ADI-Minebio: A Graph Mining Algorithm for Biomedical Data

Rodrigo de Sousa Gomide[1], Cristina Dutra de Aguiar Ciferri[2], Ricardo Rodrigues Ciferri[3] and

Marina Teresa Pires Vieira[4]

[1] Goiano Federal Institute of Education, Science and Technology, 76300-000, Ceres, GO, Brazil
rodrigo.gomide@ifgoiano.edu.br
[2] Department of Computer Science, University of São Paulo, 13560-970, São Carlos, SP, Brazil
cdac@icmc.usp.br
[3] Department of Computer Science, Federal University of São Carlos, 13565-905, São Carlos, SP, Brazil
ricardo@dc.ufscar.br
[4] Faculty of Mathematical and Nature Sciences, Methodist University of Piracicaba, 13400-911, Piracicaba, SP, Brazil
mtvieira@unimep.br

Abstract. Graph mining is concerned with mining frequent subgraph patterns over a collection of graphs, aiming to find novel and useful knowledge. It has being used to analyze data from different domains, sometimes using algorithms tailored for a specific area of knowledge. In this paper, we propose a graph-mining algorithm and its application in the biomedical domain. We introduce the ADI-bio structure, which organizes data from a database with information of a disease's patient, and also the ADI-Minebio algorithm, which performs a search on the proposed ADI-bio structure to find frequent subgraphs. Our approach is based on the ADI (adjacency index) structure and the ADI-Mine algorithm, but specifies a different structure and hence a new way of analyzing data through this structure. We also present a performance study to show the feasibility of our approach.

Categories and Subject Descriptors: H.2.1 [**Information Systems**]: Data Mining and Knowledge Discovery

Keywords: graph, graph mining, biomedical data, adjacency index

1. INTRODUCTION

Graphs are important data structures composed of nodes (i.e. vertices) and links (i.e. edges), and are typically used to model data from complex applications such as bioinformatics, social networks analysis, text retrieval, chemical compounds, protein structures and XML documents [Cook and Holder 2007], [Han and Kamber 2006]. Graph mining, or graph-based data mining, is concerned with mining frequent subgraph patterns over a collection of graphs, aiming to find novel and useful knowledge. It provides a way to analyze data when the application involves patterns that are more complex than frequent itemsets and sequential patterns [Han and Kamber 2006], and possibly the knowledge to be mined is spread in interrelated transactions.

In this paper, we propose ADI-Minebio, a graph-mining algorithm aimed at analyzing data from a database with information of treatments of diseases and side effects caused by these treatments. Side effects are usually other diseases that also need a specific treatment. Therefore, we seek to investigate relationships between treatments for the original disease and treatments for the side effects originated from the original disease or subsequent side effects. We organize this information in graphs and use the proposed ADI-Minebio algorithm to extract meaningful substructures from these graphs.

The problem of mining biomedical data arose from a need of medical experts from the Medical Faculty of Ribeirão Preto from the University of São Paulo, which investigate the Sickle Cell Anemia (SCA) and its related diseases. The motivation is to aid experts to find patterns among treatments and side effects (i.e. new diseases), by developing both a database with information of interest and a related graph-mining algorithm. Our current work is included in an environment called Information Extraction and Decision Support System in Biomedical domain (IEDSS-Bio), which is under development and addresses supporting the expert in making decisions, by extracting relevant information from biomedical documents, storing the information in a data warehouse, and mining interesting knowledge from the data warehouse [Matos et al. 2010]. It is worth to note that the database and the data warehouse are not completely loaded with real data, since the extraction of information from scientific papers is still been performed. Therefore, we use only synthetic data to assess our proposals. In this paper, our synthetic data of interest is stored in relational tables and our algorithms transform these tables in a graph database.

Regarding our proposal, the ADI-Minebio algorithm searches for frequent subgraphs in the graph database using a new index structure also proposed in this paper, called ADI-bio, which is specially organized to structure data from a database with information of patients of a given disease. Our approach is based on the ADI (adjacency index) structure, which provides an efficient way to search for frequent subgraphs in large databases that cannot be held in main memory, and also on the ADI-Mine algorithm [Wang et al. 2004], a graph-mining algorithm that improves the gSpan algorithm [Han and Yan 2002] by using the ADI structure. The main differentials of the proposed ADI-Minebio algorithm is that it uses a different index structure and introduces a new way of analyzing data based on this index, providing an adequate way of working with graph data from the biomedical domain (i.e. relationships among treatments and their side effects). Our performance tests showed the feasibility of our approach when applied to discover new knowledge from a graph database storing information about treatments and diseases.

The reminder of this paper is organized as follows. Section 2 describes the theoretical foundation and the basis of our proposals, Section 3 presents the proposed ADI-bio and ADI-Minebio algorithms, Section 4 discusses performance results used to assess our proposals, Section 5 surveys related work and Section 6 concludes the paper and presents future work.

2. THEORETICAL FOUNDATION

In this section, we describe the adjacency index (Section 2.1) and the ADI-Mine Algorithm (Section 2.2), which are used as a basis for our proposals.

2.1 Adjacency Index

An adjacency index (ADI) is an index structure that supports mining graph patterns over large databases that cannot be held into the main memory [Wang et al. 2004]. It indexes the database on three levels: the first level represents the edges, the second level represents the graph identifications and the third level represents adjacency information. Figure 1 depicts two graphs G1 and G2, as well as shows how they are indexed by ADI.

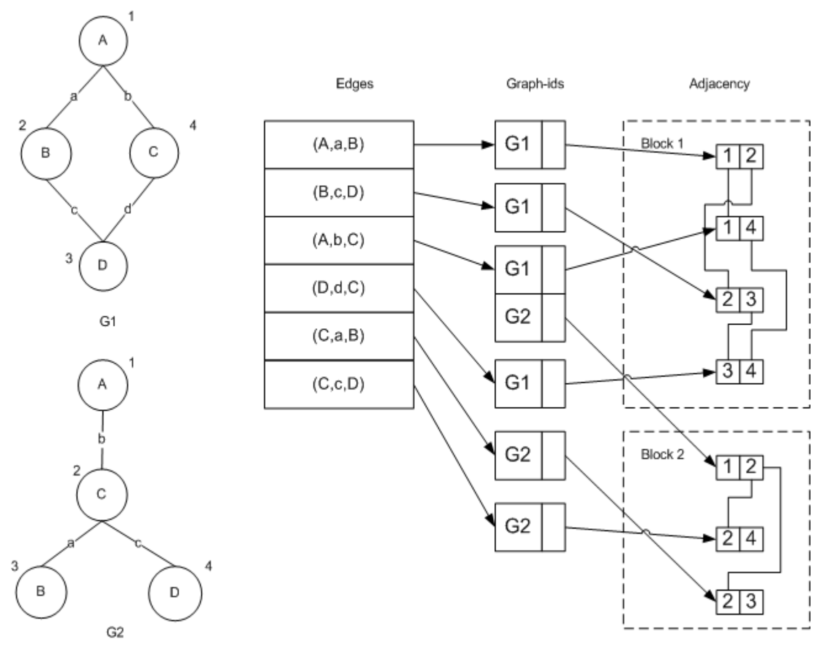

Fig. 1. An example of the ADI Structure.

The first level indexes all the edges present in all graphs through a list in the ADI structure. In ADI, an edge *e = (u, v)* is stored as a tuple *(l(u)*; *l(u:v)*; *l(v))*, where *l(u)*, *l(u:v)*, *l(v)* are the label of the first vertex, the label of the edge and the label of the second vertex, respectively. The mappings of all edges are stored sorted in an edge table. For instance, in Figure 1, the first tuple of the edge table represents an edge between the vertices A and B. Furthermore, each edge appears only once in the edge table, independently on how many times it appears in the graphs.

The second level is a linked list of all graphs that have an edge stored in the edge table. In detail, for each edge *e* of the edge table, there is a linked list of graphs represented by their identification, such that a graph will be present in this list only if it has the edge *e*. For instance, in Figure 1, both G1 and G2 contain the edge represented by (A, b, C). Note that each graph appears only once in a linked list of an edge *e*, independently on how many times *e* appears in the graph.

The third level consists of the adjacency information. In this level, all the edges of a graph are stored in blocks, such that each block represents a graph. Also, the edges are linked with adjacent edges, enabling the visualization of the connections between nodes. For instance, in Figure 1, the edge of block 1 encoded by (1, 2) is adjacent to the edge encoded by (2, 3) through the vertex encoded by 2.

Depending on the available main memory, the three levels of ADI may be stored on disk or in the main memory. If the graph database is small, the whole index can be held into the main memory. Otherwise, the levels of the index are stored on disk according to the following order: first the third level, then the second level, and finally the first level. The clustering of edges in blocks in the third level speed up the storage and the retrieval of graph data on disk.

2.2 The ADI-Mine Algorithm

The ADI-Mine algorithm [Wang et al. 2004] adapts the gSpan algorithm [Han and Yan 2002] to search for frequent subgraph patterns using the ADI structure. Therefore, like the gSpan algorithm, the ADI-

Mine algorithm is based on the depth-first search (DFS) for navigating through connected graphs and on the search for isomorphic graphs using the minimum DFS code notation.

In detail, using as input a graph database and a minimum support, the ADI-Mine algorithm produces as output a set of frequent graphs patterns described as follows. The algorithm first removes all the edges present in the edge table that do not satisfy the minimum support. Then, the algorithm tracks each remaining edge in the edge table and adds this edge to the set of frequent graphs patterns. Also, for each edge, the algorithm recursively investigates its adjacent edges to identify which ones are also frequent, using as a basis the third level of ADI. Frequent adjacent edges are also included in the frequent graphs patterns, except when they generate isomorphic graphs.

The identification of isomorphic graphs is performed by the minimum DFS code notation, a technique composed of three phases: (i) creating a DFS tree; (ii) generating the DFS code for this tree; and (iii) organizing the generated code in lexicographic order. A DFS tree corresponds to the path from the root to its rightmost vertex, which is named rightmost path. Figure 2 shows a graph composed of the vertices {v0, v1, v2, v3}, such that only the vertices {v0, v1, v3} form the DFS tree. The edges in the rightmost path, represented by dark lines, are called forward edges, while the remaining edges are known as back edges.

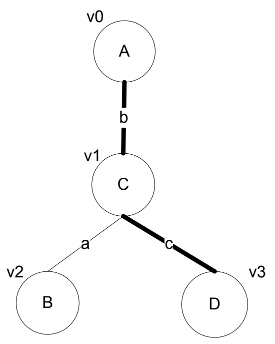

Fig. 2. An example of a graph and its DFS tree.

The DFS code corresponds to the labeling of all the edges of a graph. It is constructed using the 5-tuple *(i, j, l_i, $l_{(i,j)}$, l_j)*, where *i* and *j* represent vertices from the DSF tree, l_i is the label of the vertex *i*, $l_{(i,j)}$ is the label of the edge that connects *i* to *j*, and l_j is the label of the vertex *j*. The DFS code for the DFS tree of Figure 2 is shown in Table 1.

Table 1. The DFS code for the DFS tree of Figure 2.

# Edge	The DFS Code
0	(v0, v1, A, b, C)
1	(v1, v3, C, c, D)
2	(v1, v2, C, a, B)

The organization of the DFS code in lexicographic order is performed considering the linear order of the graph edges. Given two edges $e_{i,j}$ and $e_{x,y}$, $e_{i,j} < e_{x,y}$ when one of the following conditions is satisfied: (i) if both $e_{i,j}$ and $e_{x,y}$ are forward edges, then $j < y$ or $j = y$ and $i > x$; (ii) if both $e_{i,j}$ and $e_{x,y}$ are backward edges, then $i < x$ or $i = x$ and $j < y$; (iii) if $e_{i,j}$ is a forward edge and $e_{x,y}$ is a backward edge, then $i < y$; and (iv) if $e_{i,j}$ is a backward edge and $e_{x,y}$ is a forward edge, then $i > y$. The lexicographic order of the DFS codes shown in Table 1 is {(v0, v1, A, b, C), (v1, v3, C, c, D), (v1, v2, C, a, B)}. Two graphs are isomorphic when they are represented by the same lexicographic order.

3. GRAPH MINING FROM A BIOMEDICAL DATABASE

In this section, we define the problem of graph mining (Section 3.1), we also discuss the construction of a graph database from relational tables (Section 3.2), describe the proposed ADI-bio structure (Section 3.3) and introduce the proposed ADI-Minebio graph-mining algorithm (Section 3.4).

3.1 Problem Definition

We define our goal in mining a biomedical database as to discover a set of frequent subgraphs that represent patterns involving treatments of diseases and side effects caused by these treatments. In fact, side effects are usually other diseases that also need a specific treatment. Therefore, we seek to investigate relationships between treatments for the original disease and treatments for the side effects originated from the original disease or from subsequent side effects.

Formally, let D be a disease, T_D be one of the treatments used for a patient with D, SE be one of the side effects caused by T_D, and T_{SE} be one of the treatments used for SE. Our goal is to find existing relationships between T_D and T_{SE}, which is represented by the following sequence $D \rightarrow T_D \rightarrow SE \rightarrow T_{SE}$, where → represents the relationship of treatment-effect. Also, it is possible that a side effect SE_1 and its treatment T_{SE1} be related to another side effect SE_2 and its related treatment T_{SE2}, and so on, composing a sequence of treatments and side effects, such that $D \rightarrow T_D \rightarrow SE_1 \rightarrow T_{SE1} \rightarrow SE_2 \rightarrow T_{SE2} \dots \rightarrow SE_n \rightarrow T_{SEn}$.

Table 2 shows a sample of data related to treatments and side effects of the SCA disease and its related diseases, whose investigation represent the main motivation of our work. This *disease-treatment-effect table* contains the following data: (i) identification of the paper from which the information has been extracted; (ii) name of the disease (or the side effect) that is analyzed in the paper; (iii) name of the treatment for this disease (or for the side effect); and (iv) name of the side effect caused by the treatment. An example of sequence of treatment and side effects to be analyzed is: Sickle Cell Anemia (SCA) → Hydroxyurea (Hydro) → Constipation → Hydration (Hydt) → Vomiting (Vo) → Dexametasona (Dex).

The representation of the *disease-treatment-effect table* as a graph follows the baselines described in this section, and is discussed in Section 3.2.

Table 2. An example of data from the *disease-treatment-effect table*.

# Paper	Disease Name	Treatment Name	Effect Name
1	Sickle Cell Anemia (SCA)	Standard BFM (BFM)	Hepatomegaly (Hep)
1	Sickle Cell Anemia (SCA)	Hydroxyurea (Hydro)	Hepatomegaly (Hep)
1	Sickle Cell Anemia (SCA)	Hydroxyurea (Hydro)	Constipation (aCo)
…	…	…	…
2	Sickle Cell Anemia (SCA)	Standard BFM (BFM)	Hepatomegaly (Hep)
2	Sickle Cell Anemia (SCA)	Hydroxyurea (Hydro)	Vomiting (Vo)
…	…	…	…
3	Sickle Cell Anemia (SCA)	Standard BFM (BFM)	Hepatomegaly (Hep)
3	Sickle Cell Anemia (SCA)	Hydroxyurea (Hydro)	Constipation (aCo)
3	Sickle Cell Anemia (SCA)	Hydroxyurea (Hydro)	Vomiting (Vo)
…	…	…	…
23	Constipation (aCo)	Hydration (Hydt)	Vomiting (Vo)
24	Constipation (aCo)	Hydration (Hydt)	Vomiting (Vo)
…	…	…	…
35	Vomiting (Vo)	Dexametasona (Dex)	Jitters (Jitt)
…	…	…	…
41	Vomiting (Vo)	Dexametasona (Dex)	Jitters (Jitt)

3.2 The Process to Build a Graph Database from Relational Tables

Our graph database contains information about treatments and side effects of these treatments, and is represented as a directed labeled graph. While its vertices correspond to diseases and side effects (e.g. the target disease D or its side effects $SE_1, SE_2, \ldots, SE_n$) or correspond to treatments for a given disease or side effect (e.g. T_D, $T_{SE1}, T_{SE2}, \ldots, T_{SEn}$), its edges represent relations between diseases/side effects and treatments. Furthermore, the label of each edge represents the frequency of occurrence of this edge in the *disease-treatment-effect table*. For instance, Figure 3 shows the graph database that represents the information depicted partially in Table 2.

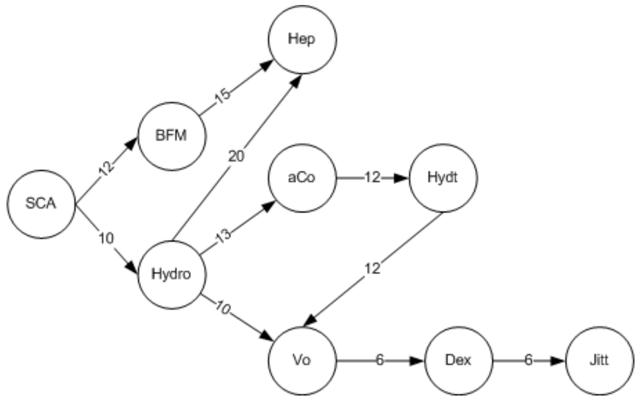

Fig. 3. The *disease-treatment-effect* graph database for the information depicted in Table 2.

The construction of the graph database starts with the vertex whose label is the disease that is the focus of analysis, i.e. the disease from which we wish to discover relationships among effects and treatments. In our example, this disease is the SCA disease (i.e. D = SCA), which is represented in the leftmost vertex in the graph of Figure 3. The treatments of SCA (i.e. T_{D1} = BFM and T_{D2} = Hydro) compose the next vertices, which are linked to the SCA vertex. Then, vertices are created for the side effects of treatments (e.g. T_{SE1} = Hep, T_{SE2} = aCo and T_{SE3} = Vo for the treatment Hydro). For each side effect introduced in the graph, we obtain additional information (i.e. its treatments and side effects) in other rows of the table, if its name is present in the column *Disease Name* in the *disease-treatment-effect table*. The label of the edges represents frequencies of a treatment-disease, such as the label 12 in the edge that connects the vertices SCA and BFM indicates that the information that the disease SCA is treated by the treatment BFM occurs twelve times, three of them shown in rows of Table 2.

Therefore, for a given disease, the resulting graph database contains: (i) a set of treatments of this disease; (ii) a set of side effects caused by the treatments; and (iii) a set of treatments that can be used to combat these side effects, which are obtained from other scientific papers that report experiences involving patients with these new diseases.

3.3 The Proposed ADI-bio Structure

In this section, we propose a new structure for the adjacency index, called ADI-bio, which is specially designed according to the characteristics of the graph database. ADI-bio is composed of two levels. The first level stores edges and frequencies of edges, while the second level stores the adjacency information. Figure 4 shows an example of the ADI-bio structure for the graph depicted in Figure 3.

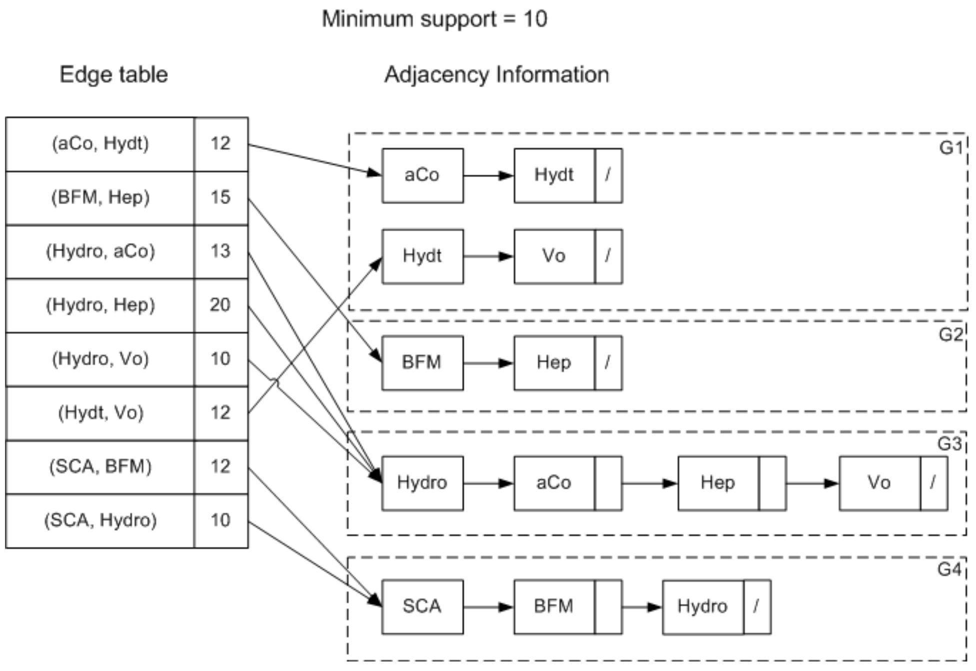

Fig. 4. An example of the ADI-bio structure.

The components of the ADI-bio structure are detailed as follows.

- *Edge Table*: contains each distinct edge of the graph database, whose label value satisfies the minimum support. The *Edge Table* is organized in alphabetical order.
- *Adjacency Information*: for each edge *e = (u, v)* of the *Edge Table*, the *Adjacency Information* contains a linked list that begins with the vertex *u*. The inclusion of the remaining components in the list should respect the following formation rules:
    - Include each vertex found in a branch from *u*, following a depth-first search in the graph, until the end of the branch or until finding a vertex that has already been included in some subgraph of the *Adjacency Information*. For instance, for the edge (aCo, Hydt), the vertices aCo, Hydt and Vo are included in the *Adjacency Information*, generating the subgraph G1, which is represented by 2 lists. The first list contains all adjacent vertices to aCo, and the second list contains all adjacent vertices to Hydt. There is no list for the vertex Vo as the minimum support is set to 10 and, according to Figure 3, the frequency of the edge (Vo, Dex) is 6.
    - Do not include a vertex that has already been inserted into a subgraph. In this case, this vertex is ignored and the next edge from the *Edge Table* is analyzed according the same described rules. For instance, the construction of the subgraph G3 regarding the edge (Hydro, aCo) is performed as follows. First, the vertices Hydro and aCo are included in G3. Then, the vertices Hydt and Vo are ignored, as they already compose the subgraph G1. In the sequence, the edge (Hydro, Hep) is analyzed, and the vertex Hep is included in G3. Finally, as the next edge of the *Edge Table* also refers to Hydro (i.e. the edge (Hydro, Vo)), the vertex Vo is also included in G3. As a result, G3 is represented by the list <Hydro, aCo, Hep, Vo>.

The algorithm to construct the ADI-bio structure is shown in Algorithm 1. It uses as inputs a graph database and a minimum support and generates as output the ADI-bio structure, named *adibiofreq*, which contains only the frequent edges. The algorithm works as follows. An auxiliary variable *adibio*, which has the same structure as *adbiofreq*, is created in line 1 to store all the vertices and edges that are found during the execution of the algorithm. The *Adjacency Information* of *adibio* is initialized in line 3. Then, the graph database is traversed (lines 4 to 15) and each element is added to the *Adjacency Information* of *adibio* if it is a vertex (lines 5 and 6) or is added to the *Edge Table* of *adibio* if it is a new edge (line 8 and 9). In the latter case, the frequency counter of this new edge is initialized to 1 (line 10) or is incremented by 1 if the *Edge Table* of *adibio* already contains the new edge (line 12). The *adibiofreq* structure is created in line 16, using as a basis the *adibio* structure. While *adibio* structure contains all the edges of the graph database and its adjacent edges, *adibiofreq* stores only the frequent edges and their correspondent adjacent edges. Finally, the *Edge Table* of *adibiofreq* is sorted in alphabetical order (line 17) and the *Adjacency Information* of *adibiofreq* is reorganized (line 18), using as a basis the formation rules.

```
Algorithm 1: ADIbio (GBD, mins_up)
     Input: GBD {graph database}, mins_up {minimum support}
     Output: adibiofreq {linked list of records containing the fields edgeTable and
                          adjacencyInformation}
     var adibio: linked list of records containing the fields
                 edgeTable and adjacencyInformation
     initialize the adjacency information (i.e. adibio.adjacencyInformation)
     for each element e of GDB do
       if e is a vertex
         store e in the adjacency information
       else
         if e is a new edge then
           insert e into edge table (i.e. adibio.edgeTable)
           adibio.edgeTable(e).countEdge ← 1
         else
           adibio. edgeTable(e).countEdge ← countEdge + 1
         endif
       endif
     endfor
     create the structure adibiofreq, composed of frequent subgraphs from adibio
     sort the edge table of adibiofreq
     reorganize the adjacency information of adibiofreq
     return adibiofreq
```

The *Adjacency Information* of the *adibiofreq* structure is used to find frequent subgraphs, as discussed in Section 3.4.

3.4 The Proposed Algorithm for Extracting Frequent Subgraphs

Aiming at extracting subgraphs from the graph database, our algorithm uses the concepts of DFS code and minimum DFS code. It also uses the ADI-bio data structure to find the frequent subgraphs. Each row of the *Edge Table* represents a frequent subgraph. The remaining subgraphs are found through the adjacent edges for each edge present in the edge table, using the *Adjacency Information*.

The ADI-Minebio algorithm is shown in Algorithm 2. It uses as input the *adibiofreq* structure generated by Algorithm 1 and generates as outputs a list of all frequent subgraphs and a list of adjacent edges. The ADI-Minebio algorithm works as follows. In lines 1 and 2, it defines two variables: (i) *LF*, which stores all the frequent subgraphs, and *lAdjacency*, which maintains the edges that compose a subgraph. In lines 3 to 9, the algorithm traverses the *Edge Table* and for each edge *e*, it performs the following steps: it adds *e* to *lAdjacency* (line 4); it obtains the minimum DFS code of the subgraph in *lAdjacency* (line 5); it adds *lAdjacency* to *LF*, since *lAdjacency* is a frequent subgraph (line 6); it finds F_e, which is a list of adjacent edges from *lAdjacency* (line 7); and finally it calls the procedure subgraph-mine to generate a new subgraph by performing an extension of the edge *e* using its adjacent edges (line 8).

To exemplify this first part of Algorithm 2, consider the processing of the first edge of the *Edge Table* of Figure 3 (i.e. *e* = (aCo, Hydt). The following partial results are obtained: (i) the list of adjacent edges *lAdjacency* = <(aCo,Hydt)>; (ii) the list of the frequent subgraphs *LF* = <{(aCo,Hydt})>; (iii) and the list F_e of all edges that are adjacent to *e,* i.e. F_e = <(Hydt,Vo)>.

The *subgraph-mine* procedure is responsible for creating new extensions of frequent subgraphs by using the edges from the *Edge Table* that are adjacent to the edges of these subgraphs. For each edge from F_e (line 13), the following steps are performed: *lAdjacency* is reset (line 14); the edge is added to *lAdjacency*, generating a new subgraph (line 14); and the DFS code of *lAdjacency* is computed (line 16). The new subgraph generated is considered only if its DFS code is minimum and the existent list of frequent subgraphs does not contain it (lines 17 to 22). If this test evaluates to true, the new subgraph is added to *LF* (line 23). Finally F_e is updated with new adjacent edges (line 24). The subgraph-mine procedure is executed recursively (line 25), until all adjacent edges have been considered.

Using the same previous example, the algorithm finds the edges adjacent to the edge (Hydt, Vo). In this case, there are no adjacent edges. As a result, the subgraph-mine procedure ends and Algorithm 2 processes the next edge present in the *Edge Table*, i.e. the edge (BFM, Hep). In the sequence, the edge (BFM, Hep) is added to the list of adjacent edges, its minimum DFS code is computed and the edge is also added to the list of frequent subgraphs. As there are no edges adjacent to the edge (BFM, Hep), Algorithm 2 continues by evaluating the next edge, i.e. the edge (Hydro, aCo).

Considering the edge (Hydro, aCo), the following adjacent edges are found: (Hydro, Hep), (Hydro, Vo), (aCo, Hydt). In lines 14 to 16, the algorithm generates the new subgraph {(Hydro, aCo), (Hydro, Hep)}, and in line 14, it updates the list of frequent edges to F_e = <(Hydro, Vo), (aCo, Hydt)>, since there are no adjacent edges to (Hydro, Hep). Using F_e, the subgraph {(Hydro, aCo), (Hydro, Hep), (Hydro, Vo)} is built, and again the list F_e is updated, producing F_e = <(ASC Hydt)>. The next subgraph generated is: {(Hydro, aCo), (Hydro, Hep), (Hydro, Vo), (aCo, Hydt)}, and the list is updated with the edge (Hydt,Vo), which is adjacent to the edge (aCo, Hydt). In the sequence, the last subgraph generated for (Hydro, aCo) is {(Hydro, aCo), (Hydro, Hep), (Hydro, Vo), (aCo, Hydt) (Hydt, Vo)}. Finally, the processing of the edge (Hydro, aCo) ends, since there are no more adjacent edges. The same process is repeated for the following edges: (Hydro, Hep); (Hydro, Vo), (Hydt, Vo), (aCo, BFM) e (aCo, Hydro).

```
Algorithm 2: ADI-Minebio (adibiofreq)
     Input: adibiofreq {linked list of records containing the fields edgeTable
                  and adjacencyInformation}
     Output: LF {list of frequent subgraphs},
             lAdjacency {list of adjacent edges}
01   var LF: list
02       lAdjacency: list
03   for i=0; i<=adibiofreq.edgeTable.lenght; i++
04     lAdjacency.add(adibiofreq.edgeTable(i))
05     compute the minimum DFS code of the subgraph from lAdjacency
06     LF.add(lAdjacency)
07     find Fe, the list of adjacent edges from lAdjacency
08     call subgraph-mine(adibiofreq.edgeTable(i), lAdjacency, Fe, LF)
09   endfor
10
11   Procedure subgraph-mine
12   Parameters: edge, lAdjacency, Fe, LF
13   for i=0; i<=Fe.length-1; i++
14       lAdjacency = new list
15       add Fe[i] in lAdjacency
16       compute D, the DFS code of lAdjacency
17       if DFS code is not minimum
18         return;
19       endif
20       if minimal DSF code is in LF
21         return;
22       endif
23       LF.add(lAdjacency)
24       update Fe using the new adjacent edges
25       call subgraph-mine(Fe[i+1],lAdjacency, Fe, LF)
26    endfor
27    return
```

3.5 Further Considerations

The ADI-Minebio algorithm enables the specialist to discover new clues and make assumptions of occurrences that are not directly related to the data obtained from experiments with patients suffering from certain disease. For instance, consider the subgraph that includes the path SCA → Hydro → aCo → Hydt in the graph of Figure 3. This subgraph represents the combination of treatments and side effects in patients with the SCA, which is interpreted as: "*patients that were treated with Hydroxyurea may also need hydration*".

In our experiments, we focused on the SCA disease. However, our approach can be used for any other type of disease. Furthermore, although the algorithm has been developed to support a biomedical issue, it can be used in any application whose data is organized similarly to the database presented here (Table 2). Our algorithm is adequate for processing any table *T(Id, A, B, C)*, such that *A* is the object upon which we wish to perform analysis (in our example, *A* = disease), *C* has the same nature (domain) of A, and also exists a relationship between values of the attributes A and B, and among values of the attributes B and C. These relationships should be represented by the following sequence: $A_1 \rightarrow B_1 \rightarrow C_1 \rightarrow B_2 \rightarrow C_2 \ldots \rightarrow B_n$.

4. PERFORMANCE EVALUATION

In this section, we present and discuss the performance evaluation carried out to validate the proposed ADI-bio structure and the proposed ADI-Minebio algorithm. In the tests, we investigate three different issues. Firstly, we address the impact of increasing number of vertices and edges in the performance for building the ADI-bio's data structure. Secondly, we investigate the impact of increasing number of vertices in the performance of the ADI-Minebio algorithm. Finally, we analyze how the minimum support impairs the performance of the ADI-Minebio algorithm.

We used the J2SE 6.0 based on the Eclipse Galileo IDE to implement the proposals. Regarding the datasets, we used synthetic data about treatments and side effects of the SCA. It is worth to note that the database and the data warehouse of this disease are not completely loaded with real data, since the extraction of information from scientific biomedical papers is still been performed. Therefore, we used only synthetic data to assess our proposals. However, to generate the synthetic data, we followed the characteristics of some real data. We found experimentally, using an automatic tool developed as a component of the IEDSS-Bio environment (Section 1), that the number of treatments by disease and the number of side effects by treatment extracted from biomedical scientific papers varies from 4 to 7 and from 1 to 7, respectively. We used these findings in the generation of our synthetic data.

The minimum support and the number of vertices and edges of the generated graphs are specified in each test described in this section. The performance tests were carried out on a computer with a 3. 2 GHz Pentium core 2 duo P8700 processor, 4 GB of main memory, a 7200 RPM SATA 750 GB hard disk, Windows Vista operating system. We gathered the elapsed time in milliseconds and seconds.

Figure 5 shows the time spent to build the ADI-bio data structure, considering graphs containing from 1,000 up to 1,000,000 vertices and from 666 up to 660,000 edges (Table 3). In this test, we used the value of 1% for the minimum support. For smaller numbers of vertices and edges (i.e. up to 100,000 vertices and 66,000 edges), which generated up to 16,500 frequent edges, the ADI-bio data structure provided a linear growth in the building cost. But, the use of larger numbers of vertices and edges and, consequently, the increase in the frequent edges, impaired the time construction of the ADI-bio data structure.

Table 3: Characteristics of the synthetic datasets:
number of vertices, number of edges and number of frequent edges

Dataset	# Vertices	# Edges	# Frequent Edges
1	1,000	666	166
2	10,000	6,600	1,650
3	50,000	33,000	8,250
4	100,000	66,000	16,500
5	250,000	165,000	41,250
6	500,000	330,000	82,500
7	750,000	495,000	123,750
8	1,000,000	660,000	165,000

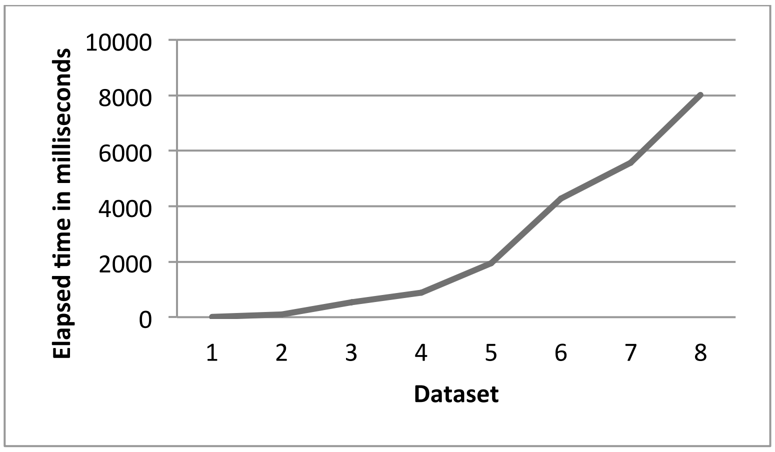

Fig. 5. Time spent to build the ADI-bio data structure.

Figure 6 shows the time spent to find frequent subgraphs patterns using the ADI-Minebio, considering graphs containing from 50,000 up to 100,000 vertices. The characteristics of the graphs are as follows. They had 70% of their vertices connected and 65% frequent edges. Also, we used the value of 1% for the minimum support. The performance results demonstrated that the ADI-Minebio algorithm provided good results, which were slightly superior to a linear growth. For instance, an increase of twice in the number of vertices (i.e. from 50,000 to 100,000 vertices) produced only an increase of four times in the elapsed time (i.e. from 50 seconds to about 200 seconds).

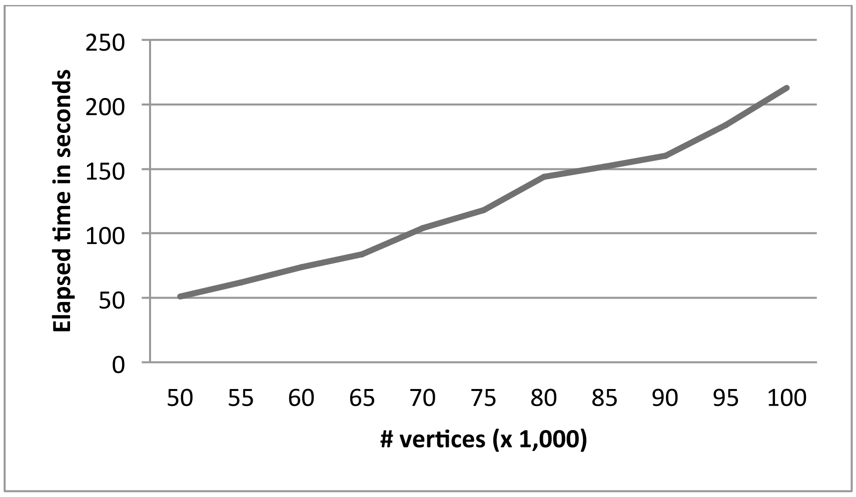

Fig. 6. Time spent to identify frequent subgraphs patterns

Table 4 shows the impact of increasing values of minimum support in the performance of the ADI-Minebio algorithm. We used the values of 90, 100 and 110 as the minimum support for a graph database containing 250,000 vertices. The increase in the minimum support generated a smaller number of frequent subgraphs. Therefore, a challenge faced by the ADI-Minebio algorithm was to detect this fact and to spend less time to produce the list of frequent subgraphs. As can be noted, the proposed algorithm greatly overcame this challenge, since it provided a reduction of 61% in the elapsed time for a reduction of only 31% in the number of frequent patterns.

Table 4. Performance results for increasing values of minimum support.

Minimum Support	Elapsed Time (s)	# Patterns
90	69	5,390
100	55	4,916
110	27	3,704

5. RELATED WORK

There are different types of algorithms for mining graphs. They are based on two approaches: based on Apriori [Agrawal and Srikant 1994] and based on pattern growth methods [Pei 2002]. While AGM [Inokuchi et al. 2000] and FSG [Kuramochi and Karypis 2001] are examples of algorithms based on Apriori, the algorithms gSpan [Han and Yan 2002], Mofa [Borgelt and Berthold 2002], CloseGraph [Han and Yan 2003], SPIN [Huan et al. 2004], and Metagraph [Deepti et al. 2008] use techniques based on pattern growth methods. Furthermore mining graphs are usually applied in different fields of knowledge. Thus, mining graphs algorithms may be tailored for specific areas of knowledge due to the characteristics of their data, as is the case reported in [Chen et al. 2009], [Lam and Chan 2008], [Lam and Chan 2008a], [Heydari et al. 2009] and [Song and Chen 2006]. Another graph-mining algorithms more related to our proposals are described in more details below.

FOGGER [Zeng, 2009] is a graph-mining algorithm based on the gSpan algorithm that addresses specifically the problem "frequent connected graph generator mining". It tackles this problem using an extended ADI data structure, called ADI++, for quickly checking subgraph isomorphism. The FOGGER also uses two pruning techniques to avoid branches of the DFS code enumeration tree and showed to be very efficient and scalable for increasing graph data volumes. On the other hand, our proposals address the complete set of frequent subgraphs, which is not tackled by the graph generator FOGGER.

The approach reported in [Wang et al. 2004] deals with a set of undirected graphs and works separately with each undirected graph. On the other hand, our approaches consider a single directed graph containing all subgraphs. We compute the frequencies of edges occurrences when the database is traversed to build the ADI-bio structure, thus facilitating the computation of the subgraphs frequency. In the approach of Wang et al. (2004), these frequencies are found in the ADI-Mine algorithm, after ADI structure is built.

Zou and Holder (2010) also investigate the issue of frequent subgraph mining. In the same way of our proposals, they consider a dataset as a single graph, but considering large graphs that are too large to whole fit in main memory. For this purpose, sampling approaches are employed and the sampling method called "random areas selection sampling" provided the best results. On the other hand, the single graph addressed by our proposals is manipulated by an optimized ADI data structure, called ADI-bio, to efficiently search for patterns of treatments and side effects in medical applications.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we focused on mining graphs in the biomedical domain, considering information related to treatments of diseases and side effects caused by these treatments. We introduced two main contributions, as described as follows.

- We proposed *the ADI-bio structure*, which is specially organized to index data from a graph database with information of patients of a disease.
- We also proposed *the ADI-Minebio algorithm*, which searches for frequent subgraphs in the graph database organized according to the ADI-bio structure.

The advantages of our proposals were investigated through performance tests considering synthetic data from the Sickle Cell Anemia. In the tests, we investigated three different issues: (i) the impact of increasing number of vertices and edges in the performance for building the ADI-bio's data structure; (ii) the impact of increasing number of vertices in the performance of the ADI-Minebio algorithm; and (iii) how the minimum support impairs the performance of the ADI-Minebio algorithm. For an increasing number of vertices, which ranged from 1,000 up to 1,000,000 vertices, the performance results showed that the ADI-Minebio algorithm provided good results, which were slightly superior to a linear growth. The results also demonstrated that the ADI-Minebio algorithm provided a great reduction in the time spent for detecting frequent subgraphs as the minimum support increased.

We plan to apply the ADI-Minebio algorithm to a database containing real data from the biomedical domain. We also plan to include information about the number of patients enrolled in the experiment as a quantitative aspect of the algorithm, as well as to extend the proposed algorithm to consider this aspect.

Acknowledgments. This work has been supported by the following Brazilian research agencies: FAPESP, CNPq, CAPES and FINEP.

REFERENCES

AGRAWALL, RAKESH AND SRIKANT, RAMAKRISHNAN. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th Very Large Data Bases Conference*. Santiago, Chile, pp. 487-499, 1994.

BORGELT, CHRISTIAN AND BERTHOLD, MICHAEL. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining.* Maebashi City, Japan, pp. 51-58, 2002.

CHEN, HUAJUN, DING, LI, WU, ZHAOHUI, YU, TONG, DHANAPALAN, LAVANYA AND CHEN, JAKE Y. Semantic Web for Integrated Network Analysis in Biomedicine. *Briefings in Bioinformatics* 10 (2): 177-192, 2009.

Cook, Diane J. and Holder, Lawrence B. *Mining Graph Data*. John Wiley, 2007.

DEEPTI, GAUR, ADITYA, SHASTRI AND RANJIT, BISWAS. Metagraph-Based Substructure Pattern Mining. In *Proceedings of the 2008 International Conference on Advanced Computer Theory and Engineering*. Washington, DC, USA, pp. 865-869, 2008.

HAN, JIAWEI AND YAN, XIFENG. Gspan: Graph-Based Substructure Pattern Mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining.* Maebashi City, Japan, pp. 721-724, 2002.

HAN, JIAWEI AND YAN, XIFENG. Closegraph: Mining Closed Frequent Graph Patterns. In *Proceedings of the 2003 Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, pp. 286-295, 2003.

HAN, JIAWEI AND KAMBER, MICHELINE. Data Mining: Concepts and Techniques. Elsevier, 2nd Ed., 2006.

HEYDARI, MEHDI, HELAL, RAED ALI AND GHAUTH, KHAIRIL IMRAN. A Graph-Based Web Usage Mining Method Considering Client Side Data. In *Proceedings of the 2009 International Conference on Electrical Engineering and Informatics*. Selangor, Malaysia, pp. 147-153, 2009.

HUAN, JUN, WANG, WEI, PRINS, JAN AND YANG, JIONG. SPIN: Mining Maximal Frequent Subgraphs from Graph Databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Washington, DC, USA, pp. 581-586, 2004.

INOKUCHI, AKIHIRO, WASHIO, TAKASHI AND MOTODA, HIROSHI. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *Proceedings of the 2000 Principles and Practice of Knowledge Discovery in* Databases. Lyon, France, pp. 13-23, 2000.

KURAMOCHI, MICHIHIRO AND KARYPIS, GEORGE. Frequent Subgraph Discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*. Washington, DC, USA, pp. 313-320, 2001.

LAM, WINNIE AND CHAN, KEITH. A Graph Mining Algorithm for Classifying Chemical Compounds. In *Proceedings of the 2008 IEEE International Conference on Bioinformatics and Biomedicine*. Washington, DC, USA, pp. 321-324, 2008.

LAM, WINNIE AND CHAN, KEITH. Analyzing Web Layout Structures Using Graph Mining. In *Proceedings of the IEEE International Conference on Granular Computing*. Hangzhou, China, pp. 361-366, 2008.

MATOS, PABLO F., PINTO, ANA CRISTINA S., CIFERRI, CRISTINA D.A., VIEIRA, MARINA T.P., PARDO, THIAGO A.S. AND CIFERRI, RICARDO R. An Environment for Data Analysis in Biomedical Domain: Information Extraction For Decision Support Systems. In *Proceedings of the 23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Cordoba, Spain, pp. 306-316, 2010.

PEI, JIAN. Pattern-Growth Methods for Frequent Pattern Mining. Ph.D. Thesis. Simon Fraser University, Burnaby, BC, Canada, 2002.

SONG, YONGLING AND CHEN, SU-SHING. Item Sets Based Graph Mining Algorithm and Application in Genetic Regulatory Networks. In *Proceedings of the IEEE International Conference on Granular Computing*. Atlanta, USA, pp. 337-340, 2006.

WANG, CHEN, WANG, WEI, PEI, JIAN, ZHU, YONGTAI AND SHI, BAILE. Scalable Mining of Large Disk-Based Graph Databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Washington, USA, pp. 316-325, 2004.

ZENG, ZHIPING, WANG, JIANYONG, ZHANG, JUN AND ZHOU, LIZHU. Fogger: an Algorithm for Graph Generator Discovery. In *Proceedings of the 2009 International Conference on Extending Database Technology*. Saint Petersburg, Russia, pp. 517-528, 2009.

ZOU, RUOYU AND HOLDER, LAWRENCE. Frequent Subgraph Mining on a Single Large Graph Using Sampling Techniques. In *Proceedings of the 8th Workshop on Mining and Learning with Graphs*. New York, NY, USA, pp. 171-178, 2010.