Int. J. Open Problems Compt. Math., Vol. 3, No. 1, March 2010
ISSN 1998-6262; Copyright © ICSRS Publication, 2010
www.i-csrs.org

A Novel Approach for Detection and Elimination of Automorphic Graphs in Graph Databases

R Vijayalakshmi, R Nadarajan, P Nirmala, and M Thilaga
[1]PSG College of Technology, Coimbatore-641004, Tamil Nadu, India
e mail: rv@mca.psgtech.ac.in

Abstract

Graphs have become indispensable in modeling and representing complicated structured data such as proteins, chemical compounds, and XML documents. Development of graph databases for use in research and development is a well-established activity in pharmaceutical and chemical industries. Storing the graphs into large databases is a challenging task as it deals with efficient space and time management. Unlike item sets in huge transactional databases, it becomes essential to ensure the consistency of graph databases since relationships among edges of a graph are predominant. One of the necessary procedures required is a mechanism to check whether two graphs are automorphic. For graphs with more than one vertex with the same label, more than one adjacency matrix representations are possible based on the ordering of vertices with identical labels and there are possibilities that the same graph is stored more than once using different adjacency matrices, leading to adverse results in mining graph databases. Difficulty in identifying and eliminating the automorphic graphs is a challenging problem to the research community. In this paper, a proficient algorithm is devised that efficiently detects and avoids the same graph getting stored into the database. The computational time is also substantially reduced compared to the canonical labeling approach used in Frequent Subgraph Discovery algorithm. The experimental results and comparisons offer a positive response to the newly proposed algorithm.

Keywords: *Graph Database, Graph Mining, Graph Automorphism, Canonical Labeling, Breadth First Search.*

1 Introduction

Many scientific and commercial applications urge for patterns that are more complex and complicated to process than frequent item sets and sequential

patterns. Such sophisticated patterns range from sets and sequences to trees, lattices and graphs. As one of the most general form of data representation, graphs easily represent entities, their attributes and their relationships to other entities. Using a graph for representing the data therefore is one of the most promising approaches to extracting knowledge from relational data. Various conferences over the past few years on mining graphs have motivated researchers to focus on the importance of mining graph data. One of the major perceptions concerned in graph mining is discovering frequent patterns [1], [2], [4], [5]. The key operation required by any frequent subgraph discovery algorithm is a mechanism to check whether two subgraphs are identical or not.

A well-known representation of graph structured data is an adjacency matrix representation. Many graph databases such as chemical graphs have more than one vertex with the same label. These graphs have more than one adjacency matrix representation based on the ordering of same vertex labels, and it becomes difficult to identify them uniquely. There are possibilities that the same graph is stored more than once in the graph database leading to adverse results of mining. Also, if stored more than once in different adjacency matrices, a single graph affects the consistency of the graph database [12], [16]. The research community faces this great challenge while storing graphs in huge databases. To avoid the ambiguity of representation and inefficiency in the graph pattern search, canonical labeling approach that produces a unique code for each graph called canonical code has been used in Frequent Subgraph Discovery algorithm (FSG) [14]. By comparing this code with the unique codes generated for all the graphs in the database, we can identify if the graph is already present in the database. In this paper, the canonical labeling approach used in FSG has been investigated and the time complexity is also analyzed. An innovative Fast-GraphAutomorphicFilter (F-GAF) algorithm has been proposed that uses an edge-based representation of graphs called grid representation to detect automorphic graphs efficiently.

The rest of the paper is organized as follows. Section II presents the formal definitions and notations used for the proposed research work. Section III reviews the related work in this area and the drawbacks of canonical labeling. Section IV introduces the proposed algorithm F-GAF using the novel edge-based graph representation. Section V discusses the empirical performance evaluation of F-GAF using synthetic graph datasets.

2 Definitions and Notations

This section introduces the various definitions and notation used for this work.

Definition 1 Labeled Graph

A labeled graph G is a 4-tuple, G = (V,E,μ,ν) where V is a finite set of vertices, E ⊆ V×V is a set of edges, μ: V→L_V denotes a vertex labeling function and ν:

$E \rightarrow L_E$ denotes a edge labeling function. The following definitions assume a graph database GD and the graphs $G_1=(V_1,E_1,\mu_1,\nu_1)$ and $G_2=(V_2,E_2,\mu_2,\nu_2)$.

Definition 2. Graph Isomorphism

Given a pair of labeled graphs G_1, and G_2, an isomorphism from G_1 to G_2 is a bijection from V_1 to V_2 such that the induced action on E_1 is a bijection onto E_2. For every edge $e_1=(u,v) \in E_1$, there exists an edge $e_2=(f(u),f(v)) \in E_2$ such that $\nu_1(e_1)=\nu_2(e_2)$. For every edge $e_2=(u,v) \in E_2$, there exists an edge $e_1=(f^{-1}(u),f^{-1}(v)) \in E_1$ such that $\nu_1(e_1)=\nu_2(e_2)$.

Definition 3. Automorphism

An automorphism between two graphs G_1, G_2 is an isomorphism mapping where $G_1 = G_2$. That is, it is a graph isomorphism from a graph G to itself. The graph G_2 shown in Figure 1 is automorphic to G_1.

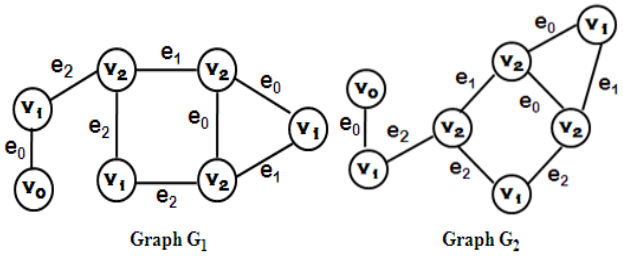


Fig. 1: Graph Automorphism ($G_1 = G_2$)

Definition 4. Canonical Label

The canonical label of a graph G, called cl(G), is defined as a unique code (e.g., string) that is invariant on the ordering of the vertices and edges in the graph [3], [12]. As a result, two graphs will have the same canonical label if they are automorphic.

3 Related Work

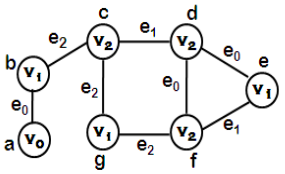
The significance of using graphs to represent complex datasets has been recognized in different disciplines such as chemical domain [5], [8], [17], [18], computer vision [15], image and object retrieval [6], [10], and machine learning [4], [13], [20]. The graph isomorphism problem takes up an important position in the world of complexity analysis. It is one of the few problems that is in NP complete[11].

3.1 Canonical labeling

Canonical labels play a critical role in the frequent subgraph discovery [3], [14]. While in frequent item set mining it is trivial to ensure that the same item set is checked no more than once in the search (using an arbitrary, but fixed global order of the items), in frequent subgraph mining it is one of the core problems to find

how to avoid redundant search. Since the same graph can be grown in several different ways by adding the same nodes and edges in different orders, it is difficult to guarantee that each graph is considered only once. Therefore methods that rule out redundant search are very important to make the algorithms efficient. However, the problem of determining canonical label of a graph is equivalent to determining automorphism between graphs. This is because if two graphs are automorphic with each other, their canonical labels must be identical.

A simple way of defining the canonical label of an undirected graph is to use the string obtained by concatenating the upper triangular elements of the graph's adjacency matrix when this matrix has been symmetrically permuted such that this string is the lexicographically largest (or smallest) among the strings obtained from all such permutations. To obtain a unique code, all the vertex labels of the graph are associated with unique identifiers to recognize the vertices distinctively. A graph G_1 having vertices with vertex labels $v_0,v_1,\ldots$ and unique identifiers a,b,…and one of its adjacency matrices are given in Figure 2. The edges in Graph G_1 have edge labels $e_0,e_1,\ldots$



id / label	a v_0	b v_1	c v_2	d v_2	e v_1	f v_2	g v_1
a	0	e_0	0	0	0	0	0
b	e_0	0	e_2	0	0	0	0
c	0	e_2	0	e_1	0	0	e_2
d	0	0	e_1	0	e_0	e_0	0
e	0	0	0	e_0	0	e_1	0
f	0	0	0	e_0	e_1	0	e_2
g	0	0	e_2	0	0	e_2	0

Fig. 2: Graph G_1 and one of its Adjacency Matrix

The canonical code of Graph G_1 is '$e_0 0 e_2 0 0 e_1 0 0 0 e_0 0 0 0 e_0 e_1 0 0 e_2 0 0 e_2$'

If a graph contains |V| vertices, the worst case time complexity to compute its canonical code is O(|V|!) since |V|! permutations of vertices have to be checked before selecting the minimum(or maximum) code. To narrow down the search space, the vertices are partitioned by their degree and labels using a well-known technique called the *vertex invariants* [16].

The adjacency matrix of the graph G_1 shown in figure 2 is partitioned into three groups based on degree of vertices and vertex labels. One of its combinations (a-beg-cdf) is shown in Figure 3.

id	a	b	e	g	c	d	f
label	v_0	v_1	v_1	v_1	v_2	v_2	v_2
partition	0	1			2		
a	0	e_0	0	0	0	0	0
b	e_0	0	0	0	e_2	0	0
e	0	0	0	0	0	e_0	e_1
g	0	0	0	0	e_2	0	e_2
c	0	e_2	0	e_2	0	e_1	0
d	0	0	e_0	0	e_1	0	e_0
f	0	0	e_1	e_2	0	e_0	0

Fig. 3: Partitioned Adjacency Matrix of Graph G_1

All possible permutations of vertex labels inside the partitions 1 and 2 would generate 3! * 3! = 36 combinations, out of which the minimum (or maximum) code is considered as the canonical code of the graph.

3.2 Weakness of canonical labeling

Assume a graph G is represented as an adjacency matrix with *n vertices* and *p partitions*. If there is more than one vertex with same vertex labels, the vertex labels need to be partitioned based on vertex degrees and different classes of vertex labels within each partition. If all vertex labels of n vertices are distinct, then, p becomes equal to n thus making partitioning impossible.

The two cases of partitioning are (i) each partition has n_i vertices with same vertex labels,(where $|n_i|$ is the number of vertices in the i^{th} partition) ($1 \leq i \leq p$) and (ii) all n vertices have same vertex labels and degrees, hence only one partition (a complete graph falls under this case).

To demonstrate the aforementioned cases, consider a graph G with 19 vertices. For case (i), assume p=5, n_1= 3, n_2= 5, n_3= 4, n_4= 4, and n_5= 3. Then, the number of canonical codes, N is 3! ∗ 5! ∗ 4! ∗ 4! ∗ 3! (=24,88,320). Each canonical code has 209 elements as a string (similar to the one shown for the adjacency matrix in Figure 2). To obtain a minimum (or maximum) canonical code among 24,88,320 codes, a string comparison algorithm is needed which further increases the number of comparisons. For case (ii) p=1 and N=19!, which means 121,645,100,408,832,000 canonical codes are generated and tested. For finding canonical labels for graphs with self-loops, the computation time is still higher [3].

4 Proposed Work

Since the canonical code computation consumes more time if the candidate patterns are regular and relatively large [14], the Frequent Subgraph Discovery algorithm requires more time. To avoid the difficulty in computation for finding

all possible permutations of identical vertices inside p partitions, an efficient algorithm Fast-Graph Automorphic Filter (F-GAF) has been developed that uses an edge-based representation of graphs. Given a graph database GD, the proposed algorithm checks the automorphism of graphs without generating huge number of permutation matrices unlike the canonical labeling.

The notations used in the F-GAF algorithm are listed in Table 1.

Table 1: Notations used in F-GAF algorithm

Notation	**Meaning**
GD	Graph Database
G	Graph
G_k	Input Graph
$N \leftarrow \|V\|$	Number of vertices in G
$E \leftarrow \|E\|$	Number of edges in G
$e \leftarrow (S, I_S, D, I_D, E)$	An Edge Tag
$EA(G) \leftarrow \{(e_1,e_2, \ldots ,e_{\|E\|})\}$	Edge Array of G
N	Number of distinct vertex labels in G
$V \leftarrow \{V_1,V_2,\ldots,V_N\}$	Set of vertex labels in G, $1 \leq \|V_i\| \leq N$
$v \leftarrow \{v_1, v_2,\ldots v_n\}$	Distinct vertex labels, $1 \leq \|v_i\| \leq N$, $1 \leq i \leq n$
D	Degrees of distinct vertex labels
$\beta \leftarrow \{V_1(V_{11},\ldots),V_2(V_{21},\ldots),\ldots,V_N(V_{N1},\ldots)\}$	Set of all vertex labels with their neighbours in G
$v_{Degree}(G) \leftarrow \{v_1(d_{11},d_{12},\ldots,d_{1j}),$ $v_2(d_{21},d_{22},\ldots,d_{2j}),\ldots,v_n(d_{31},d_{32},\ldots,d_{3j})\}, 1 \leq j \leq N$	Collection of vertex degrees of G
$N\beta(G) \leftarrow \{V_1(nd_{11},nd_{12},\ldots),$ $V_2(nd_{21},nd_{22},\ldots) ,\ldots,V_N(nd_{N1},nd_{N2},\ldots)\}$	Set of all vertex labels with the degrees of neighbours in G
$GC(G) \leftarrow \{N,E,EA(G), v_{Degree}(G),N\beta(G)\}$	Grid Code of G

The three phases of F-GAF algorithm is shown in Figure 4. In the preprocessing phase, the input graph G_k is expressed as an edge array. The feature extraction phase constructs a grid code of G_k that contains all the features needed for identifying automorphism. Pattern matching phase compares this grid code with the grid codes of the graphs in the database.

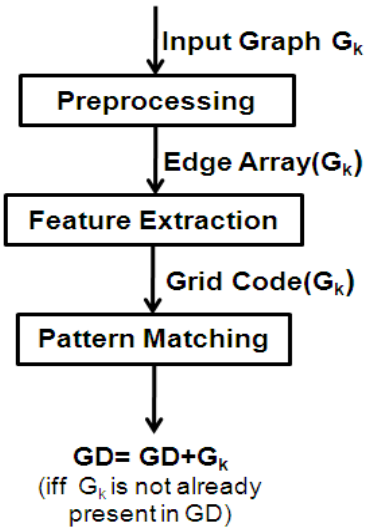


Fig. 4: Phases of F-GAF Algorithm

The phases of the proposed algorithm are described in the following sections.

4.1 Preprocessing

In the pre-processing phase, the input graph G_k is visualized as being placed on a grid of rows and columns. Each vertex lies at the intersection of a row and a column. Figure 5 shows the grid representation of G_k.

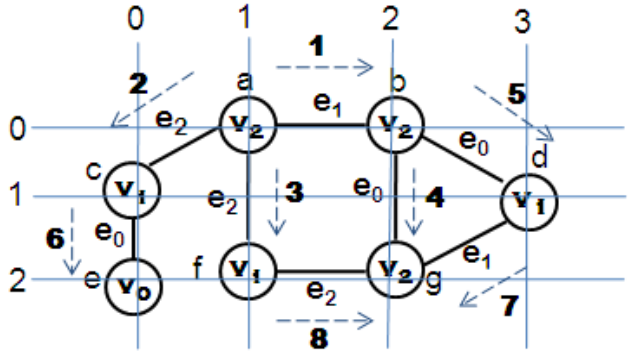


Fig. 5: Grid representation of Graph G_k

Each edge of the graph is a 5-tuple called *edge tag* represented as (S, I_S, D, I_D, E), S - Source Vertex Label, I_S - Identifier of S, D - Destination Vertex Label, I_D - Identifier of D, and E - Edge Label.

Each edge tag is read into *Edge Array (EA)*, which is a collection of information such as number of vertices N, number of edges E and collection of edge tags using Breadth First Traversal, starting from 0^{th} row, 0^{th} column including self loop first (if any), in a left-to-right, top-to-bottom approach so as to include the edges only

once. A unique identifier is assigned to all the vertices. This method of traversing and representing each edge of the graph *exactly once* is known as *Grid Traversal Technique*. Using this technique, the graph shown in Figure 5 is encoded in to a set of edge tags as $\{(v_2,a,v_2,b,e_1)$, (v_1,c,v_2,a,e_2), (v_1,f,v_2,a,e_2), (v_2,b,v_2,g,e_0), (v_1,d,v_2,b,e_0), (v_0,e,v_1,c,e_0), (v_1,d,v_2,g,e_1), and $(v_1,f,v_2,g,e_2)\}$.

For an undirected graph, the edge tags are arranged in the lexicographic order of source, destination and edge labels. Since no edges are repeated, the number of edge tags in the edge array is the same as the number of edges in the graph. The efficiency of this representation has been tested against the traditional adjacency matrix and adjacency list representations on various types of graph data such as complete, sparse and non-sparse graphs by performing Depth First Traversal. This representation itself is a Breadth First Traversal of all edges in lexicographic order. The comparisons prove that the new representation is efficient in terms of time and space complexities. The results of these comparisons are shown in the Table 2, Section V.

4.3 Feature Extraction

The grid code is considered as a feature vector consisting of the edge array, distinct vertex labels and the degrees v_{Degree} and all vertex labels with degrees of each of its neighbours $N\beta$ of k^{th} graph. The grid code of G_k generated in this phase is represented as $GC(G_k)=\{N,E,EA(G_k),v_{Degree}(G_k),N\beta(G_k)\}$. The grid code of a graph is considered as its unique code like the canonical code that uniquely identifies a graph.

4.4 Pattern Matching

In this phase, the grid code of G_k is compared with those of the other graphs in GD to check automorphism.

After computing the grid code of the k^{th} graph, the algorithm compares $GC(G_k)$with each graph $GC(G_i)$, $1\leq i \leq k$. If the grid code of G_k has the same values for N, E, EA, v_{Degree} and $N\beta$ as that of G_i, the algorithm concludes that the graphs are automorphic and terminates without including the grid code of G_k to GD. In this process of comparison in the specified order, if any of these parameters are different, the algorithm immediately concludes that the graphs are different and after adding the grid code of G_k to GD terminates the process.

4.2 Fast-Graph Automorphic Filter(F-GAF) Algorithm for detecting Graph automorphism

In the preprocessing phase, the algorithm takes the edge array of G_k as input constructed using the grid traversal technique. The Fast-Graph Automorphic Filter (F-GAF) algorithm outlines the feature extraction and the pattern matching phases.

Algorithm Fast-GraphAutomorphicFilter (F-GAF)

Input: GD←{GC(G_1),GC(G_2),...,GC(G_{k-1})}, EA(G_k)←input graph

Output: GD←{GC(G_1),GC(G_2),...,GC(G_{k-1}),GC(G_k)}, if GC(G_k) does not exist already in GD.

k←1; GD←ϕ

//Feature extraction

GC(G_k)=GridCodeGen(EA(G_k))

//Pattern Matching

If k=1, then

GD←GD+GC(G_k);

return

Else

For each graph G_i with N_{Gi}= N_{Gk} and E_{Gi}= E_{Gk} and e_{Gi} = e_{Gk} and $v_{iDegree}(G_i) = v_{iDegree}(G_k)$, do

If ($N\beta_i(G_i) = N\beta_k(G_k)$))

Report 'G_i and G_k are same'

Reject G_k

Else

GD←GD+GC(G_k)

Return

Algorithm GridCodeGen(GA(G_k))

Construct $v_{Degree}(G_k)$,β_k, and $N\beta_k$

Grid Code GC(G_k) ←{GA(G_k), $v_{Degree}(G_k)$,$N\beta_k$}

Return GC(G_k)

The key improvement of F-GAF algorithm over canonical labeling is that it significantly narrows down the search space and drastically reduces the time

needed for identifying automorphic graphs. The v_{Degree} and Nβ are compared only if N, E and EA are same. This avoids the time consuming comparisons. On the other hand, the canonical code generation itself consumes more time in canonical labelling,

4.3 Analysis and Illustration of F-GAF Algorithm

The analysis of the computational time of F-GAF algorithm is described as follows.

Using a hash based implementation, the algorithm takes **2E** comparisons to compute $v_{Degrees}$ and β from the edge tags (where E is the number of edges in the input graph). To compute Nβ, the number of comparisons needed is also **2E**. Hence, the total time of GridCodeGen(G_k) is **4E**.

The maximum number of comparisons needed to check whether G_k is automorphic to any of the k-1 graphs would be $\mathbf{k(1+1+(3E)+(2N)^2+(N*(N-1))}$, as the number of vertices and edges, grid arrays(only source, destination vertices, and edge labels are compared), v_{Degree}, Nβ of k graphs are compared. Therefore, at the worst case, the total number of comparisons of F-GAF algorithm would be $\mathbf{T(F\text{-}GAF) = 4E+k(2+3E+(5N)^2\text{-}N)}$

For the comparison of F-GAF with canonical labeling, we used the chemical datasets in GD acquired from http://pubchem.ncbi.nlm.nih.gov/. For illustration, the chemical compound *Diazepam* shown in Figure 6 is used to describe the efficiency of F-GAF algorithm that detects avoids automorphic graphs getting stored into chemical databases over canonical labelling.

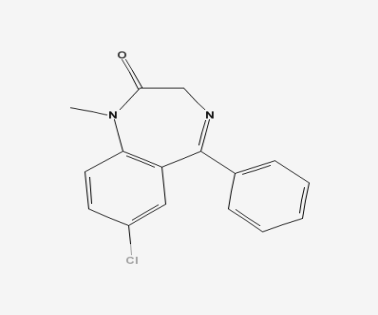


Fig. 6: Chemical compound Diazepam

The grid representation of Diazepam is given in Figure 6. Here, the vertex labels C, O, N, and Cl represent the atoms, edge labels s and d represent single and double bonds and numbers represent unique identifiers of vertices.

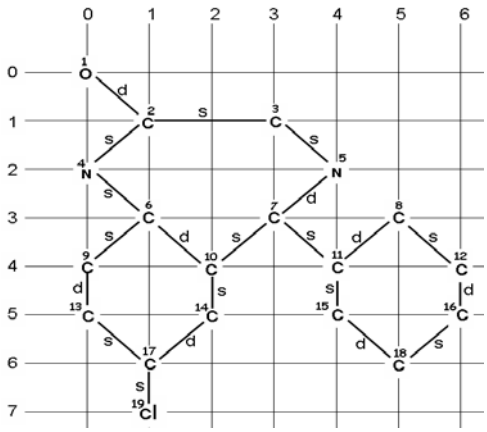


Fig. 6: Grid Representation of Diazepam

The grid code of Diazepam is written as

GC(Diazepam)={19,21,((**O**,**C**,d,1,2),(**C,C**,s,2,3),(**C,N**,s,2,4),(**C,N**,s,3,5),
(**N,C,**s,4,6),(**N,C,**d,5,7),(**C,C**,s,6,9),(**C,C**,d,6,10),(**C,C**,s,7,10),(**C,C**,s,7,11),
(**C,C**,d,8,11),(**C,C**,s,8,12),(**C,C**,d,9,13),(**C,C**,s,10,14),(**C,C**,s,11,15),
(**C,C**,d,12,16),(**C,C**,s,13,17),(**C,C**,d,14,17),(**C,C**,d,15,18),(**C,C**,s,16,18),
(**C,Cl**,s,17,19)),$v_{Degree}(G_{Diazepam})$, $N\beta_{Diazepam}$}.

The degrees of different vertices with same vertex labels is written as

$v_{Degrees}$ (Diazepam)=

Identifiers 2 3 6 7 8 9 10 11 12 13 14 15 16 17 18
Degrees {C{ **3, 2, 3, 3, 2, 2, 3, 3, 2, 2, 2, 2, 2, 3, 2**},

19 4 5 1
Cl{1}, N{2, 2}, O{1}}

The set of vertices with their neighbours and their degrees is written as

β(Diazepam)=

2 3 1 4 3 2 5 6 4 9 10 7 5 10 11 8 11 12 9 6 13
{C{C, O, N},C{C, N},C{N, C, C},C{N, C, C},C{C, C},C{C, C},

10 7 6 14 11 7 8 15 12 18 16 13 9 17 14 10 17 15 11 18
C{C, C, C},C{C, C, C},C{C, C, C},C{C, C, C},C{C, C, C},C{C, C, C},

16 12 18 17 19 13 14 18 15 16 19 17 4 2 6 5 3 7 1 2
C{C, C},C{Cl, C, C},C{C, C}, Cl{C}, N{C, C},N{C, C}, O{C} }

The set of vertices with degrees of all its neighbours computed from $v_{Degrees}$ (Diazepam) and β(Diazepam) is written as

Nβ(Diazepam)=

{ 2 C (2,1,2) 3 C (3,2) 6 C (2,2,3) 7 C (2,3,3) 8 C (3,2) 9 C (3,2) 10 C (3,3,2) 11 C (3,2,2) 12 C (2,2)
13 C (2,3) 14 C (3,3) 15 C (3,2) 16 C (2,2) 17 C (1,2,2) 18 C (2,2) 19 Cl (3) 4 N (3,3) 5 N (2,3) 1 O (3) }

The number of comparisons for computing the $v_{Degrees}$ and β is 2 * 21. The same number of comparisons are needed for Nβ as well. Hence the total time of GridCodeGen(Diazepam) is 2 * 42.

Assume there are already 100 graphs in GD. To compare 101th graph with 100 graphs, at the worst case the number of comparisons taken by F-GAF would be 84+100(1+1+(3*21)+((5*19)^2)-19) = 907184.

On the other hand, the canonical labeling works with the partitioned adjacency matrix of Diazepam given in Figure 7.

Id	1	19	4	5	3	8	9	12	13	14	15	16	18	2	6	7	10	11	17
Label	O	Cl	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Degree	1		2											3					
1 O	0	0	0	0	0	0	0	0	0	0	0	0	0	d	0	0	0	0	0
19C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s
4 N	0	0	0	0	0	0	0	0	0	0	0	0	0	s	s	0	0	0	0
5 N	0	0	0	0	s	0	0	0	0	0	0	0	0	0	0	d	0	0	0
3 C	0	0	0	s	0	0	0	0	0	0	0	0	0	s	0	0	0	0	0
8 C	0	0	0	0	0	0	0	s	0	0	0	0	0	0	0	0	0	d	0
9 C	0	0	0	0	0	0	0	0	d	0	0	0	0	0	s	0	0	0	0
12 C	0	0	0	0	0	s	0	0	0	0	0	d	0	0	0	0	0	0	0
13 C	0	0	0	0	0	0	d	0	0	0	0	0	0	0	0	0	0	0	s
14 C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s	0	d
15 C	0	0	0	0	0	0	0	0	0	0	0	0	d	0	0	0	0	s	0
16 C	0	0	0	0	0	0	0	d	0	0	0	0	s	0	0	0	0	0	0
18 C	0	0	0	0	0	0	0	0	0	0	d	s	0	0	0	0	0	0	0
2 C	d	0	s	0	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6 C	0	0	s	0	0	0	s	0	0	0	0	0	0	0	0	0	d	0	0
7 C	0	0	0	d	0	0	0	0	0	0	0	0	0	0	0	0	s	s	0
10 C	0	0	0	0	0	0	0	0	0	s	0	0	0	0	d	s	0	0	0
11 C	0	0	0	0	0	d	0	0	0	0	s	0	0	0	0	s	0	0	0
17C	0	s	0	0	0	0	0	0	s	d	0	0	0	0	0	0	0	0	0

Fig. 7: Partitioned adjacency matrix of Diazepam

The number of permutations to compute all possible canonical codes is 2!*9!*6!. Therefore, 522547200 distinct codes each of length 161 are generated. The number of comparisons to find unique code accounts to 84130099200. This unique code of Diazepam is then compared with the 100 graphs in GD leading to an additional 16100 comparisons at the worst case.

This study shows that F-GAF performs extremely well compared to canonical labeling.

4.4 Experimental Results of Graph Traversal

A comprehensive study on synthetic data sets to compare (i) time taken for DFS using adjacency list, adjacency matrix and grid representation (ii) Canonical labeling Vs. F-GAF Algorithms was conducted.

The time taken for depth first traversal of graphs using adjacency list, adjacency matrix and grid representation for various types of graphs such as complete, non-sparse, and sparse graphs was studied. The results are given in Table 2. This work explores the need for a generic representation of graphs, more specifically, chemical graphs.

Table 2: Time taken for depth first traversal for complete, sparse and non-sparse graphs

Execution time for DFS in seconds				
Complete Graphs				
No of Vertices	**No of Edges**	**Adjacency List**	**Adjacency Matrix**	**Grid Representation**
4	7	0.494505	0.274725	0.049451
5	11	0.549451	0.384615	0.214286
6	14	0.714286	0.494505	0.126374
7	18	0.494505	0.659341	0.351648
8	23	0.879121	0.769231	0.576923
Sparse Graphs				
4	3	0.494505	0.164835	0.010989
4	4	0.43956	0.164835	0.016484
6	5	0.659341	0.21978	0.021978
6	5	0.659341	0.21978	0.021978
9	8	0.989011	0.32967	0.071429
Non-Sparse Graphs				
5	7	0.549451	0.274725	0.038462
6	8	0.659341	0.32967	0.071429
6	11	0.659341	0.43956	0.126374
9	8	0.989011	0.32967	0.065934

4.5. Performance Evaluation of F-GAF

Synthetic data sets were chosen for the current experiments since the algorithm has to analyze all possible types of graphs like complete, sparse and non-sparse

graphs to prove its efficiency. Table 3 shows the behaviour of F-GAF against canonical labeling. The experimental analysis of chemical graph data sets given in Table 3 also reveals an optimistic performance of F-GAF algorithm over canonical labeling. The results are obtained on chemical graph data sets with different number of graphs.

Table 3: Time taken to detect Automorphic graphs using canonical labeling and F-GAF

Number of graphs in GD=100					
Number of Vertices V	**% of vertices having same labels and degrees**	**Time in seconds**			
		Canonical Labeling	**F-GAF**		
			Number of Edges		
			Sparse	**Non-Sparse**	**Complete**
			1000	**3000**	**4950**
100	6	2.09	4.55	14.55	22.44
	7	41.7	4.55	14.55	22.44
	8	704.8	4.55	14.56	22.44
	9	9052.6	4.56	14.56	22.45
	10	135789.9	4.56	14.56	22.45
			23940	**55860**	**79800**
400	2.0	704.8	123.57	288.96	414.39
	2.5	135789.9	123.58	288.98	414.40
	2.75	2036835.7	123.58	288.99	414.44
	3.0	30552525.9	123.80	289.02	414.46
			37425	**87325**	**124750**
500	1.6	704.8	192.04	449.44	631.92
	1.8	9052.6	192.07	449.45	631.93
	2.0	135789.9	192.09	449.45	631.93
	2.2	2036835.7	192.12	449.46	631.94
	2.4	30552525.9	192.18	449.46	631.94
			149850	**349650**	**499500**
1000	0.8	704.8	406.11	767.96	1101.03
	1.0	135789.9	406.12	767.97	1101.04
	1.2	30552525.9	406.12	767.97	1101.05

From the empirical analysis, it is found that the canonical labeling is based on the number of vertices as it is a vertex based representation. On the other hand, F-GAF is edge-based. Hence the time of execution varies for sparse, non-sparse, and complete graphs.

The time complexity of the canonical labeling algorithm varies from polynomial (if few vertices have same labels and same degree) to exponential (if more vertices have same labels and same degrees). The algorithm used in the proposed study takes almost fixed time for each type of graphs irrespective of the number of partitions. When most of the vertex labels are same in a graph (such as the carbon atoms being the vertex labels in chemical graphs), the time complexity of canonical labeling algorithm is very huge when compared to F-GAF. This algorithm saves a good amount of storage space, when it is used in chemical graphs data set. This is because, almost all chemical graphs are non-sparse but not complete.

5 Conclusion

In canonical labeling, all possible permutations of v vertices with the same vertex labels and degrees result in |v|! to identify automorphism. In the proposed algorithm, the search space is drastically narrowed down by exactly locating the vertex labels avoiding redundant storage of the same graph into the graph database. Instead of generating all possible permutation matrices, it uses a simple string comparison procedure to find whether the two graphs match.

6 Open Problems

The active research areas include frequent subgraph mining, maximal subgraph mining, etc. The efficient storage and retrieval of graph data is still a challenging problem. The proposed representation could be used for representing graph data such as chemical graphs, biological networks, and web graphs.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. of the 20th Int. Conf. on Very Large Databases (VLDB)*, pages 487–499. Morgan Kaufmann, September 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. L. P. Chen, editors, *Proc. of the 11th Int. Conf. on Data Engineering (ICDE)*, pages 3–14. IEEE Press, 1995.

[3] Akihiro Inokuchi, Takashi Washio, Hiroshi Motoda: Complete Mining of Frequent Patterns from Graphs. *Mining Graph Data. Machine Learning* 50(3): 321-354 (2003)

[4] C.-W. K. Chen and D. Y. Y. Yun. Unifying graph-matching problem with a practical solution. *In Proc. Of International Conference on Systems, Signals, Control, Computers*, September 1998.

[5] R. N. Chittimoori, L. B. Holder, and D. J. Cook. Applying the SUBDUE substructure discovery system to the chemical toxicity domain. *In Proc. of the 12th International Florida AI Research Society Conference*, pages 90–94, 1999.

[6] V. A. Cicirello. Intelligent retrieval of solid models. *Master's thesis, Drexel University*, Philadelphia, PA, 1999.

[7] Diane J. Cook, Lawrence B. Holder, Mining Graph Data, *John Wiley & sons, Inc.*, 2007.

[8] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. *In Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[9] M. Deshpande and G. Karypis. Automated approaches for classifying structures. *In Proc. of the 2nd Workshop on Data Mining in Bioinformatics (BIOKDD '02)*, 2002.

[10] D. Dupplaw and P. H. Lewis. Content-based image retrieval with scale-spaced object trees. In M. M. Yeung, B.-L. Yeo, and C. A. Bouman, editors, *Proc. of SPIE: Storage and Retrieval for Media Databases*, volume 3972, pages 253–261, 2000.

[11] S. Fortin. The graph isomorphism problem. *Technical Report TR96-20*, Department of Computing Science, University of Alberta, 1996.

[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *In Proc. of ACM SIGMOD Int. Conf. on Management of Data*, Dallas, TX, May 2000.

[13] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the SUBDUE system. *In Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[14] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. *Proc. 1st IEEE Int. Conf. on Data Mining (ICDM 2001*), San Jose, CA), 313–320. IEEE Press, Piscataway, NJ, USA 2001.

[15] D. A. L. Piriyakumar and P. Levi. An efficient A* based algorithm for optimal graph matching applied to computer vision. *In GRWSIA-98*, Munich, 1998.

[16] R. C. Read and D. G. Corneil. The graph isomorph disease. *Journal of Graph Theory*, 1:339–363, 1977.

[17] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. *In Proceedings of the 7th International Workshop on Inductive Logic Programming,* volume 1297, pages 273–287, 1997.

[18] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1–6, 1997.

[19] R. Vijayalakshmi, R. Nadarajan, and B. Malar. A study of substructure similarity search in graph data bases using grid-based approach. *In Proceedings of the First International Conference on Information Processing (ICIP)*, pages 481-490, 2007.

[20] K. Yoshida and H. Motoda. CLIP: Concept learning from inference patterns. *Artificial Intelligence*, 75(1):63–92, 1995.