FSP: Frequent Substructure Pattern Mining

Shuguo Han Wee Keong Ng Yang Yu
CAIS, School of Computer Engineering
Nanyang Technological University, Singapore
{hans0004, awkng, yuya0001}@ntu.edu.sg

***Abstract*—Graphs have become increasingly important in modeling the complicated structures. *Mining frequent subgraph patterns* is an important research topic in *graph mining* that helps to analyze the structured database. It has been applied in many applications, such as chemistry, biology, computer networks, and World-Wide Web. In this paper, we propose a new algorithm called FSP (Frequent Substructure Pattern mining), which improves the state-of-the-art algorithm — gSpan. Our algorithm has reduced the number of graph and subgraph isomorphism tests and the number of accessing the graph database. The performance of FSP was evaluated base on a chemical compound dataset, which is widely used by subgraph mining algorithms. The experimental results show that FSP overcomes with the state-of-the-art gSpan algorithm.**

I. Introduction

As a two-dimensional or more data structure, graph has been used extensively in various domains, such as biochemistry[12], computational biology[11], pattern recognition [8], computer vision [8], semi-structure indexing [5], text retrieval [3], etc.

Frequent subgraph mining has a lot of real life applications. For example, in Web Mining, when a large quantity of XML documents are collected, frequent subgraph mining can be used to cluster based on their common structure. Therefore, frequent subgraph mining has raised great interests from the researchers and industries.

The core problem of *frequent subgraph mining* is graph isomorphism test, which tests whether two candidate graphs are structurally identical. It belongs to NP-complete problems [4], which means the algorithms runs in non-deterministic polynomial time. So efficient and approximate algorithms are to be discovered to overcome this problem. In addition, as techniques of structured data mining are explored, the data structure used becomes more and more complex, and the scale of data sets also increased conspicuously.

The above discussion implies that there is a need for algorithms in graph mining to be designed with high accuracy and efficiency. In this paper, we propose a new graph mining algorithm, called FSP, to address the problems in the following sections.

II. Related Work

In this section, the related work based on the categorized approaches (i.e., Breadth-First Approach and Depth-First Approach) is reviewed.

The first frequent subgraph mining under breadth-first approach was proposed by Inokuchi et al. [7] to mine the unconnected subgraphs, called ACM. It is based on the Apriori algorithm developed by Agrawal and Srikant [1] originally for mining frequent itemsets. ACM uses an iterative process to generate candidate subgraphs and discover their support in the database.

MoFa proposed by Borgelt and Berthold [2] and gSpan proposed by Yan and Han [13] are the first depth-first graph miners. MoFa generates fragments by embedding them in all appropriate molecules in parallel. Extension is restricted to those fragments. gSpan developed a novel lexicographic ordering as a canonical representation to eliminate the duplicate subgraphs. In gSpan, the right-most extension was introduced to reduce the generation of duplicate graphs.

Huan *et al.* proposed another depth-first algorithm, called FFSM [6]. As an important shortcoming of the method, FFSM requires to check the frequency of an additional set of subgraphs that are not canonical. Recently, another GASTON was proposed by Nijssen and Kok [10] based on "quick-start principle". The algorithm enumerated paths and trees that have the polynomial algorithms to detect the subgraph isomorphism first. Only at the end, GASTON generates general graphs with cycles by joining. To detect the duplicate subgraphs, a well-known algorithm $Nauty$ [9] is used to handle the NP-complete subgraph isomorphism problem.

III. Frequent Substructure Pattern Mining

In SectionIII-A, we show the background of canonical graph representation to be used in the following sections. In Section III-B, we present DFS balance to fast prune the duplicate subgraphs. Section III-C present the structure similarities in the search space to reduce the number of accessing the database.

A. Canonical Graph Representation

1) DFS Subscripting: When performing a depth-first search(DFS) in a graph, we can construct a depth-first search tree. For example, if Figure 1 (a) is a frequent subgraph in the sample dataset, the solid edges, called forward edge, in Figure 1(b)-(d) construct five different DFS trees for this graph. Other dashed edges are called backward edge. Figure 1 (b)-(d) are all the same graph as the one in Figure 1 (a).

2) Canonical Labeling and Minimum DFS Code: One manner to solve graph isomorphism problem is to calculate label of two graphs. If their canonical labels are the same, then these graphs are isomorphic to each other. There are few algorithms to compute the canonical label. gSpan developed a new labeling system from Nauty, called *minimum* DFS *Code*.

(Minimum DFS Code) Given a graph G, $Z(G) = \{code(G,T)|\ \forall T$, T is a DFS tree for G.$\}$, based on DFS

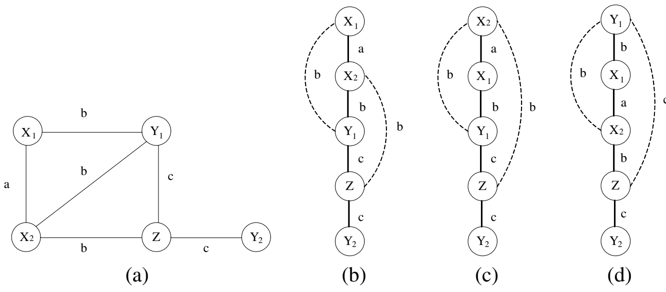


Fig. 1. Depth-First Search Trees and Forward/Backward Edge Set of the Subgraph

edge#	(b)	(c)	(d)
0	(0,1,X,a,X)	(0,1,X,a,X)	(0,1,Y,b,X)
1	(1,2,X,b,Y)	(1,2,X,b,Y)	(1,2,X,a,X)
2	(2,0,Y,b,X)	(2,0,Y,b,X)	(2,0,X,b,Y)
3	(2,3,Y,c,Z)	(2,3,Y,c,Z)	(2,3,X,b,Z)
4	(3,1,Z,b,X)	(3,0,Z,b,X)	(3,0,Z,c,Y)
5	(3,4,Z,c,Y)	(3,4,Z,c,Y)	(3,4,Z,c,Y)

TABLE I
DFS CODE FOR THE SUBGRAPHS IN FIGURE 1(B) TO (D)

lexicographic order in [13], the minimum one, $min(Z(G))$, is called **Minimum DFS Code** of G. it is also the canonical label of G.

We use $min(G)$ to denote $min(Z(G))$, and $min(\alpha)$ to denote the minimum DFS code of the graph represented by the DFS code α. The codes generated by the possible DFS trees in Figure 1(b) to (d) are shown in Table III-A1. The minimum DFS code of the graph in Figure 1 (a) is Code(c).

It has been proved that given two graphs G and G', G is isomorphic to G' if and only if min$(G) = min(G')$. Thus the problem of mining frequent connected subgraphs is equivalent to mining their corresponding minimum DFS codes.

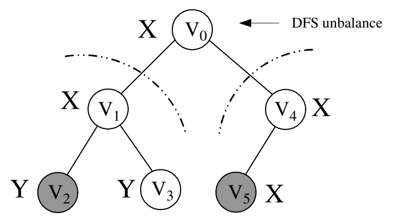
B. DFS Balance

Each DFS code contains a forward edge set and a backward edge set. The DFS tree of the subgraph are formed by the forward edge set in the DFS code.

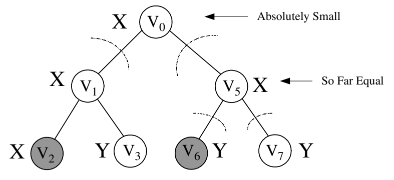
In the DFS tree of a graph, a vertex,except the root, and its descendant vertices form a new sub-DFS tree. So that the sub-DFS tree is rooted at this vertex, it is not be the root of the DFS tree.

DFS Balance: A graph is said to be ***DFS balance*** if and only if for any vertex in the graph, all of the sub-DFS trees rooted at its children should be disposed in descending order under DFS lexicographic order. Otherwise, it is **DFS unbalance**. For a vertex in the DFS tree of the graph, if the DFS Code of the sub-DFS trees rooted at its children are in descending order, then we say that it is DFS Balance at this vertex.

Once a graph is DFS Balance at vertex V_i, there are also two situations. If the sub-tree formed by its latter children is always smaller than the former one, then we say it is **absolutely small** at this vertex, as no matter how many new nodes are added to the last child, the DFS code of latter children is always smaller than the former one unless a new child is added to the parent vertex. For example, as shown in Figure III-B (a), the sub-tree rooted at V_1 is smaller than the one rooted at V_4, as the label weight of darkened vertices are different. So no matter how many nodes are added to the latter sub-tree, there will be no change on the relationship of these two sub-trees so it is absolutely small.



(a) DFS unbalance



(b) DFS balance

Another situation is for the existing nodes in the latter sub-tree, it is equal to the code in the former tree with the same DFS code length. We call this situation **so far equal**. If more node is added to the latter sub-tree, then it could be smaller or larger than the former sub-tree. Figure III-B (b) shows an example in such case. You can see at vertex V_5, the sub-trees formed by its children are *so far equal*, as their DFS code are exactly the same. However, if on more node is added to the latter sub-tree, then V_5 will be DFS unbalance as latter child's sub-DFS tree is smaller than the former one's.

Fast Pruning Rule: If one DFS code is not DFS Balance, then it must be non-minimum.

Proof omitted due the paper space.

C. Structure Similarities in DFS Search Space

During study the search space of gSpan, we found that there are structure similarities among the DFS search space, which would be very useful for applications in find frequent substructures. There are two important structure similarities:

The first one is ***if a subgraph is grown by a backward edge from its parent, then its children has a one to one relationship with its later siblings.*** For example, in Figure 2, Figure 2 is a sample frequent subgraph in search space, figure (b) to (g) are its structurally full children set (We ignored the variation of the vertex labels and edge labels in this case, as they will not affect the structure relationship among subgraphs). And Figure 2 (b.0) to (b.4) are the full children set of Figure 2 (b). We can see that all of figure (b)'s siblings are a subgraph of one of its children. The only difference among its children and siblings are the backward edge it grow from its parent, Figure 2 (a).

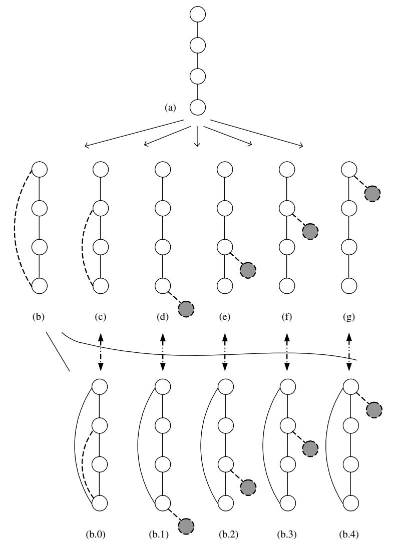


Fig. 2. Structure Similarities: backward edge extension

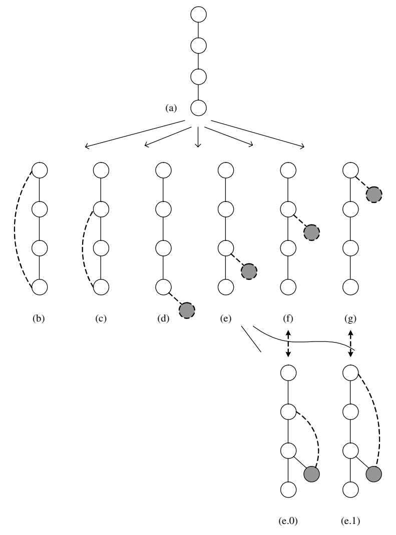


Fig. 3. Structure Similarities: forward edge extension

The second structure similarity is that ***if a subgraph is grown by a forward edge from its parent, then all of its children with a backward edge grow from it has a one to one relationship with its later siblings.*** This similarity is not as straight forward as the first one, before the one-to-one match, one transformation would be applied to the children set of the subgraph. However, the sibling node is still a subgraph of the corresponding child node. As you can see in Figure 3, Figure (e.0) and (e.1) are the children set with a backward edge grown from Figure (e). They have a one-to-one relationship with the Figure 3 (f) and (g). The transformation is simple. First of all, change the last backward edge of its children into forward edges, then delete the forward edge, which current subgraph grew from the base subgraph, in its children.

Under these two similarities, we can see that the sibling nodes are always a subgraph of a child node if the condition of the similarities are satisfied. So while we are exploring the DFS Search Space, after we find out the frequencies of the children nodes/subgraphs, we can always assign the graph index to the sibling subgraph they are reflected to due to the Frequency Antimonotone. We name this property **Backward Graph Index Assignation**.

IV. The FSP Algorithm

In this section, we formulate our FSP algorithm based on DFS code, DFS lexicographic order, DFS balance, DFS code tree structure similarities and their properties.

Algorithm 1 FSP Algorithm

```
 1: Sort labels of the vertex set $\mathbb{V}$ and edge set $\mathbb{E}$ in $\mathbb{GDB}$ by
    their frequency;
 2: Remove infrequent vertices and edges in $\mathbb{V}$ and $\mathbb{E}$;
 3: Relabel the remaining vertices and edges in descend-
    ing/ascending frequency;
 4: Initialize search space $\mathbb{SP}$ with all frequent $1-$edge sub-
    graphs;
 5: for all subgraph $\mathbb{G} \in \mathbb{SP}$ do
 6:   Build_DFS_Tree($\mathbb{G}$);
 7:   if $\mathbb{G}$.startingVertex() $\neq \mathbb{V}$.getFirst() then
 8:     $\mathbb{V}$.deleteFirst();
 9:   end if
10: end for
```

Line $(1-4)$ removes the infrequent vertices and edges from the graph set $\mathbb{GDB}$. If a vertex or edge is not frequent, then it could not be contained in a frequent subgraph. Relabel the remaining ones in proper order (ascending order or descending order by requirement) by frequency. Add all frequent $1-$edge subgraphs into the DFS code tree, or the search space $\mathbb{SP}$ in lexicographic order.

For each frequent $1-$edge subgraph, **Build_DFS_Tree** grows all the DFS nodes in the subtree rooted at this graph. This procedure will be elaborated later. Line 7 shrink the vertex set when all the subgraphs start from the vertex is explored.

Algorithm 2 Build_DFS_Tree($\mathbb{G}$))

Require: A candidate subgraph $\mathbb{G}$

```
1: getFrequency($\mathbb{G}$);
2: if $\mathbb{G}$ is frequent then
3:   Generate_Children($\mathbb{G}$);
4:   for all child $\in \mathbb{G}$ do
5:     Build_DFS_Tree($\mathbb{G}$);
6:   end for
7:   backward_Assignation($\mathbb{G}$);
8: end if
```

A graph $\mathbb{G}$ with minimum DFS code is passed into this function. Line 1 check the frequency of the subgraph in proper graph set, which is performing the subgraph isomorphism test between $\mathbb{G}$ and the graphs. The graph set to be tested is the graph indexes of $\mathbb{G}$'s parent subgraph. If the subgraph is frequent in the database, then generate all the children of this subgraph. Line 4 is a recursive call to further grow the subgraph in DFS manner.

After all the children are explored, the subgraph $\mathbb{G}$ are passed into the subprocedure **backward_Assignation**, which will find out whether there are structure similarities between the children of $\mathbb{G}$ and its siblings. If similarities are found, then the subprocedure will pass all the graph index from child subgraphs of $\mathbb{G}$ to its corresponding siblings. So the number of subgraph isomorphism check is minimized by the size of child subgraph's graph index.

Algorithm 3 Generate_Children($\mathbb{G}$)

```
1:  for all valid positions do
2:    for all valid edges $e$ do
3:      $\mathbb{G}' = \mathbb{G}$
4:      $\mathbb{G}'$.addBackwardEdge();
5:      fastPruning($\mathbb{G}'$);
6:      MinCheck($\mathbb{G}'$);
7:      if $\mathbb{G}'$ is minimum DFS code then
8:        $\mathbb{G}$.addedChild($\mathbb{G}'$);
9:      end if
10:   end for
11: end for
12: for all valid positions do
13:   for all valid edges $e$ do
14:     $\mathbb{G}' = \mathbb{G}$
15:     $\mathbb{G}'$.addForwardEdge();
16:     fastPruning($\mathbb{G}'$);
17:     MinCheck($\mathbb{G}'$);
18:     if $\mathbb{G}'$ is minimum DFS code then
19:       $\mathbb{G}$.addedChild($\mathbb{G}'$);
20:     end if
21:   end for
22: end for
```

Procedure **Generate_Children** generate all the possible child subgraphs for subgraph $\mathbb{G}$. Once a candidate child is generated, the procedure will perform the *fastPruning* and *minCheck*. FastPruning procedure apply the DFS balance concept, perform a DFS balance by level. MinCheck is an application for graph isomorphism check between two graphs. Due to the DFS lexicographic order defined in the gSpan, if the DFS code of a subgraph is not the minimum DFS code, then this subgraph must have been discovered before and it could be pruned from the search space.

V. Experiments and Performance Evaluation

All experiments of FSP and gSpan are done on a 3.0 GHz Intel Pentium(R)4 PC with 3.0GB of RAM, running Window XP Professional with SP2.

A. Chemical Compound Dataset

The chemical dataset that we used in our experiments was obtained from the URL [1]. The entire data set contains 340

Functions	10%	15%	20%	25%	30%	40%
gSpan	4183	1510	547	368	202	48
-p	4182	1510	549	360	201	50
-p -b	4185	1511	548	360	201	48
-p -s	793	302	123	82	49	17
-b	4177	1518	547	359	201	48
-s	797	299	123	83	49	17
-o	386	208	115	82	63	33
-o -p	372	205	113	82	63	33
-o -p -b	370	205	113	89	64	33
-o -p -s	103	51	31	23	18	12
-o -b	385	210	114	83	63	32
-o -s	104	51	31	23	18	12

TABLE II
Runtime Results

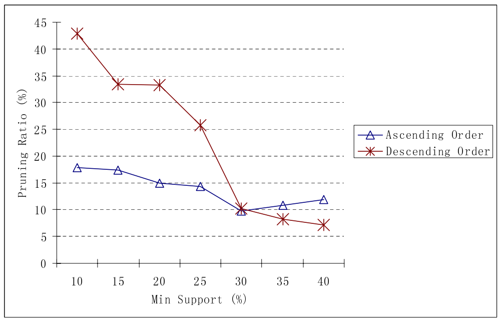


Fig. 4. Pruning Ratio

chemical compounds and for each compound it lists the set of atoms that it is made off and how these atoms are connected together, i.e., their bonds. Each atom is specified as a pair of *element-name* and *element-type*, and each bond is specified by its bond-type. The entire dataset contains 24 different element names [2], 66 different element types, and four different types of bonds.

B. Experimental Results and Analysis

There are four options when running the program: (1)"-o" indicates using descending label order is used, default is ascending order; (2) "-s" indicates enable structural data isomorphism test, default is back-track algorithm; (3) "-p" indicates enable fast pruning function, by default, this function is disabled; (4) "-b" indicates enable backward assignation function, by default, this function is disabled.

The runtime results are shown in Table II for clarification. The unit of all the data is in second. From Table II, we can see that the performance are mainly affected by the structural subgraph tests and the descending order. Other options could hardly differentiate the runtime results from one to another.

The runtime runtime improvements by '-o', '-s' and both '-o' and '-s'over gSpan are shown in Table II. As you can see from the table, the performance of structural subgraph search over gSpan from 64.6% to 80.9%. And performance of descending label order over gSpan from 31.3% to 90.8%. And overall improvements is from 75.0% to 97.5%.

[1]http://web.comlab.ox.ac.uk/oucl/research/areas /machlearn/PTE/

[2]As, Ba, Br, C, Ca, Cl, Cu, F, H, Hg, I, K, Mn, N, Na, O, P, Pb, S, Se, Sn, Te, Ti and Zn.

In Table II, we can see that **fast pruning** function could improve the performance even though it is not that large. As shown in Figure 4, at minimum support of 10% and 15% with descending order and backward assignation enabled, fast pruning could perform around 2% faster.

VI. Conclusion

In this paper, we have developed an algorithm that can perform frequent subgraph mining with a better performance in both theoretical analysis and practical results. The algorithm proposed in the paper has the better performance over all other algorithms in graph mining so far.

The contribution of this paper is summarized as follows: (1) improving the canonical graph representation used in gSpan, and defining relationship between sub-DFS trees whose root share the same parent node in the candidate subgraphs; (2) discovering the structural similarities in the DFS Search Space, and presenting the two one-to-one reflection between children sub graph and latter sibling subgraphs; (3) applying the two techniques to subgraph mining problem to reduce the number of graph and subgraph isomorphism tests effectively; and (4) implementing the FSP algorithm and other current related techniques and comparing the results.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 51, Washington, DC, USA, 2002. IEEE Computer Society.

[3] G. Cong, L. Yi, B. Liu, and K. Wang. Discovering frequent substructures from hierarchical semi-structured data. In *SDM*, 2002.

[4] T. H. Cormen, R. L. Rivest, C. Stein, and C. E. Leiserson. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[5] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997.

[6] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 549, Washington, DC, USA, 2003. IEEE Computer Society.

[7] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD '00: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 13–23, London, UK, 2000. Springer-Verlag.

[8] T.-L. Liu and D. Geiger. Approximate tree matching and shape similarity. In *ICCV International Conference on Computer Vision (1)*, pages 456–462, 1999.

[9] B. D. McKay. *Practical graph isomorphism*. Congressus Numerantium, 1981.

[10] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652, Seattle, WA, USA, 2004.

[11] L. B. H. Shaobing Su, Diane J. Cook. Knowledge discovery in molecular biology: Identifying structural regularities in proteins. *Intelligent Data Analysis*, pages 413–436, 1999.

[12] Y. Takahashi, M. Sukekawa, and S. ichi Sasaki. Automatic identification of molecular similarity using reduced-graph representation of chemical structure. *Journal of Chemical Information and Computer Sciences*, 32(6):639–643, 1992.

[13] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 721, Washington, DC, USA, 2002. IEEE Computer Society.