Structural Graph-Based Representations Used for Finding Hidden Patterns

Ivan Olmos[1], Jesus A. González[2]

[1] Universidad Autónoma de Puebla,
Av. San Claudio y 14 Sur, Ciudad Universitaria,
Puebla, México
ivanop_rkl@yahoo.com.mx
[2] Instituto Nacional de Astrofísica, Óptica y Electrónica,
Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, México
jagonzalez@inaoep.mx

Abstract. In graph-based data mining (GBDM) tasks, an accurate data representation is fundamental for finding hidden patterns. However, there does not exist a standard representation to describe structural data because of the specific domain characteristics. Then, different graph topologies could be used as data representation, which is a challenge for GBDM tools. In this paper we explore a methodology for discovering hidden patterns in domains where is used a graph based representation. Our methodology is divided in three phases: first, we propose a formal graph notation used to symbolizes our graphs; second, we perform the data mining phase using the SI-COBRA and the SUBDUE tools; finally, we show how could be interpreted the outputs of these tools. We performed a set of experiments in two different domains where our methodology was applied: the web log and the SAT domains. With these examples we show how it is possible to symbolize our graphs with our notation, and also perform the GBDM task with the selected tools.

1 Introduction

The main problem in data mining (DM) tasks consists of the extraction of useful knowledge. This task is performed discovering similarities between objects from real and theoretical domains [1]. For almost all cases, original data is transformed into a suitable representation, so that it can be processed. One of the most important challenges is to preserve its original information because the success in the DM task depends of this [2]. In order to achieve it, different data representations have been studied with the aim of finding the the most suitable for each domain [3, 4]. Unfortunately, each domain has its own data characteristics, so it is necessary to propose particular representations for each domain.

Graphs are a powerful and flexible knowledge representation used to model simple and complex structured domains [3]. The representation power and flexibility is the main advantage of why the graph-based representation model has been adopted by researchers in different areas such as machine learning and DM [3, 1].

Graph mining is a widespread studied problem, where several works have been developed, each of them with different objectives. For example, Subdue [3] and gSpan [2] are algorithms that finds hidden patterns in labeled graphs, but the first one implements a computationally-constrained beam search with the aim to reduce the computational cost and the second one explore any possible pattern with a list-code representation. There are algorithms capable to find specific patterns in labeled graphs, based on the subgraph isomorphism problem (SIP), such as SI-COBRA [9]. Some algorithms impose topological restrictions on the input graphs [5]. There are other graph projects such as Ullman [6], VF2 [7] and Nauty [8] that are not able to work with labeled graphs, because many of them are oriented only to solve mathematical problems.

On the other hand, in these researchers have been proposed different graph notations. The most widely used graph notation for graphs is $G = (V, E)$, where V is a set (not empty) of vertices and E is a set of edges, $E \subseteq V \times V$. However, this notation is not suitable for DM, because data information is represented through labels that are attached to the vertices and edges. In works oriented to GBDM, appears other notations where labels are represented in different ways. For example, in gSpan a graph is represented by a 4-tuple $G = (V, E, L, l)$, where L is a set of labels and l is a assigning function of labels. On the other hand, Bunke and Jiang [14] proposed a similar notation, where $G = (V, E, \alpha, \beta)$, but the assigning functions of vertices and edges labels are separated in two functions, α and β.

Based on the above mentioned, in this work we explored a standard methodology that could be applied in data mining tasks. Our methodology is based on three phases. We first used a flexible notation for representing labeled/unlabeled graphs, that help us to formalize our graph representations. Second, we proposed to use data mining tools capable to work with labeled graphs (the SI-COBRA [9] and the SUBDUE [3] tools) without topological restrictions, which are capable to discover or search hidden patterns in graphs. And finally, an interpretation phase, which describes the way in which the algorithms outputs should be interpreted.

With the aim to show the flexibility of our methodology, we performed a set of experiments in two different domains: the web log and the SAT domains. The web log domain was used to discover hidden patterns that represents user's behavior in a web site. This means that at the beginning of the task we did not know the structure of the patterns to be found. On the other hand, we were interested in finding if specific structures exists in SAT instances that represents solutions of the problem..

The paper has the following structure. Section 2 gives the basic graph notation of our methodology. In section 3 are introducing the graph based data mining tools used in this work. The graph-based representation proposed for the web log and the SAT domains are introduced in Section 4. In Conclusion we discuss obtained results and impact of our work.

2 Graph Notation

As we mentioned before, graphs have been used in several research areas. Different authors define a graph with some variations, according to their requirements. Based on ideas of previous works [2, 14], we propose to use a formal graph notation capable to represent labeled/unlabeled graphs with a 6-tuple $G = (V, E, L_V, L_E, \alpha, \beta)$, where:

- $V = \{v_i | i = 1, \ldots, n\}$ is the finite set of vertices, $V \neq \emptyset$, and $n = \#vertices$ in the graph
- $E \subseteq V \times V$ is the finite set of edges, $E \subseteq \{e = \{v_i, v_j\} | v_i, v_j \in V, 1 \leq i, j \leq n\}$
- L_V is a set of vertex labels
- L_E is a set of edge labels
- $\alpha : V \rightarrow L_V$ is a function assigning labels to the vertices
- $\beta : E \rightarrow L_E$ is a function assigning labels to the edges

It is important to stress that this graph notation is very flexible, because it is possible to represent any graph topology with only three simple steps:

1. The indexing phase, where is associated an unique key or identification to any vertex and edge belonging to the graph. This step is performed with an arbitrary enumeration, represented by a numerical subscript associated to each vertex and edge.
2. Establish the set of vertices and edges labels, represented by L_V and L_E. These sets could be defined by extension or generalization.
3. Define the labels functions. In this step, we establish an association between elements of the graphs (vertices and edges) with their respective labels. This association is represented by the functions α and β for vertices and edges respectability.

Note that indices attached in the indexing phase must not be used as identifier because they do not represent data information, despite these values are unique for each vertex and edge.

Before to introducing examples were we used this notation, we present some important graph topologies. There exist four well known graph-topologies used for data representations: chain (Fig. 1.a), wheel or ring (Fig. 1.b), tree (Fig. 1.c) and star (Fig. 1.d). Based on these topologies, it is possible to construct new hybrid topologies, such as a backbone (Fig. 2.a), a backbone-tree (Fig. 2.b), a wheel-star (Fig. 2.c), a semi-wheel star with trees (Fig. 2.d), a wheel-star with trees (Fig. 2.e), and a fully connected graph (Fig. 2.f).

Based on these topologies, we introduce some examples where is used our notation. For the sake of simplicity, we assume that the input data come from of the most common data structure: a table, see Fig. 3.

First, we define what a table is. Let $\mathbf{A} = \{A_1, \ldots, A_m\}$ be a set of "m" attributes (an attribute is a descriptive property or characteristic of an object), where the domain of $A_x \in \mathbf{A}$ is denoted by $D(A_x)$. A table $\Re$, defined over $\mathbf{A}$ is a set of rows called registers $\wp$ where $\Re = \{\wp : \wp =< l_1, \ldots, l_m >\}$, and $\forall l_i : l_i \in D(A_i), 0 < i \leq m$. We consider that all registers $\wp \in \Re$ are indexed

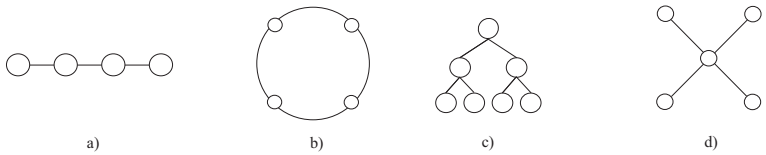

Fig. 1. Four Basic Graph Topologies: a) Chain, b) Wheel or Ring, c) Tree, and d) Star

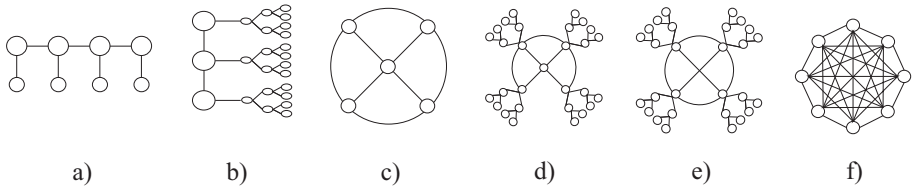

Fig. 2. Different Hybrid Topologies: a) Backbone, b) Backbone-tree, c) Wheel - Star, d) Semi-Wheel-Star with Trees, e) Wheel-Star with Trees, and f) Fully Connected Graph

with a numerical value x, represented by $\wp_x$, with the aim to uniquely identify any register of the relation.

OUTLOOK	TEMP	HUMIDITY	WINDY	PLAY
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes

Fig. 3. Example of a Relational Data Table for the Enjoy Sport Domain.

Example 1: Chain topology. Consider the table $\Re = \{\wp_1, \ldots, \wp_7\}$ of Fig. 3, where $\mathbf{A} = \{$Outlook, Temperature, Humidity, Windy, Play$\}$ and $D(Outlook) = \{sunny, overcast, rainy\}$, $D(Temperature) = \{hot, mild, cool\}$, $D(Humidity) = \{TRUE,\ FALSE\}$, and $D(Play) = \{no, yes\}$. If we would like to represent each $\wp_x \in \Re$ with a chain topology ($\wp_x =< l_{x,1}, \ldots, l_{x,m} >$, $l_{x,j} \in D(A_j)$, $1 \leq j \leq m$), then we generate a set $G = \{G_1, \ldots, G_7\}$, where $\forall \wp_x \in \Re : \wp_x$ is represented by G_x. As an example, the first register "sunny, hot, high, FALSE, no" is represented by $G_1 = (V_1, E_1, L_{V_1}, L_{E_1}, \alpha_1, \beta_1)$ such as is shown in Fig. 4 a), where vertices and edges are indexing from left to right (this indexing could be performed in a different way). Then, $V_1 = \{v_j | j = 0, \ldots, 5\}$ and $E_1 = \{e = \{v_j, v_{j+1}\} : 0 \leq j \leq 4\}$.

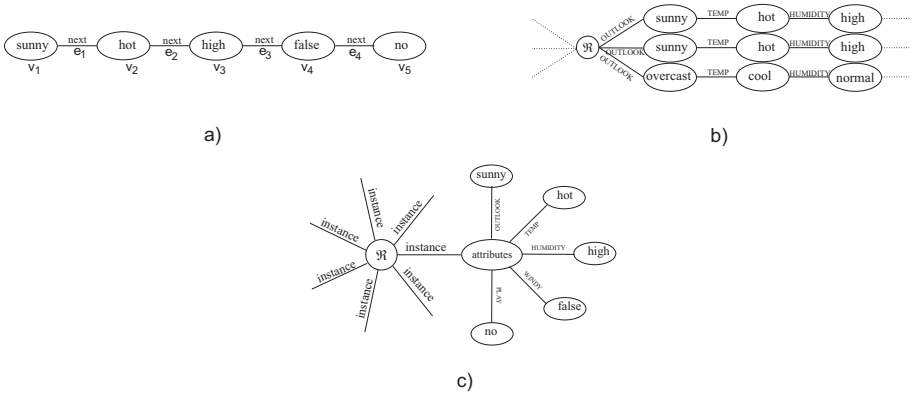

Fig. 4. Example of a Reduction from a Table: a) Chain Topology; b)Star Topology; c) Hybrid Representation.

After that, L_{V_1} and L_{E_1} are defined. In our case, $L_{V_1} = \{sunny, hot, high, FALSE, no\}$, whereas $L_{E_1} = \{next\}$ (but it is possible to select any other label). Finally, we establish the association between vertices / edges and their labels such as is shown in Fig. 4 a). Then, $\alpha_1 = \{\alpha_1(v_1) = sunny, \alpha_1(v_2) = hot, \alpha_1(v_3) = high, \alpha_1(v_4) = FALSE, \alpha_1(v_5) = no\}$ and $\beta_1 =: E_1 \rightarrow L_{E_1}$, where $\beta_1(v_j, v_{j+1}) = A_{j+1}, 0 \leq j < m$.

Example 2: Star topology An alternative way for representing $\Re$ is through a graph with a star topology (Fig. 1 d. shows the basic geometric form of a star). Consider that we would like to represent our graph such as is shown in Fig. 4 b). The idea consists of representing with a central node the name of the relation, and its registers with branches based on chains. Evidently, the formal notation of each branch is easy to build based on idea of Example 1. Therefore, the full graph is building with the union of the chains plus the central node with its respective edges. Formally, let $\Re$ be a relation with n registers and m attributes A $= \{A_1, \ldots, A_m\}$. Let $G_x = (V_x, E_x, L_{V_x}, L_{E_x}, \alpha_x, \beta_x)$ be a graph representation for the $x - th$ register $\wp_x \in \Re$, where:

- $V_x = \{v_j | j = 1, \ldots, m\}$
- $E_x = \{e = \{v_j, v_{j+1}\} : 1 \leq j \leq m-1\}$
- $L_{V_x} = D(A_1) \cup \cdots \cup D(A_m)$
- $L_{E_x} = \{A_1, \ldots, A_m\}$
- $\alpha_x : V_x \rightarrow L_{V_x}$, where $\alpha_x(v_j) = l_{x,j}, 1 \leq j \leq m$
- $\beta_x : E_x \rightarrow L_{E_x}$, where $\beta_x(v_j, v_{j+1}) = A_{j+1}, 1 \leq j < m$

Let $G_{set} = \{G_1, \ldots, G_n\}$ be a set of graphs, where $G_x \in G_{set}$ is the graph-based representation of $\wp_x \in \Re$. A graph-based star representation of $\Re$ is a graph $G = (V, E, L_V, L_E, \alpha, \beta)$, where:

- $V = \{v_0\} \cup \bigcup_{x:G_x \in G_{set}} V_x$

- $E = \bigcup_{x:G_x \in G_{set}} E_x \cup \{e_i : 0 < i \leq n\}$, where $e_i = \{v_0, v_1^i\}$, v_1^i is the vertex v_1 of G_i, $G_i \in G_{set}$.
- $L_V = D(A_1) \cup \cdots \cup D(A_m) \cup \{\Re\}$
- $L_E = \{A_1, \ldots, A_m\}$
- $\alpha = \bigcup_{x:G_x \in G_{set}} \alpha_x \cup \{\alpha(v_0) = \Re\}$
- $\beta = \bigcup_{x:G_x \in G_{set}} \beta_x \cup \{\beta(\{v_0, u\}) = A_1, \alpha(v_0) = \Re, u \in V, u \neq v_0\}$

Example 3: Hybrid Star Topology Based on Example 2, it is possible to derive a new and more complex structure, where each branch is also a star, which represents registers with their respective attributes. This idea is shown in Fig. 4 c), where the first register of Fig. 3 is represented. Based on this idea, the formal representation of this topology is defined as follows. Let $\Re$ be a relation with n registers and m attributes $\mathbf{A} = \{A_1, \ldots, A_m\}$, where each register $\wp_x \in \Re$ $(1 \leq x \leq n)$ is represented by $\wp_x =< l_{(x,1)}, \ldots, l_{(x,m)} >$. A graph-based hybrid star representation of $\Re$ is a graph $G = (V, E, L_V, L_E, \alpha, \beta)$, where:

- $V = V_{Inst} \cup V_{Attrib} \cup \{v_\Re\}$, where $V_{Inst} = \{v_{(i,j)} : 1 \leq i \leq n, 1 \leq j \leq m\}$, $V_{Attrib} = \{v_1, \ldots, v_n\}$
- $E = E_1 \cup \cdots \cup E_n \cup E'$, where:
  - $\forall E_i \in E, 1 \leq i \leq n : E_i = \{\{v_i, v_{(i,j)}\} : v_{(i,j)} \in V_{Inst}, v_i \in V_{Attrib}, 1 \leq j \leq m\}$
  - $E' = \{\{v_\Re, v_i\} : v_\Re \in V, v_i \in V_{Attrib}, 1 \leq i \leq n\}$
- $L_V = D(A_1) \cup \cdots \cup D(A_m) \cup \{\Re, attributes\}$
- $L_E = \{A_1, \ldots, A_m\} \cup \{instance\}$
- $\alpha : V \rightarrow L_V$, where:
  - $\alpha(v_{(i,j)}) = l_{(i,j)}$ if $v_{(i,j)} \in V_{Inst}, 1 \leq i \leq n, 1 \leq j \leq m$
  - $\alpha(v_i) = attributes$ if $v_i \in V_{Attrib}, 1 \leq i \leq n$
  - $\alpha(v_\Re) = \Re$
- $\beta : E \rightarrow L_E$, where:
  - $\beta(\{v_i, v_{(i,j)}\}) = A_j, 1 \leq i \leq n, 1 \leq j \leq m, v_i \in V_{Attrib}, v_{(i,j)} \in V_{Inst}$
  - $\beta(\{v_\Re, v_i\}) = instance, v_\Re \in V, v_i \in V_{Attrib}$

With these examples, we illustrated the flexibility of the proposed notation for representing different graph topologies in a formal way. In areas such as GBDM this notation could be used to characterize patterns in a formal way.

Based on our notation, the next step in our methodology is the GBDM phase. Since we are considering graphs without topological restrictions, then we used the SI-COBRA and the SUBDUE data mining tools that support this type of graphs. A brief description of these algorithms is introduced in Section 3.

3 Two Algorithms to Work with Labeled Graphs

In this section we introduce two algorithms to work with labeled graphs and solve the SI and the GBDM problems. The first one is the SI-COBRA algorithm [9] that we use to solve the subgraph isomorphism problem with directed-undirected, labeled-unlabeled graphs. This algorithm is useful for problems where

it is necessary to test if there exists a substructure in a set of data (represented by graph G) with an equal topology with respect to a specific structure to search for (represented by graph G'). The second one is the SUBDUE algorithm [3], which is capable to find common structures in a set of input graphs. Subdue is able to work with directed-undirected, labeled-unlabeled graphs. This algorithm is useful in tasks where we work with structural domains and it is necessary to perform a data mining process using a graph based representation.

3.1 The SI-COBRA Algorithm

The SI-COBRA algorithm [9] finds the instances of a graph G' in a graph G, where a linear sequence of codes is used to represent the graphs. This algorithm starts by building a graph model of G', represented by a linear sequences of codes, called DFC'. The DFC' model is a sorted sequence of codes, where the degree of the vertices, the numbers of instances of each label and the lexicographic order based on $L_V EV$.

Based on DFC' (the model of G'), the matching process tries to build a linear sequence DFC of G, using a step by step expansion without candidate's generation based on backtracking, punning phases and a width-depth search. The size of DFC must be the same of DFC' and the corresponding codes entries must also be identical. If DFC could be generated, then a subgraph isomorphic to G' is found. This idea is shown in Fig. 5. This algorithm is complete, and it is possible to find all the instances of a graph G' in a graph G, just by leaving the algorithm run over all the possibilities.

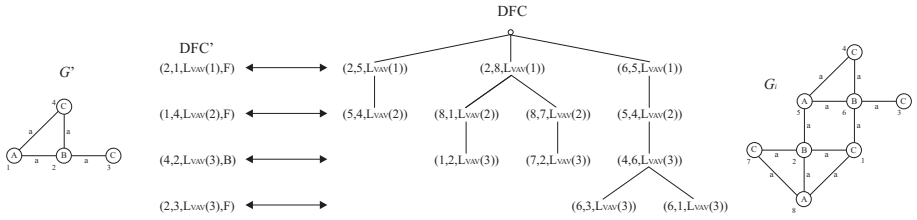

Fig. 5. Example of a Width-Depth Search using L_{VEV} Codes

In this section, we showed a brief description of the SI-COBRA algorithm. In section 3.2 we present a description of the SUBDUE algorithm, which is capable to solve the GBDM problem.

3.2 The SUBDUE Algorithm

Subdue [3] is a Data Mining tool that achieves the mining task using a model evaluation method called "Minimum Encoding" that is a technique derived from the minimum description length principle [10] and chooses as best substructures

those that minimize the description length metric that is the length in number of bits of the graph representation. The number of bits is calculated based on the size of the adjacency matrix representation of the graph. According to this, the best substructure is the one that minimizes I(S) + I(G|S), where I(S) is the number of bits required to describe substructure S, and I(G|S) is the number of bits required to describe graph G after being compressed by substructure S.

The main discovery algorithm is a computationally constrained beam search. The algorithm begins with the substructures matching a single vertex in the graph. These substructures are then expanded by a vertex and an edge or by an edge in all the possible ways according to the input graph and this process is repeated with the generated substructures. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit. The best substructure found by Subdue is used to compress the input graph, which can then be input to another iteration of Subdue. After several iterations, Subdue builds a hierarchical description of the input data where later substructures are defined in terms of substructures discovered on previous iterations.

Fig. 6 shows a simple example of Subdue's operation. Subdue finds four instances of the triangle-on-square substructure in the geometric figure. The graph representation used to describe the substructure, as well as part of the input graph, is shown in the middle.

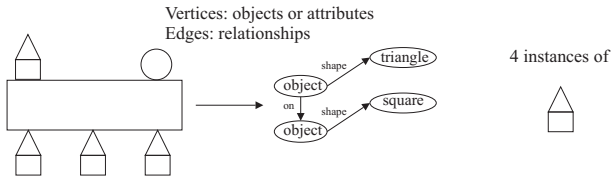

Fig. 6. Subdue's Example.

4 Graph-Based Representations for Two Real World Domains

In sections 2 and 3 we introduced a formal graph notation and describe data mining tools that could be used as part of our methodology for discovering hidden patterns. In this section we explain with examples how our methodology is applied to two real world domains.

4.1 Graph-Based Representation for the SAT Problem

The satisfactibility problem, or SAT for short, has been studied in different research areas where it is necessary to find or prove non-contradictory states. The

SAT problem consists of a set of m boolean variables and a set of n clauses in conjunctive normal form (CFN), where the goal of the problem is to determine whether there exists a true assignment to the variables such that the formula becomes satisfied. One way to solve instances that come from a SAT problem consists of transforming the SAT instance into a graph and find a solution through the SIP. Traditionally this process is performed with unlabeled graphs, where is search for a clique and $\#vertices = \#clauses$ [13]. However, this representation is not the best for the SIP because during the search process combinations that do not represent any possible solutions are explored. With the aim to reduce this problem, in this work we propose to use labeled graphs. We start with a brief introduction to the SAT problem and then we describe its graph based representation.

Let $U = \{u_1, \ldots, u_m\}$ be a set of boolean variables. A clause over U is a set of literals over U (u and $\bar{u}$ are literals if u_i is a variable over U), where the literals are joined by disjunctions. For example, $\{u_1, \bar{u_2}, u_3\}$ is a clause with 3 literals. A clause C is satisfied by a truth assignment if and only if $\exists u_x \in C : t(u_x) = T$, $t : U \rightarrow \{true - T, false - F\}$. A set of clauses $\hat{C} = \{C_1, \ldots, C_n\}$ over U is satisfiable if and only if $\forall C_x \in \hat{C} : \exists u \in C_x : t(u) = T$, that is, there exists some truth assignment of U that satisfies all clauses in $\hat{C}$. The problem to find a truth assignment that satisfies $\hat{C}$ over U is known as *satisfiability* or SAT, for short.

As an example, consider that there exist a set of clauses $\hat{C} = \{C_1, C_2, C_3\}$, $C_1 = \{a, \bar{b}\}$, $C_2 = \{c, \bar{a}\}$, $C_3 = \{b, c\}$. Clearly, solutions of this examples are $\{a, c\}, \{a, c, b\}, \{\bar{b}, c\}$ and $\{\bar{b}, \bar{a}, c\}$. In these solutions we can see that they have 1, 2 or 3 literals. However, any SAT solution always include a literal per each clause. For example, the solution $\{a, c\}$ means that the literal c appears in C_2 and C_3. Therefore, every SAT solution with n clauses has n literals.

Because of this, it is possible to represent $\hat{C}$ with a graph $\hat{G}$, where each literal is represented by a vertex in the graph, all vertices that come from the same clause have an unique label, and there exists an edge between each pair of vertices if they come from different clauses and the corresponding literals (associated to the vertices) are not the same or if they are the same, then both of them are negated or non-negated (**restriction 1**). Based on this idea, clauses $\hat{C} = \{C_1, C_2, C_3\}$ of our example are represented by the graph of Fig. 7 a), where each possible solution is represented by a clique. As an example, the solution $\{a, c\}$ is represented by v_1, v_3 and v_6, which are liked by edges $\{v_1, v_3\}, \{v_3, v_6\}$ and $\{v_6, v_1\}$. Then, it is possible to represent a SAT problem with two graphs $\hat{G}$ and G', where $\hat{G}$ represents the clauses of the SAT problem, and G' the structure of any valid solution. This idea is shown in Fig. 7 b) and c).

This idea could be generalized for any k-SAT problem. Let $(U, \hat{C})$ be an instance of k-SAT. If $\hat{C}$ over U is satisfiable, this means that $\exists u_x \in C : t(u_x) = T$, $\forall C_x \in \hat{C}$. That is, $\forall C_x$, $C_y \in \hat{C}$, if $u \in C_x$ and $w \in C_y$, are literals solutions of $\hat{C}$, then $t(u) = t(w) = T$. Considering the above mentioned, we define a graph representation of $\hat{C}$ as follows.

Let $\hat{C} = \{C_1, \ldots, C_n\}$ be a set of clauses over $U = \{u_1, \ldots, u_m\}$. A graph-based representation of $\hat{C}$ is a graph $\hat{G} = (V, E, L_V, L_E, \alpha, \beta)$ where:

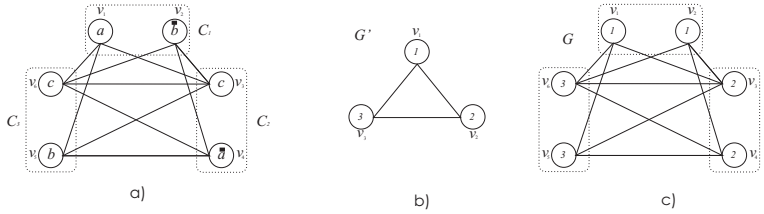

Fig. 7. Example of a Reduction from a 2-SAT Instance to a SI Instance.

- $V = \bigcup_{x:C_x \in \hat{C}} V_x$, where $V_x = \{v_i : f(x,u) = v_i,\ \forall u \in C_x\}$, $i = 1, \ldots, k$, $k = 2$ for 2-SAT, $k = 3$ for 3-SAT and so on. $f(x,u)$ is a function where x represents the index of C_x and u is a literal in C_x. f maps each pair of inputs to a vertex v_i
- $E = \{\{v_i, v_j\} : v_i, v_j \in V$ and
  a) if $v_i \in V_x$, then $v_j \notin V_x$
  b) if $f(x,w) = v_i$ and $f(y,u) = v_j$, then $u \neq \bar{w}\}$
- $L_V = \{1, 2, 3, \ldots, n\}$, $|L_E| = 1$
- $\alpha : V \to L_V$, where: $\alpha(v_i) = x$ if $f(x,u) = v_i$
- $\beta : E \to L_E$

Note that each literal in each clause $C_x \in \hat{C}$ is mapped to one vertex in $\hat{G}$. Also, each clause C_x derives a set of vertices V_x, where each vertex $v \in V_x$ has the label x and V comes from the union of the sets V_x. On the other hand, E is built based on restriction 1.

Now, we show how a SAT-problem can be reduced to an instance (G', G) of the SI-problem. First, we define the graph $G = \hat{G}$ and $G' = (V', E', L'_V, L'_E, \alpha', \beta')$, where: $V' = \{v'_i : i = 1, \ldots, n\}$ (n is the number of clauses), $E' = \{\{v'_i, v'_j\} : i \neq j\}$, $L'_V = L_V$, $L'_E = L_E$, $\alpha' : V' \to L'_V$ where $\alpha(v_i) = i$ and $\beta' : E' \to L'_E$. Note that G' is a fully connected graph with n vertices, because we need to find a subgraph where there might exist an edge between each pair of vertices and a vertex represents one literal for each clause. Therefore, G' represents a possible solution of $(U, \hat{C})$. Then, we build an instance (G', G) of the SI-Problem where we want to decide if $\exists S : S$ subgraph of G and S isomorphic of G'. If (G', G) is a yes-instance of the SI-Problem, then S satisfies $\hat{C}$ through the literals represented by the vertices in G'. So, $\hat{C}$ is satisfiable and the SAT-instance $(U, \hat{C})$ is also a yes-instance.

Since we need to solve the SIP on labeled graphs, we propose to use the SI-COBRA algorithm. The first step is to represent the instance (G', G) with a text file where some lines represent vertices (the syntax of these lines is "v id label", where "id" is the index value and "label" is its label) and others represents edges (the syntax is "e v1 v2 label", where "v1" and "v2" are the vertices that define the edge and "label" is its label). For example, clauses of Fig. 7 are represented in the SI-COBRA input format as in shown in Fig. 8. Note that all edge's labels are represented with "1" but could be selected any

other value. After running SI-COBRA, we found four different subgraphs that are isomorphic to G', which are shown in Fig. 8 b). The interpretation of these results is easy: consider "Graph 1" of Fig. 8 b). Since we know that $f(1, a) = v_1, f(1, \bar{b}) = v_2, f(2, c) = v_3, f(2, \bar{a}) = v_4, f(3, b) = v_5$ and $f(3, c) = v_6$ (see the example of Fig. 7), then we conclude that v_1 is associated to literal a, v_3 with c and v_5 with b. Then, we conclude that $\hat{C}$ is satisfiable with literals a, b and c.

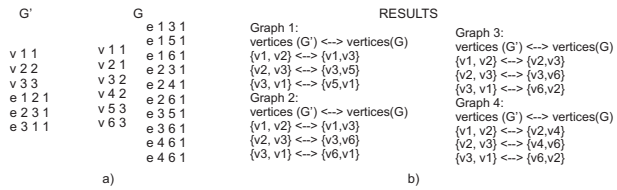

Fig. 8. Example of an Input and Output of the SI-COBRA Algorithm based on 7.

4.2 Web-Log Domain

Web sites design and administration is becoming a complex task as a consequence of the large amount of data that is kept in them [11]. An accurate design considers the users access to the web site, with the aim to improve the user satisfaction at web site navigation time. This information is available in the web sites servers because all received visitors are recorded in the web access log [11]. This file contains information divided in a set of entries, where each entry has the fields: IP (remote host name IP), login (remote login name), request (request line exactly as it came from the client) and others. Fig. 9 shows three lines of a web log file.

```
213.98.220.123  - - [07/Apr/2002:08:06:35  -0600] "GET /~ jaime/imagenes/neutral1. jpg HTTP/1.1"
200 1346 "http:// cseg.inaoep.mx/~jaime/archivos_too/Activides_too.html" " Mozilla/4.0
(compatible; MSIE 5.0; Windows 98;   DigExt)"
213.98.220.123  - - [07/Apr/2002:08:07:53  -0600] "GET /~ jaime/archivos_too/UML/tarea3. pdf
HTTP/1.1" 200 9271 "http:// cseg.inaoep.mx/~jaime/archivos_too/Activides_too.html" " Mozilla/4.0
(compatible; MSIE 5.0; Windows 98;   DigExt)"
166.114.204.156  - - [07/Apr/2002:09:10:29  -0600] "GET
/~jaime/archivos_too/UML/UML_ tutoriales .html HTTP/1.1" 200 7473
"http://www. google.com/search?q=UML+ Tutoriales&ie=UTF8&oe=UTF8&hl=es&lr=lang_es" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1)"
```

Fig. 9. Three Lines of a Web Log File.

The first step for transforming a web log L into a graph-based representation is to remove useless data from the web log file. This phase is called data cleaning, and commonly entries that represent images, audio files, scripts, system messages, and filed requests are removed [12].

The second step consist of designing the way in which a web log could be represented through graphs. In this step is take into account the patterns to search

for. For example, if we need to find the most visited resources in the web site, then the most important files to use are the host "IP" address and the "Request" fields. With the aim to identify those resources, first we establish a possible visit pattern. One of the most common is to know the sequence of resources visited of the site. For this problem, we propose two graph representations: chains and stars. If we used a chain topology, each vertex could symbolizes full or partial paths. For the sake of simplicity, consider that each vertex represents a full access path and all of them are linked by edges (labeled with "follows") in a sequence. This idea is shown in Fig. 10 a). Based on our methodology, we can formalize this representation as follows: let $L = \{L_1, \ldots, L_s\}$ be the set of entries from a web log file grouped by their IP's, where $\forall L_x \in L : L_x = \{l_j : j \leq \#\text{entries in } L \text{ with the same IP}\}$, $l_j =< u_i, t_j, rs_x, [...] >$. A graph-based sequential representation of $L_x \in L$ is a graph $G_x = (V_x, E_x, L_{V_x}, L_{E_x}, \alpha_x, \beta_x)$, where:

- $V_x = \{v_i : i = 1, \ldots, |L_x|\}$
- $E_x = \{\{v_i, v_{i+1}\} : i = 1, \ldots, |L_x| - 1\}$
- $L_{V_x} = \{rs_j : rs_j \text{ is defined in } l_j \in L_x\}$
- $L_{E_x} = \{follows\}$
- $\alpha_x : V_x \rightarrow L_{V_x}$, where $\alpha(v_i) = rs_j$ if rs_j is defined in $l_i \in L_x$
- $\beta_x : E_x \rightarrow L_{E_x}$, where $\forall\{v_i, v_{i+1}\} \in E_x : \beta(\{v_i, v_{i+1}\}) =$follows

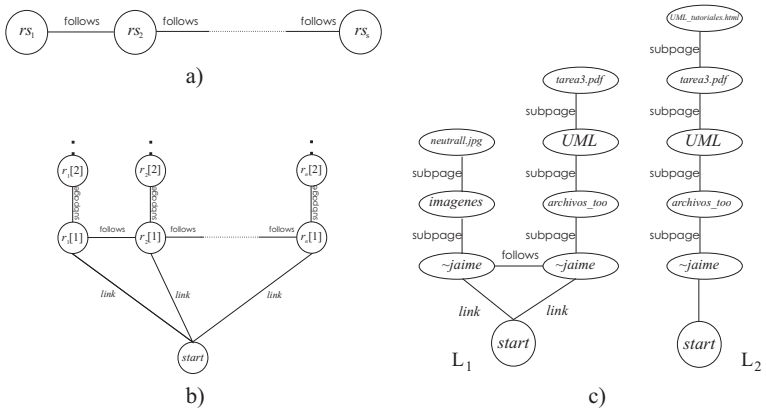

Fig. 10. Representations for a Web Log File.

This representation is oriented to find continuous access sequences but it does not alternate resources. In order to overcome this problem, we propose an alternative representation where the access path rs is decomposed in its hierarchical levels creating sequences $rs =< rs[1], rs[2], \ldots, rs[m] >$, where m is the depth of the path. As an example, resource $rs =$ "$/\ jaime/archivos\_too/UML/UML\_tutoriales.html$" is decomposed into $rs =< rs[1] =$ "$\ jaime$", $rs[2] =$ "$archivos\_too$", $rs[3] =$ "UML", $rs[4] =$ "$UML\_tutoriales.html$" $>$. Moreover, all path resources are

linked by a central node called "start", hence it is possible to find accessed resources in a non-sequential way. This idea is shown in Fig. 10 b). As an example, L_1 of Fig. 10 c) represents the first two lines of Fig. 9, because they have the same IP and L_2 represents the third line.

Formally, this representation is divided in two parts: the graph representation of a path resource divided in its levels, Representation 1, and the graph representation of a web log (similar to the Example 3), Representation 2.

Representation 1 of a Web log Let $rs_x = \backslash path_1 \backslash path_1 \backslash \ldots \backslash path_m$ be a resource represented by its WWW address path with m levels of depth that come from an entry $l_k =< u_i, t_j, rs_x, [...] >$ from a web log file. A graph-based representation of rs_x is a graph $G_x = (V_x, E_x, L_{V_x}, L_{E_x}, \alpha_x, \beta_x)$ where:

- $V_x = \{v_i : i = 1, \ldots, m\}$
- $E_x = \{\{v_i, v_{i+1}\} : i = 1, \ldots, m-1\}$
- $L_{V_x} = \{path_i : i = 1, \ldots, w\}$, where $w \leq m$
- $L_{E_x} = \{subpage\}$
- $\alpha_x : V_x \rightarrow L_{V_x}$, where $\alpha_x(v_i) = path_i$, $i = 1, \ldots, m$
- $\beta_x : E_x \rightarrow L_{E_x}$, where $\forall \{v_i, v_{i+1}\} \in E_x : \beta(\{v_i, v_{i+1}\}) = follows$, $i = 1, \ldots, m-1$

Representation 2 of a Web log Let $L = \{L_1, \ldots, L_s\}$ be a set of entries grouped by their IP's from a web log file, where $\forall L_k \in L : L_k = \{l_{k,j} =< u_i, t_j, rs_x, [...] >, j \leq w\}$ (w = #entries in L with the same IP). Let $G_{k,j} = (V_{k,j}, E_{k,j}, L_{V_{k,j}}, L_{E_{k,j}}, \alpha_{k,j}, \beta_{k,j})$ be the graph-based representation of resource rs_x recorded in $l_{k,j} \in L_k$, where its set of vertices is $V_{k,j} = \{v_1^{k,j}, \ldots, v_n^{k,j}\}$. A graph-based hierarchical representation of L_k is a graph $G_k = (V_k, E_k, L_{V_k}, L_{E_k}, \alpha_k, \beta_k)$, where:

- $V_k = \bigcup_{i=1,\ldots,w} V_{k,i} \cup \{v_0\}$
- $E_k = \bigcup_{i=1,\ldots,w} E_{k,i} \cup \{\{v_0, v_1^{k,i}\} : i = 1, \ldots, w\} \cup \{\{v_1^{k,i}, v_1^{k,i+1}\} : i = 1, \ldots, w-1\}$
- $L_{V_k} = \bigcup_{i=1,\ldots,t} L_{V_{k,i}} \cup \{start\}$
- $L_{E_k} = \{subpage, link, follows\}$
- $\alpha_k : V_k \rightarrow L_{V_k}$, where $\alpha_k = \bigcup_{i=1,\ldots,w} \alpha_{k,i} \cup \{\alpha_k(v_0) = start\}$
- $\beta_k : E_k \rightarrow L_{E_k}$, where $\bigcup_{i=1,\ldots,w} \beta_{k,i} \cup \{\beta_k(\{v_1^{k,i}, v_1^{k,i+1}\}) = follows, i = 1, \ldots, w-1\} \cup \{\beta_k(\{v_0, v_1^{k,i}\}) = link, i = 1, \ldots, w\}$

The next step in our methodology consist of finding the hidden patterns. With the aim to show this, we worked with a web log that come from of the computer science department at the INAOE web site. We used 11 different web log files with 25,000 to 300,000 records each one. In this problem we used the SUBDUE system as data mining tool, because we did not known any possible patterns a priori. The SUBDUE system reports the discovered patterns divided in iterations, where each pattern is reported based on the vertices and edges indices used in the input graphs, and the number of instances per each pattern is reported too. In the first iteration are discovered patterns based on the original input graphs, but in the next iterations the original input graphs are compressed [3].

We performed a set of experiments looking for the most visited pages. Some of these results are shown in Fig. 11. We found patterns with a variable length. As an example, in Fig. 11 a) represents a pattern with 6 vertices. Moreover, SUBDUE reports that this pattern appear 48 times. Based on this result, we conclude that these users first visited page "/univ/" and then visited pages "/univ/logo.html", "/univ/blanco.html" and so on. It is evident that this result is easy to understand. Therefore, this representation is suitable to find sequential patterns describing the path followed by users throughout the web site.

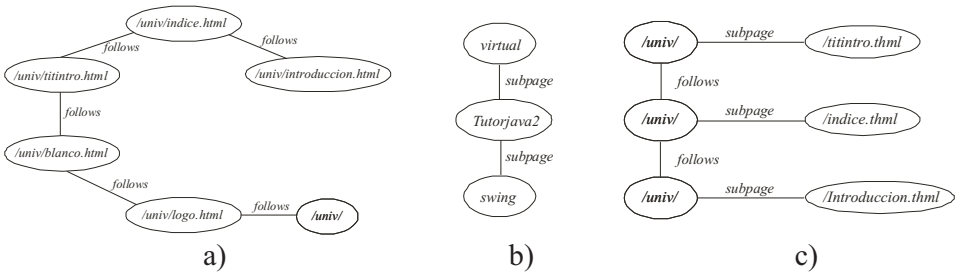

Fig. 11. Discovered Substructures in a Web Log File based on a Chain Topology.

However, if our interest is to find more complex access patterns including jumps between pages (as an example, a user could visit a web page from another page located in the same server using a hyperlink, but the new page might not keep the same path of the previous one), we use Representation 2. As an example, with this representation we found the graphs shown in Fig. 11 b) and c). Note that some edges are labeled as "follows" and others with the "subpage" label. This means that vertices linked with a "follows" edge have a higher hierarchy (these pages represent a home page) than vertices linked with a "subpage" edge (these pages do not represent a home page).

With these simple examples, we showed how our methodology help us to find hidden patterns in graphs. Moreover, we propose a notation that could be used to formalize graph-based representations without topological restrictions.

4.3 Conclusions

In this work we explored a GBDM methodology that could be used in many problems where it is not possible to use specialized tools with specific graph topologies. In our methodology we proposed to use a standard graph notation (useful for representing labeled graphs) and the SI-COBRA and the SUBDUE systems as data mining tools. Finally, we showed some examples where a graph-based representation is used in two domains: web log and SAT.

In this work we presented two different graph-based representations for the web log domain, with the aim to discover those most visited resources. In our experiments we found that the SUBDUE algorithm and our proposed representations were an accurate combination to find the most visited resources of the site, where the highest visit frequency was 120.

For the SAT problem, we proposed a transformation process that help us to find solutions of SAT instances based on the subgraph isomorphism problem, where labeled graphs and the SI-COBRA algorithm were used. In our results we can see that any valid solution is reported by the SI-COBRA algorithm. In future works, we will prove the correctness of this approach.

Finally, we want to stress the importance of selecting accurate tools for searching of finding patterns. In this sense and based on our experience, the SI-COBRA and SUBUDE systems are powerful tools for labeled graphs, which could be used in data mining task where a graph-based representation is suitable for our data.

References

1. Kuramochi, M.; Karypis, G. *An Efficient Algorithm for discovering Frequent Subgraphs.* Tech. Report. Dept. of Computing Science, University of Minnesota. June (2002).
2. Xifeng, Y. Jiawei H. *gSpan: Graph - Based Substructure Pattern Mining.* Technical Report. University of Illinois. (2002).
3. Cook, D. J.; Holder, L. B. *Substructure discovery using minimum description length and background knowledge.* Journal of Artificial Intelligence Research, 1:231-255, 1994.
4. Inokuchi, A and Washio, T Motoda. *Complete Mining of Frequent Pattern from Graphs: Mining Graph Data.* Machine Learning, Kluwer Academic Publishers. 321-354 (2003).
5. E. Luks. Isomoprhism of Bounded Valence can be tested in Polynomial Time. Journal of Computer and System Sciences 25 (1982) 42 65.
6. Ullman, J. R. *An Algorithm for subgraph Isomorphism.* Journal of the Association for Computing Machinery, Vol. 23, Issue 1, 31-42, 1976
7. L. P. Cordella; et. al. *An improved algorithm for matching large graphs.* Proceedings of the 3rd IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition.
8. D. M. Brendan. *Practical graph isomorphism.* Congressus Numerantium, 30:45/87.
9. Olmos, Ivan. Gonzalez, Jesus A., Osorio, Mauricio. *Subgraph Isomorphism Detection using a Code Based Representation.* Proceedings of the 18th International FLAIRS Conference, 2005.
10. Rissanen, J. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Company (1989).
11. Nanopoulos, A. Katsaros,; Manolopoulos, D. Y. *Effective Prediction of Web-user Accesses: A Data Mining Approach.* Proceeding WEBKDD Workshop, San Fransisco, CA. (2001).
12. Tanasa, D. and Trousse, B. *Advanced Data Preprocessing for Intersites Web Usage Mining.* IEEE Intelligent Systems. Vol. 19, Issue 2 (2004) 59 - 65.

13. Garey, M.R.; Johnson, D.S. and Stockmeyer, L. *Some Simplified NP-complete problems*. Annual ACM Symposium on Theory of Computing. Proceedings of the sixth annual ACM symposium on Theory of computing. Pp. 47-63, 1974.
14. Bunke, H and Jiang, X. *Graph Matching and Similarity*. Teodorescu, H.-N., Mlynek, D., Kandel, A., Zimmermann, H.-J. (eds.): Intelligent Systems and Interfaces. Kluwer Academic, 2000.