A Comparative Study of Subgraph Matching Isomorphic Methods in Social Networks

TINGHUAI MA1,2 (Member, IEEE), SIYANG YU1, JIE CAO 1, YUAN TIAN3, ABDULLAH AL-DHELAAN3, and MZNAH AL-RODHAAN3

1School of Computer Software, Nanjing University of information science Technology, Jiangsu, Nanjing 210-044, China(e-mail: thma@nuist.edu.cn)
2CICAEET, Nanjing University of information science Technology, Jiangsu, Nanjing 210-044, China.
3Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh 11362, Saudi Arabia

Corresponding author: Tinghuai Ma (e-mail: thma@nuist.edu.cn).

This work was supported in part by the National Science Foundation of China (No. 61572259, No.U1736105) and the National Social Science Foundation of China (No.16ZDA054). The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group no. RGP-VPP-264.

Abstract

With the fast development of social networks, more and more data has been generated. Finding useful information among these data is important. Subgraph matching is the method that can be used in social networks for social search, recommender systems and so on. In this paper, we choose five typical graph matching methods which are VF2, SPath, TurboISO, BoostIso and RI to value their performance and scalability in social networks. These methods are verified by three social network datasets whose node’s number vary from thousands to millions. According to the experiments, we can find that VF2 and RI is applicable for rather small graphs while SPath performs better in large graph when average degree of graph is small. When graph with high average degree, TurboISO and BoostIso performs better. What’s more, we also compare the efficiency of different matching order when searching. The order choosing strategy proposed by RI is better.

Index Terms

social network, subgraph matching, matching order

I. INTRODUCTION

Graph is a kind of data structure that can represent relationships between entities consisting of a set of vertices and edges [1]. It is an important data type in social networks, protein-protein interaction and so on. In social networks, entities can be represented by vertices and edges are used to represent relationships between entities. Subgraph matching aims to find the same structure as the query graph from data graph which is a NP-hard problem [2]. In social networks, subgraph matching methods can be used in several areas of social networks such as recommender systems, social search and so on [3]–[7].

Recommender systems. Graph matching can be used to find a team or a talented person in LinkedIn which companies required. For example, as is shown in Fig. 1, if a company wants to found a team for marketing management, they need to recruit a Marketing Director, a Marketing Clerk, a Regional Manager and a Planning Clerk. The relationships among these employees required by this company can be transformed into a query graph, as show in Fig. 1. In this graph, edges represent the leadership and cooperation relationships among people, nodes represent job seekers and their professional fields are represented by node labels. Adopted subgraph matching, the query graph can be found in database of LinkedIn, and thus the company can find staffs meeting the demand. A system called ExpFinder [8] has been released to do this job.

Social search. Social search means that an entity with specific types of connections or attributes can be found by using subgraph matching [9]. It can be used to judge the kinship patterns among people. What’s more, it can be used in academic field. For example, we can form the relationships between experts we want to find into a query graph and find the same structure by subgraph matching in academic data sets such as paper reference network. Therefore, cooperation relationships among experts can be...
found.

**Group finding.** As shown in Fig. 2, a social network contains many groups [10]. By finding a subgraph containing partial information of a group in data graph [11], we can easily find the group we want. That can be used to find criminal gangs. Members of the same criminal gang must have close connection, that is to say, a criminal gang must be in a same community. Converting the information of convicts which police has known to a graph, we can use graph matching methods to find the community which containing the criminal gang [12].

II. BACKGROUND

A. PROBLEM DEFINITION

Topology of a social network data can be presented by a graph, where the formal definition of graph is described as Definition 1.

**Definition 1. (Graph)** A graph is a set \(G(V, E, L)\), where \(V\) represents nodes in graph, \(E\) represents edges in graph and \(L\) represents labels of nodes in graph. \(|V_g|\) represents the number of vertices in graph \(G\) and \(|E_g|\) represents the number of edges in graph \(G\).

Because dataset size of social network is large due to its great number of users, only using DFS (Depth-First-Search) method to searching query graphs is inefficient. Therefore, most of methods speed up the searching process by construct index for data graph. Index construction is defined as follows.

**Definition 2. (Graph Isomorphism)** Assuming that there is a data graph \(G(u)\) and a query graph \(Q(v)\), graph isomorphism problem is to find the same structure as \(Q\) in \(G\) which is a bijective function \(f : u \rightarrow v\), where \(v\) represent nodes in \(Q\) and \(u\) represent nodes in \(G\), and \(l(v) = l(u)\) which means that the label of \(v\) is equal to the label of \(v'\). For example, in Fig. 3, \(u_1\) is isomorphic to \(v_1\), \(u_2\) is isomorphic to \(v_2\), \(u_3\) is isomorphic to \(v_3\), \(u_4\) is isomorphic to \(v_4\) and \(u_5\) is isomorphic to \(v_5\).

The common procedure of subgraph matching algorithms can be described as Algorithm 1. Line 3 to line 6 is the filter period. During filter, candidate set \(C(v)\) of each query node \(v\) is initialized firstly. After that, different pruning strategies...
are used to prune out invalid nodes. As for searching process ranging from line 7 to line 11, matching state \( M(G) \) are initialized first. Then, matching order of pattern \( p \) (path or node) is decided. At last, all embeddings are to be found starting from line 8 where isJoinable\((p_i)\) is used to judge whether pattern \( p_i \) can be added into \( M(G) \) or not.

**Algorithm 1** Exact Subgraph Matching Method

1: Input: Data graph \( G \) and query graph \( Q \)
2: Output: All embeddings of \( Q \) in \( G \)
3: Initialize \( C(v), M(G) \)
4: \( \text{while } \forall v \in V_Q \text{ do} \)
5: \( \text{Pruning invalid candidate nodes in } C(v) \)
6: \( \text{end while} \)
7: Finding matching order of pattern \( p \)
8: \( \text{while } \mid M(G) \mid \text{ doesn’t equals to } \mid V(Q) \mid \text{ do} \)
9: \( \text{if } \text{isJoinable}(p_i) \text{ then} \)
10: \( \text{Match}(p_{i+1}) \)
11: \( \text{end if} \)
12: \( \text{end while} \)

**B. RELATED WORK**

Ullmann et al. first proposed a method to solve the graph isomorphism problem in 1976 [21] without any pruning and indexing strategies. Because there are too many invalid node need to be judged in Ullmann which leads to large redundant computation, VF2 was proposed [22] to solve this problem. Different from choosing the matching order randomly, VF2 choose the node which would be matched next among the neighborhood of nodes which have been matched. What’s more, several simple pruning strategies are applied to reduce the number candidate matching nodes before searching process. These lead to a notable improvement of performance.

To find isomorphic graphs in large network fast, methods as VF2 with poor indexing strategy can’t meet this requirement. To optimize it, SPath [23] implement path-at-a-time pattern during the searching process. It decomposes the query graph into several paths and finds embeddings of each path which would be joined later. Most of algorithms introduced above focus more on searching pattern to reduce the time complexity. Turbo\textsuperscript{ISO} [24] and BoostIso [25] try to find a greater matching order to make subgraph matching more efficient. What’s more, compressing graph method is implemented to reduce the space complexity. Candidate region exploration is proposed in Turbo\textsuperscript{ISO} to find a good matching order among query nodes. And query graph is compressed in Turbo\textsuperscript{ISO}. BoostIso first transforms the data graph into a hyper graph. What’s more, a proper matching order among candidate nodes of a query graph is determined. BoostIso [25] is a method that is proposed to make the matching process faster, it can be combined with another subgraph matching algorithms and have rather good performance than use the matching algorithms merely. What’s more, Bonnici V et al. [26] proposed RI algorithm, and further optimize the matching order of questions after that.

Also, in recent years, some methods are proposed to solve more complex graph matching problem. For example, Shemshadi et al. [27] proposed a method for graphs with multi-Labeled nodes. And graph matching for integrating multiple data sources are discussed by Zhang et al. [28]. Methods mentioned above first are algorithms that find query graph in a big data graph. Using subgraph matching to answering natural language questions over knowledge graphs is solved in [29]. There is another kind of indexing subgraph matching algorithm is used to find embeddings of query graph among a set of data graph [30]–[32]. Most of these methods mine frequent substructures among graphs and construct index for them. We don’t concentrate on this kind of algorithms, thus detailed development of it is omitted there.

Different from existing research, in this paper, query order and filtering strategies are two aspects we discuss to compare the performance of existing subgraph isomorphic algorithms. As for the matching order, we compare the performance of each method by finding subgraph from the same candidate set. What’s more, we also use several real social networks datasets to compare the performance and scalability of five typical methods. Therefore, advantages and disadvantages of each algorithm are found which is useful to find a suitable one to apply to social networks according to different situations.

**III. FILTERING AND VERIFYING OF SUBGRAPH MATCHING METHODS**

Subgraph matching is a NP-complete problem, it is time-consuming to enumerate all possibilities. Therefore, "filter and verify" is the framework for subgraph matching to find all the isomorphic subgraphs on a large graph.

**A. FILTER STRATEGY**

Filter is used to find vertices in the candidate matching set that are invalid. If vertices in the candidate matching set...
that do not meet the matching conditions can be pruned out as early as possible, the search efficiency of the algorithm will be greatly improved. Therefore, it is very important to propose a good filtering method.

**VF2.** Pruning strategy was first proposed in VF2. Before that, the isomorphic subgraphs are found by enumeration. A node \( u \) in data graph \( G \) has the possibility to match with a node \( v \) if the degree of node \( u \) is greater than or equal to the degree of node \( v \). Degree of a vertex is considered by VF2.

**SPath.** With the increase of the size of graphs, pruning methods as VF2 adopted which only consider the degree of a vertex cannot prune many invalid vertices if the structure of graph is complex. SPath improves its pruning ability by constructing index for each node in data graph using \( k \)-neighborhood signature. In detail, \( NS(u) \) is defined to save neighborhood labels and tags of a node \( u \) within \( k \) distance. For example, in Fig. 4, we assume that \( k \) equals to 2, \( NS(u_1) = \{a:\{u_1\}\}, \{a:\{u_5, u_6\},b:\{u_2\}\}, \{b:\{u_1\},c:\{u_3\},d:\{u_4\}\}\}. A data node \( u \) can be added into candidate sets of a query node \( v \) only if \( NS(v) \subseteq NS(u) \), which can greatly reduce the size of candidate sets.

**FIGURE 4:** An example of neighborhood signature

Though it is time-consuming to construct neighborhood signature for each node in data graph, searching time can be reduced apparently due to effective pruning.

**Turbo\textsubscript{ISO}.** Two pruning strategies are adopted by Turbo\textsubscript{ISO}: (1) If \( u \) is the candidate matching node of \( v \), then the number of the neighborhood of \( u \) must be no less than that of \( v \); (2) If \( u \) is the candidate matching node of \( v \), the adjacent label set of \( v \) must be contained by the adjacent label set of \( u \).

What’s more, Han et al. thinks that nodes in query graph which share the same neighbors can be represented by one vertex, which will make it more efficient. Therefore, they compress the query graph by using a node to represent several vertices. NEC tree is designed to rewrite query graph to avoid redundant enumeration. Vertices in the tree are called NEC vertex where a NEC vertex contains all query vertices with the same labels and neighborhoods. A ranking function that equals to the frequency of the label of \( v \) over the degree of vertex \( v \) as shown in equation 1 to find the root node of NEC tree.

\[
\text{Rank}(v) = \frac{\text{freq}(q,L(v))}{\text{deg}(v)} \tag{1}
\]

After choosing the start node, a depth first searching process is implemented to build NEC tree. Nodes having the same neighborhood label set and label are represented in one node in NEC tree. For example, in Fig. 5(a), the neighborhood of \( v_2 \) and \( v_4 \) is \( v_1 \), therefore, they can be represented by one node which is \( v_1' \) in Fig. 5(b), where Fig. 5(a) is a query graph and Fig. 5(b) is the consistent NEC tree.

**FIGURE 5:** An example of NEC tree

After getting the NEC tree, there still another filter strategy: the number of nodes in candidate set of a NEC vertex \( v' \) must be no less than the number of nodes that represented by \( v' \).

**BoostIso.** During the filter stage, BoostIso compresses both data graph and query graph compared with Turbo\textsubscript{ISO}. Actually, BoostIso can be combined with other subgraph matching algorithms to make them more efficient.

Syntactic containment and syntactic equivalence are two principles that BoostIso abide by to transform data graph into hyper graph. A vertex \( u \) syntactic contain another vertex \( v \) whose label is the same as \( u \) if and only if the neighborhood node set of \( v \) is contained by the neighborhood node set of \( u \). For example, in Figure 6(a), the neighborhood node set of \( u_2 \) is \( \{u_1, u_5\} \) and the neighborhood node set of \( u_4 \) is \( \{u_1\} \), therefore, \( u_4 \) is syntactic contained by \( u_2 \). Similarly, syntactic equivalence means that the neighborhood node set of \( v \) equals to that of \( u \). Nodes syntactic equaling to each other are saved in a vector \( SEC(v) \).

**FIGURE 6:** An example of syntactic containment

BoostIso converts the data graph into an hyper graph which is defined as: \( G_{sh} = (V_{sh}, E_{sc}, E_{se}, \Sigma_{sh}, L_{sh}). V_{sh} \) is the set of hyper nodes where every \( SEC(v) \) is a hyper node, thus the number of hyper nodes are \( |SEC(v)| \). \( E_{sc} \) is a set of undirected edges of \( G_{sh} \). \( E_{se} \) is a set of directed edges. Only if a hyper node \( h \) syntactic contains \( h' \) can there exists a directed path between them. And \( L_{sh} \) is the label set of data graph. By constructing hyper graph, the
size of data graph has been reduced greatly. Therefore, as for pruning, only the hypernodes labeled with \( L_\alpha(u) \) and having no SC-Parents can they be added into the candidate set.

**RI.** There are two main filter principles used by [26] which are introduced above. Actually, compared with filtering, RI focuses more on the determination of matching order.

### B. MATCHING ORDER

1) **Framework of searching**

The procedure of searching follows the framework as algorithm 2 shows. We can see that if the candidate set is same, matching order is of great importance for the performance of searching. A good search order can greatly improve the efficiency of search.

**Algorithm 2 Algorithm of searching**

1: Input: Data graph \( G \), query graph \( Q \) and Candidate set for each query vertex
2: Output: All embeddings of \( Q \) in \( G \)
3: Finding matching order
4: SubgraphSearch(v):
5: \( \text{while } |M(G)| \) doesn’t equals to \( |V(Q)| \) do
6: \( \text{for } i \in C(v) \) do
7: if isJoinable(i) then
8: updateState()
9: SubgraphSearch(v+1)
10: end if
11: restoreState()
12: end for
13: end while

The strategies for choosing the matching order of the typical subgraph matching algorithms are introduced as below.

**VF2.** Instead of finding candidate nodes for each query node and searching by breath first search in a random order of query nodes, VF2 only choose the first node to be matched randomly. After that, each node to be selected should be connected with at least one node that are already matched which makes it more efficient compared to ULLMANN.

**Spath.** As for searching process, different from other algorithms which match one query node per time, path-at-a-time is used in SPath, which means that SPath matches more than one node per time. To make sure the counterpart of the shortest chosen paths of query graph in data graph is also the shortest, SPath first decompose query graph into several sets of shortest paths originated from each query node \( v \) according to equation 2. Equation 2 calculate value of distance \( k \) when the number of \( k \)-neighborhood label is different at first time. That is to say, the longest paths of shortest paths sets equals to \( k^* \). After that, paths with smaller size of candidate sets and shorter size of paths will be chosen eventually. What’s more, chosen paths should cover all query node.

\[
k^* = \arg \min_{k} \left\{ \left| \bigcup_{k<k_0} S_k(u) \right| - \left| \bigcup_{k<k_0} S_k(v) \right| > 0 \right\}, \forall l \in \Sigma
\]  

\[\text{(2)}\]

**Turbo_ISO.** Different candidate embeddings of a query graph \( Q \) in data graph \( G \) may locate in different region in \( G \). Due to the graph structure of each region is different, during the search step, if the subgraph in each region is found using the same matching order, it will result in redundant computation. That is to say, it is more efficient if matching order of each region is determined according to their own structural feature respectively. To find a good matching order for each candidate region, query vertices with less candidate vertices are favored.

What’s more, enumeration of all vertices in data graph and query graph to find the embeddings is unnecessary. A COMB/PER(Combination/Permutation) strategy is taken to find the embeddings. COMB means that Turbo_ISO choose \(|\text{NEC}|\) candidate nodes to judge every time. \(|\text{NEC}|\) means the number of vertices that a NEC node contains. Meanwhile, if the candidate nodes selected can be matched with the NEC node, the PER strategy then permute the possible matching. For example, in Fig. 6, the vertex set of a NEC vertex \( v_1 \) is \( \{v_2, v_3\} \) and \( C(v_2') \) equals to \( \{u_2, u_3, u_4\} \). COMB choose two candidate nodes every time. If \( u_2 \) and \( u_3 \) are chosen and they can matched with \( v_2 \), then PER will product all the matching pairs which is \( \{v_2-u_2, v_3-u_3\} \) and \( \{v_2-u_3, v_3-u_2\} \). Computation is greatly reduced due to NEC tree and the COMB/PER strategy.

**BoostISO.** As for the matching order of candidate nodes of a query node, dynamic relationship table is proposed to determine it. When initializing the candidate matching set \( C(v) \), only vertex \( u \) which isn’t syntactic contained by any vertex can be added into the candidate matching set. Otherwise, only after all vertices containing \( u \) have been matched can the node \( u \) be added into the candidate set. This strategy makes it possible to avoid duplication computation early. This is because that if a vertex \( u \) is pruned, all the nodes which \( u \) contains will not be added into the table. In other words, if a vertex \( u \) is pruned, all vertex contained by \( u \) can be pruned, too.

**RI.** Bonnici V et al. propose that it is better to verify the node which connected to the matched vertices earlier. In detail, they compute three value of query nodes to determine the matching order which are \( V_{m,vis} \), \( V_{m,neig} \) and \( V_{m,unv} \). \( V_{m,vis} \) contains the matched nodes which \( m \) is connected with. \( V_{m,unv} \) contains the neighborhood nodes of \( m \) which are connected with the matched nodes. And \( V_{m,vis} \) represents the number of the neighbors of \( m \) which don’t belong to \( V_{m,vis} \) and \( V_{m,neig} \).

### C. COMPLEXITY

In this subsection, we compare the time complexity of five typical subgraph matching methods. Actually, besides the
time complexity of searching, the performance of searching are influenced by matching order and the scale of candidate sets. Therefore, we only compare the time complexity of filtering.

<table>
<thead>
<tr>
<th>Filter Name</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>VF2</td>
<td>(O(</td>
</tr>
<tr>
<td>SPath</td>
<td>(O(</td>
</tr>
<tr>
<td>TurboISO</td>
<td>(O(</td>
</tr>
<tr>
<td>BoostISO</td>
<td>(O(</td>
</tr>
<tr>
<td>RI</td>
<td>(O(</td>
</tr>
</tbody>
</table>

### IV. EXPERIMENTS

In this section, we compare the performance of five typical subgraph matching algorithms mentioned above. What’s more, we compare the performance of typical determining matching order strategies by adopting the same searching process under the same candidate sets. All of algorithms to be compared in this section are implemented in C++. And all experiments were conducted on laptop with a 1.70GHZ processor and 8GB of RAM.

**Data set.** We use three real social network data sets to test the performance and scalability of typical subgraph matching methods. All datasets of social networks we have used are downloaded from SNAP, where they provides tons of large social network datasets including DBLP, Youtube and so on [40]. Datasets there are undirected graph containing nodes and undirected edges. Average degree represents the average neighborhood number of a data graph which indicates if a graph is sparse. We vary the size and degrees of the data graphs to compare the performance and scalability of these methods. Three real datasets are used in our experiment including Email-Eu, Email-Erozon and DBLP. Table 2 shows detailed information of datasets we used. As we can see from Table 2, size of these three datasets becomes larger gradually where the size of Email-Eu is smallest and size of DBLP is the largest. In addition, average degrees of these four datasets are different where Email-Eu is denser than others and DBLP is the sparsest among these four datasets. All of these data graphs are undirected graphs with vertex labels.

To compare the performance of different matching order under the same candidate sets, GraphGen [42] are used to generate synthetic graphs. The parameter of the synthetic graph we used is shown as table 3. We generate three networks using the benchmark for experiment. For subsequent comparison experiments, we randomly assign a label value to each node. There are 50 distinct labels.

**Query set.** As for the query graph, we test the matching performance by varying the size of graphs. We choose the size of query graph ranging from 4 to 24, which means that the number of edges of query graph are 4, 8, 12, 16, 20 and 24 respectively. 1000 connected query subgraphs of each size are chosen randomly. And the finally results are average number of these groups. As for choosing query graph, we first choose a vertex among the vertex set of data graph at random and then a random walk strategy is used to find the rest nodes until size of query set meets the need. That is to say, we first choose a start vertex from data graph randomly, and then choose an adjacent node of vertex which have been chosen until the size of query graph meets the request.

### A. COMPARING THE PERFORMANCE OF FIVE TYPICAL ALGORITHMS

In this subsection, we compare the performance of algorithms introduced above over several social network datasets. Because BoostISO is a method which is used to speed up matching process, so we use BoostISO to speed up TurboISO to compare with other algorithms. Therefore, VF2, SPath, TurboISO, TurboISO-Boost and RI will be evaluated in this subsection. Source codes of SPath are provided by authors in [39].

Social networks have the basic structural characteristics of complex networks. What’s more, it shows some unique structural features due to its sociality which is different from other complex networks. Our main purpose is to compare the performance of these algorithms with different situation(size of data set and average degree are varied) when they are applied to social networks. According to the unique feature of social networks, we use real social networks datasets to compare the typical graph matching algorithms which are introduced in Section 3 instead of using synthetic datasets.

**Performance criteria.** All of the algorithms to be tested are based on the framework: “filter and search”. According to the framework of subgraph matching methods, filtering time and searching time are two main criteria to determine whether a matching method is good or not. We measure these criteria mentioned above by varying the average degree and size of the data set. Our main purpose is to test the performance of these methods and find out the suitable methods under different situations. In addition, we all choose the best value of parameters according to the original paper of each algorithm. That is to say, all of these algorithms are compared under their best circumstances.

1) Email-Eu dataset

Dataset of Email-Eu shows information about all incoming and outgoing email between members of the European research institution.

Fig. 7 shows the runtime of the filtering phase over Email-Eu. Since filtering is related to the query graph, when the size of the query graph is different, the corresponding running time also differs. Although we can conclude that the performance of SPath during filtering is worse compared with other algorithms from fig. 7. However, in fact, It takes the longest time for BoostISO to filter. This is because it needs to create a hypergraph for the target graph before filtering which requests tons of operations about target graph. For Email-Eu, it takes 0.0429 minutes to establish a
TABLE 2: Information of datasets in our experiment

<table>
<thead>
<tr>
<th></th>
<th>Nodes</th>
<th>Edges</th>
<th>Distinct Labels</th>
<th>Average_degree</th>
<th>Size/MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email-Eu</td>
<td>1005</td>
<td>11966</td>
<td>50</td>
<td>23</td>
<td>0.17</td>
</tr>
<tr>
<td>Email-Erozon</td>
<td>36692</td>
<td>183831</td>
<td>200</td>
<td>10</td>
<td>3.03</td>
</tr>
<tr>
<td>DBLP</td>
<td>317080</td>
<td>1049866</td>
<td>50</td>
<td>4</td>
<td>23.3</td>
</tr>
</tbody>
</table>

TABLE 3: Information of datasets in our experiment

<table>
<thead>
<tr>
<th></th>
<th>Nodes</th>
<th>Edges</th>
<th>Distinct Labels</th>
<th>Average_degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN1</td>
<td>5000</td>
<td>98820</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>SYN2</td>
<td>10000</td>
<td>195744</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>SYN3</td>
<td>20000</td>
<td>391813</td>
<td>50</td>
<td>20</td>
</tr>
</tbody>
</table>

hypergraph. Since the coordinate span of fig.7 is large if the runtime of hypergraph constructing is considered, it is not conducive to indicating the trend of performance of other algorithms. Therefore, we discuss the filtering performance of BoostIso separately here, which means that the runtime of BoostIso in fig.7 is the filtering time required by BoostIso after the hypergraph is established. In fact, when the graph changes infrequently, the hypergraph can only be calculated once, which is one reason why we discuss it separately.

Fig. 8 shows the number of iterations when searching over Email-Eu. As we can see from Fig. 8, as the size of the query graph increases, the query time of the VF2 algorithm grows fastest. This is because the filtering strategy adopted by VF2 in the filtering phase is relatively simple (it can be seen from Fig. 7 that the filtering time of VF2 is the shortest), so the candidate matching set obtained is large in scale. The same is true for RI, but since it optimizes the matching order, the number of iterations is effectively reduced. What’s more, BoostIso performs best under Email-Eu especially when the size of query graph is large.

FIGURE 7: Time of filtering over Email-Eu

Fig. 9 shows the runtime of the filtering phase over Email-Erozon. And Fig. 10 shows the number of iterations when searching over Email-Eu. In Fig. 9, we still don’t show the time of creating hypergraph by BoostIso over Email-Erozon. The constructing time is 39.43 minutes. The reason why SPath’s filtering time is much higher than other algorithms is that it needs to construct index for the whole target graph and which is time-consuming. The cost time of constructing index increases as the size of the graph increases. Unlike BoostIso, we do not only show the filtering time after the index is constructed. That is to say, in Fig. 9, the time for constructing index is considered. This is because even though the time of creating index is considered, the total time spent in the filtering phase of SPath is still on the same order of magnitude as other algorithms.

3) DBLP dataset

DBLP is a co-authorship network. Vertices there represent writers of papers and edges represents that two vertices (writers) connected by edges have published at least one paper together.

Fig. 11 shows the runtime of the filtering phase over DBLP. And Fig.12 shows the number of iterations with varying size of query over DBLP. It should be noted that BoostIso can’t finish finding subgraphs of DBLP. Apparently, there must be some implement problems though it can be executed well in another two data sets. The same problem have also occurred in [41], they communicated with authors can’t solve this problem. So, we don’t compare BoostIso

2) Email-Erozon dataset

Email-Erozon represents communications among users. Nodes represent users of email. Communication between users is represented by edges. Because label information of nodes and edges are not contained in the dataset of email-erzoon getting from SNAP, we randomly assign labels ranging from 0 to 200 to nodes in the dataset. What’s more, there is no edge label.

Fig. 9 shows the runtime of the filtering phase over Email-Erozon. And Fig. 10 shows the number of iterations when searching over Email-Eu. In Fig. 9, we still don’t show the time of creating hypergraph by BoostIso over Email-Erozon. The constructing time is 39.43 minutes. The reason why SPath’s filtering time is much higher than other algorithms is that it needs to construct index for the whole target graph and which is time-consuming. The cost time of constructing index increases as the size of the graph increases. Unlike BoostIso, we do not only show the filtering time after the index is constructed. That is to say, in Fig. 9, the time for constructing index is considered. This is because even though the time of creating index is considered, the total time spent in the filtering phase of SPath is still on the same order of magnitude as other algorithms.

VOLUME 4, 2016

7
with other methods under the DBLP data set.

As we can see in this figure, when size of query graph up to 8, VF2 can’t finish searching process in a reasonable time. And when size of query graph up to 12, RI can’t finish searching process in a reasonable time. This is because that VF2 adopted poor pruning strategies, with the increase of the number of nodes, the candidate sets become larger and larger. Though a good matching order is chosen, RI meets the same problem as VF2. Therefore, RI and VF2 is suitable for small graphs. Among these methods, SPath is more efficient.

As we can see from Fig. 12, different from other dataset, the iterations decreases when the size of the query graph exceeds 12 over DBLP. The rather low average degree and the largest degree are two main aspects which makes there is not so many candidate nodes for each query node when the query size is large.

As we can see from the experiments over these three datasets, when the query size is small, though the iterations of VF2 are larger than other algorithms, but the overall performance is good because the less filtering time. But when the size of target graph increases, it can’t finish the search in a reasonable time as shown in Fig. 12 even the filtering time is less than the other algorithms. What’s more, because of the better matching order, the iterations of RI are less than that of VF2. However, it still can’t finish the search in a reasonable time when the size of graph is large. Therefore, both VF2 and RI can be adopted to find isomorphic subgraphs for small target graphs.

Compared Fig. 8, Fig. 10 and Fig. 12, we can find the other three methods performs better during searching compared with VF2 and RI. That is because that these methods adopt more complex searching and pruning ways to make searching efficient. That is to say, simpler searching strategy as VF2 is more efficient in small graphs but the scalability of it is worse than other methods.

As we can see in Fig. 10 and Fig. 12, SPath performs best during searching over DBLP, but it performs not as good over Email-Erozon. The main reason is that DBLP has lower average degree and the average degree of Email-Erozon is higher. The searching strategy of Spath is path-based. High average degree means that more possible pathes need to be judged. TurboISO and BoostIso perform stable over Email-Erozon and DBLP. The size of DBLP and Email-Erozon
is large which makes the graph compression strategy used by BoostIso and Turbo_ISO more effective. The main idea of graph compression is to find the similar nodes in graph and make the graph a smaller one. What’s more, Turbo_ISO compresses the query graphs and BoostIso compresses both query and data graph. That’s why BoostIso performs better. Therefore, Spath is suitable for graphs whose average degree is not so large. Though we can’t get the performance of BoostIso in DBLP, but according to the original paper and analysis of its performance in the other data sets described above, Turbo_ISO and BoostIso can well adapted to graphs with large degree because of graph compression. As for the filtering time, although the filtering time is higher than that of VF2 and RI, more redundant candidate nodes can be pruned out by complex pruning strategies, which guarantee that the isomorphic subgraphs can be found in a reasonable time when the target graph is large. What’s more, if the target graph is not dynamic, the index of SPath and BoostIso can be created only once.

Therefore, as analysis above, we can conclude that VF2 and RI are suitable for small graphs, while SPath, BoostIso and Turbo_ISO can be used when the size of graph is large. When the average degree is high, BoostIso and Turbo_ISO are better while SPath performs better for sparse graphs.

B. EVALUATION THE EFFECTIVE OF MATCHING ORDER

As we can see from the experiments above, the searching performance are not only influenced by the matching order but also the effective of the filtering strategies. That is to say, because of the candidate sets gotten from each algorithms are different, we can not simply know the effective of the choosing of matching order. Therefore, to know the influence of matching order for the performance of searching, we find the isomorphic subgraphs on the same candidate sets by using different matching order.

Actually, the matching order of SPath and BoostIso is greatly influenced by the structure constructed during filtering period which can not be used in many situations. Therefore, we just compare the matching order choosing strategies of VF2, RI and Turbo_ISO which are often used in nowdays. To make it clearer, performance of choosing nodes randomly is also compared.

Fig. 13, Fig. 14 and Fig. 15 shows the number of iterations when searching with the same candidate sets over SYN1, SYN2 and SYN3 respectively. As we can see from these figures, order choosing strategy proposed by RI performs better. After that, order choosing strategy proposed by Turbo_ISO also performs better than the rest two methods.

V. FUTURE WORKS

The fast growth speed of social networks such as Facebook and Twitter makes data size expend constantly, which leads to large computational cost. For example, by the end of 2016, The number of monthly Facebook users reached 1.860 billion [40] leading to huge amounts of data. What’s more, complex as social networks, it is time-consuming to process it. Therefore, finding a fast and effective subgraph matching approach is of great importance. There are three main aspects of subgraph matching algorithms that need to be further developed.

Distributed storage and parallel processing. Faster searching speed is required due to data size becomes larger and larger. Parallel processing can be used to speed up subgraph searching and distributed file system can be used for dataset storage. Distributed system framework such as Spark can meet the need described above. So finding good strategies to adopt subgraph matching methods on Hadoop or Spark is an important work.

Dynamic updating. Updating frequently is another characteristic social networks have [43]. There are about 1.23 billion people use facebook per day, and every operation they carry out would cause update of database [44]. It is inefficient to find the result again once the database updated. So finding an efficient and effective method to apply to the dynamic data graph is vital. There are some methods using incremental processing technology to solve the dynamic updating problem [18], [45], [46]. But a more complete and efficient framework for large data set is needed. It’s just
starting of this method, there’s a long way to go.

More efficient approaches of approximate algorithms for large and noisy networks. Sometimes, the data we get is noisy and exact algorithms can’t be used due to its strict constraint on structure. Approximate subgraph matching method is a good choice under this situation. However, many of approximate methods are not designed for large networks. Therefore, with the fast growing speed of data size, it is desired to find more efficient approximate methods to solve complex subgraph matching problems which can’t be solved by exact methods.

VI. CONCLUSION

In this paper, we compare the performance of exact subgraph matching methods in social networks by using a series of real-life datasets. According to the experiments, due to its poor pruning strategies VF2 is applicable for rather small graphs while SPath performs better in large graph when average degree of graph is small. When it comes to graph with high average degree, TurboISO performs well. BoostsO also have good performance for speed up other methods. As for the efficiency of matching order, the method proposed by RI is better. In summary, as an important methods applied in social networks, subgraph matching can be used to find social relationships and so on. Rich information of nodes should be utilized in future to make subgraph matching more useful and accurate in social networks.

REFERENCES


[46] T. Ma, W. Shao, Y. Hao, J. Cao, Graph Classification Based on Graph Set Reconstruction and Graph Kernel Feature Reduction, Neurocomputing 296 (2018) 33-45.

***

VOLUME 4, 2016

11