TẠP CHÍ KHOA HỌC
TRƯỜNG ĐẠI HỌC QUY NHƠN

Thuật toán khai phá đồ thị con thường xuyên theo mô hình MapReduce

Trần Thiên Thành*, Trần Thị Liên, Nguyễn Thị Kim Phượng

Khoa Công nghệ thông tin, Trường Đại học Quy Nhơn, Việt Nam

Ngày nhận bài: 05/05/2020; Ngày nhận đăng: 03/07/2020

TÓM TẮT

Trong bài báo này chúng tôi trình bày về thuật toán gSpanMR, một thuật toán khai phá đồ thị con thường xuyên theo mô hình lập trình MapReduce. Thuật toán gSpanMR (gSpan MapReduce) được phát triển dựa trên kết quả mã hóa đồ thị bằng DFS code của thuật toán gSpan và thuật toán khai phá đồ thị con thường xuyên trên mô hình MapReduce FSM-H. Thuật toán được cài đặt và thực nghiệm trên cụm máy tính sử dụng nền tảng Hadoop cho thấy tốc độ có cải tiến so với thuật toán FSM-H.

Từ khóa: *Khai phá đồ thị con thường xuyên, MapReduce, DFS Code.*

**Tác giả liên hệ chính.*
Email: tranthienthanh@qnu.edu.vn

JOURNAL OF SCIENCE
QUY NHON UNIVERSITY

A frequent subgraph mining algorithm on the MapReduce model

Tran Thien Thanh*, Tran Thi Lien, Nguyen Thi Kim Phuong

Faculty of Information Technology, Quy Nhon University, Vietnam

Received: 05/05/2020; Accepted: 03/07/2020

ABSTRACT

In this paper, we present the gSpanMR algorithm, a frequent subgraph mining on the MapReduce programming model. The gSpanMR algorithm (gSpan MapReduce) was developed based on the results of encoding graphs using the DFS code in gSpan algorithm and FSM-H algorithm on the MapReduce model. gSpanMR algorithm has been installed and tested on computer clusters using Hadoop platform, which shows better execution time than FSM-H algorithm.

Keywords: *Frequent subgraph mining, MapReduce, DFS Code.*

1. INTRODUCTION

Frequent subgraph mining has been an emerging data mining problem with many scientific and commercial applications. There exist many algorithms for solving the in-memory version of frequent subgraph mining task, most notable among them are AGM,[1] FSG,[2] gSpan,[3] Gaston,[4] and DMTL.[5] These methods assume that the dataset is small and the mining task finishes in a reasonable amount of time using an in-memory method. To consider the large data scenario, a few traditional database based graph mining algorithms, such as, DB-Subdue,[6] and DB-FSG[7] and OOFSG[8] are also proposed. The kernel of frequent subgraph mining is subgraph isomorphism test. Lots of well-known pair-wise isomorphism testing algorithms were developed. However, the frequent subgraph mining problem was not explored well. One approach used by many researchers to solve problems with large data and complexity is to use parallel computational models on computer clusters. We use the results of DFS code proposed in gSpan algorithm[3] combined with the idea of implementing FSM-H algorithm[9] on Hadoop to build the gSPanMR algorithm to frequent subgraphs mining on the MapReduce programming model.

The paper is organized as follows: Section 1 introduces research issues; Section 2 presents some concepts of frequent subgraph mining and two algorithms gSpan, FSM-H; Section 3 presents the details of the gSpanMR algorithm; Section 4 presents the experimental time comparison of FSM-H and gSpanMR algorithms; The final section are conclusions and some further developments of the paper.

2. FREQUENT SUBGRAPH MINING

Definition 1 (*Labeled graph*)[3] A labeled graph can be represented by a 4-type, $G = (V, E, L, l)$, where:

*Corresponding author.
Email: tranthienthanh@qnu.edu.vn

• V is a set of vetices,

• $E \subseteq V \times V$ is a set of edges,

• L is a set of labels,

• l: $V \cup E \rightarrow L$ is a function assigning labels to the vetices and the edges.

Definition 2 (*Isomorphism graph*)[3] An issomophism is a bijective function f: $V(G) \rightarrow V(G')$, such that:

• $v \in V(G)$, $l_G(v) = l_{G'}(f(v))$,

• $(u, v) \in E(G)$, $(f(u), f(v)) \in E(G')$ and $l_G(u, v) = l_{G'}(f(u), f(v))$.

Definition 3 (*Frequent subgraph*)[3] Given a graph dataset, $D = \{G_1, G_2, \ldots, G_n\}$ and a minimum support *minSup*. A graph G is a *frequent subgraph* in D with *minSup* if $\sup(G, D) \geq \text{minSup}$, where $sup(G, D) = |\{G_i \in D: G \text{ is isomophism to a subgraph of } G_i\}|$.

In the following section, I will discuss the concepts of DFS code and some properties that help to build gSpan algorithm.

Given an undirected connected graph G. The *DFS tree* of graph G is a tree built from the vertices and the edges of the graph G by DFS algorithm.[10] The DFS tree creates a linear order between vertices and edges of the graph in their order of visit.

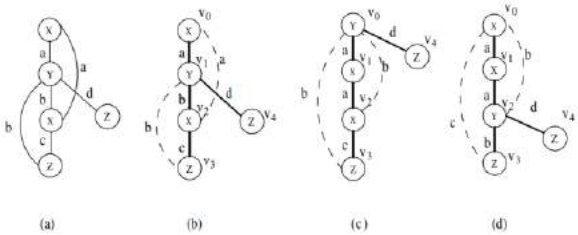

Figure 1. DFS tree and Forward/Backward edge[3]

Definition 4. (*DFS code*)[3]. Given a DFS tree T for a graph G, an edge sequence (e_i) such $e_i < e_{i+1}$, where $i = 0, 1, \ldots, |E(G)|$-2, (e_i) is called a DFS code, denoted as $code(G, T)$.

An edge $e = (v_i, v_j)$ in a DFS code is a 5-tupe: $(i, j, l_i, l_{(i,j)}, l_j)$. Table 1 shows the corresponding DFS codes for Figure 1 (b), (c), (d).

Table 1. DFS codes for Figure 1 (b), (c), (d)

Edge no.	(Fig. 1b) α	(Fig. 1c) β	(Fig. 1d) γ
0	(0,1,*X*,*a*,*Y*)	(0,1,*Y*,*a*,*X*)	(0,1,*X*,*a*,*X*)
1	(1,2,*Y*,*b*,*X*)	(1,2,*X*,*a*,*X*)	(1,2,*X*,*a*,*Y*)
2	(2,0,*X*,*a*,*X*)	(2,0,*X*,*b*,*Y*)	(2,0,*Y*,*b*,*X*)
3	(2,3,*X*,*c*,*Z*)	(2,3,*X*,*c*,*Z*)	(2,3,*Y*,*b*,*Z*)
4	(3,1,*Z*,*b*,*Y*)	(3,0,*Z*,*b*,*Y*)	(3,0,*Z*,*c*,*X*)
5	(1,4,*Y*,*d*,*Z*)	(0,4,*Y*,*d*,*Z*)	(2,4,*Y*,*d*,*Z*)

Definition 5 (*DFS code Lexicographic Order*)[3] If $\alpha = code(G_\alpha, T_\alpha) = (a_0, a_1, \ldots, a_m)$ and $\beta = code(G_\beta, T_\beta) = (b_0, b_1, \ldots, b_n)$ then $\alpha \leq \beta$ if either of the following is true:

i) $\exists t$, $0 \leq t \leq min(m, n)$, $a_k = b_k$ for $k < t$, $a_t < b_t$

ii) $a_k = b_k$ for $0 \leq k \leq m$, and $n \geq m$.

Definition 6 (Minimum DFS code)[3] Given a graph G, $Z(G) = \{code(G, T) \mid T \text{ is a DFS tree for } G\}$, based on DFS lexicographic order, the minimum one, $min(Z(G))$, is called *Minimum DFS code* of G, denoted as $min(G)$.

The minimum DFS code of a G graph is also called a labeling scheme for graph G. Based on this labeling, we have the result of isomorphism of the two graphs.

Theorem 1[3] Given two graphs G and G', G is isomophism to G' if and only if $min(G)=min(G')$.

Definition 7 (*DFS code's Parent and Child*)[3] Given a DFS code $\alpha = (a_0, a_1, \ldots, a_m)$, any DFS code valid $\beta = (a_0, a_1, \ldots, a_m, b)$ is called α's *child*, and α is called *β's parent*. We denote $children(\alpha) = \{\beta \mid \text{ is } \beta\text{'s parent}\}$.

The growth from the DFS code α to valid DFS code β is necessary for frequent subgraph mining. In fact, to construct a valid DFS code, b must be an edge which only grows from the vertices on the rightmost path. In figure 2 the graph shown in 2(a) has several potential children with one edge growth, which are shown in 2(b)-2(f) (assume the drakened vertices constitute the rigthmost path).

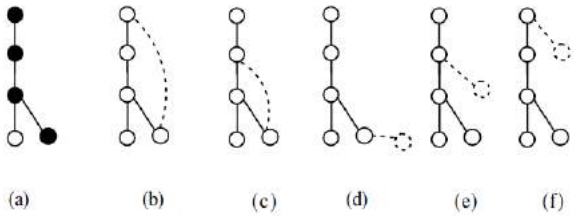

Figure 2. DFS code growing[3]

The DFS code together with the parent-child relationship forms a tree, called the *DFS code tree*. In the DFS code tree each node is a DFS code of a graph. The nodes at the i^{th} level are DFS codes of graphs with i-1 edges. Figure 3 illustrates the DFS code tree.

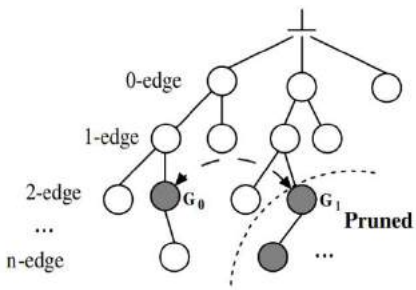

Figure 3. DFS code tree[3]

We can truncate the branches of the DFS code tree that do not contain the minimum DFS code with the following result:

Theorem 2 (*DFS code Growth*)[3] Given a DFS code β, if $\alpha = min(\beta)$, $\alpha < \beta$. Let $D_\gamma = \{\eta \mid \eta < \gamma\}$, $\forall\ \sigma \in children(\beta)$, $min(\sigma) \in D_\alpha \cup children(\alpha) \subseteq D_\beta$.

From the above results, the gSpan algorithm starts from the 1-edge frequent subgraph and grows for more-edge graphs until the graph expands infrequently.

However, the gSpan algorithm complexity is an exponential function in the worst case scenario. Therefore, frequent subgraph mining on large graph sets is still a challenge.

To contribute to addressing this challenge, an approach is to build algorithms based on the MapReduce model to perform in parallel on the computer cluster to reduce execution time. MapReduce model is a parallel programming model on cluster of computers proposed by Google[11]. Then Apache Hadoop is an open source platform that supports this programming model and is widely used in research and applications.

The FSM-H algorithm is an approach to frequent subgraphs mining on the MapReduce model. The FSM-H algorithm works on a graph set of data sets consisting of k parts, one on a data node. The multi-step iterative algorithm, at the i^{th} step, is the candidate i-edge subgraph from the i-1-edge frequent subgraph, then calculates the support for each candidate at each node and passes it to the reduce function summarizes and determines which subgraphs are often saved along with a list of graphs that contain it as input for the next step. The process is repeated until no more frequent subgraphs are discovered.

The FSM-H algorithm uses minimal DFS code to identify isomorphic subgraphs. However, the this algorithm has some limitations during the frequent subgraph mining: The generation of candidate i-edge subgraphs from i-1-edge frequent subgraphs is still redundant because it contains many duplicate (isomorphic) subgraphs. So data transfer and calculation are also more. For each i-edge subgraph in the candidate set, the algorithm finds the minimum DFS code to homogeneous subgraphs from each other, thereby sending the results to the reduce function. The algorithm to find the minimum DFS code is exponentially complex. If instead of finding the minimum DFS code by checking the DFS code is minimal, the computational complexity of the algorithm will be reduced if the DFS code is not minimal but the correctness of the algorithm is still guaranteed. To overcome these limitations, we improved the FSM-H algorithm based on the properties of DFS code as shown in the following section.

3. gSpanMR ALGORITHM

To overcome the limitations of the FSM-H algorithm, we use the following property in generating i-edge subgraph candidates from regular i-1-edge frequent subgraphs.

Property 1 Given $\alpha = (a_0, a_1, \ldots, a_n)$ is a minimum DFS code of a graph G. If $\beta = (a_0, a_1, \ldots, a_n, b)$ is a minimum DFS code then the following is true:

i) b is greater than or equal to a_0.

ii) If b is a backward edge of the form $(v_i, v_j), j < i$ then b is greater than or equal to the edge of α that contains v_j.

iii) If b is a forward edge of the form $(v_i, v_j), j > i$ then b is greater than or equal to the edge of α that contains v_i.

Proof.

Let G' be the graph of DFS code β.

i) Assume $b < a_0$. We build a DFS code β' of graph G' that b is the first edge of the DFS code. It is easy to see $\beta' < \beta$, which contradicts the hypothesis β is the minimum DFS code.

ii) If a_k $(0 \le k \le n)$ contains vertices v_j and $a_k > b$ then DFS code $\beta' = (a_0, a_1, ..., a_{k-1}, b', a'_n, ..., a'_{k+1}, a_k)$, is a DFS code of G' and $\beta' < \beta$ should conflict with the hypothesis β is the minimum DFS code of G'. Let $b = (i, j, l_i, l_{(i,j)}, l_j)$ we denote $b' = (j, i, l_j, l_{(j,i)}, l_i)$.

iii) If $b < a_k$ $(0 \le k \le n)$ then $\beta' = (a_0, a_1, \ldots, a_{k-1}, b, b_k, \ldots, b_n)$ is a DFS code of G', and $\beta' < \beta$, should conflict with the hypothesis is the minimum DFS code of G'.

We build an algorithm to generate DFS codes k+1-edge from minimum DFS code k-edge and graph G, based on property 1.

Algorithm 1. Generate DFS codes k+1-edge

Input: Minimum DFS code k-edge *dfsMin*, graph G

Output: $\{\beta \mid \beta \in children(dfsMin), \beta$ is a DFS code of a subgraph in $G\}$

Format: **Canditate_generation**(*dfsMin, G*)

Action:

1. $C = \varnothing$
2. *rmp* = **rightMostPath**(*dfsMin*)
3. *vMax* = *rmp*[0]
4. *minLabel* = *dfsMin*[0].*fromLabel*

// Backward edges

5. **for each** *v* **in** *rmp*:
6. $e = (vMax, v)$
7. **if** *e* **in** *G* **and** *e* **not in** *dfsMin* **then**
8. **for each** *e*' **in** *dfsMin*:
9. **if** *e*' > *e* **then continue**
10. $dfs = dfsMin \cup \{e\}$
11. $C = C \cup dfs$

// Forward edge from *vMax*

12. **for each** *v* **in** *G.adjList*(*vMax*):
13. $e = (vMax, v)$
14. **if** *e* **not in** *dfsMin* **and** *label*(*e*) >=*minLabel* **then**
15. $dfs = dfsMin \cup \{e\}$
16. $C = C \cup dfs$

//Forward edges of the form (u,v), where u in *rmp*

17. **for each** *v* **in** *rmp*, $v \neq vMax$:
18. **for each** *v*' **in** *G.adjList*(*v*):
19. $e = (v, v')$
20. **if** *e* **not in** *dfsMin* **then**
21. **for each** *e*' **in** *dfsMin*:
22. **if** *e*'>*e* **then continue**
23. $dfs = dfsMin \cup \{e\}$
24. $C = C \cup dfs$
25. **return** *C*

The correctness of the algorithm is confirmed by property 1. The time complexity of algorithm 1 in the worst case is O ($k.m$), where k is the number of edges in *dfsMin*, and $m = |E(G)|$.

In the gSpanMR algorithm we check if a DFS code is the minimum DFS code? The algorithm is as follows:

Algorithm 2. Check the minimum DFS code.

Input: DFS code for check *dfsCode*

Output: **True** if *dfsCode* is minimum, **False** otherwise

Format: **isMin**(*dfsCode*).

Action:

1. g = **to_Graph**(*dfsCode*)
2. *minDFS* = ∅
3. **return tryIsMin**(*dfsCode*, *minDFS*, 0)

Procedure **tryIsMin**(*dfsCode, minDFS, i*) checks whether the i^{th} edge of *dfsCode* is a edge in the minimum DFS code; *minDFS* is the minimum DFS code *i*-1 edge.

Sub tryIsMin(*dfsCode*, *minDFS*, *i*)

1. **if** *i* > **length**(*dfsCode*) **then return true**
2. *list* = List of smallest edges in *g* can grow for *minDFS*
3. **for each** *e* **in** *list*:
4. **if** *e* < *dfsCode*[*i*] **then return False**
5. *minDFS* = *minDFS* ∪ {*e*}
6. *chk* = **tryIsMin**(*dfsCode*, *minDFS*, *i*+1)
7. if *chk* = **False then return False**
8. *minDFS* = *minDFS* – {*e*}

Algorithm 2 uses branch-and-bound method, so in practice it is better than the algorithm to find the minimum DFS code of a graph, although the worst case complexity of algorithm 2 is $O(2^n)$, where *n* is the number of edges of *dfsCode*.

Based on the gSpan algorithm and the approach of the FSM-H algorithm, we developed the gSpanMR algorithm with 3 phases:

Phase 1 (Preparation): this phase reads data from HDFS, explores frequent edges of a set of graphs and writes the results to HDFS to prepare for phase 2.

Phase 2 (Mining): this phase explores frequent *k*+1-edge subgraphs from frequent k-edge subgraphs. This phase consists of many steps starting from *k* = 0 until no more frequent subgraphs are explored.

Phase 3 (Collection): this phase collects all sub-graphs often exploited in phase 2.

Algorithm 3. gSpanMR

Input: *D* is set of graphs, minimum support *minSup*

Output: Set of frequent subgraphs of *D* with minimum support *minSup*

Action:

// Phase 1

key: id of a graph

value: a graph

Mapper_Preparation(**Text** *key*, **Text** *value*)

1. *intermediate_key = null*
2. *intermediate_value* = **serialize**(*value*) // Convert graph into byte sequence
3. **omit**(*intermediate_key*, *intermediate_value*)

key: a DFS Code

values: byte sequences of graphs

Reducer_Preparation(**DFSCode** *key*, **ByteWritable** *values*)

1. **forall** *value* **in** *values*:
2. **write_to_file**(*key*, *value*)

// Phase 2

key: a DFS code *k*-edge

value: byte sequence of graphs that contain key

Mapper_Mining(**DFSCode** *key*, **ByteWritable** *value*)

1. *listGraph* = **convert_to_List_Graph**(*value*) //convert byte sequence to list of graphs
2. *C* = ∅
3. **forall** *g* **in** *listGraph*:
4. *C* = *C* ∪ **Canditate_generation**(*key, g*)
5. **forall** *c* **in** *C*:
6. **if isMin**(*c*):
7. *v* = **serialize**(*c.OL*)
8. **emit**(*c, v*)

key: a DFS Code

values: byte sequences of graphs

Reducer_Mining(**DFScode** *key*, **ByteWritable** <*values*>)

1. **forall** *value* **in** *values*:
2. *support*+=**getsupport**(*value*)
3. **if** *support* ≥ *minSup*:
4. **forall** *value* **in** *values*:
5. **write_to_file**(*key*, *value*)

// Phase 3

key: a DFS code *k*-edge

value: byte sequence of graphs that contain key

Mapper_Collection(DFSCode *key*, **ByteWritable** *value*)

1. *gText* =

convert_DFSCode_to_Graph_Text(*key***)**

//Convert DFS code to graph in format text

2. **emit(***gText*, **null)**

key: a abugraph in format text

values: null

Reducer_Collection(Text *key,* **Text** *values***)**

1. **write_to_file(***key***)**

4. IMPLEMENTATION AND EXPERIMENT

We have implemented the gSpanMR algorithm on the Hadoop platform in the Java programming language. We conducted experiments on a system of 5 PCs with CPU: Intel Core 2 Duo E8400 3.00GHz, RAM: 4GB, installed Ubuntu 14.04, Hadoop 2.7.3, including 1 MasterNode and 4 NameNode. Three datasets are created from *Graphgen* tool with the number of graphs are 5000, 7000 and 10000 graphs respectively, the number of vertices in the graph is from 20 to 40 vertices, the minimum support is 20%. Experimental results as shown in Table 2.

Table 2. Runtime of FSM-H and gSpanMR

Number of graphs	FSM-H	gSpanMR
5000	1050s	820s
7000	2134s	1721s
10000	3930s	3122s

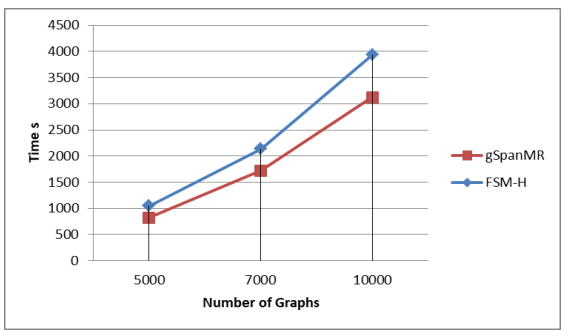

Figure 4. Experimental results comparing the execution time of FSM-H and gSpanMR

Through experimental results show that the execution time of the algorithm gSpanMR less FSM-H algorithm. The more the graphs increase, the more the difference in time increases due to the reduction of many subgraph candidates and the minimum DFS code test.

5. CONCLUSIONS

With the improved subgraph candidates and testing the minimum DFS code in the gSpanMR algorithm, it has been shown to improve the time it takes to perform the frequent subgraph mining algorithm. In the future, we will continue to improve gSpanMR algorithm and implement gSpanMR algorithm on Spark environment to limit the reading and writing data in external memory.

ACKNOWLEDGEMENTS

This study is conducted within the framework of science and technology projects at institutional level of Quy Nhon University under the project code T2019.625.20.

REFERENCES

1. Inokuchi, T. Washio, and H. Motoda. *An apriori-based algorithm for mining frequent substructures from graph data*, 4th Eur. Conf. Principles Data Mining Knowl. Discov., PKDD, Lyon, France, September 13-16, 2000.
2. M. Kuramochi and G. Karypis. *Frequent subgraph discovery*, 1st IEEE International Conference on Data Mining, IEEE, San Jose, California, USA, 29 November - 2 December 2001.
3. Yan and J. Han. *gSpan: Graph-based substructure pattern mining*, Int. Conf. Data Min. IEEE, Maebashi City, Japan, 9-12 December 2002.
4. S. Nijssen, and J. Kok. *A quickstart in frequent structure mining can make a difference*, 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, ACM, Seattle, WA, USA, 22-25 August, 2004.

5. V. Chaoji, M. Hasan, S. Salem, and M. Zaki. An integrated, generic approach to pattern mining: Data mining template library, *Data Mining Knowledge Discovery*, **2008**, *17*(3), 457–495.

6. S. Chakravarthy, R. Beera, and R. Balachandran. *Db-subdue: Database approach to graph mining*, 8th Adv. Knowl. Discov. Data Mining, IEEE, Sydney, Australia, 26-28 May, 2004.

7. S. Chakravarthy and S. Pradhan. *Db-FSG: An SQL-based approach for frequent subgraph mining*, 19th Int. Conf. Database Expert Syst. Appl., IEEE, Turin, Italy, 1-5 September, 2008.

8. Srichandan and R. Sunderraman. *Oo-FSG: An object-oriented approach to mine frequent subgraphs*, 9th Australasian Data Mining Conf., IEEE, Ballarat, Australia, December, 2011.

9. Mansurul A. Bhuiyan and Mohammad Al Hasan. An Iterative MapReduce Based Frequent Subgraph Mining Algorithm, *IEEE Transactions on Knowledge and Data Engineering*, **2015**, *27*(3), 608-620.

10. Cormen, H. Thomas, Leiserson, E. Charles, Rivest, L. Ronald. *Introduction to Algorithms*, 1st ed., MIT Press and McGraw-Hill, NewYork City, 1990.

11. J. Dean and S. Ghemawat. *Mapreduce: Simplified Data Processing on Large Clusters*, 6th Symposium on Operating System Design and Implementation, San Francisco, OSDI, December 6-8, 2004.