# Complete and incomplete approaches for graph mining*

Amina Kemmar [1], Yahia Lebbah [1], Mohammed Ouali [1], and Samir Loudni [2]

University of Oran, Es-Senia, Lab. LITIO,
B.P. 1524 EL-M'Naouar, Oran, Algeria
[2] University of Caen - Campus II, Department of Computer Science, France
{kemmami,ylebbah}@yahoo.fr,mohammed.ouali@gmail.com,samir.loudni@unicaen.fr

**Abstract.** In this paper, we revisit approaches for graph mining where a set of simple encodings is proposed. Complete approaches are those using an encoding allowing to get all the frequent subgraphs. Whereas incomplete approaches do not guarantee to find all the frequent subgraphs. Our objective is also to highlight the critical points in the process of extracting the frequent subgraphs with complete and incomplete approaches. Current canonical encodings have a complexity which is of exponential nature, motivating this paper to propose a relaxation of canonicity of the encoding leading to complete and incomplete encodings with a linear complexity. These techniques are implemented within our graph miner GGM (Generic Graph Miner) and then evaluated on a set of graph databases, showing the behavior of both complete and incomplete approaches.

**Keywords:** Graph mining, frequent subgraph, pattern discovery, graph isomorphism.

## 1 Introduction

Graph-mining represents the set of techniques used to extract interesting and previously unknown information about the relational aspect from data sets that are represented with graphs.

We revisit some approaches for graph mining where a set of simple encodings is proposed. Complete approaches are those using an encoding enabling to get all the frequent subgraphs. Whereas incomplete approaches do not guarantee to find all the frequent subgraphs. Our objective is also to highlight the critical points in the process of extracting the frequent subgraphs. The introduced techniques are implemented within GGM (generic graph miner). We provide an experimentation with GGM showing the behavior of complete and incomplete approaches. It is not proven if the canonical encoding of graphs is in the class of NP-complete problems, nor in polynomial class. This is also verified in practice,

* This work is supported by TASSILI research program 11MDU839 (France, Algeria).

since that all the current canonical encodings have complexities which are of exponential nature. This motivates deeply our work on proposing a relaxation of the canonicity of the encoding, leading us to what we qualified in this paper *complete* and *incomplete* encodings with low polynomial complexities.

The following section 2 introduces preliminaries on graph mining and the current approaches to solve frequent subgraph discovery problem. Section 3 explains our graph mining algorithm GGM. Experimental results of GGM are given in section 4. Section 5 concludes the paper and addresses some perspectives.

## 2 Frequent subgraph discovery problem

An undirected graph $G = (V, E)$ is made of the set of vertices $V$ and the set of edges $E \subseteq V \times V$. Each edge $(v_1, v_2)$ is an unordered pair of vertices. We will assume that the graph is labeled with vertex labels $L_V$ and edge labels $L_E$; the same label can be assigned to many vertices (or edges) in the same graph. The size of a graph $G = (V, E)$ is defined to be equal to $|E|$.

**Definition 1 (Frequent Subgraph discovery).** *Given a database $\mathcal{G}$ which contains a collection of graphs. The frequency of a graph $G$ in $\mathcal{G}$ is defined by $freq(G, \mathcal{G}) = \#\{G' \in \mathcal{G} | G \subseteq G'\}$. The support of a graph is defined by*

$$support(G, \mathcal{G}) = freq(G, \mathcal{G})/|\mathcal{G}|.$$

*The frequent subgraph discovery problem consists to find all connected undirected graphs $F$ that are subgraphs of at least $minsup|\mathcal{G}|$ graphs of $\mathcal{G}$:*

$$F = \{G \in \mathcal{G} | support(G, \mathcal{G}) \geq minsup\},$$

*for some predefined minimum support threshold* minsup *that is specified by the user.*

Generally, we can distinguish between the methods of discovering frequent subgraphs according to the way the three following problems are handled:

**Candidates generation problem** This is the first step in the frequent subgraph discovery process which depends on the search strategy. It can be done with breadth first or depth first strategies. With breadth first strategy, all $k$-candidates (i.e., having $k$ edges) are generated together, then $(k+1)$-candidates and so on; making the memory consumption huge [4][3]. But with a depth approach, the $k$-candidates are iteratively generated, one by one.

**Subgraph encoding problem** When some new candidate is produced, we should verify that it has been already generated. This can be resolved by testing if this new candidate is isomorphic to one of the already generated subgraphs. The canonical DFS code [6] is usually used to encode the generated frequent subgraphs. By this way, verifying that the new candidate is isomorphic to one of the already generated candidates is equivalent to testing if its encoding is equal to the encoding of some already generated candidate.

**Frequency computation problem** If some new candidate is declared to be not isomorphic to any of the already produced candidates, we should compute its frequency. It could be done by finding all the graphs of the database which contain this new candidate.

In the following section, we present a new algorithm GGM - Generic Graph Miner - for finding connected frequent subgraphs in a graphs database. We propose also some simple encodings to handle efficiently the frequency counting problem.

## 3 GGM, a generic graph miner

GGM finds frequent subgraphs, parameterized with some encoding strategies detailed in section 3.1. It is generic, because we aim to make the key steps of GGM easily parameterized.

**Algorithm 1** GGM($\mathcal{G}$, $f_{min}$)

**Require:** $\mathcal{G}$ represents the graph dataset and $f_{min}$ the minimum frequency threshold.
**Ensure:** $\mathcal{F}$ is the set of frequent subgraphs in $\mathcal{G}$.
1: $\mathcal{F} \leftarrow \emptyset$
2: $\mathcal{E} \leftarrow$ all frequent edge labels in $\mathcal{G}$
3: $\mathcal{N} \leftarrow$ all frequent node labels in $\mathcal{G}$
4: $\mathcal{P} \leftarrow$ Generate-Paths($\mathcal{N}$, $\mathcal{E}$, $\mathcal{G}$, $f_{min}$)
5: $\mathcal{T} \leftarrow$ Generate-Trees($\mathcal{P}$, $\mathcal{E}$, $\mathcal{G}$, $f_{min}$)
6: $\mathcal{C} \leftarrow$ Generate-Cyclic-Graphs($\mathcal{T}$, $\mathcal{E}$, $\mathcal{G}$, $f_{min}$)
7: $\mathcal{F} \leftarrow \mathcal{P} \cup \mathcal{T} \cup \mathcal{C}$
8: RETURN $\mathcal{F}$

The general structure of the algorithm is illustrated in algorithm 1. The algorithm initializes the frequent subgraphs with all frequent edges and nodes within the graph database $\mathcal{G}$. Then, the algorithm proceeds with three separated steps:

1. enumerating frequent paths from the frequent nodes,
2. generating the frequent trees from the frequent paths by keeping the same extremities of each initial path,
3. extending the frequent paths and trees by adding an edge between two existing nodes to obtain cyclic graphs.

This approach is inspired from GASTON [5] in which these three steps are repeated for each discovered subgraph. In other words, GASTON loops on the above three steps, whereas in our approach, they are executed one time only.

### 3.1 Graph encoding

The canonical labeling is used to check whether a particular candidate subgraph has already been generated or not. However, developing algorithms that can efficiently compute the canonical labeling is critical to ensure that the mining algorithm can scale to very large graph datasets. There exists different ways to assign a code to a given graph, but it must uniquely identify the graph such

that if two graphs are isomorphic, they will be assigned the same code. Such encoding is called a *canonical encoding*. It is not proven if the canonical encoding of graphs is in the class of NP-complete problems, nor in polynomial class. This is also verified in practice, since that all the current canonical encodings have complexities which are of exponential nature.

The idea of our encoding is to use a non-canonical encoding, resulting in two kinds of encodings : complete and incomplete.

**Definition 2 (Complete and incomplete encodings).** *Let $f$ be an encoding function. For any two distinct non-isomorphic graphs $G_1$ and $G_2$, $f$ is complete if $f(G_1) \neq f(G_2)$. Otherwise, $f$ is said to be incomplete.*

**DFS based complete encoding** This encoding is a relaxation of that defined in [6]. Such encoding is processed by taking only one walk through a depth first search (and not the minimum as in [6]). It is straightforward that this encoding is complete, and the same graph can be generated several times as illustrated in Figure 1 which shows that two isomorphic graphs can have different codes. For the graph (a) in Figure 1, there exists several DFS codes. Two of them, which are based on the DFS trees in Figure 1(b)-(c) are listed in Table 1.

| edge | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Fig 1.(b) | $(1, 2, X, s, Y)$ | $(2, 3, Y, t, X)$ | $(3, 1, X, s, X)$ | $(3, 4, X, q, Z)$ | $(4, 2, Z, t, Y)$ | $(2, 5, Y, r, Z)$ |
| Fig 1.(c) | $(1, 2, Y, s, X)$ | $(2, 3, X, s, X)$ | $(3, 1, X, t, Y)$ | $(3, 4, X, q, Z)$ | $(4, 1, Z, t, Y)$ | $(1, 5, Y, r, Z)$ |

**Table 1.** DFS codes for Figure 1 (b)-(c)

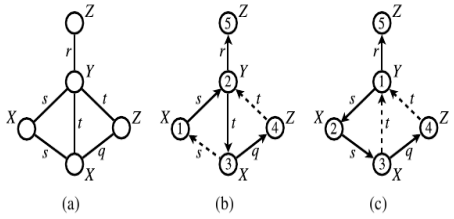


**Fig. 1.** Different DFS trees associated to the labeled graph (a)

Since this encoding visits only once the edges, it is then straightforward that the worst-case complexity is $O(m)$, where $m$ is the number of edges.

**Edge sequence based incomplete encoding** Given a graph $G$ and an edge $e_{ij} = (v_i, v_j) \in G$, where $deg(v_i) \leq deg(v_j)$, the edge $e_{ij}$ is represented by the 5-tuple: $(deg(v_i), deg(v_j), l_v(v_i), l_e(v_i, v_j), l_j(v_j))$,
where $deg(v)$ is the degree of $v$ in the graph $G$, $l_v(v)$ and $l_e(e)$ are the labels of the vertex $v$ and the edge $e$ respectively.
Given a graph $G$, we denote SEQ-DEG($G$) the sequence of its edge codes :

$$SEQ - DEG(G) = code(e_1)code(e_2)...code(e_{|E|})$$

where $e_i <_l e_{i+1}$, the relation $<_l$ defines a lexicographic order between edges (e.g. $(2, 3, X, s, Y) < (2, 3, Z, s, Y)$). The code associated to the graph (a) of Figure 1 is: (1,4,Z,r,Y)(2,3,X,s,X)(2,3,Z,q,X)(2,4,X,s,Y)(2,4,Z,t,Y)(3,4,X,t,Y).

Enumerating the edges is done with $O(m)$, where $m$ is the number of edges, but sorting lexicographically the edges requires $O(m\ log(m))$ which is the worst case complexity of sorting algorithms. Thus, in final, the complexity of this encoding is $O(m\ log(m))$. This encoding is not complete because we can find examples of non-isomorphic graphs having the same code.

## 4 Experimental Results

We performed a set of experiments to evaluate the performance of our algorithm GGM on two kinds of graph databases. The first databases of large graphs contain some molecular structures of chemical compounds (PTE [1]). The second databases of small graphs are extracted from the database PTE (PTE1,PTE2,PTE3). The characteristics of these datasets are illustrated in Table 2. All experiments were done on 2.4Ghz Intel Core 2 Duo T8300 machines with 2GB main memory, running the Linux operating system.

| Name | #graphs | #nodes | #edges | average #nodes | average #edges | #node labels | # edge labels |
|---|---|---|---|---|---|---|---|
| PTE1 | 1 | 8 | 7 | 8 | 7 | 2 | 1 |
| PTE2 | 5 | 98 | 102 | 19 | 20 | 10 | 3 |
| PTE3 | 20 | 519 | 530 | 25 | 26 | 10 | 4 |
| PTE | 340 | 9189 | 9317 | 27 | 27 | 66 | 4 |

**Table 2.** Caracteristics of graph datasets used in the experiments.

| Dataset | PTE1 | PTE2 | PTE3 |
|---|---|---|---|
| MinFreq | 1 | 3 | 8 |
| Gaston | 0,00 | 0,00 | 0,00 |
| GGM SEQ-DEG | 0,02 | 0,13 | 0,17 |
| GGM DFS | 0,22 | 3,42 | 3,94 |

**Table 3.** Runtimes in second of Gaston and GGM on simple graph datasets.

| MinSup % = MinFreq | 20% = 68 | | | 50% = 170 | | | 60% = 204 | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | GASTON | SEQ-DEG | DFS | GASTON | SEQ-DEG | DFS | GASTON | SEQ-DEG | DFS |
| #freq. paths | 53 | 53 | - | 17 | 17 | 17 | 9 | 9 | 9 |
| #freq. trees | 124 | 97 | - | 15 | 14 | 66 | 2 | 2 | 6 |
| #freq. cyclic graphs | 13 | 12 | - | 2 | 2 | 10 | 0 | 0 | 0 |
| # Total | 190 | 162 | - | 34 | 33 | 93 | 11 | 11 | 15 |
| runtime (s) | 0,02 | 2,29 | >2000 | 0,01 | 0,60 | 4,29 | 0,01 | 0,26 | 0,68 |

**Table 4.** Results of Gaston and GGM (with the DFS encoding and SEQ-DEG) on the graph dataset PTE. MinSup represents the minimum support threshold and MinFreq the minimum frequency.

We have done a comparison between our algorithm and the Gaston tool. Table 3 (resp. Table 4) shows the results of our algorithm with the first database (resp. second database). The minimum frequency was set to 1 for the first results. So, this table presents the runtimes to generate all the subgraphs of each database. While for the second case, we choose different MinFreq expressed also by MinSup (i.e. minimum support). For instance, for the PTE database which contains 340 graphs, the number of graphs that are subgraphs of at least

[1] The Predictive Toxicology dataset (PTE) can be downloaded from `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PTE`.

60% = $\frac{204}{340}$ graphs of PTE, is given by $\#Total = \#freq.paths + \#freq.trees + \#freq.cyclic\ graphs$. From this frequency, we see that there is no frequent cyclic graphs (i.e. #freq. cyclic graphs = 0).

Concerning the results on both databases, the complete encoding is usually less performant than the incomplete one. This is explained by the fact that the complete encoding handles larger set of candidates than the incomplete one. For the incomplete encoding, the number of frequent subgraphs discovered by GGM is not too far from that of Gaston. We notice also that from the frequency of 170 graphs, the result is the same.

## 5 Conclusion

In this paper, we have presented algorithm GGM for the frequent subgraph discovery problem in a graph datasets. We pointed out the key points in the graph mining process. We combined several strategies inspired from existing algorithms to implement *GGM*. The two important points in the process of discovery are the generation of new candidates and the frequency counting. Our experimentations show the effectiveness of the incomplete approach compared to the complete one. It shows the importance of handling a reasonable amount of candidates. The main perspective is to improve our incomplete encoding. We have to experiment our incomplete approach on huge graph databeses such as those coming from chemistery. Actually, we have implemented the whole mining algorithm in GGM and confront its performance to Gaston.

## References

1. Bettina Berendt, Andreas Hotho, and Stum Gerd. Towards semantic web mining. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 264–278, London, UK, UK, 2002. Springer-Verlag.
2. Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowl. and Data Eng.*, 17:1036–1050, August 2005.
3. Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 13–23, London, UK, 2000. Springer-Verlag.
4. Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society.
5. Siegfried Nijssen and Joost N. Kok. The gaston tool for frequent subgraph mining. *Electron. Notes Theor. Comput. Sci.*, 127:77–87, March 2005.
6. Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 02:721–724, 2002.
7. Ken'ichi Yoshida and Hiroshi Motoda. Clip: concept learning from inference patterns. *Artif. Intell.*, 75:63–92, May 1995.