TGP: Mining Top-K Frequent Closed Graph Pattern without Minimum Support*

Yuhua Li, Quan Lin, Ruixuan Li, and Dongsheng Duan

School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, P. R. China
linquan.hust@gmail.com
{idcliyuhua,rxli}@hust.edu.cn
duandongsheng@smail.hust.edu.cn

Abstract. In this paper, we propose a new mining task: mining top-k frequent closed graph patterns without minimum support. Most previous frequent graph pattern mining works require the specification of a minimum support threshold to perform the mining. However it is difficult for users to set a suitable value sometimes. We develop an efficient algorithm, called TGP, to mine patterns without minimum support. A new structure called Lexicographic Pattern Net is designed to store graph patterns, which makes the closed pattern verification more efficient and speeds up raising support threshold dynamically. In addition, Lexicographic Pattern Net can be stored in the file through serialization, so it dosen't need generate candidate patterns again in the next mining. It is found in the preliminary experiments that TGP can find top-k frequent closed graph patterns completely and accurately. TGP is especially valuable for mining frequent patterns when user cannot provide appropriate minimum support. Furthermore, TGP can be extended to mine other kinds of graphs or dynamic graph streams easily.

Keywords: data mining, top-k, frequent closed graph pattern

1 Introduction

Frequent subgraph mining[1–4, 7] has long been viewed as a challenging problem in data management. From the view of generation of candidate subgraph, there are two typical algorithms: Apriori and FP-growth[5]. Apriori and Fp-growth based mining framework need a min-support threshold to ensure the generation of candidate subgraph correct and necessary. Unfortunately, this framework, though simple, leads to the following two problems.

First, *min_support* is difficult to choose. If the *min_support* is too small, thousands of patterns will be mined out but a lot of which are meaningless.

* This work is supported by National Natural Science Foundation of China under Grant 70771043, 60873225, 60773191. National High Technology Research and Development Program of China under Grant 2007AA01Z403, Natural Science Foundation of Hubei Province under Grant 2009CDB298

On the other hand, if the *min_support* is too big, large patterns will not be contained in mining result because frequent counts of lager patterns are often small. Choosing a suitable *min_support* needs users to try many times, which is a great burden. Second, the mining result contains a lot of redundant graph pattern. According to the Apriori property, all the subgraphs of a big frequent graph pattern are frequent and should be mined out, wasting a lot time but meaningless.

To solve the two problems above, the existing mining framework must be changed. The second problem has been noted and examined by researchers recently proposing to mine frequent closed graph patterns instead[6, 8]. Mining closed patterns is more efficient and the mining result is compact without loss of information. Therefore, mining closed pattern will be a good choice for mining frequent patterns.

TSP [11]and TFP [12] have solved the $top-k$ frequent closed item and sequence pattern mining respectively, but mining $top-k$ frequent closed graph pattern is still a problem now because graph mining itself is more complex than item and sequence.

Our research focuses on the pattern of money laundering for detecting suspicious transactions in financial trade network. From domain knowledge we know that money laundering is a kind of group crime. We have a transaction database which contains many detected money laundering cases which can be modeled with trade graphs. Now we need to find $top-k$ frequent transaction patterns which can be regarded as money laundering patterns. There are similar problems in fraud detection and crime clue mining.

To solve these problems, we propose a new mining task: mining $top-k$ frequent closed graph patterns of size bigger than *min_size* without *min_support*. We propose the Lexicographic Pattern Net model based on Lexicographic Sequence Tree, which are used to store candidate graph pattern. Based on Lexicographic Pattern Net, a new efficient algorithm, called TGP, is developed to solve this mining task. Experiments show that TGP solves this new mining task very well. Applying TGP into our problem, we can find out most frequently used money laundering patterns quickly. Among these patterns, some match the domain knowledge very well, and others can be derived indirectly.

The rest of the paper is organized as follows. In Section 2, we define our new mining task and compare the difference between TGP and CloseGraph. In Section 3 we propose a new pattern storage structure Lexicographic Pattern Net and the mining process is described in Section 4. Experimental result is analyzed in Section 5. Extending TGP to mining other kinds of graphs is discussed in Section 6, and we conclude our study and discuss the future works in Section 7.

2 Problem Definition

In this section we propose the new mining task: $top-k$ frequent closed graph pattern. We denote the vertex set of a graph g by $V(g)$, the edge set by $E(g)$, then the graph can be denoted by $G(V(g), E(g))$. Since most of interesting graph

patterns are connected graphs, the mentioned graph is simply connected with undirected graph with labeled vertex and labeled edge without multiple edges. Graphs with self-loop and directed graph are discussed in Section 6.

Definition 1. ***Top-k Frequent Closed Graph (TGP)*** *Subgraph g is a* ***frequent graph pattern*** *in a labeled graph dataset D, if its support in D is no less than $min\_support$. A grpah pattern g is a* ***closed graph pattern*** *if there exists no graph pattern g' such that(1)$g \sqsubset g'$ and (2)$support(g) = support(g')$. A closed graph pattern g is a* ***top-k closed graph pattern of minimum size*** *min_size if there exist no more than $(k-1)$ closed graph patters whose size is at least min_size and whose support is higher than that of g.*

Our new mining task adopts the advantages of CloseGraph. Comparing CloseGraph with traditional frequent subgraph, CloseGraph discards some patterns whose supports equal to their super patterns and such super patterns exist in the mining result. So the mining result of CloseGraph is more meaningful. Furthermore, CloseGraph adopts early termination to speed up pruning, so it's more efficient. TGP eliminates redundant subpatterns, and can control the amount of mining results and the size of pattern, which makes the mining result more meaningful. Furthermore, there is no need for user to specify a *min_support*.

3 Pattern Extension And Storage

To solve our new mining task, a new structure called DFS Code Net is proposed to store all the patterns from one graph and another structure called Lexicographic Pattern Net is proposed to store the merge result of DFS Code Nets for all graphs in graph dataset.

3.1 Graph Representation

Han, J. and Yan X. have used DFS Subscripting and Right-Most Extension [4] to generate DFS Codes for a graph. According to DFS Lexicographic Order, there must be a minimum DFS Code for the graph, which is used to denote the graph uniquely. This method has been proved to be an efficient coding technology especially in algorithms based on FP-growth. Our TGP algorithm is also based on FP-growth, so we also denote a graph by its minimum DFS Code.

3.2 DFS Code Net

DFS Code Tree has been used widely in graph mining algorithms based on FP-growth. It is a hierarchical tree structure for storing graph patterns. Every node of DFS Code Tree stores a DFS Code, which represents a graph pattern. The Pattern its child nodes represent is its supper pattern. But this structure is not perfect to mine closed pattern, because closed pattern verification need

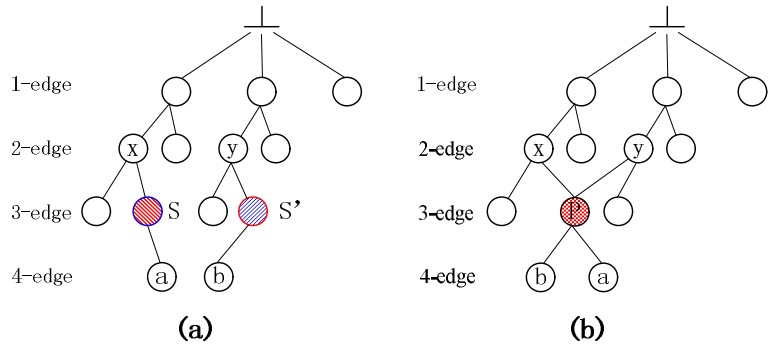

Fig. 1. (a) is a DFS Code Tree, nodes S and S' present for same graph pattern. (b) is a DFS Code Net, nodes S and S' in (a) are merged into node P. There exist no other nodes presenting the same graph pattern with P.

compare its frequent count with all of its direct subpatterns. We cannot get frequent count directly from DFS Code Tree structure because a pattern has several codes according to different extension orders as shown in Fig. 1(a).

The DFS code of S and S' are different but represent same pattern P. If we want to get all the sub patterns of P, we have to find all P's DFS Code nodes and their parents such as node x and node y, but x and y may present for same graph pattern. So it is inefficient. We improve the DFS Code Tree to DFS Code Net, which is a hierarchical net structure, as Fig. 1(b)shows.

Algorithm 1: *Generate_DFS_Code_Net*(G)

input : Graph G
output: DFS Code Net D

```
1  Initial DFS Code Net D to a Null Net;
2  foreach Edge e of G do
3      Create Node N(C),insert(N);
4      for Node N in Net D do
5          if N can be extended then
6              Pattern P= Extend(N),C=DFS_Code(P);
7              if exist Node N_child and DFS_Code(N_child) represent P then
8                  DFS_Code(N_child) = min(C, C'),N'child = N_child;
9              end
10             else create Node NewN(C),insert(NewN), N'child = NewN;
11         end
12         else break;
13     end
14 end
15 return D
```

DFS Code Net construction as algorithm 1 is a NP-Complete problems, it has the same time complexity with DFS Code Tree [4] with small additional cost. A node of DFS Code Net stores the minimum DFS Code of a graph pattern,

and it is unique. If the graph size of g is n, then there is just one $n-th$ level node storing the minimum DFS Code of g, representing graph g itself.

3.3 Lexicographic Pattern Net

Lexicographic Pattern Net stores all patterns and their frequent counts for graph set, which is improved on the Lexicographic Sequence Tree. The structure of Lexicographic Pattern Net is the same as the DFS Code Net, but the node of Lexicographic Pattern Net stores the pattern's frequent count and graph ID which contains the pattern. Lexicographic Pattern Net is generated from DFS Code Nets by a kind of incremental method. Fig. 2 shows two graph g_1,g_2 and their DFS Code Nets N_1,N_2.

Algorithm 2 describes how to insert a DFS Code Net to a Lexicographic Pattern Net.

Algorithm 2: $Insert\_DFS\_Code\_Net(D,L)$

input : DFS Code Net D,Lexicographic Pattern Net L
output: Lexicographic Pattern Net L after inserting

```
1  get root node R of D;
2  if R can be found in D then
3  |  foreach Node N of subnet R of D do
4  |  |  frequent++;
5  |  end
6  |  return L;
7  end
8  else
9  |  insert R to L by lexicographic order;
10 |  if R has children then
11 |  |  foreach subpattern R' of R do
12 |  |  |  call Insert_DFS_Code_Net(R', L);
13 |  |  end
14 |  end
15 |  else return L;
16 end
```

Algorithm 2 bases on an easy theory, that is if the root nodes of two DFS Code Nets are same, they represent for same graph pattern, so the two DFS Code Nets must be same. By this theory we can calculate the frequent count of nodes of Lexicographic Pattern Net quickly. In the inserting process if the root of DFS Code Net can be found in the Lexicographic Pattern Net, we can insert this net at one time just by raising the frequent count of nodes of corresponding subnet in Lexicographic Pattern Net. The Lexicographic Pattern Net N_{12} in Fig. 2 shows the Lexicographic Pattern Net generated by N_1 and N_2.

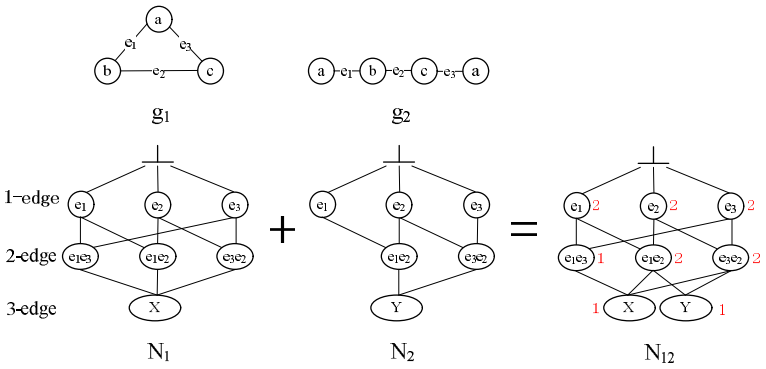

Fig. 2. DFS Code Nets N_1 and N_2 are generated from graphs g_1 and g_2 respectively. Nodes X and Y represent for the whole graphs g_1 and g_2. The Lexicographic Pattern Net N_{12} is merged from DFS Code Nets N_1 and N_2. The red numbers beside nodes are their frequent counts.

4 Mining Process

Section 3 introduces how to generate a Lexicographic Pattern Net from a graph set. In this section we explain step by step how to use Lexicographic Pattern Net to mine $top-k$ frequent closed graph pattern.

4.1 Closed Graph Pattern Verification

Our task is mining $top-k$ closed patterns, so we should guarantee that at least k closed patterns can be found. CloSpan [13] stores candidates for closed patterns during the mining process and in the last step it finds and removes the non-closed ones. This approach can not be used in mining $top-k$ closed patterns because it needs to know which patterns are closed and accumulates at least k closed patterns before it starts to raise the minimum support. Thus closed pattern verification must be done during the mining process. CloseGraph adopts Early Termination and Detecting Failure of Early Termination to find closed patterns directly during the mining process, but it still has the above problem. Lexicographic Pattern Net solves this problem very well. We can get all the direct superpatterns of any pattern, and verify that this pattern is closed quickly.

For example, if we want to verify whether the pattern e_2 in Fig.2 is closed or not, we get its direct superpatterns e_1e_2 and e_3e_2. Their frequent count is the same, so the pattern e_2 is not closed. Comparing frequent count e_3 with e_1e_3, they are different, so e_3 is closed. Lexicographic Pattern Net make closed pattern verification easily, that's why we cost a lot of time to construct it.

4.2 Applying Minimum Size Constraint

DFS Code Net and Lexicographic Pattern Net is a level structure, so the depth of node is the size of pattern. Thus it is easy to apply minimum size constraint

by add a controlled condition in Algorithm 2. That is when the size of R is *min_size*, the subpatterns of R need not be inserted any more.

4.3 Find $top-k$ patterns

In order to mine most frequent closed patterns, an appropriate minimum support should be found to judge whether a pattern is a $top-k$ frequent pattern or not. We adopt a Support Threshold Raising method to find the minimum support. To raise the minimum support quickly and correctly, a simple data structure called *closed_pattern_order_array* is used to store current $top-k$ closed graph pattern ordering by their frequent count. The size of *closed_pattern_order_array* is equal or greater than k according to the definition of TGP. When new patterns insert into it, low frequent patterns are removed to make sure it always store top-k frequent closed patterns already known.

Algorithm 3: $top-k(L, min\_size, k)$

input : Lexicographic Pattern Net L, minimum size min_size, $top-k$
output: closed_pattern_order_array A contains $top-k$ patterns

```
1  foreach Pattern P of L at min_size level do
2      if P is a closed Patten then insert P to A;
3      else
4          Find Closed Patterns from P's children;
5          insert to A;
6      end
7  end
8  remove unfrequent Patterns in A,keep size of A to k;
9  min_frequent=minimum frequent of Patterns in A;
10 while A changed do
11     foreach Pattern P of A do
12         Find Closed Patterns from P's children that
           frequent > min_frequent; insert to A;
13         remove unfrequent Patterns in A,keep size of A to k;
14         min_frequent=minimum frequents of Patterns in A;
15     end
16 end
17 return A;
```

The length of *closed_pattern_order_array* is bigger than or equal to k, and it may change during the mining process. Algorithm 3 shows how to generate the final *closed_pattern_order_array* by scanning the Lexicographic Pattern Net selectively.

Algorithm 3 need not scan every node of Lexicographic Pattern Net, and the Lexicographic Pattern Net makes the checking of closed pattern very easy. According to Apriori, the frequent count of sub pattern is small or equal to that of their parents, which reduces the search space. Experiments show that Algorithm 3 is efficient.

4.4 TGP Algorithm

Algorithm 4 describes our new mining framework. This new framework generates a Lexicographic Pattern Net which contains all sub patterns and the relationships between them. When Lexicographic Pattern Net is constructed, it can be reused for different mining task. For example, we can edit mining tasks line 6 in Algorithm 4. Experiments indicate that the runtime of line 6 is less than 1% of the whole runtime, so this new framework has a good performance in multitask.

Algorithm 4: $TGP(D, min\_size, k)$

input : graph set D,minimum size min_size,$top-k$ value k
output: closed_pattern_order_array A contains $top-k$ patterns

```
1 Initial Lexicographic Pattern Net L;
2 foreach graph g of D do
3 |   DFS Code Net N= DFS_Code_Net(g);
4 |   Insert_DFS_Code_Net(N,L);
5 end
6 closed_pattern_order_array A=top-k(L,min_size,k);
7 return A;
```

5 Experimental Evaluations

A comprehensive performance study has been conducted in our experiments on money laundering case dataset, synthetic datasets and processed chemical compound datasets. TGP algorithm is implemented in java, compiled by Sun's stand compiler V1.5, and interpretively executed by Sun's stand java VM V1.5. All the experiments are done on a 2.4GHZ Intel Pentium-4 PC with 1GB main memory, running Windows XP.

Money Laundering Case Dataset. We collect 200 money laundering cases. Regarding the characteristics of money laundering, several important attributes are extracted such as money amount, account type, etc. After pretreatment, these cases are modeled as 200 graphs which contain 10 nodes and 15 edges averagely. For covering up money laundering, financial criminals usually adopt some tricks such as laundering large amounts through several transactions or through long path. By analyzing money laundering action, it's found that most of them consist of three or more transactions. So we set min_size=3, $top-k$=20. Comparing the mined-out patterns with research results of financiers, most of these patterns such as Fig. 3 (a), (b) and (c) show match very well, some can be derived indirectly and some doesn't match very well. These mismatched patterns as Fig. 3(d) shows are proved to be new money laundering patterns which have not been abstracted by financiers.

Synthetic Dataset. The synthetic datasets are generated using a graph procedure called GraphGen designed by us. The procedure has 3 parameters: G, N and E. Parameter G controls the amount of output graph and every graph contains N nodes and E edges averagely. Two contrast experiments have been

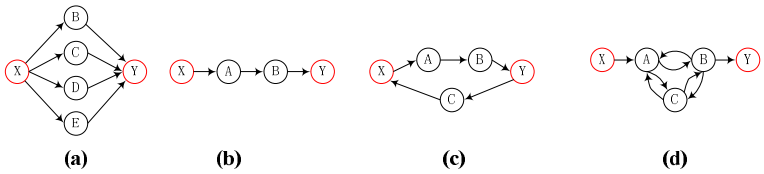

Fig. 3. (a), (b) and (c) are patterns matching Anti Money Laundering and(d) are patterns occurring frequently in real world.

conducted to analyze the performances of TGP on different graphs. The datasets of the first contrast experiment are generated by GraphGen with parameter G=300, N=10, E=10∼16 in Fig. 4(a) shows the experiment results.

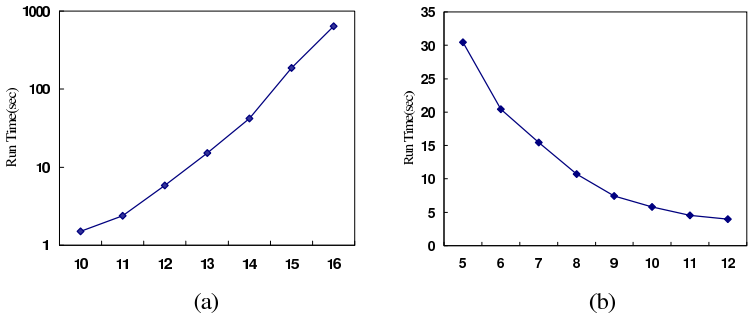

Fig. 4. (a) shows the TGP performance for different graph size (average size from 10 to 17). (b) shows performance for different graph density (graph size is fixed to 12, nodes count form 5 to 12).

The result shows that runtime grows exponentially as the graph size increases because it is a NP-Complete problem.

The datasets of the second contrast experiments are generated by GraphGen with parameter G=300, N=5∼12, E=12. This dataset contains graphs of different density. As Fig. 4(b) shows, when graphs are more density, it costs more times. For graphs of same size, if graph contains fewer nodes, it contains more sub graphs. For example, for a 5 nodes complete graph, it contains 25 sub graphs while for a 5 nodes tree, it contains 15 sub graphs at most. So it costs more time during pattern extension and combination.

Processed Chemical Compound Dataset. The chemical compound dataset can be retrieved through this URL.[1] The original dataset contains 340 compounds, 24 different atoms, 66 atom types, and 4 types of bonds. The dataset is sparse, containing average 27 vertices per graph and 28 edges per graph. The largest one contains 214 edges and 214 vertices. Because TGP have no *min_support* to pruning, complete *DFS_Code_Net* must be generated. But generating complete *DFS_Code_Net* is a NP-Complete problem, it's impossible to generate *DFS_Code_Net* for big graph. Experiment shows that when graph has

[1] http://oldwww.comlabox.ac.uk/oucl/groups/machlearn/PTE.

more than 25 edges, the runtime is longer than 4000 seconds. Furthermore, 3 synthetic datasets are generated: Data1015 (the graph size are from 10-15 at random), Data1520 (the graph size are from 15-20 at random) and Data2025 (the graph size are from 20-25 at random).

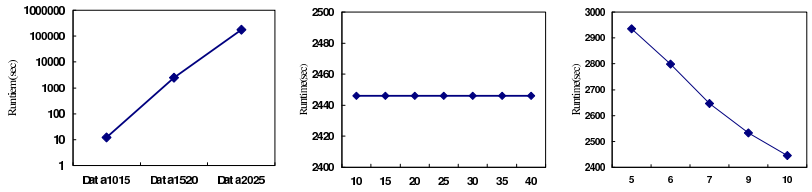

Fig. 5. (a) shows performance on three different data sets. (b) shows performance for different $top-k$ values on Data1520. (c) shows performance for different *min_size* value on Data1520.

Fig. 5(a) shows the runtime for the 3 test dataset for mining top-20 frequent closed graph with the minimum size 10. This result shows that runtime grows linearly with the graph size.

Fig. 5(b) shows the runtime for different $top-k$ values on Data1520. The runtime changes very little for different $top-k$ values because mining patterns on the Lexicographic Pattern Net cost less than 1% time of the whole runtime.

Fig. 5(c) shows the runtime for different *min_size* values on Data1520. *Min_size* determines which levels of the DFS Code Net should be combined to Lexicographic Pattern Net. The bigger the *min_size* is, the smaller the subnet to be combined is. But combining costs litter time to DFS Code Net generation, the whole runtime doesn't change much.

6 Discussion

So far, we have proposed and investigated a general framework, TGP, for mining closed, labeled connected, undirected, frequent subgraph patterns. But in real world, graph is various. Here we show how the framework can be extended to mine other kinds of graphs.

1. **Mining directed graphs**. We have used Extinction Direction Code [14] to solve mining directed graph pattern. This method improves DFS Code by adding a direction code. If extension direction is the same with the edge direction, the direction code is 1, otherwise, it is 0.
2. **Mining unlabeled and partially labeled graphs**. For this kind of graph, we can transform it to labeled graph by adding specified label (this label does not appear in graph) to unlabeled nodes and edges.
3. **Mining non-simple graph**. Non-simple graphs may have self-loop and multiple edges. We can change extension order to solve it. In order to accommodate self-loops, the growing order can be changed to backward edges,

self-loops and forward edges. Multiple edges can appear in these three kinds of edges. If we allow two neighbored edges in a DFS code to share the same vertices, actually the definition of DFS lexicographic order can accommodate multiple edges.

4. **Mining graph stream**. Dynamic graph stream [16] is a special graph set. Different from normal graph set, the graph count of graph stream is not fixed, so the mining framework based on Apriori cannot work on this kind of dataset. TGP algorithm can solve this problem very well because it is incremental. All graph pattern information in it is stored in the Lexicographic Pattern Net, and added into Lexicographic Pattern Net one by one. TGP is a stream framework. And for dynamic data, time tags can be add to Lexicographic Pattern Net nodes for mining frequent graph patterns in a time slice.

7 Conclusion

In this paper, we have studied the problem of mining $top-k$ frequent closed graph patterns with size no less than *min_size* and proposed an efficient mining algorithm TGP. This new algorithm adopts Lexicographic Pattern Net to store patterns and relationship between patterns, which makes close pattern verification easily during the mining process. So the algorithm can raise *min_support* correctly and ensures the complete and right results. Comparing TGP with CloseGraph, TGP doesn't need users to provide a proper *min_support* which is often difficult to set and it can find specified number of most frequent closed graph patterns. In addition, TGP can run with a parameter *min_size* to filter out small patterns. So the patterns mined out by TGP are practical and in many cases more preferable than the traditional minimum support threshold based graph pattern mining. Moreover TGP mining framework is incremental, so it is suitable for mining graph stream.

We have done experiment in financial dataset and received good results. In future, we will widen the range of TGP application, such as mining most frequent molecular formula in poisonous chemical substances dataset to find poisoning molecular structure. TGP is a NP-Complete problem and it cannot prune before extending patterns to *min_size* for ensuring its veracity and completeness. So for huge graph it doesn't work well. In future we will study how to sacrifice some accuracy and completeness to prune during generating DFS Code Net to apply TGP on huge graphs. Moreover, the extension of TGP for mining other complicated structured patterns, such as directed graph pattern, unlabeled and partially labeled graph pattern, is valuable for future research.

References

1. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: First IEEE International Conference on Data Mining (ICDM'01), pp. 313–320. (2001).

2. Vanetik, N., Gudes, E., Shimony. S.E, Computing Frequent Graph Patterns from Semistructured Data. In: Second IEEE International Conference on Data Mining (ICDM'02), pp.458–465. (2002)
3. Han, J., Wang, W., Prins, J.: Efficient Mining of Frequent Subgraph in the presence of Isomorphims. In: 3rd IEEE International Conference on Data Mining (ICDM'03), pp. 549–552. (2003)
4. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Second IEEE International Conference on Data Mining (ICDM'02), pp. 721–723. (2002)
5. Christian, B. An Implementation of the FP-growth Algorithm. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (OSDM '05), pp. 1–5. (2005)
6. Han, J., Yan, X.: CloseGraph: Mining Closed Frequent Graph Patterns. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '03), pp. 286–295. (2003)
7. PTamas, H., PJan, R., Stefan W.: Frequent subgraph mining in outerplanar graphs. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'06), pp. 1097–1011. (2006)
8. Yan, X., Zhou, J., Han, J.: Mining closed relational graphs with connectivity constraints. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD'05), pp. 324–333. (2005)
9. Thomas, L.T., Valluri, S.R., Karlapalem, K.: MARGIN: Maximal Frequent Subgraph Mining. In Proceedings of the Sixth International Conference on Data Mining (ICDM'06), pp. 1097–1101. (2006)
10. Wang, N., Parthasarathy, S., Tan, K., Tung, A.K.H.: CSV: visualizing and mining cohesive subgraphs. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data(KDD2008), pp. 445–458 (2008)
11. Wang, J., Han, J., Lu, Y., Tzvetkov, P.: TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets. IEEE Trans. Knowl. Data Eng, pp. 652–664.(2005)
12. Tzvetkov, P., Yan, X., Han, J.: TSP: Mining Top-K Closed Sequential Patterns. In: 3rd IEEE International Conference on Data Mining (ICDM'03), pp. 347–354 (2003)
13. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: SIAM International Conference on Data Mining (SDM'03), pp. 166–177. (2003)
14. Li, Y., Lin, Q., Zhong, G., Duan. D.: A Directed Labeled Graph Frequent Pattern Mining Algorithm based on Minimum Code, In: The 3rd International Conference on Multimedia and Ubiquitous Engineering (MUE'09), pp. 353–359 (2009).
15. Maunz, A., Helma, C., Kramer, S.: Large-scale graph mining using backbone refinement classes. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD' 09), pp. 617–626 (2009)
16. Muthukrishnan, S.: Data streams: algorithms and applications. In 14th ACM-SIAM Symposium on Discrete Algorithms. (SODA'2003), pp.413–413 (2003)