Efficient Computation of the Degree of Belief in a Propositional Theory

Carlos Guillén[1,2], Guillermo De Ita[1], Aurelio López-López[2]

[1] Universidad Autónoma de Puebla,
[2] Instituto Nal. de Astrofísica Óptica y Electrónica, (INAOE), México
{cguillen,deita,allopez}@ccc.inaoep.mx

Abstract. We design a novel model-based approach for reasoning, considering the reasoning process as the computation of the degree of belief of an intelligent agent. We assume a Knowledge Base (KB) expressed by a two Conjunctive Form (2-CF) Σ. We represent Σ using an appropriate logical structure. Our representation of a 2-CF is highly expressive since it supports efficient reasoning, even in the cases where the formula-based representation does not allow to do it. Indeed, we can compute the degree of belief with new information (a literal or a binary clause) efficiently, providing an efficient scheme of inductive reasoning.
Exploiting this logical structure of a 2-CF, we propose a way to determine the relative value for every element in the KB which is an essential problem when new knowledge is aggregated and is necessary to maintain the consistency of the KB.
Keywords: #SAT Problem, Automated Reasoning, Degree of Belief, Belief Revision.

1. Introduction

A widely accepted framework for reasoning in intelligent systems is the knowledge-based system approach. The main idea is to represent the knowledge in an appropriate structure or language with a well defined semantics assigned to the sentences [9]

The AI goal of providing a logic model of human agent capability of reasoning, in the presence of incomplete and changing information, has proven to be very difficult to achieve. For example, to decide whether a KB Σ implies a sentence α (denoted as $K \models \alpha$) is a co-NP-Hard problem, even in the propositional case [9]. Many other forms of reasoning which have been developed to avoid, at least partly, these computational difficulties, have also shown to be hard to compute.

As it has been pointed out in [1, 4, 11, 13], an important problem to explore is the computational complexity of the reasoning methods. Although each method is clearly intractable in the general case, a precise determination of the complexity is left as an open issue. Furthermore, it is not clear under which restrictions such methods would become tractable.

In our case, we perform reasoning as the computation of the proportion of the number of models of Σ which remains after a new formula F is processed

(here as processing we mean calculating a degree of belief, updating or a doing belief revision). In order to clarify a frontier between efficient and exponential-time reasoning, we start working with a KB Σ in two conjunctive form (2-CF), and we consider queries also in 2-CF. We want to compute the degree of belief that an agent A has in a new piece of information F based on the conditional probability of F with respect to Σ and denoted as $P_{F|\Sigma}$, and calculated as the fraction of models of Σ that satisfy F.

Computing the degree of belief leads us to count the number of models of a formula Σ (denoted as #SAT(Σ)) which is a classical #P-complete problem. #SAT looks harder than SAT problem, for example, the 2-SAT problem (SAT restricted to consider 2CF's)is solved in linear time. However, the corresponding counting problem #2-SAT is #P-complete. #2-SAT remains #P-complete if we consider only monotone formulas or Horn formulas [14].

#SAT is of special concern to Artificial Intelligence (AI), and it has a direct relationship to Automated Theorem Proving, as well as to approximate reasoning [1, 3, 6, 10, 13]. An interesting area of research has been the identification of restricted cases for which #SAT can be solved efficiently.

We extend here some of the procedures presented in [2, 3, 7] for the #2-SAT problem and show how to apply them to the computation of the degree of belief. Furthermore, we show that our logical structure representation of the KB allows the efficient computation of the degree of belief for some classes of 2-CFs.

Given our logical structure representation of a 2-CF, we propose a way to determine the relative values for every element of the KB, which is an essential problem for performing reasoning in efficient way, for example in the area of update-belief revision, where is essential to know how to incorporate dynamically a single or a sequence of changes into an initial Knowledge Base.

2. Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x_i or a negated variable $\overline{x_i}$. As usual, for each $x_i \in X$, $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$.

A *clause* is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in I\!N$, a *k-clause* is a clause consisting of exactly k literals and, a *$(\leq k)$-clause* is a clause with at most k literals. A variable $x \in X$ *appears* in a clause c if either x or $\overline{x}$ is an element of c.

A *Conjunctive Form* (CF) F is a conjunction of clauses (we also consider a CF as a set of clauses). We say that F is a monotone CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses and, $(\leq k)$-CF denotes a CF containing clauses with at most k literals. A $k\mu$-CF is a formula in which no variable occurs more than k times. A $(k, j\mu)$-CF ($(\leq k, j\mu)$-CF) is a k-CF ($(\leq k)$-CF) such that each variable appears no more than j times.

We use $\upsilon(X)$ to express the set of variables involved in the object X, where X could be a literal, a clause or a Boolean formula. For instance, for the clause $c = \{\overline{x_1}, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$. And $Lit(F)$ is the set of literals which appear in

a CF F, i.e. if $X = v(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$. We also denote $\{1, 2, ..., n\}$ by $[\![n]\!]$.

An assignment s for F is a Boolean function $s : v(F) \to \{0, 1\}$. An *assignment* can be also considered as a set of non complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l *true* and $\overline{l}$ *false*. Considering a clause c and assignment s as a set of literals, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$, and if for all $l \in c$, $\overline{l} \in s$ then s falsifies c.

If $F_1 \subset F$ is a formula consisting of some clauses of F, then $v(F_1) \subset v(F)$, and an assignment over $v(F_1)$ is a *partial* assignment over $v(F)$. Assuming $n =\mid v(F) \mid$ and $n_1 =\mid v(F_1) \mid$, any assignment over $v(F_1)$ has 2^{n-n_1} extensions as assignments over $v(F)$.

Let F be a Boolean formula in Conjunctive Form (CF), F is *satisfied* by an assignment s if each clause in F is satisfied by s. F is *contradicted* by s if any clause in F is contradicted by s. A model of F is an assignment for $v(F)$ that satisfies F.

Given F a CF, the SAT problem consists of determining if F has a model. The #SAT problem consists of counting the number of models of F defined over $v(F)$. #2-SAT denotes #SAT for formulas in 2-CF. We also denote #SAT(F) by $\mu_{v(F)}(F)$ or just $\mu(F)$ when $v(F)$ is clear from the context.

Up to now, the maximum subclass of 2-CF where #2SAT is solved efficiently is for the class $(2, 2\mu)$-CF (formulas in 2-CF where each variable appears twice at most) [13, 14]. Here, we extend such class for considering the topological structure of the undirected graph induced by the formula.

3. Logical Structure for Representing a 2-CF

Computing the degree of belief of an agent is a hard task in general, even if the KB Σ of the agent is expressed by 2-CF's. We design a logical structure for representing a 2-CF in a way that computing the degree of belief in new formulas can be done efficiently, for a restricted class of Boolean formulas.

An initial Knowledge Base (KB) is given by a 2-CF Σ. Σ is represented by a signed constrained directed graph G_Σ. We compute, for every node $x \in G_\Sigma$ (variable of the formula Σ), a pair (α_x, β_x), where α_x indicates how many times the variable x is 'true' and β_x indicates the number of times that the variable x can take value 'false' in the set of models of Σ. We show in following sections that under this representation of Σ, we can compute efficiently the degree of belief in a literal or in a binary clause including variables not used previously in Σ, providing so an efficient inductive scheme of reasoning.

3.1. The Graph Representation of the Knowledge Base

Let Σ be a 2-CF, the *constrained graph* of Σ is the undirected graph $G_\Sigma = (V(\Sigma), E(\Sigma))$, with $V(\Sigma) = v(\Sigma)$ and $E(\Sigma) = \{\{v(x), v(y)\} : \{x, y\} \in \Sigma\}$, that is, the vertices of G_Σ are the variables of Σ, and for each clause $\{x, y\}$ in Σ there is an edge $\{v(x), v(y)\} \in E(\Sigma)$.

Each edge $c = \{\upsilon(x), \upsilon(y)\} \in E$ is associated with an ordered pair (s_1, s_2) of signs, assigned as labels of the edge connecting the variables appearing in the clause. The signs s_1 and s_2 are related to the signs of the literals x and y respectively. For example, the clause $\{\overline{x} \vee y\}$ determines the labelled edge: "$x \overset{-\ +}{\frown} y$" which is equivalent to the edge "$y \overset{+\ -}{\frown} x$".

Some authors have considered the signs of the literals in the clauses of a 2-CF F by using orientation of the edge corresponding to the clause [13, 14], and then the problem of counting models of F is seen as counting the number of orientations in its respective constrained graph, which has no sink (vertex with out-degree zero).

A graph with labelled edges on a set S is a pair (G, ψ), where $G = (V, E)$ is a graph, and ψ is a function with domain E and range S. $\psi(e)$ is called the label of the edge $e \in E$. Let $S = \{+, -\}$ be a set of signs . Let $G = (V, E, \psi)$ be a signed graph with labelled edges on $S \times S$. Let x and y be nodes in V. If $e = \{x, y\}$ is an edge and $\psi(e) = (s, s')$, then s (s') is called the *adjacent sign* to x (y).

Given a 2-CF Σ, a *connected component* of G_Σ is a maximal subgraph such that for every pair of vertices x, y, there is a path in G_Σ from x to y. We say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ.

Let Σ be a 2-CF. If $\mathcal{F} = \{G_1, \ldots, G_r\}$ is a partition of Σ (over the set of clauses appearing in Σ), i.e. $\bigcup_{\rho=1}^{r} G_\rho = \Sigma$ and $\forall \rho_1, \rho_2 \in [\![r]\!], [\rho_1 \neq \rho_2 \Rightarrow G_{\rho_1} \cap G_{\rho_2} = \emptyset]$, we say that $\mathcal{F}$ is a *partition in connected components* of Σ if $\mathcal{V} = \{\upsilon(G_1), \ldots, \upsilon(G_r)\}$ is a partition of $\upsilon(\Sigma)$.

If $\{G_1, \ldots, G_r\}$ is a partition in connected components of Σ, then:

$$\mu_{\upsilon(\Sigma)}(\Sigma) = \left[\mu_{\upsilon(G_1)}(G_1)\right] * \ldots * \left[\mu_{\upsilon(G_r)}(G_r)\right] \tag{1}$$

In order to compute $\#SAT(\Sigma)$, first we should determine the set of connected components of Σ, this can be done in linear time. The different connected components of G_Σ constitute the partition of Σ in its connected components, even if G_Σ is disconnected. Then, computing $\#SAT(\Sigma)$ is reduced to compute $\mu_{\upsilon(G)}(G)$ for each connected component G of Σ.

From now on, when we mention a formula Σ, we assume that Σ is a connected component graph. We say that a 2-CF Σ is a *path*, a *cycle*, a *tree* or a *grid* if its corresponding constrained graph G_Σ is a path, a cycle, a tree, or a grid, respectively.

Notice that each connected component G_i is independent of any other G_j, $i \neq j$, since $\upsilon(G_i) \cap \upsilon(G_j) = \emptyset$. Of course, if the number of models in any connected component is zero, then the total number of models in Σ is zero too.

Let $G_\Sigma = (V, E, \{+, -\})$ be a signed connected graph of an input formula Σ in 2-CF. Let v_r be the node of minimum degree in G_Σ which is chosen to start a depth-first search. We obtain a spanning tree T_G with v_r as the root node and a set of fundamental cycles $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ and where each back edge $c_i \in E$ marks the beginning and the end of a fundamental cycle.

Given any pair of cycles C_i and C_j of $\mathcal{C}$, $i \neq j$, if C_i and C_j share edges, we call them *intersecting* cycles; otherwise, they are called *independent* cycles. Let

A_G be the depth-first search graph of G_Σ formed by the spanning tree T_G and the set of fundamental cycles $\mathcal{C}$.

We translate A_G to a Directed Acyclic Graph (DAG), denoted by D_G, assigning an orientation to each edge $\{u, v\}$ in A_G by directing: $u \to v$ if v is the parent node of u in T_G.

We apply a topological sorting procedure on D_G, obtaining an ordered number $'o'$ associated with each node in D_G such that $o(u) < o(v)$ whenever $u \to v$. This order number indicates the order for processing the nodes in D_G when we compute the value $\#\mathrm{SAT}(\Sigma)$ in the next subsection.

3.2. Computing the Number of Models on the Directed Acyclic Graph

Let Σ be a simple path, we can compute the number of models of Σ applying on G_Σ the following matrix operators:

$$T_{++} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, T_{+-} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, T_{-+} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, T_{--} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \tag{2}$$

The subscripts " $++$", " $+-$", " $-+$", and " $--$" correspond to the signs of the literals associated to the source and target nodes, respectively. For example, if $\Sigma = \{\{x, \overline{y}\}, \{\overline{y}, \overline{z}\}, \{z, w\}\}$ (see figure 1), then we use the operators given in (2) as follows.

We begin with the 2-vector $(1, 1)$ to compute in two rows the number of models of Σ. In the first row, we compute incrementally the number of models of Σ where x is 'true' ($x = 1$), and in the second row the number of models of Σ with x 'false' ($x = 0$), so that the sum of entries of the resulting vector is the number of models of Σ. The vector $q_x = (\alpha_x, \beta_x)$ is called the *charge* of the node x in Σ.

For example, the processing of the path illustrated in figure 1 is:

$$(1,1)\underline{T_{+-}} \to (1,2)\underline{T_{--}} \to (2,3)\underline{T_{++}} \to (5,2)$$

and therefore $\#SAT(\Sigma) = 7$.

In order to count the number of models on simple cycles, we consider 2×2-matrices and the matrix operators $\Psi_{ss'}$ defined on $\mathbb{N}^4$ for $s, s' \in \{+, -\}$, as follows:

$$\Psi_{ss'} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = T_{ss'} \diamond \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3}$$

Where "$\diamond$" is the Hadamard product ($(a_{ij}) \diamond (b_{ij}) = (a_{ij} b_{ij})$ for (a_{ij}) and (b_{ij}) $m \times n$-matrices).

Then, the operators in (2) are applied on each node and the operator in (3) is applied when the final arc that closes the cycle is processed. Then, the sum of entries of the resulting matrix is the number of models of Σ.

Example 1 *Let* $\Sigma = \{\{x, \overline{y}\}, \{\overline{y}, \overline{z}\}, \{z, w\}, \{\overline{x}, w\}\}$ *be a simple cycle (see figure 2), then we proceed as follows.*

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \underline{T_{+-}} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \underline{T_{--}} \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix} \underline{T_{++}} \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix} \underline{\Psi_{-+}} \begin{pmatrix} 3 & 2 \\ 0 & 1 \end{pmatrix}$$

Therefore, Σ *has 6 models.*

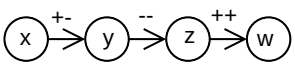

Fig.1 Example of a path

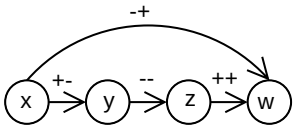

Fig.2 Example of a cycle

If G_Σ is a tree, we compute $\#SAT(\Sigma)$ applying the following algorithm

Procedure $Count\_Models(\Sigma)$
Input: Σ a constrained graph of a 2-CF
Output: $q = (m, n)$, where $m + n = \#\text{Sat}(\Sigma)$
$\mathbf{q}_x := (1, 1)\ \forall x \in \upsilon(\Sigma)$, $\upsilon := \upsilon_1(\Sigma)$, $F := \Sigma$
S1) **While** $F \neq \emptyset$ do
S4) $\quad x \in \upsilon \wedge (c \in F : x \in \upsilon(c))$
S5) $\quad F := F \setminus \{c\}$
S6) $\quad \upsilon := \upsilon_1(F)$
S7) $\quad \mathbf{q}_{c\setminus x} = \mathbf{q}_{c\setminus x} \diamond T_c \mathbf{q}_x$
S8) **Return** $\mathbf{q}_{c\setminus x}$

Where $\upsilon_1(F)$ is the set of variables of F with degree 1, T_c denotes the operator $T_{ss'}$ such that s is the sign of the variable $x \in \upsilon(c)$ of degree 1, and s' is the sign of the variable $y \in \upsilon(c) \setminus \{x\}$.

Example 2 *Let* $\Sigma = \{\{x, \overline{y}\}, \{\overline{y}, \overline{z}\}, \{z, w\}, \{y, u\}, \{z, v\}\}$ *be a tree (see fig. 3).*

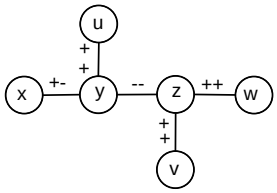

Fig.3 A tree G_Σ

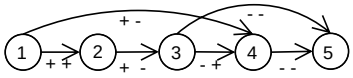

Fig.4 The DAG of the formula Σ_1

Applying the algorithm $Count\_Models$ to Σ from example 2, we have:

1. $\mathbf{q}_x = \mathbf{q}_y = \mathbf{q}_z = \mathbf{q}_w = \mathbf{q}_u = \mathbf{q}_v = (1, 1)$, $\quad F := \Sigma$
2. $\mathbf{q}_y := \mathbf{q}_y \diamond T_{+-}\mathbf{q}_x = (1, 1) \diamond T_{+-}(1, 1) = (1, 2)$, $\quad F := F \setminus \{\{x, \overline{y}\}\}$

3. $\mathbf{q}_y := \mathbf{q}_y \diamond T_{++}\mathbf{q}_u = (1,2) \diamond T_{++}(1,1) = (2,2),\ \ F := F \setminus \{\{u,y\}\}$
4. $\mathbf{q}_z := \mathbf{q}_z \diamond T_{++}\mathbf{q}_v = (1,1) \diamond T_{++}(1,1) = (2,1),\ \ F := F \setminus \{\{v,z\}\}$
5. $\mathbf{q}_z := \mathbf{q}_z \diamond T_{++}\mathbf{q}_w = (2,1) \diamond T_{++}(1,1) = (4,1),\ \ F := F \setminus \{\{z,w\}\}$
6. $\mathbf{q}_z := \mathbf{q}_z \diamond T_{--}\mathbf{q}_y = (4,1) \diamond T_{--}(2,2) = (8,4),\ \ F := \emptyset.$

Therefore $\#SAT(\Sigma) = 12$, and according to the last step, there are 8 models where z has value true and 4 models where z has value false.

Given a connected graph G_Σ, traversing G_Σ in depth-first search leads to a tree that we denote as A_Σ. We can express $G_\Sigma = A_\Sigma + E_a$, where E_a is the set of back edges that form the fundamental cycles found during the depth-first search. It is not difficult to follow the order given by the topological order procedure to compute $\#SAT(\Sigma)$.

For computing $\#SAT(\Sigma)$ for any general 2-CF Σ, we use $Count\_Models$ for pre-processing G_Σ obtaining a new $G_{\Sigma'}$ such that $\#\mathrm{SAT}(\Sigma) = \#\mathrm{SAT}(\Sigma')$ but in $G_{\Sigma'}$ there is no node of degree 1. Then $Count\_Models$ updates the charges already computed on each node in Σ'.

Now the processing continues on Σ'. For each arc $b = (s,t)$ which embraces an original fundamental cycle in G_Σ, according to the signs in b, each column $\begin{pmatrix} a \\ b \end{pmatrix}$ in the current matrix M where $b > 0$ is translated to $\begin{pmatrix} a & 0 \\ 0 & b_t \end{pmatrix}$ if s has sign '+' or such column is translated to $\begin{pmatrix} a_t & 0 \\ 0 & b \end{pmatrix}$ if s has sign '-' and $a > 0$.

Notice that the number of the node t in which the cycle closes is used as subindex. Furthermore, if there are already subindexes in the current column $\begin{pmatrix} a_i \\ b_j \end{pmatrix}$ such subindexes are preserved in the new columns aggregated to M.

Columns with similar subindexes are added in order to reduce them into a single column. The following example, allow to illustrate the processing of cycles.

Example 3 *Let us consider* $\Sigma_1 = \{\{x_1,x_2\},\{x_2,\overline{x}_3\},\{\overline{x}_3,x_4\},\{\overline{x}_4,\overline{x}_5\},\{x_1,\overline{x}_4\},\{\overline{x}_3,\overline{x}_5\}\}$ *whose constrained graph* G_{Σ_1} *is illustrated in figure 4. Note that in each node* 4 *and* 5 *a cycle is closed. The computation of* $\#SAT(\Sigma_1)$ *is done as follows.*

$\begin{pmatrix} 1 & 0 \\ 0 & 1_4 \end{pmatrix} \underline{T_{++}} \begin{pmatrix} 1 & 1_4 \\ 1 & 0 \end{pmatrix} \underline{T_{+-}} \begin{pmatrix} 1_5 & 0 & 1_{4,5} & 0 \\ 0 & 2 & 0 & 1_4 \end{pmatrix} \underline{\Psi_{+-}T_{-+}} \begin{pmatrix} 1_5 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1_4 \end{pmatrix} \underline{\Psi_{--}T_{--}}$
$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 1 & 4 & 0 & 1 \end{pmatrix}$. Therefore $\#SAT(\Sigma_1) = 9$.

Although the procedures showed in this subsection have exponential time complexity for 2-CF's in general, we have shown in [2, 7] that if the constrained graph G_Σ associated with a 2-CF Σ does not contain intersecting cycles, then we can compute $\#\mathrm{SAT}(\Sigma)$ in polynomial time.

3.3. Computing the Charges of a 2-CF

In this section, we present a novel method for computing the charges (α_x, β_x) for all variables x in a 2-CF Σ, assuming that the previous algorithm has been applied for computing $\#\mathrm{SAT}(\Sigma)$.

Let $A_0, \ldots, A_m$ be the sequence of matrices obtained during the application of the above procedure for computing #SAT(Σ). Given this sequence, we build a new sequence of matrices $B_m, \ldots, B_0$ computed as:

$$\begin{array}{l} B_m = \quad A_m \\ B_{m-i} = balance(A_{m-i}, B_{m-i+1}),\ i = 1, ..., m \end{array} \tag{4}$$

where $balance(A, B)$ is a new matrix operator between two matrices of $2 \times n$ dimensions and which produces as result a new matrix of dimension $2 \times n$, whose $i-th$ column $\begin{pmatrix} c \\ c' \end{pmatrix}$ is built from the $i-th$ columns $\begin{pmatrix} a \\ a' \end{pmatrix}$ and $\begin{pmatrix} b \\ b' \end{pmatrix}$ of A and B, respectively, as follows:

$$< \begin{pmatrix} a \\ a' \end{pmatrix}, \begin{pmatrix} b \\ b' \end{pmatrix} > \rightarrow \begin{pmatrix} c \\ c' \end{pmatrix}$$

If $a \neq a'$ then the values c and c' are the closest integers to the values $r \cdot a$ and $r \cdot a'$ respectively, and where $r = (b + b')/(a + a')$, or $\begin{pmatrix} c \\ c' \end{pmatrix} = \begin{pmatrix} b \\ b' \end{pmatrix}$ otherwise.

Notice that the essence of the recurrence *balance* is to apply the inverse operation of that used in each step of the computation of #SAT(Σ), and following the inverse order used in the construction of the sequence $A_0 \ldots, A_m$.

If initially the matrices A and B do not have the same dimensions, e.g A is a $2 \times n_1$ matrix and B is a $2 \times n_2$ matrix and $n_1 < n_2$, then we reduce the dimension (n_2) of B to the dimension (n_1) of A, summing columns and closing lines of computation according to the order in which they were extended (when a new cycle was found) during the computation of #SAT(Σ).

Therefore, let $x_0, \ldots, x_m$ be the topological order of the nodes in G_Σ, we associate with each node x_i, $i = 0, \ldots m$ the matrix B_i from the sequence $B_m, \ldots, B_0$. And then, the charge q_i of the variable x_i, $i = 0, \ldots, m$ is determined as: $q_i = \begin{pmatrix} a_i \\ a'_i \end{pmatrix}$ where a_i and a'_i are the sum of the entries of the first and second row of B_i, respectively.

For example, let us consider the formula Σ_1 showed at the end of section 3.2. For this formula, we have:

$$A_0 = \begin{pmatrix} 1\ 0 \\ 0\ 1 \end{pmatrix}, A_1 = \begin{pmatrix} 1\ 1 \\ 1\ 0 \end{pmatrix}, A_2 = \begin{pmatrix} 1\ 0\ 1\ 0 \\ 0\ 2\ 0\ 1 \end{pmatrix}, A_3 = \begin{pmatrix} 1\ 2\ 0\ 0 \\ 0\ 2\ 0\ 1 \end{pmatrix}, A_4 = \begin{pmatrix} 0\ 2\ 0\ 1 \\ 1\ 4\ 0\ 1 \end{pmatrix}$$

Applying the recurrence (4), we obtain the series $B_m, \ldots, B_0$ and the charges $q_m, \ldots, q_0$ as:

$$B_4 = A_4 \ \Rightarrow q_4 = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

$$B_3 = balance(\begin{pmatrix} 1\ 2\ 0\ 0 \\ 0\ 2\ 0\ 1 \end{pmatrix}, \begin{pmatrix} 0\ 2\ 0\ 1 \\ 1\ 4\ 0\ 1 \end{pmatrix}) = \begin{pmatrix} 1\ 2\ 0\ 0 \\ 0\ 4\ 0\ 2 \end{pmatrix} \Rightarrow q_3 = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

$$B_2 = balance(\begin{pmatrix} 1\ 0\ 1\ 0 \\ 0\ 2\ 0\ 1 \end{pmatrix}, \begin{pmatrix} 1\ 2\ 0\ 0 \\ 0\ 4\ 0\ 2 \end{pmatrix}) = \begin{pmatrix} 1\ 0\ 0\ 0 \\ 0\ 6\ 0\ 2 \end{pmatrix} \Rightarrow q_2 = \begin{pmatrix} 1 \\ 8 \end{pmatrix}$$

$$B_1 = balance(\begin{pmatrix} 1\ 1 \\ 1\ 0 \end{pmatrix}, \begin{pmatrix} 1\ 0\ 0\ 0 \\ 0\ 6\ 0\ 2 \end{pmatrix}) = balance(\begin{pmatrix} 1\ 1 \\ 1\ 0 \end{pmatrix}, \begin{pmatrix} 1\ 0 \\ 6\ 2 \end{pmatrix}) = \begin{pmatrix} 6\ 0 \\ 1\ 2 \end{pmatrix}$$

$$\Rightarrow q_1 = \begin{pmatrix} 6 \\ 3 \end{pmatrix}$$

$$B_0 = balance(\begin{pmatrix} 1\ 0 \\ 0\ 1 \end{pmatrix}, \begin{pmatrix} 6\ 0 \\ 1\ 2 \end{pmatrix}) = \begin{pmatrix} 7\ 0 \\ 0\ 2 \end{pmatrix} \Rightarrow q_0 = \begin{pmatrix} 7 \\ 2 \end{pmatrix}.$$

It is not hard to propagate the computation of the charges for the nodes which were pre-processed by the *Count_Models* algorithm and that do not appear in $G_{\Sigma'}$, in fact, the same operator *balance* is applied for this task.

Notice that the computation of the charges of a 2-CF Σ has the same complexity order that the one used for computing #SAT(Σ). Thus, if the constrained graph of Σ does not contain intersecting cycles, then we compute all the charges of Σ in polynomial time.

4. Propositional Reasoning Based on the Number of Models

Independently that the computation of the charges of a KB Σ might require in the worst case an exponential time, knowing the charges as well as an evaluating order (G_Σ) are potentially useful in the area of propositional reasoning. For example in the field of model-based diagnosis, where is essential to determine all single assumptions (literals or clauses) that resolve an inconsistency, the evaluation order and the charges in Σ are useful for determining the best order in which the test of flaws in the digital circuits for consistency checking can be done [12, 8].

Other kind of applications where the charges can be useful is in the field of system planning, where the charges of a plan can be used for recognizing the frequency use of some operators, or in order to recognize which is the most frequently scenario of a trajectory that achieves some subgoals.

Other line of applications is in belief revision and updating knowledge bases. For example, consider an initial KB containing the clauses: $\{x\}$ and $\{\overline{x}, y\}$. One obvious implication of those clauses is the fact $\{y\}$. If a new information $\{\overline{y}\}$ has been observed by an intelligent agent, contradicting the assumption that y is true, so we will have to give up some (or all) of our previous beliefs, or it will lead to an inconsistent KB.

Many approaches for incorporating dynamically a single or a sequence of changes into an initial Knowledge Base (KB) have been proposed [1, 4, 6, 11]. Almost all of these proposals are plagued by serious complexity-theoretic impediments, even in the Horn case [4, 6]. More fundamentally, these schemes are not inductive, in the sense that they can lose in a single update any positive properties of the structure of the KB.

Even in the previous trivial example, it is not clear which approach should be taken. Usually, extra-logical factors should be taken into account, like the source and reliability of each piece of information or some kind of bias towards or against updates. For example, some methods for revision are based on some implicit bias, namely an a priori probability that each element of the domain theory requires revision [10].

Contrary to assign the probabilities to each element of the theory Σ by an expert or simply chosen by default, the charges of the variables provide a degree of their reliability in Σ. Furthermore, the dynamic updating of our representation of the KB gives the additional advantages that the relative values of the element of Σ can be adjusted automatically, in response to newly-obtained information.

In the rest of this chapter, we present only one of the applications in which the built logical structure and the charges of a 2-CF are useful. We try here the problem of computing the degree of belief of an intelligent agent on new information.

4.1. Computing the Degree of Belief of an Intelligent Agent

A generalization of deductive inference which can be used when a knowledge base is augmented by, e.i., statistical information, is to use the computation of a degree of belief, as an effort to avoid the computationally hard task of deductive inference.

If we assign an equal degree of belief to all 'basic situations' that appear in a knowledge base Σ of an intelligent agent, then we can compute the probability that Σ can be satisfied. If Σ involves n variables, the probability to satisfy Σ, is: $P_\Sigma = Prob(\Sigma \equiv \top) = \frac{\#SAT(\Sigma)}{2^n}$, where $\top$ stands for the logical value true and $Prob$ is used to denote the probability [13].

We are interested in the complexity of the computation of the degree of belief of a query (propositional formula) F with respect to Σ, which is considered as the fraction of models of Σ that are consistent with F, that is, the conditional probability of F with respect to Σ, denoted by $P_{F|\Sigma}$ and computed as: $P_{F|\Sigma} = Prob((\Sigma \wedge F) \equiv \top | \Sigma \equiv \top) = \frac{\#SAT(\Sigma \wedge F)}{\#SAT(\Sigma)}$.

We start assuming that the KB Σ is a satisfiable 2-CF, then $\#\mathrm{SAT}(\Sigma) > 0$ and $P_{F|\Sigma} = \frac{\#SAT(\Sigma \wedge F)}{\#SAT(\Sigma)}$ is well-defined. One important goal of research is to recognize the class of formulas for Σ and F where the computation of $P_{F|\Sigma}$ can be done efficiently, and for this, we need an appropriate representation of the knowledge base, as well as smart algorithms for solving #SAT.

Let Σ be a KB in 2-CF and let F be a query which is a unitary or binary clause, since for this case we have an inductive scheme for the incremental recompilation of knowledge which keeps the structure of the KB with updating actions. We assume that Σ has been processed by the procedures of the previous section and the charges of all variables in Σ have been stored in the array $vars\_pairs$.

Suppose that an agent A has to take an action according to a set of options $Q = \{l_1, \ldots, l_k\}$. The classic deduction method suggests to choose the literals

$l \in Q$ which are consistent with Σ, that is, checking the satisfiability of $(\Sigma \cup l)$ for each $l \in Q$, recognizing equal value to all literals satisfiable with Σ.

But $P_{l|\Sigma}$ provides more information than knowing that $(\Sigma \cup l)$ is satisfiable. $P_{l|\Sigma}$ gives the proportion of the original models of Σ that remains models for $(\Sigma \cup l)$. This class of information is crucial when the agent A has to take an action depending on the strategic value of each alternative $l \in Q$. Indeed, A can decide its action according to the option $l \in Q$ which maximizes its degree of belief. Then, we show now how to compute $P_{l|\Sigma}$ and for this, we have two cases:

1. $l \in Lit(\Sigma)$: As every variable $x_i \in v(\Sigma)$ has associated its respective charge (α_i, β_i), we use $v(l)$ as a pointer for the array $vars\_pairs$ in order to recover $(\alpha_{v(l)}, \beta_{v(l)})$. Then, $P_{F|\Sigma} = \frac{\beta_{v(l)}}{\mu(\Sigma)}$ if l is a negated variable or $P_{l|\Sigma} = \frac{\alpha_{v(l)}}{\mu(\Sigma)}$ otherwise.
2. $l \notin Lit(\Sigma)$: Then we have new information not considered before and the original probability space for computing the conditional probability $P_{F|\Sigma}$ has to be extended.
   When new pieces of information that did not originally appear in the sample space have to be considered, then we introduce in the area of updating the degree of belief by doing an extension of the original probability space [5]. Let consider here, a more general case.
   Let $F = (\bigwedge_{j=1}^{k} l_j)$ be a conjunction of literals such that every variable of F does not appear in $v(\Sigma)$. Let $\mid v(\Sigma) \mid = n$. There are 2^n assignments defined over $v(\Sigma)$ and 2^{n+k} assignments defined over $v(\Sigma) \cup v(F)$, then we *update* the domain of the probability space for computing $P_{F|\Sigma}$, as: $P_{F|\Sigma} = \frac{Prob_{(\Sigma \wedge F)}}{Prob_{\Sigma}} = \frac{\frac{\mu(\Sigma \wedge F)}{2^{n+k}}}{\frac{\mu(\Sigma)}{2^n}} = \frac{\mu(\Sigma \wedge F)}{2^k \cdot \mu(\Sigma)}$. As G_F and G_Σ are two independent connected components and $\mu(\bigwedge_{l \in F} l) = \prod_{l \in F} \mu(l) = 1$, then:

$$P_{F|\Sigma} = \frac{\mu(\Sigma) \cdot \mu(F)}{2^k \cdot \mu(\Sigma)} = \frac{\mu(\Sigma)}{2^k \cdot \mu(\Sigma)} = \frac{1}{2^k} \tag{5}$$

   Indeed, for the case $k = 1$, an agent A believes in new information not related with its knowledge base with a reliability of 0.5. Since we extend the models of Σ for considering a new variable $v(l)$, half of those extension assignments have $v(l)$ true and the other half have $v(l)$ false. We have that the fraction of models of Σ which are consistent with $\{l\}$ is 1/2 and the other half is consistent with $\{\overline{l}\}$.

Notice that in the first case, $P_{l|\Sigma}$ is computed by one pointer access, one comparison and one division, then it has a constant time complexity. In general, least of the time for computing $P_{l|\Sigma}$ is spent for determining the position of $v(l)$ in the array $vars\_pairs$. If we are not using hash techniques or pointers for ordering the variables, then in the worst case, we need a logarithmic time on the number of variables of the KB for determining the position of $v(l)$ in $vars\_pairs$ (applying for instance a binary search).

Consider now the case when the set of options $Q = \{c_1, \ldots, c_k\}$ is a set of binary clauses and the agent A has to determine its future action based on the

options codified by each clause. For example, A chooses its future action based on the clause $c \in Q$ that maximizes its degree of belief with respect to the KB. Let $c = \{x, y\}$ be any clause of Q, we have four cases for the computation of $P_{c|\Sigma}$:

1. $x \notin \Sigma$ and $y \notin \Sigma$: There are three models out of the four assignments of $\upsilon(c)$ and, as the constrained graphs G_Σ and G_c are independent, then $P_{c|\Sigma} = \frac{\mu(\Sigma)\cdot\mu(c)}{\mu(\Sigma)\cdot 2^2} = 3/4$, since we extend the probability space with the new two variables: $\upsilon(x)$ and $\upsilon(y)$. This case is computed in constant time.
2. $x \in Lit(\Sigma)$ and $y \notin Lit(\Sigma)$: Then $\upsilon(x)$ is searched on the array *vars_pairs* in order to retrieve $(\alpha_{\upsilon(x)}, \beta_{\upsilon(x)})$. According to the sign of $x \in c$ we have that $\mu(\Sigma \wedge c) = 2 \cdot \alpha_{\upsilon(x)} + \beta_{\upsilon(x)}$ if x appears as unnegated variable in c, otherwise $\mu(\Sigma \wedge c) = 2 \cdot \beta_{\upsilon(x)} + \alpha_{\upsilon(x)}$. Then;
   $$P_{c|\Sigma} = \begin{cases} \frac{2\cdot\alpha_{\upsilon(x)}+\beta_{\upsilon(x)}}{\mu(\Sigma)} & \text{if } x \text{ appears as unnegated variable,} \\ \frac{2\cdot\beta_{\upsilon(x)}+\alpha_{\upsilon(x)}}{\mu(\Sigma)} & \text{otherwise} \end{cases}$$
   Notice that this case has a logarithmic time complexity over $\mid \upsilon(\Sigma) \mid$ since it mainly depends on retrieving the pair $(\alpha_{\upsilon(x)}, \beta_{\upsilon(x)})$ from the array of variables.
3. $x \in Lit(\Sigma)$, $y \in Lit(\Sigma)$ and $c \in \Sigma$: since c has been already computed in $\mu(\Sigma)$, $\mu(\Sigma \wedge c) = \mu(\Sigma)$ and then $P_{c|\Sigma} = 1$. This case obtains the maximum possible value for $P_{c|\Sigma}$, so any alternative of action of the agent will take this option.
4. $x \in Lit(\Sigma)$, $y \in Lit(\Sigma)$ and $c \notin \Sigma$: Let consider for this option a more general situation, explained at once.

Let $F = (\bigvee_{j=1}^{k} l_j)$ be a clause with k literals. Considering F as a set of literals, let $A = \{l \in F | \upsilon(l) \notin \upsilon(\Sigma)\}$ be the literals in F whose variables do not appear in $\upsilon(\Sigma)$ and let $F' = F - A$ be the literals in F whose variables appear in $\upsilon(\Sigma)$, let $t = \mid A \mid$.

We compute $\mu(\Sigma \wedge F)$ by extending the models of Σ with the new variables $\upsilon(A)$ and eliminating from this extended assignments those which falsify $(\Sigma \wedge F)$, that is, $\mu(\Sigma \wedge F) = \mu(\Sigma) \cdot 2^t - \mu(\Sigma \wedge \overline{F})$, where $\overline{F} = (\overline{\bigvee_{j=1}^{k} l_j}) = (\bigwedge_{j=1}^{k} \overline{l}_j)$.

As G_A is a connected component independent of $G_{\Sigma \cup F'}$, then $\overline{F} = \overline{A} \cup \overline{F'}$ and $\mu(\Sigma \wedge \overline{F}) = \mu(\Sigma \wedge \overline{F'}) \cdot \mu(\overline{A}) = \mu(\Sigma \wedge \overline{F'})$ since $\mu(\overline{A}) = 1$, then:

$$P_{F|\Sigma} = \frac{\mu(\Sigma \wedge F)}{2^t \cdot \mu(\Sigma)} = \frac{\mu(\Sigma) \cdot 2^t - \mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} = 1 - \frac{\mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} \tag{6}$$

We can consider $\overline{F'}$ as a partial assignment on the number of variables in $(\Sigma \wedge F)$ since it consists of a set of literals. Let $s = (\bigwedge_{j=1}^{k} \overline{l}_j)$ be an initial partial assignment defined over $\upsilon(\Sigma) \cup \upsilon(\overline{F})$ which consists of 2^{n+t} assignments. We can consider s as a partial assignment and try to extend it in order to count the total number of satisfying assignments for $(\Sigma \wedge \overline{F})$. If we consider s as a set of unitary clauses then s could be used in a unit reduction process with Σ, in order to build extended satisfying assignment s' for $(\Sigma \wedge \overline{F}')$.

We call the *reduction* of Σ by a literal $l \in Lit(\Sigma)$ (also called $forcing\ l$) and denoted by $\Sigma[l]$ to the set of clauses generated from Σ by
1) removing all clause containing l (called subsumption rule),
2) removing $\overline{l}$ from all the remaining clauses (called unit resolution rule).

The *unit reduction* on a formula Σ consists of, given a unitary clause (l), performing a reduction of Σ for the literal of the unitary clause, that is, $\Sigma[l]$. Given the partial assignment $s = (\bigwedge_{j=1}^{k} \overline{l}_j)$, we define the reduction of Σ by s, as: $\Sigma[s] = \Sigma[\overline{l}_1][\overline{l}_2]\dots[\overline{l}_k]$. For short, we write $\Sigma[\overline{l}_1, \overline{l}_2, \dots, \overline{l}_k]$ instead of $\Sigma[\overline{l}_1][\overline{l}_2]\dots[\overline{l}_k]$. We denote with Σ' the resulting formula of applying unit reduction on Σ and s, that is, $\Sigma' = \Sigma[s]$.

Notice that a unit resolution rule can generate new unitary clauses. Furthermore, the unit resolution rule allows to extend the partial assignment s by the new unitary clauses appearing in this process, that is, $s = s \cup \{u\}$ where u is obtained by unit resolution rule in $\Sigma[s]$. If a pair of contradictory unitary clauses are obtained during this process then $\mu(\Sigma \wedge \overline{F}) = 0$.

Unit Propagation $UP(\Sigma, s)$ is the iterative process of doing unit reduction applying a set of unitary clauses (in our case s) over Σ until there are no more applications of *unit reductions* on the resulting formulas Σ'.

When a subsumption rule is applied, we have to consider the set of variables in Σ which can be *eliminated* from Σ. For example, if we apply a subsumption rule on $(x) \wedge (x \vee y)$, both clauses are eliminated from Σ but if y has only one occurrence in Σ, then the subsumption rule eliminate y, but the total number of models for Σ' has to consider that y can take any logical value. We introduce a new set $Elim\_vars$ containing the eliminated variables by the application of the subsumption rule. $Elim\_vars$ is checked in each application of the subsumption rule.

Then, the partial assignment s is applied on Σ in order to simplify the original KB by a more simple KB Σ', that is, $\Sigma' = UP(\Sigma, s)$ and then $\mu(\Sigma \wedge \overline{F}) = \mu(\Sigma') * 2^{|Elim\_vars|}$.

If $UP(\Sigma, \overline{c})$ generates the nil clause, then $\mu(\Sigma \wedge \overline{c}) = 0$ and this means that the initial clause c is logically deduced from Σ ($\Sigma \models c$), and then $P_{c|\Sigma} = 1 - \frac{\mu(\Sigma \wedge \overline{c})}{\mu(\Sigma)} = 1$. Furthermore, the generation of the nil clause in $UP(\Sigma, \overline{c})$ takes a linear time on the number of clauses of Σ. Then, if we have the logical structure representation of Σ, the logical deduction task of proving $\Sigma \models c$ for c a unitary or binary clause is solved in linear time.

Of course, if $\Sigma \not\models c$ and $v(c) \subset v(\Sigma)$ then there are cases where the computation of $\mu(\Sigma \wedge c)$ could require almost the same time that computing $\mu(\Sigma)$. However, as the resulting formula Σ' of $UP(\Sigma, s)$ is a subset of Σ, then $G_{\Sigma'}$ is a subgraph of G_Σ. In fact $G_{\Sigma'}$ is formed by substructures of G_Σ which are already computed during the computation of $\mu(\Sigma)$ and then, it is not necessary to re-compute such substructures. Thus, we only have to re-compute on the trajectories from x to y (the variables in c) what is modified from G_Σ to $G_{\Sigma'}$.

Example 4 *Let* $\Sigma_1 = \{\{x_1, x_2\}, \{x_2, \overline{x}_3\}, \{\overline{x}_3, x_4\}, \{\overline{x}_4, \overline{x}_5\}, \{x_1, \overline{x}_4\}, \{\overline{x}_3, \overline{x}_5\}, \{x_2, \overline{x}_6\}\}$ *be an initial KB and the clause* $c = \{\overline{x}_2, \overline{x}_5\}$. *We want to compute*

$P_{c|\Sigma_1}$. *We know that* $\mu(\Sigma_1) = 15$. $\overline{F} = (x_2) \wedge (x_5)$ *and the partial assignment to start the computation of* $\mu(\Sigma_1 \wedge \overline{F}')$ *is* $s = \overline{F}$. $\Sigma_1' = \Sigma_1[x_2, x_5] = \{\{\overline{x}_3, x_4\}, \{\overline{x}_4\}, \{x_1, \overline{x}_4\}, \{\overline{x}_3\}\}$, *the clauses* c_1, c_2 *and* c_7 *from* Σ_1 *were subsumed and then* $Elim\_vars = \{x_1, x_6\}$ *are the eliminated variables in this iteration. The new unitary clauses* $\{\overline{x}_4\}$ *and* $\{\overline{x}_3\}$ *are generated, extending the partial assignment as* $s = (x_2, x_5, \overline{x}_4, \overline{x}_3)$ *and such new unitary clauses are used in the Unit Propagation procedure, then* $\Sigma_2' = \Sigma_1'[\overline{x}_4, \overline{x}_3] = \emptyset$. *Then* $\mu(\Sigma \wedge \overline{F}) = \mu(\Sigma_2') \cdot 2^{|Elim\_vars|} = 1 \cdot 2^2 = 4$. *And according to equation (6), we have that* $P_{c|\Sigma_1} = 1 - \frac{4}{15} = \frac{11}{15}$.

We can extend this process of deciding which action an intelligent agent has to take if the set of options are codified, for example, by a set of 2-CF's, that is, $Q = \{F_1, \ldots, F_k\}$ constitutes the set of alternative actions for an intelligent agent. Of course, for this latter case, the complexity time of computing $P_{F_i|\Sigma}, i = 1, \ldots, k$ is at least up to now, exponential and in this case, the previous knowledge about $\#\mathrm{SAT}(\Sigma)$ and the charges of the variables may be not enough to reduce the time complexity of computing $\#\mathrm{SAT}(\Sigma \cup F)$.

5. Conclusions

We have designed an appropriate logical structural representation of a 2-CF knowledge base. Our model-based system of reasoning includes cases where the formula-based approach does not support efficient reasoning. We show that using our logical structural representation, we can compute the degree of belief $P_{F|\Sigma}$ efficiently, when F is a query composed by literals or a binary clause which includes variables not appearing before in Σ. Indeed, for this case, we have an inductive scheme and the incremental recompilation of knowledge keeps the initial structure of the KB without losing with the updates, any positive properties of the structure of the KB, providing so, an efficient scheme of reasoning for an intelligent agent who has its knowledge base represented by a 2-CF.

Exploiting this logical structural representation of a 2-CF, we also propose a way to determine the relative value for all element in the KB which is an essential problem in some applications of deductive reasoning. Furthermore, the dynamic updating of our logical representation of the KB provides the additional advantage that the relative value of the elements of Σ could be adjusted automatically in response to newly-obtained information.

References

1. Darwiche A., On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, *Jour. of Applied Non-classical Logics*, 11(1-2), (2001), pp. 11-34.
2. De Ita G., Tovar M., Vera E., Guillén C., Designing Efficient Procedures for #2SAT, Proceedings of the 12th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-12), (2005), pp. 28-32.

3. De Ita G., Tovar M., Applying Counting Models of Boolean Formulas to Propositional Inference, *Advances in Computer Science and Engineering Researching in Computing Science* Vol. 19, (2006), pp. 159-170.
4. Eiter, T., Gottlob, G., On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals, *Artificial Intelligence* 57,1992.
5. Fagin R., Halpern J. Y., A new approach to updating beliefs, *Uncertainty in Artificial Intelligence* 6, eds. P.P. Bonissone, M. Henrion, L.N. Kanal, J.F. Lemmer, (1991), pp. 347-374.
6. Gogic G., Papadimitriou C.H., Sideri M., Incremental Recompilation of Knowledge, *Journal of Artificial Intelligence Research* 8, (1998), pp. 23-37.
7. Guillen C., López A., De Ita G, Model Counting for 2SAT Based on Graphs by Matrix Operators, *Jour. Engineering Letters*, Vol. 15, No. 2, (2007), pp.259-265.
8. Han B., Lee Shie-Jue, Yang Hsin-Tai, A Model-Based Diagnosis System for Identifying Faulty Components in Digital Circuits, *Applied Intelligence* 10, (1999), pp.37-52.
9. Khardon R., Roth D., Reasoning with Models, *Artificial Intelligence*, Vol. 87, No. 1, (1996), pp. 187-213.
10. Koppel M., Feldman R., Maria Segre A., Bias-Driven Revision of Logical Domain Theories, *Jour. of Artificial Intelligence Research 1*, (1994), 159-208.
11. Liberatore P., Schaerf M., The Complexity of Model Checking for Belief Revision and Update, Procc. Thirteenth Nat. Conf. on Art. Intellegence (AAAI96), 1996.
12. Peischl B., Wotawa F., Computing Diagnosis Efficiently: A Fast Theorem Prover For Propositional Horn Theories, Proc. of the 14th Int. Workshop on Principles of Diagnosis, (2003), pp.175-180.
13. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence* 82, (1996), pp.273-302.
14. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.