redundancy rate? We designed two types of experiments. A type of experiment is to keep the average degree of target graph constant, but the number of vertexes changes. Another experiment is to keep the number of vertexes constant, but the average degree of target graph changes. For this goal, we randomly generate several new target graphs, Table 2 depicts the parameter characteristics of these graph.

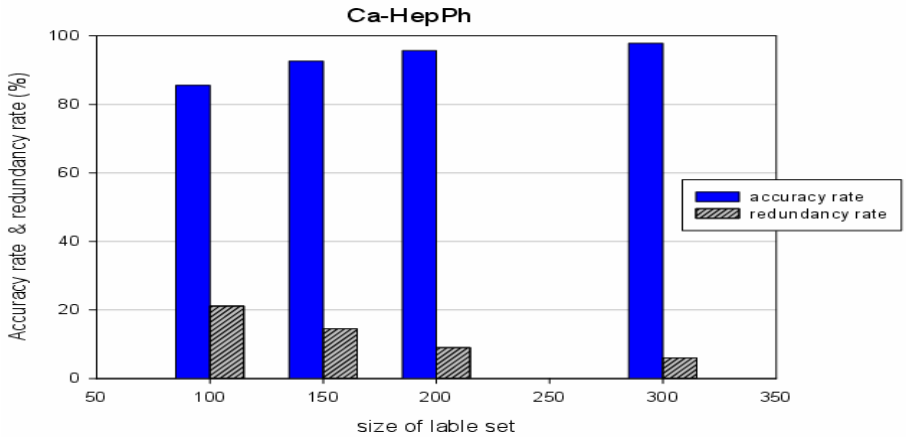

Fig. 4. Accuracy rate and redundancy rate of Ca-HepPh

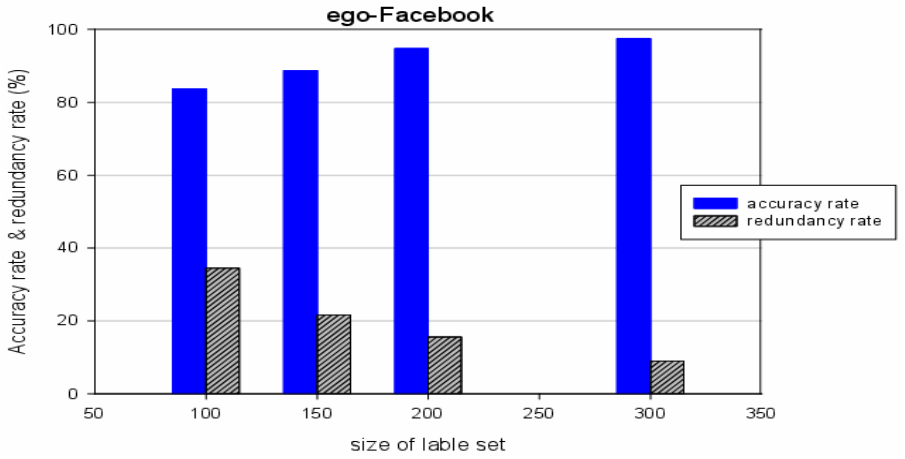

Fig. 5. Accuracy rate and redundancy rate of ego-Facebook

Table 2. The parameters of target graphs (random graph)

Graph	Vertices	Edges	AD	ND	ACC	ASPL
G1	1000	10000	20	4	0.019	2.641
G2	3000	30000	20	4	0.007	2.945
G3	6000	60000	20	4	0.003	3.203
G4	8000	80000	20	5	0.002	3.323
G5	1000	20000	40	3	0.04	2.151
G6	1000	30000	60	3	0.059	1.964
G7	1000	40000	80	3	0.079	1.921

We did the same search experiment on these random graphs just like what we did on the real-world graphs. We found that there was no significant impact on the accuracy and redundancy rate of the algorithm by simply increasing the number of vertices or edges. However, the size of the label set/vertex degree, we called label density, is the key factor that affects the accuracy and redundancy rate of HLMA. As shown in Fig. 6, Fig. 7, Fig. 8, and Fig. 9, we have made several experiments under different label density. It can be seen that the higher the label density, the higher the accuracy rate and the lower the redundancy rate. When the label density was a constant, the impact, increasing the number of vertexes and edges of the target graph, on query accuracy and redundancy rate is not obvious.

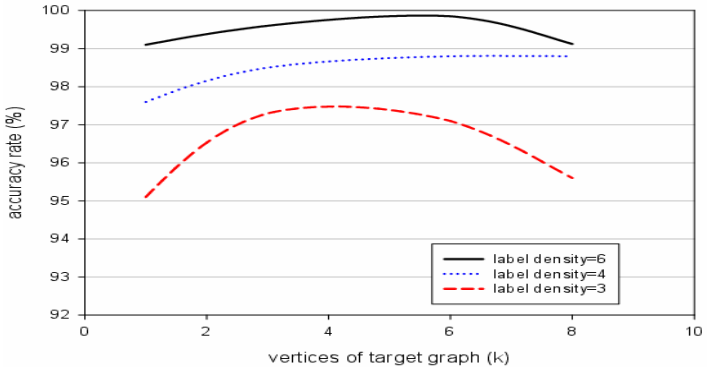

Fig. 6. Accuracy rate and the number of vertexes in target graph

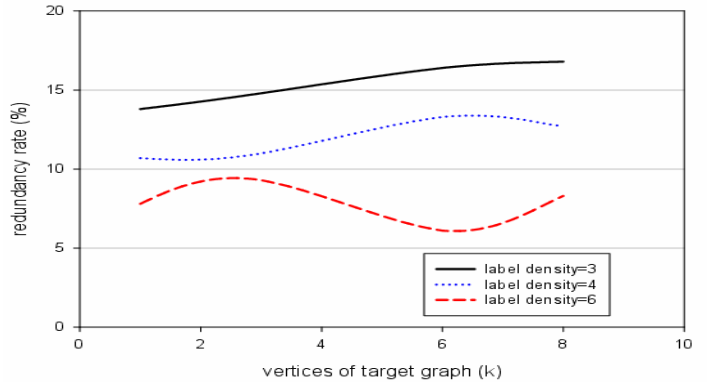

Fig. 7. Redundancy rate and the number of vertexes in target graph

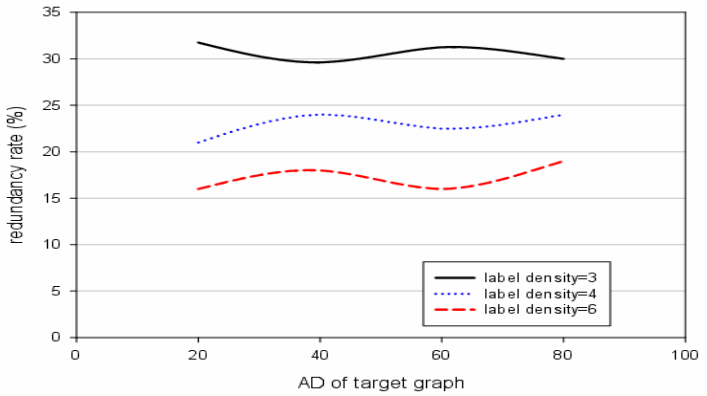

Fig. 8. Accuracy rate and the AD in target graph

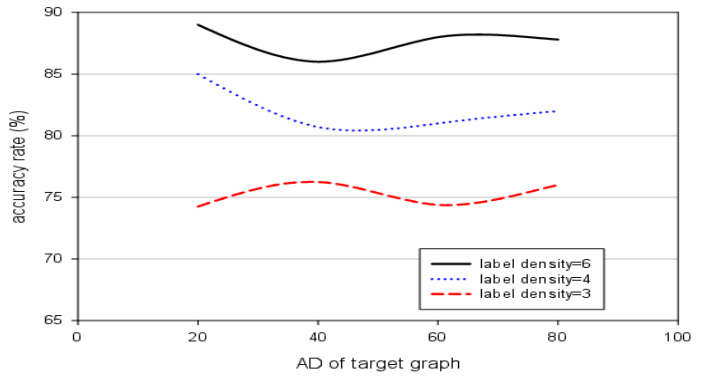

Fig. 9. Redundancy rate and the AD in target graph

7.3 Time Efficiency

The execution time of the above algorithms can be divided into two parts: h-list generation time and graph search time. Fig. 10 depicts the h-list's generation time changed with the number of vertexes in target graph changed. Fig. 11 shows the relationship between graph search costs with the number of vertexes in target graph. Among them, the query graphs are the trees with 20 vertexes. The average degree of target graphs is 20 and h=1. We can see that for the target graph with ten thousand vertexes, the search time is only 14.7ms. In fact, h-list generation is the most time-consuming part of HLMA. As shown in Fig. 11, the generation time of h-list is almost linear.

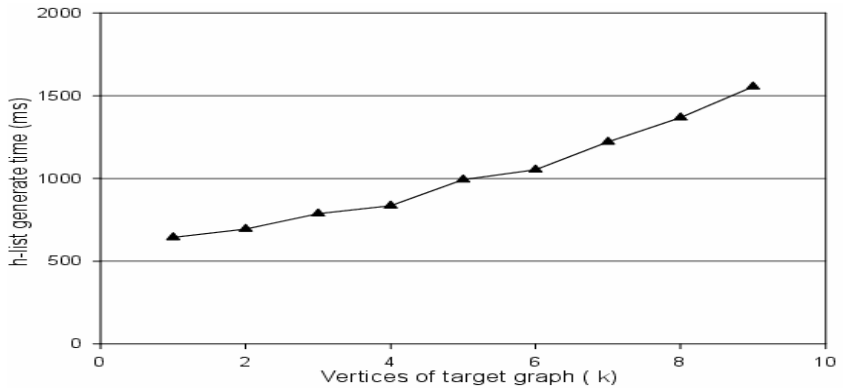

Fig. 10. h-list generation time and the number of vertexes in target graph

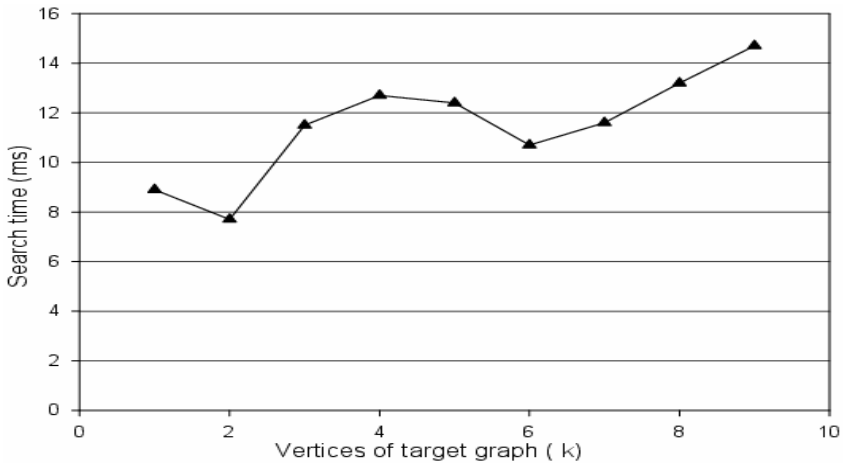

Fig. 11. Graph search time and the number of vertexes in target graph

7.4 Compared with Path Layer Matrix

As we all know, the path layer matrix is a useful graph invariant, which contains rich graph information. For graph whose vertexes are less than 11, it is known that the same path layer matrix is closely related to graph isomorphic. However, for the graph with 14 vertexes, there are examples of non-isomorphism graphs with the same path layer matrix. Fig. 12 and Fig. 13 are examples of such graphs. Fortunately, these cases will be distinguished by HLMA very soon. This shows that h-list is more powerful than the path layer matrix.

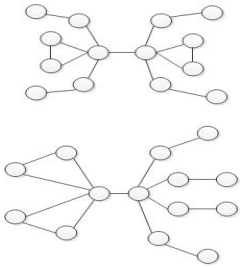

Fig. 12. Non-isomorphism with same path layer matrix

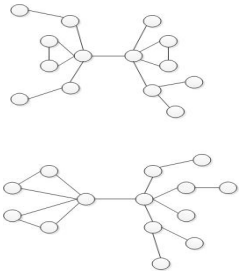

Fig. 13. Non-isomorphism with same path layer matrix

8 Conclusion

In the paper, we proposed a novel approximate subgraph matching algorithm HLMA based on neighborhood information. We conduct experiments on real world graphs and randomly generated graphs. The experimental results show the time complexity of HLMA is a roughly linear relationship with the number of vertices of a target graph, and the accuracy of HLMA is not sensitive to change in scale of graph, but the value of label density. When label density is higher than 2.5, the accuracy rate of HLMA is close to 100%.

Inexact matching has more significance in the analysis of social networks. For inexact matching problem, we need to loosen the matching conditions mentioned. Here we have two kinds of effective means to achieve this goal. (1) As proposed in Ness and Sigma, using a cost function based on edge missing. We can add up all the path vectors in a label vector set, then calculate the matching cost according to edge missing. The more edge missed, the higher the price is. (2) We can take advantage of the hierarchical nature of h-list and adopt the idea similar with SAGA. Label gap will be introduced in the label matching step as a penalty factor. Unlike exact match, in the scoring stage, each match in a candidate set is no longer assigned to the same initial similarity score, but a different value according to the edge missing or penalty factor. In future work, we will further improve and test HLMA for the inexact matching.

Acknowledgements

This research was supported by the Key Projects of Science and Technology Research of Hebei Province Higher Education [ZD2017304], and is supported by the Fundamental Research Funds for the Central Universities (2019JBM025), and was funded by Science and Technology Plan of China Administration of Market Supervision [2019MK003].

References

[1] E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF. <https://www.researchgate.net/publication/225070173_SPARQL_query_language_for_RDF>, 2006.

[2] J. Ingraham, V. Garg, R. Barzilay, T. Jaakkola. Generative models for graph-based protein design, in: Proc. Neural Information Processing Systems (NeurIPS), 2019.

[3] T. Price, F. I Pena III, Y.-R. Cho, Survey: enhancing protein complex prediction in PPI networks with GO similarity weighting, Interdisciplinary Sciences Computational Life Sciences 5(3)(2013) 196-210.

[4] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, International Journal of Pattern Recognition and Artificial Intelligence 28(1)(2014) 1450001.

[5] R. Singh, J. Xu, B. Berger, Global alignment of multiple protein interaction networks with application to functional orthology detection, in: Proc. National Academy of Sciences 105(35)(2008) 12763-12768.

[6] M. Mongiovi, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, R Sharan, Sigma: a set-cover-based inexact graph matching algorithm, Journal of Bioinformatics and Computational Biology 8(2)(2010) 199-218.

[7] X. Xu, X. Wang, K.M. Kitani, Error correction maximization for deep image hashing, in: Proc. British Machine Vision Conference (BMVC), 2018.

[8] Y. Tian, R.C. Mceachin, C. Santos, J.M. Patel, SAGA: a subgraph matching tool for biological graphs, Bioinformatics 23(2)(2007) 232-239.

[9] H. He, A.K. Singh. Closure-tree: an index structure for graph queries, in: Proc. 22nd International Conference on Data Engineering, 2006.

[10] H. Tong, C. Faloutsos, B. Gallagher, T. Eliassi-Rad, Fast best-effort pattern matching in large attributed graphs, in: Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007.

[11] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, S. Tao, Neighborhood based fast graph search in large networks, in: Proc. ACM SIGMOD International Conference on Management of data, 2011.

[12] H. He, A.K. Singh, Graphs-at-a-time: query language and access methods for graph databases, in: Proc. SIGMOD, 2008.

[13] J. Lee, W.-S. Han, R. Kasperovics, J.-H. Lee, An in-depth comparison of subgraph isomorphism algorithms in graph databases, PVLDB 6(2)(2013). DOI: 10.14778/2535568.2448946.

[14] X. Yan, P.S. Yu, J. Han, Graph indexing: A Frequent Structure-based Approach. In: Proc. SIGMOD, 2004.

[15] H. Shang, Y. Zhang, X. Lin, J.X. Yu, Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, PVLDB 1(1)(2008) 364-375.

[16] J. Cheng, Y. Ke, W. Ng, A. Lu. Fg-index: towards verification-free query processing on graph databases, in: Proc. SIGMOD, 2007.

[17] J. Liu, B. Xu, X. Xu, T. Xin, A link prediction algorithm based on label propagation, Journal of Computational Science 16(2016) 43-50.

[18] C. Choi, Y.S. Lee, S.E. Yoon, Discriminative subgraphs for discovering family photos, Computational Visual Media 2(3)(2016) 1-10.

[19] X. Kong, P.S. Yu, X. Wang, A.B. Ragin, Discriminative feature selection for uncertain graph classification. <https://arxiv.org/pdf/1301.6626.pdf>, 2013.

[20] M. Han, H. Kim, G. Gu, K. Park, W. Han, Efficient subgraph matching: harmonizing dynamic programming, adaptive matching order, and failing set together, in: Proc. SIGMOD, 2019.

[21] S. Sun, Q. Luo, Scaling up subgraph query processing with efficient subgraph matching, in: Proc. IEEE 35th International Conference on Data Engineering, 2019.

[22] Y. Tian, J.M. Patel, TALE: a tool for approximate large graph matching, in: Proc. ICDE, 2008.

[23] Y. Wang, H. Wang, J. Li, Efficient graph similarity join for information integration on graphs, Frontiers of Computer Science 10(2)(2016) 317-329.

[24] B. Xu, T. Xin, Y. Wang, Y. Zhao, Local random walk with distance measure, Modern Physics Letters B. 27(8)(2013) 1-9.

[25] V.A. Skorobogatov, A.A. Dobrynin, Metric analysis of graphs, Commun. Math. Chem. 4(23)(1988) 105-151.

[26] J. Leskovec, J.J. Mcauley, Learning to discover social circles in ego networks, in: Proc. Advances in Neural Information Processing Systems, 2012.

[27] J. Leskovec, High energy physics - phenomenology collaboration network. <http://snap.stanford.edu/data/ca-HepPh.html>.

[28] J. Leskovec, Social circles: Facebook. <http://snap.stanford.edu/data/egonets-Facebook.html>.