Lower and upper queries for graph-mining*

Amina Kemmar [1], Yahia Lebbah [1,3], Samir Loudni [2], and Mohammed Ouali [1]

[1] Laboratoire LITIO, Université d'Oran
BP 1524, El-M'Naouer, 31000 Oran, Algérie
[2] Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
[3] Laboratoire I3S/CNRS, Université de Nice - Sophia Antipolis, Nice, France
{kemmami,ylebbah}@yahoo.fr,samir.loudni@unicaen.fr,mohammed.ouali@univ-oran.dz

Abstract. Canonical encoding is one of the key operations required by subgraph mining algorithms for candidates generation. They enable to query the exact number of frequent subgraphs. Existing approaches make use of canonical encodings with an exponential time complexity. As a consequence, mining all frequent patterns for large graphs is computationally expensive. In this paper, we propose to relax the canonicity property, leading to two encodings, *lower and upper encodings*, with a polynomial time complexity, allowing to tightly enclose the exact set of frequent subgraphs. These two encodings allow two kinds of queries, lower and upper queries, to get respectively a subset and a superset of frequent patterns.
Lower and upper encodings have been integrated in `Gaston`. Experiments performed on large and dense synthetic graphs show that, these two encodings are very effective compared to `Gaston` and `gSpan`, while on large real world sparse graphs they remain very competitive.

1 Introduction

Frequent subgraph pattern mining is one of the most well-studied problems in the graph mining domain. It concerns the discovery of subgraph patterns that occur frequently in a collection of graphs or in a single large graph. This problem arises in many data mining tasks that include: chemoinformatics [13] and computational biology [6], to name but a few. In [10], an approach based on closed graphs is proposed, to predict biological activity of chemical compounds. Closed sets of graphs allow one to adapt standard approaches from applied lattice theory and Formal Concept Analysis to graph mining. An other approach is proposed in [2], where graphs are represented by (multi)sets of standard subgraphs representing substructures which are biologically meaningful, given by domain expert. In this paper, we consider a *database of graphs*, where each graph represents a transaction.

* This work is supported by TASSILI research program 11MDU839 (France, Algeria). This work is also supported by PNR research project 70/TIC/2011 (Algeria).

There are two general approaches used to solve the frequent subgraph pattern mining problem. The first approach, represented by [8,9], extends the Apriori-based candidate generation approach [1] to graph pattern mining. The second approach, represented by [3,7,11,15], adopt a pattern-growth principle [5] by growing patterns from a single graph directly. All these algorithms are complete: they are guaranteed to discover all frequent subgraphs.

Due to the completeness requirements, most of these algorithms need to perform graph isomorphism operations in order to check whether two subgraphs are identical or not in the candidate generation. This problem is equivalent to compare *canonical encodings* [4] of graphs. Even though these two problems are not known to be either in P or in NP-complete, in practice however, most existing canonical encodings have complexities which are of exponential nature. As a consequence, many existing complete algorithms impose strong limitations on the types of graph datasets that can be mined in a reasonable amount of time, as those derived from chemical compounds [3,7,8,13,15]: graphs that are sparse, contain small frequent subgraphs, and have very few vertex and edge labels. For large and dense graphs that contain frequent cyclic subgraphs, mining the complete set of frequent patterns is computationally expensive, since numerous subgraph isomorphisms are performed, making these algorithms not effective.

To overcome these limitations, we propose in this paper a *relaxation of the canonicity property*, leading to two encodings, *lower and upper encodings*, with a polynomial time complexity. The two encodings have been integrated in `Gaston`, leading to two approximate algorithms, called `Gaston-Low` and `Gaston-Up`. Because of its incomplete nature, the number of patterns discovered by `Gaston-Low` (resp. `Gaston-Up`) is a lower (resp. upper) bound of the exact number of frequent graphs. Thus, these two encodings allow one to enclose the number of frequent graphs generated by a canonical encoding. The lower and upper encodings allow two kinds of queries: lower and upper queries. The lower query uses the lower encoding and enables to get a subset of frequent patterns. And the upper query uses the upper encoding and enables to get a superset of frequent patterns.

Experiments performed on synthetic and real world datasets show that, on large and dense graphs, `Gaston-Low` and `Gaston-Up` are very effective compared to `Gaston` and `gSpan`, while on large real-life sparse graphs it remains very competitive.

This paper is organized as follows. Section 2 introduces preliminaries on graph mining. Section 3 reviews some canonical encodings. Section 4 motivates our proposals. Section 5 explains our two encodings. Section 6 is devoted to experimentations. Finally, we conclude and draw some perspectives.

2 Preliminaries

In this section we briefly review some basic concepts and fix the notations.

A *labelled graph* can be represented by a 4-tuple, $G = (V, E, L, l)$, where V is a finite set of vertices, $E \subset V \times V$ is a set of edges, L is a set of labels, and $l : V \cup E \rightarrow L$ is a function assigning a label to every element of $V \cup E$.

A *dense graph* is a graph in which the number of edges is close to the maximal number of edges. The graph density D is defined as: $D = 2|E|/(|V|(|V|-1))$. $(|V|(|V|-1)/2)$ is the number of edges in a complete graph. Clearly, the given formula of D computes the proximity of the number of edges to the maximum number of edges. A graph is *cyclic* if it has many cycles. As the number of cycles increases, the number of edges increases, and then the density of the graph increases too.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs.

- G_1 and G_2 are *isomorphic* if there is a bijection function $f : V_1 \to V_2$ satisfying: (i) $\forall\, u \in V_1, l_{G_1}(u) = l_{G_2}(f(u))$, (ii) $\forall\, \{u, v\} \in E(G_1), \{f(u), f(v)\} \in E(G_2)$, and (iii) $\forall\, \{u, v\} \in E(G_1)\ l_{G_1}(\{u, v\}) = l_{G_2}(\{f(u), f(v)\})$.
- G_2 is a *subgraph* of G_1, iff $V_2 \subset V_1$, and $E_2 \subset E_1 \wedge (\forall \{v_1, v_2\} \in E_2 \Rightarrow v1 \in V_2$ and $v_2 \in V_2)$.
- G_1 is *subgraph isomorphic* to G_2, denoted $G_1 \subseteq G_2$, if G_1 is isomorphic to a subgraph of G_2. Deciding whether a graph is subgraph isomorphic to another graph is NP-complete [14].

An *encoding function* is a function that assigns to a given graph a *code* (i.e. a sequence of bits, a string, or a sequence of numbers).

Definition 1 (Canonical encoding). *The encoding function ϕ is canonical when for any two graphs G_1, G_2: G_1 and G_2 are isomorphic iff $\phi(G_1) = \phi(G_2)$.*

Given a transaction database $\mathcal{D}$ which contains a family of graphs. The *frequency* of a graph G in $\mathcal{D}$ is defined by $freq(G, \mathcal{D}) = \#\{G_{\mathcal{D}} \in \mathcal{D} \,|\, G \subseteq G_{\mathcal{D}}\}$. The *support* of a graph is defined by

$$support(G, \mathcal{D}) = \frac{freq(G, \mathcal{D})}{|\mathcal{D}|}$$

Let s_{min} be some predefined minimum support. The *Frequent Subgraph Discovery Problem* $FSDP$ consists in finding connected undirected graphs G' that are subgraphs of at least $(s_{min} \times |\mathcal{D}|)$ graphs of $\mathcal{D}$:

$$FSDP(\mathcal{D}, s_{min}) = \{G' | support(G', \mathcal{D}) \geq s_{min}\}.$$

The query $FSDPquery(\mathcal{D}, s_{min}, \phi)$ consists in finding graphs $FSDP(\mathcal{D}, s_{min})$ by using some encoding function ϕ.

Generally, we can distinguish between the algorithms for computing frequent subgraphs according to the way the three following problems are handled:

1. **Candidates generation problem:** The candidates are initialized with frequent edges (1-candidates). The k-candidates (i.e., having k edges) are generated, by adding one edge to each $(k-1)$-candidate. This process can be done with a breadth-first strategy as well as a depth-first strategy.
2. **Subgraph encoding problem:** A canonical encoding is assigned to each generated graph. Verifying that the candidate is new, consists in checking that its encoding does not belong to the encodings of the generated candidates. This paper contributes in this step (see section 5).

3. **Frequency computation problem:** Once a new candidate is generated, we have to compute its frequency. It can be achieved by finding all the transactions of the database that contain this new candidate.

3 Canonical encoding

Developing algorithms that can efficiently compute the canonical encoding is critical to ensure the scalability to very large and dense graph datasets. It is not proven if the canonical encoding of graphs is in the class of NP-complete problems, nor in polynomial class [4].

The encoding function of the FSG algorithm [9] is based on exploring the adjacency matrices of the considered graph. To get the canonical encoding, gSpan algorithm [15] performs various DFS searches on the graph. In [11], the authors have proposed an efficient miner algorithm in which they used an appropriate canonical encoding for the three graph structures: paths, trees and cycles. Paths and trees are encoded efficiently in polynomial time. But for cyclic graphs, the encoding is of exponential nature. As pointed out in [12], the more there are cycles in the graph, the more its encoding is expensive. In fact, for cyclic graphs, the complexity of the encoding in Gaston is $O(|C|(2|C|)!|V||E|^{|C|})$, where E represents the number of edges in the cyclic graph, $|C]$ is the minimal number of edges to remove to obtain a spanning tree. Clearly, $|C|$ is strongly related to the number of cycles in the graph.

So, to overcome this limitation, we propose in this paper two efficient encodings for cyclic graphs: *lower-encoding* and *upper-encoding* (see section 5). Lower-encoding allows one to generate a significant subset of the frequent cyclic graphs, whereas the upper-encoding allows one to generate all frequent graphs with duplicates. Roughly speaking, these two encodings define an interval enclosing the exact frequent graphs. The following section shows the effectiveness of our approach for dense graph datasets.

4 Motivating example

Let us consider the graph database shown in Figure 1. When executing the state of the art `Gaston` miner [11] to extract all subgraphs with a support $s_{min} = 50\%$, it takes 10 minutes, and the number of frequent cyclic graphs found is $1,334,095$, which is very considerable compared to the size of the database. This is essentially due to the canonical encoding which is expensive for these graphs. Our main idea is to use a non-canonical encodings, called *lower and upper encodings*, to enclose the complete frequent subgraphs, whilst ensuring reasonable computing times. Applying our lower and upper encodings on the example of Figure 1 gives rise to the following results: (i) the lower query enables to generate $1,309,066$ cyclic graphs in 1.1 minutes. Moreover, the percentage of missed frequent cyclic subgraphs is about 1.87%, which remains reasonably small compared to the saved CPU time, (ii) the upper query allows one to generate $1,468,364$ subgraphs in 2.43 minutes.

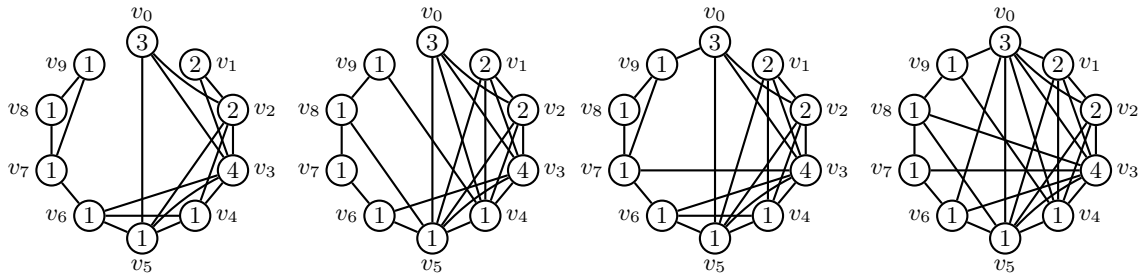


Fig. 1. An example of a dense graph transaction database

5 Lower and upper queries

Due to the incomplete nature of the lower encoding, the number of discovered frequent graphs is a lower bound of the exact number of frequent graphs, and the converse for the upper encoding. These two encodings allow two kinds of queries on the transaction database: the lower query and the upper query.

5.1 A lower-bounding graph encoding and the lower query

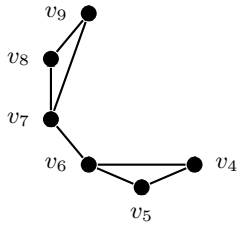
Definition 2 (Lower Encoding).
Let ϕ_L be an encoding function. ϕ_L is lower if the following property holds: Given two graphs G_1 and G_2, if G_1 and G_2 are isomorphic then $\phi_L(G_1) = \phi_L(G_2)$.

Definition 2 establishes the main property of a lower encoding. Based on this definition, we propose an instance of a lower encoding, called `Node-Seq`, that fully makes use of different invariants (i.e., degrees and labels). It is built by concatenating codes of all vertices of the graph, $\texttt{vcode}(v_i)$, $i = 1..|V|$, according to a lexicographic order.

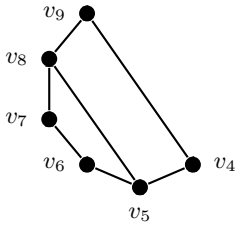
Definition 3 (Node-Seq). *The node sequence code of $G = (V, E)$ is obtained by concatenating codes of all its vertices:* $\texttt{Node-Seq}(G) = (\ \texttt{vcode}(v_1), \ldots, \texttt{vcode}(v_{|V|}))$, *where* $\texttt{vcode}(v_i) <_l \texttt{vcode}(v_{i+1})$, *and the relation $<_l$ defines a lexicographic order among vertices. The vertex code is defined by* $\texttt{vcode}(v_i) = (deg(v_i), lab(v_i), \langle lab(e_{i1}), deg(v_{i1}), lab(v_{i1})\rangle, \ldots, \langle lab(e_{im}), deg(v_{im}), lab(v_{im})\rangle)$, *where:*

- *$deg(v_i)$ is the degree of v_i,*
- *$\langle lab(e_{ij}), deg(v_{ij}), lab(v_{ij})\rangle$ is defined as follows: $lab(e_{ij})$ is the label of the j^{the} edge incident to v_i, and m is the number of its incident edges; $deg(v_{ij})$ (resp. $lab(v_{ij})$) is the degree (resp. label) of the vertex associated to the endpoint of the edge e_{ij} incident to v_i.*

Let us note that edges e_{ij} incident to v_i are ordered according to their labels. If these labels are equal, we consider the degrees and the labels of their endpoint vertices.



(SG_1)



(SG_2)

$v_i \in SG_1$	$v_i \in SG_2$	vcode(v_i)
v_4	v_4	$(2,1,1,2,1,1,3,1)$
v_5	v_6	$(2,1,1,2,1,1,3,1)$
v_6	v_5	$(3,1,2,1,1,2,1,1,3,1)$
v_7	v_8	$(3,1,2,1,1,2,1,1,3,1)$
v_8	v_7	$(2,1,1,2,1,1,3,1)$
v_9	v_9	$(2,1,1,2,1,1,3,1)$

$$\texttt{Node-Seq}(SG_1) = \texttt{Node-Seq}(SG_2) = (2,1,1,2,1,1,3,1)(2,1,1,2,1,1,3,1)(2,1,1,2,1,1,3,1)$$
$$(2,1,1,2,1,1,3,1)(3,1,2,1,1,2,1,1,3,1)(3,1,2,1,1,2,1,1,3,1)$$

Fig. 2. Example of two non-isomorphic graphs having the same encoding Node-Seq

Proposition 1. Node-Seq *is a lower encoding and it computes a lower bound of the exact number of frequent graphs. It can be achieved in $O(n\ m\ log(m) + n\ log(n))$ in the worst case, where $n = |V|$ and $m = |E|$.*

Sketch of the proof: Since Node-seq encoding uses only labels, degrees, and a lexicographic sorting, it is necessarily a lower-encoding. Its non-canonicity is stated by the counter-example given in Figure 2. In fact, (SG_1) and (SG_2) are non-isomorphic, but have the same Node-seq encoding. Traversing the vertices is done in $O(n)$. The incident edges of each vertex are enumerated and sorted at most $O(m\ log(m))$ times. The whole n encodings are sorted. Thus, the time complexity is $O(n\ m\ log(m) + n\ log(n))$.□

Let be ϕ_L some lower encoding function, and ϕ_C a canonical encoding. The lower query $FSDPquery(\mathcal{D}, s_{min}, \phi_L)$ computes a subset of frequent subgraphs:

$$FSDPquery(\mathcal{D}, s_{min}, \phi_L) \subseteq FSDPquery(\mathcal{D}, s_{min}, \phi_C).$$

5.2 An upper-bounding graph encoding and the upper query

Definition 4 (Upper Encoding). *Let ϕ_U be an encoding function. ϕ_U is upper if the following property holds: Given two graphs G_1 and G_2, if G_1 and G_2 are not isomorphic then $\phi_U(G_1) \neq \phi_U(G_2)$.*

We propose an upper encoding function which is inspired from the lower encoding, in three steps:

1. Let n be the number of vertices of the considered graph $G(V, E)$ to encode. We associate to each vertex vi a code $lst(v_i)$ containing respectively, its

degree, its label, and a sorted list of degrees and labels of its incident vertices. Then, the n vertices are sorted according to their codes $lst(v_i), i = 1..n$. Following the given order, the first vertex receives identifier 1, the second vertex identifier 2, and so on until the last vertex which receives the identifier n.

2. Each vertex v_i is represented with the following code $\texttt{vcode}(v_i) = (Id(v_i), lab(v_i), \langle lab(e_{i1}), Id(v_{i1})\rangle, \ldots, \langle lab(e_{im}), Id(v_{im})\rangle)$, where:
   - $Id(v_i)$ is the identifier of v_i, generated in the previous step,
   - $lab(e_{ij})$ is the label of the j^{the} edge incident to v_i, and m is the number of its incident edges,
   - The incident edges $e_{ij}, i = 1..m$ are ordered according to their labels and degrees.
3. Finally, the upper encoding, called `ID-Seq` , is built by concatenating codes of all vertices of the graph, according to the order found in the first step.

Proposition 2. `ID-Seq` *is an upper encoding and it computes an upper bound of the exact number of frequent graphs. It can be achieved in* $O(n\ log(n) + 2n\ m\ log(m))$ *in the worst case, where* $n = |V|$ *and* $m = |E|$.

Sketch of the proof: This upper encoding uses the identifiers of the vertices, which allows one to encode the topological structure of the graph. That is why we can not find two non isomorphic graphs with the same `ID-Seq`. The proof of the complexity is the same as the one given for `Node-Seq` encoding except that we should take additionally into account the first step. For each vertex v_i, the sorting operation to generate $lst(v_i)$ requires $O(m\ log(m))$. The complexity of the first step is $n\ m\ log(m)$, which is the additional processing time compared to the complexity of the lower encoding `Node-Seq`. □

Since assigning identifiers to vertices may be done in different ways, two isomorphic graphs can be explored multiple times giving rise to different `ID-Seq` encodings. Thus, the way the vertices are ordered and then numbered, is essential to avoid duplicates. Our vertices ordering seems a good compromise in our experimentations. But, if we want a canonical encoding, we should explore all of the orderings, which is clearly of exponential nature.

Let be ϕ_U some upper encoding function, and ϕ_C a canonical encoding. As stated for the lower encoding functions, the upper query $FSDPquery(\mathcal{D}, s_{min}, \phi_U)$ computes a superset of frequent subgraphs:

$$FSDPquery(\mathcal{D}, s_{min}, \phi_C) \subseteq FSDPquery(\mathcal{D}, s_{min}, \phi_U).$$

6 Experimental evaluation

In this section, we study the performance of the proposed encodings. We used both real and synthetic datasets. For comparison, we considered two algorithms, `gSpan` and `Gaston`.

We have integrated our lower and upper encodings in the `Gaston` algorithm to encode cyclic graphs through `Gaston-Low` [1] and `Gaston-Up` respectively. `Gaston-Low` enables to compute a lower query with `Node-Seq` encoding, whereas `Gaston-Up` an upper query with `ID-Seq` encoding. First, we generated a series of synthetic graph datasets with different settings of parameters of our graph generator *DenseGraphGen*. We study the influence of different parameters (density, frequency and the number of edge and node labels) on the performances of our encodings on synthetic datasets and we compare our results with those obtained by `Gaston` and `gSpan` (Section 6.1). Second, we compare the results of the four algorithms on real datasets (Section 6.2).

For each experiment, we report the CPU time (in seconds), the number of frequent cyclic graphs and their densities. For all experiments, we impose a time limit of $3,600$ seconds. When an algorithm cannot complete the extraction within the time limit, it will be indicated by the symbol $(-)$ in the table. All experiments were conducted on 2.50GHz Intel core i5-321M machines with 8GB main memory, running the Linux operating system.

6.1 Performances evaluation on synthetic datasets

The synthetic datasets are generated using our graph generator `DenseGraphGen`. The datasets are obtained as follows: First, we generate a dense graph G_D to guarantee that potentially frequent subgraphs are enough dense. Second, we generate $|N|$ transactions. Each transaction uses G_D as a kernel. Third, to make different all the transactions of the dataset, for each transaction, we add $|V|$ nodes and $|E|$ edges connecting randomly nodes of V with those of G_D. Then, $|EI|$ edges are added randomly between nodes of G_d. Finally, $|L_v|$ and $|L_e|$ labels are chosen randomly.

GD. Name	cyclic0	cyclic1	cyclic2	cyclic3	cyclic4	cyclic5	cyclic6	cyclic7	cyclic8
Avg. #edges	38	50	62	73	85	96	107	118	134
Avg. density	0.2	0.26	0.32	0.38	0.44	0.50	0.56	0.62	0.70

Table 1. Characteristics of the graph datasets used for our experiments: $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, and $|Lv| = |Le| = 10$.

We generated a series of graph datasets using different graph kernels (with different densities: $|E|$ is varying). Such a choice enables us to obtain increasing cyclic graphs in each dataset. Their main characteristics are given in Table 1.

6.1.1 Influence of the density Table 2 compares the performances of the four algorithms on the datasets illustrated in Table 1, with $s_{min} = 100\%$. In all of these experiments, $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, $|Lv| = |Le| = 10$.

From table 2, we can draw the following remarks. First, for small values of the density, `Gaston-Low`, `Gaston-Up` and `Gaston` perform quite similarly both in terms of runtime and number of frequent cyclic subgraphs discovered. Second,

[1] The implementation of Gaston-LB and Gaston-UB can be downloaded from `https://sites.google.com/site/akemmar/LUmining`

	runtime (s)				# cyclic Gr.		
GD. Name	Gaston-Low	Gaston	Gaston-Up	gSpan	Gaston-Low	Gaston	Gaston-Up
cyclic0	0.24	**0.2**	0.3	22.03	**1,496**	**1,496**	**1,496**
cyclic1	1.19	**0.88**	1.6	117.67	**10,729**	**10,729**	**10,729**
cyclic2	5.73	**4.89**	8.03	–	**59,996**	**59,997**	**59,997**
cyclic3	**13.03**	**13.02**	18.04	–	**132,915**	**132,915**	**132,915**
cyclic4	**58.38**	115.93	80.35	–	**619,081**	**619,081**	620,446
cyclic5	**122.39**	387.97	**168.19**	–	1,221,435	1,223,554	1,223,862
cyclic6	**303.46**	1,516.99	**409.4**	–	2,999,054	2,999,054	2,999,054
cyclic7	**744.19**	–	**1,045.94**	–	6,928,308	–	6,928,308
cyclic8	–	–	–	–	–	–	–

Table 2. Characteristics of the graph datasets used for our experiments: $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, and $|Lv| = |Le| = 10$. Performances of `Gaston`, `gSpan`, `Gaston-Low` and `Gaston-Up` on different synthetic datasets of increasing density ($s_{min} = 100\%$).

`gSpan` is not competitive, even for very low densities. Third, for high values of the density (≥ 0.44), the difference in performance between `Gaston-Low`, `Gaston-Up` and `Gaston` widens considerably: `Gaston-Low` (resp. `Gaston-Up`) is up to 5 (resp. 4) times faster than `Gaston`. Moreover, the percentage of frequent cyclic subgraphs omitted by `Gaston-Low` is very negligible. For all datasets, `Gaston-Low` finds at least 99.82% of all frequent cyclic subgraphs. For the *cyclic*7 dataset, `Gaston` is not able to complete the extraction of all frequent subgraphs in one hour. Indeed, with the increase of density value, the search space of graph isomorphism for the candidate generation and the running time increases drastically and `Gaston` spends more time to encode these cyclic graphs. This is not the case for `Gaston-Low`, which requires less time thanks to its polynomial time encoding. For `Gaston-Up`, the number of frequent cyclic graphs is the same for all datasets, except for *cyclic*4 and *cyclic*5 for which the percentage of duplicates is at most 0.2%. From these results, the two encodings succeeded to enclose tightly the exact number of subgraphs within a very competitive runtime.

6.1.2 Influence of the support threshold Table 3 shows the results obtained by the three algorithms on the dataset *cyclic*5 with values of s_{min} ranging from 100% to 40%. The results of `gSpan` are not reported since it fails to complete the extraction of all frequent subgraphs in the time limit. As we can see, as s_{min} decreases, the running time of `Gaston` increases drastically and becomes prohibitively expensive below a certain threshold, while `Gaston-Low` remains effective: it outperforms `Gaston` by a factor from 2 to 3. Moreover, `Gaston-Low` finds a significant number of frequent cyclic subgraphs with a percentage of missed subgraphs of at most 0.17%. For `Gaston-Up`, the upper encoding generates more graphs than `Gaston`, particularly for small values of s_{min}. However, `Gaston-Up` takes less time than `Gaston` thanks to its polynomial time encoding.

6.1.3 Influence of the number of edge and node labels For these series of experiments, we considered the same number of labels for nodes and edges, given by the parameter $|L|$. Table 4 shows the results obtained by the three

	runtime (s)			# cyclic Gr.		
$s_{min}(\%)$	Gaston-Low	Gaston	Gaston-Up	Gaston-Low	Gaston	Gaston-Up
100	**186.05**	476.94	**210.78**	1,221,435	1,223,554	1,223,862
80	**163.44**	464.55	**190.43**	1,225,059	1,225,678	1,225,830
70	**248.5s**	584.49	**257.13**	**1,484,175**	**1,484,175**	1,805,651
65	**219.72s**	494.67	**252.62**	**1,228,323**	**1,228,323**	1,517,283
60	**482.45s**	842.23	**554.09**	**1,819,606**	**1,819,606**	3,132,861
50	**1,743.58**	2,156.34	**1,787.1**	**4,104,185**	**4,104,185**	**6,671,993**
40	–	–	–	–	–	–

Table 3. Comparing the performances of Gaston, Gaston-Low and Gaston-Up on the *cyclic*5 dataset for different minimum support threshold s_{min}.

algorithms on the dataset *cyclic*6 with values of $|L|$ ranging from 1 to max. $|L| = max$ means that all node and edge labels are different.

Algorithm	runtime (s)			# cyclic Gr.		
GD. Name	Gaston-Low	Gaston	Gaston-Up	Gaston-Low	Gaston	Gaston-Up
cyclic6-Lmax	**299.9**	1,679.33	**496.6**	3,200,963	3,200,963	3,200,963
cyclic6-L20	**279.09**	1,489.16	**432.33**	3,200,963	3,200,963	3,200,963
cyclic6-L15	**291.87**	1,488.12	**518.57**	3,200,963	3,200,963	3,200,963
cyclic6-L10	**287.32**	1,494.4	**503.14**	3,200,963	3,200,963	3,200,963
cyclic6-L5	**650.27**	–	**641.33**	3,157,417	–	3,171,563
cyclic6-L3	**1,704.35**	2,939.96	1,933.91	3,057,301	3,057,301	3,192,363
cyclic6-L2	–	–	–	–	–	–
cyclic6-L1	–	–	–	–	–	–

Table 4. Comparing the performances of Gaston, Gaston-Low and Gaston-Up on the *cyclic*6 dataset for different values of labels. $S_{min} = 100\%$, $|N| = 1,000$, $|V| = 10$, $|E| = 5$ and $|EI| = 5$.

From the results of Table 4, we can observe three interesting points. First, as $|L|$ increases, the overall running time of the three algorithms decreases. Indeed, the higher this value, the less the number of isomorphisms are performed, leading to fast candidate generation. Second, compared with Gaston, Gaston-Low performs faster. For $|L| \geq 10$, Gaston-Low finds the exact number of frequent cyclic subgraphs and it is up to 5 orders of magnitude faster than Gaston. For $|L| = 5$, Gaston cannot complete the extraction of all frequent subgraphs in the time limit (i.e. one hour), while Gaston-Low finds a great number of frequent cyclic subgraphs in a reasonable amount of time. Third, again, Gaston-Up clearly outperforms Gaston. Moreover, it finds the exact number of frequent cyclic subgraphs, except for cyclic6-L3 where the percentage of duplicates is at most 4%.

6.2 Real world datasets

To study the behaviour of our encoding on datasets which are not dense and contain a few number of frequent cyclic graphs, we performed experiments on real world datasets coming from the biological domain encoding molecules. Each molecule corresponds to a graph where atoms are represented using nodes and bonds between them are represented by edges. They are obtained from different

sites: (a) The National Cancer Institute (NCI); (b) Developmental Therapeutics Program (DTP); (c) The Prediction of Ames Mutagenicity(PAM) [2].

Algorithm						runtime (s)				# cyclic Gr.		
GD. Name	$\|N\|$	$\|V\|$	$\|Lv\|$	$\|Le\|$	D	Gaston-Low	Gaston	Gaston-Up	gSpan	Gaston-Low	Gaston	Gaston-Up
CAN2DA99	5,210	25	82	3	0.06	**1.38**	**1.34**	**1.50**	25.41	**51** (13/2)	**52** (13/2)	139
Open-2012-05-01	189	28	20	3	0.09	1.63	**0.41**	5.12	–	8,098 (23/2)	10,532 (23/2)	59,333
new-oct99-aug00-3D	1,170	50	23	3	0.08	**2.33**	**2.26**	3.03	104.15	632 (15/7)	721 (15/9)	2,195
2DA99 2012-05-01	108,954	19	131	3	0.06	9.21	**8.86**	11.15	146.70	**17** (10/1)	**17** (10/1)	49
aug00-2D	188,937	38	186	3	0.06	**208.57**	**207.66**	**246.58**	–	239 (12/21)	271 (12/21)	909
Chemical-340	340	26	161	3	0.13	**0.02**	**0.02**	**0.04**	0.62	**64** (11/10)	**65** (11/10)	210
Compound-422	422	39	21	4	0.10	**0.44**	**0.22**	**0.92**	7.68	**2,529** (20/1)	**2,533** (20/1)	3,794
mechset3d	1,536	46	42	4	0.03	**2.77**	**2.77**	**3.5**	100.00	513 (13/10)	573 (13/11)	1,788
cans03sd	4,2247	26	161	3	0.13	**9.57**	**9.43**	**10.72**	131.08	**39** (13/1)	**39** (13/1)	107
divii	29	22	8	2	0.09	75.98	**16.26**	261.68	–	411,922 (28/2)	429,009 (28/2)	1,198,933
N6512	6,512	17	28	4	0.32	**1.65**	**1.62**	**2.42**	34.40	**66** (14/7)	**72** (14/8)	220

Table 5. Comparing the performances of `Gaston-Lower`, `Gaston-Upper`, `Gaston` and `gSpan` on real world datasets for $s_{min} = 10\%$. $|N|$: number of graphs, $|V|$: the average number of vertices and D: the average density of the dataset.

Table 5 shows the runtime and the number of frequent patterns found by the four algorithms for different datasets. We also report for each algorithm, the size of the largest frequent patterns obtained (number of its edges) and the number of its occurrence (in parenthesis). Comparing the relative performance of `Gaston-Low` and `Gaston` on the NCI datasets, we can see that overall, they perform quite similarly in terms of runtime even on large datasets (i.e. `NCI-2DA99` and `NCI-aug00-2D`). Moreover, the percentage of patterns missed by `Gaston-Low` remains reasonably low, except for `NCI-Open` dataset where this percentage is about 23%. Even though `Gaston-Low` is incomplete, the size of the greatest subgraph is practically the same than the one found by `Gaston`. Despite the great number of duplicates generated, `Gaston-Up` remains comparable with `Gaston-Low` and `Gaston` in terms of runtime. Finally, compared to `gSpan`, the three other algorithms achieve better performance, with a factor from 16 to 45. The same conclusions can be drawn for DTP and PAM datasets.

7 Conclusion

In this paper, we have proposed a relaxation of the canonicity property, leading to lower and upper encodings, giving rise to lower and upper queries, for the

[2] (a) `http://cactus.nci.nih.gov/download/nci/`; (b) `http://dtp.nci.nih.gov/`; (c) `http://doc.ml.tu-berlin.de/toxbenchmark/index.html#v2`

frequent subgraph discovery problem. Our encodings can be achieved in a polynomial time complexity. The two encodings have been integrated in the state of the art `Gaston` miner. Experiments we performed on synthetic databases as well as on a real world dataset from the biological domain show that our lower encoding is very effective on dense graphs: the lower query is able to extract a larger number of frequent cyclic subgraphs in a reasonable amount of time, while `Gaston` needs much more time to extract the complete set of frequent subgraphs. On the other hand, the upper query allows one to have a tight and valid approximation of the missed graphs using the upper-encoding. As future works, we intend to improve our lower and upper encodings in order to enhance their performances on non-dense graphs. We should also investigate more deeply real world datasets, in order to show the effectiveness of our approach on dense graphs.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB'94*, pages 487–499. Morgan Kaufmann, 1994.
2. V. G. Blinova, D. A. Dobrynin, V. K. Finn, Sergei O. Kuznetsov, and E. S. Pankratova. Toxicology analysis by means of the jsm-method. *Bioinformatics*, 19(10):1201–1207, 2003.
3. C. Borgelt and M. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of ICDM '02*, pages 51–58, Washington, DC, USA, 2002. IEEE Computer Society.
4. Scott Fortin. The graph isomorphism problem. Technical report, TR-96-20, Departement of computer science, University of Alberta, Canada, 1996.
5. J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explor. Newsl.*, 2(2):14–20, December 2000.
6. J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *J. of Computational Biology*, 12(6):657–671, 2005.
7. J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of I ICDM '03*, pages 549–552, Washington, DC, USA, 2003. IEEE Computer Society.
8. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of ECML-PKDD'00*, pages 13–23, London, UK, 2000. Springer-Verlag.
9. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of ICDM'01*, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society.
10. Sergei O. Kuznetsov and Mikhail V. Samokhin. Learning closed sets of labeled graphs for chemical applications. In *Proceedings of the 15th international conference on Inductive Logic Programming*, ILP'05, pages 190–208, Berlin, Heidelberg, 2005. Springer-Verlag.
11. S. Nijssen and J. Kok. The gaston tool for frequent subgraph mining. *Electron. Notes Theor. Comput. Sci.*, 127:77–87, March 2005.
12. Siegfried Nijssen. *Mining Structured Data*. PhD thesis, Leiden University, 2006.
13. G. Poezevara, B. Cuissart, and B. Crémilleux. Extracting and summarizing the frequent emerging graph patterns from a dataset of graphs. *J. Intell. Inf. Syst.*, 37(3):333–353, 2011.

14. Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.
15. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 02:721–724, 2002.