Efficient Computation of the Degree of Belief for a Subclass of Two Conjuctive Forms

Guillermo De Ita[1,2], Carlos Guillén[1], Ali Khanafer[2]

[1] Universidad Autónoma de Puebla, México
[2] INRIA Lille, Nord Europe, Francia,
{deita,cguillen}@ccc.inaoep.mx, khanafer.aly@gmail.com

Abstract. Assuming a Knowledge Base Σ expressed by a two Conjunctive Form (2-CF), we show that if the constrained graph of Σ is acyclic or if its cyles are independent one to the other, then it is possible to count efficiently the number of models of Σ. We determine a set of recurrence equations which allow us to design an incremental technique for counting models by a simple traversal of the constrained graph.
One of the advantages of such technique, furthermore that it has linear time complexity, is that applying it backwards allows us to determine the exact number of the logical values (the charge) that each variable has in the set of models. The charge of each variable allow us to design an efficient scheme of reasoning, even in the case where the formula-based representation does not allow to do it. Indeed, we can compute the degree of belief for new information (a literal or a binary clause) efficiently.
Keywords: #SAT Problem, Automated Reasoning, Degree of Belief.

1. Introduction

The Artificial Intelligence 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 [8]. 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 [2, 10, 12], 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, for processing here we mean to compute the degree of belief, updating or doing belief revision.

In order to clarify a frontier between efficient and exponential-time reasoning, we start working with a Knowledge Base (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 has in a new piece of information F based on the conditional probability of F with respect to Σ, denoted as $P_{F|\Sigma}$, and calculated as the fraction of models of Σ that satisfy F.

One important goal of research is to recognize the class of formulas for Σ and F where the degree of belief $P_{F|\Sigma}$ can be computed efficiently, and for this, the development of smart algorithms for solving the #SAT problem is key.

#SAT consists of counting the number of models of logical formulas. #SAT is a classical #P-complete problem, and an interesting area of research has been the identification of restricted cases for which #SAT can be solved efficiently. #SAT remains #P-complete even if we consider only monotone or Horn propositional formulas [12].

We focus toward the identification of structures on the constrained graph of formulas in 2-CF, such that these structures help us to compute #SAT efficiently. Furthermore our interest is to design procedures which count the number of models of a given formula in an incremental way, at the same time that we are traversing by the constrained graph of the formula.

Given our logical structure representation of a 2-CF, we propose procedures for determining 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.

We use $\upsilon(X)$ to express the variables involved in the object X, where X could be a literal, a clause or a CF. For instance, for the clause $c = \{\overline{x_1}, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$. $Lit(F)$ is the set of literals appearing in F, i.e. if $X = \upsilon(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$. We denote $\{1, 2, ..., n\}$ by $[\![n]\!]$.

An assignment s for F is a boolean function $s : \upsilon(F) \to \{0, 1\}$. 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 from F, and $\upsilon(F_1) \subset \upsilon(F)$, an assignment over $\upsilon(F_1)$ is a *partial* assignment over $\upsilon(F)$. Assuming $n =|$

$\upsilon(F)$ | and $n_1 =$| $\upsilon(F_1)$ |, any assignment over $\upsilon(F_1)$ has 2^{n-n_1} extensions as assignments over $\upsilon(F)$.

Let F be a 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 $\upsilon(F)$ that satisfies F. 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 $\upsilon(F)$. #2-SAT denotes #SAT for formulas in 2-CF. We also denote #SAT(F) by $\mu_{\upsilon(F)}(F)$ or just $\mu(F)$ when $\upsilon(F)$ is clear from the context.

3. Logical Structure for Representing a 2-CF

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

In order to compute $\#SAT(\Sigma)$, first we should determine the set of connected components of G_Σ and this can be done in linear time. Then, $\#SAT(\Sigma)$ is translated to compute $\#SAT(G)$ for each connected component G of G_Σ [2, 3, 12]. From now on, when we mention a 2-CF 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.

Each edge has associated an ordered pair (s_1, s_2) of signs, assigned as labels. For example, the signs s_1 and s_2 for the clause $\{\overline{x} \vee y\}$ are related to the signs of the literals x and y respectively, then $s_1 = -$ and $s_2 = +$ and the edge is denoted as: $x \overset{-\ +}{\text{---}} y$ which is equivalent to the edge $y \overset{+\ -}{\text{---}} x$.

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 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* of x (y).

Let $G_\Sigma = (V, E, \{+, -\})$ be a signed connected graph of an input formula Σ in 2-CF. Let v_r be a node of G_Σ 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 $\mathcal{C} = \{C_1, C_2, ..., C_k\}$, where each back edge $c_i \in E$ marks the beginning and the end of a fundamental cycle.

Given any pair of fundamental 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_Σ 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_Σ, assigning an orientation to each edge $\{u, v\}$ in A_Σ by directing: $u \rightarrow v$ if v is the parent node of u in T_G, see figure 2. We apply a topological sorting procedure on D_Σ, obtaining an ordered number $'o'$ associated with each node in D_Σ such

that $o(u) < o(v)$ whenever $u \rightarrow v$. This order number indicates the order for processing the nodes in D_Σ when we compute the value #SAT(Σ) in the next subsection.

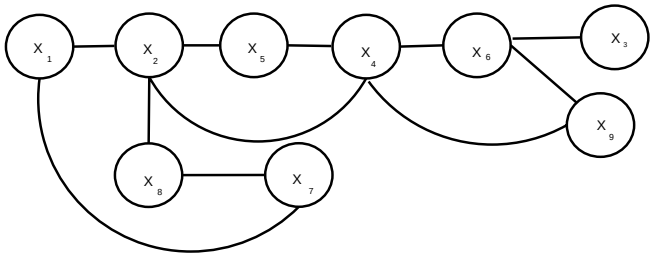

Fig. 1. A depth-first graph A_Σ

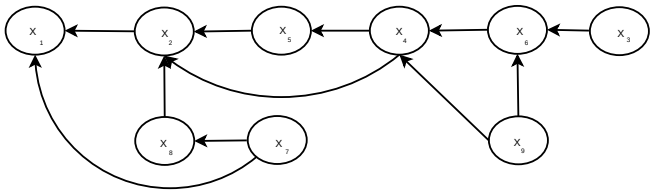

Fig. 2. The DAG D_Σ associated with the previous constrained graph

3.1. Linear procedure for Counting the Number of Models for a 2-CF

In previous papers, we have presented a set of recurrence equations to compute #SAT(Σ) by traversing D_Σ [3, 4, 7]. We recall now these recurrence relations to be applied according to the topological order of its nodes.

For each variable $x \in v(\Sigma)$ a pair (α_x, β_x) called the *initial charge*, is used for indicating the number of logical values: 'true' and 'false' respectively, that x initially took when the value #SAT(Σ) is being computed.

When we count models over a constrained graph D_Σ, we use *computing threads*. A computing thread is a sequence of pairs $(\alpha_i, \beta_i), i = 1, \ldots, m$ used for computing the number of models over a path of m nodes. A main thread, denoted by *Lp*, is associated with the spanning tree of D_Σ. This thread is always active until the process of counting finishes completely. We sketch now the general procedure for computing #SAT(Σ).

Procedure $Count\_Models(\Sigma)$
Input: D_Σ: a constrained graph of a 2-CF Σ
Output: The charge (α_y, β_y) associated with the root node y, where $\#\mathrm{SAT}(\Sigma) = \alpha_r + \beta_r$

Each node and edge of D_Σ is visited according with the topological order, and at the same time a pair (α_x, β_x) is computed for each node $x \in D_\Sigma$, in the following way:

1. $(\alpha_x, \beta_x) = (1, 1)$ if x is a leaf node in D_Σ.
2. When there is an unique edge between two nodes, e.g. $x \overset{s_1\ s_2}{\rightarrow} y$ is an edge with signs s_1 and s_2 then according with the signs (s_1, s_2) the pair (α_y, β_y) is computed from (α_x, β_x) (in parallel way it is done for all computing active thread), as:

$$(\alpha_y, \beta_y) = \begin{cases} (\beta_x \quad\quad, \alpha_x + \beta_x) \text{ if } (s_1, s_2) = (-, -) \\ (\alpha_x + \beta_x, \beta_x \quad\quad) \text{ if } (s_1, s_2) = (-, +) \\ (\alpha_x \quad\quad, \alpha_x + \beta_x) \text{ if } (s_1, s_2) = (+, -) \\ (\alpha_x + \beta_x, \alpha_x \quad\quad) \text{ if } (s_1, s_2) = (+, +) \end{cases} \tag{1}$$

   We denote with $' \rightarrow'$ the application of anyone of the four previous rules in the recurrence (1).
3. If v is a parent node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, as we have already visited all child nodes, then each pair $(\alpha_{u_i}, \beta_{u_i})$ $i = 1, ..., k$ has been determined based on recurrence (1). And then, let $\alpha_v = \prod_{j=1}^{k} \alpha_{u_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{u_j}$. We denote with $\succ$ the application of this multiplicative rule.
4. If the node y begins of a cycle C then a new computing thread L_C is created. The initial pair associated to the computing thread L_C is $(0, \beta_y)$ if $s_1 = +$, or $(\alpha_y, 0)$ if $s_1 = -$, where (α_y, β_y) is the current pair for the node y in the thread Lp and s_1 is the sign of x in the back edge of the cycle C.
5. If y ends a cycle C, then the current pair $(\alpha_{y_c}, \beta_{y_c})$ in the the extra-computing thread L_C is substracted to the current pair (α_y, β_y) of the main thread Lp, according with the sign s_2 in the back edge of the cycle C, in the following way: if $s_2 = +$ then $(\alpha_y, \beta_y) = (\alpha_y, \beta_y - \beta_{y_c})$ or $(\alpha_y, \beta_y) = (\alpha_y - \alpha_{y_c}, \beta_y)$ if $s_2 = -$. We denote by ' $\curvearrowleft$ ' the binary operation between the two active threads for computing the initial charge of the node which closes a cycle.
6. If y is the root node of D_Σ then returns its computed charge: (α_y, β_y).

To illustrate this procedure, we apply it on the DAG of figure 2, assuming a monotone 2-CF, that is, all variable appear in nonnegative way and then the last rule of recurrence (1) is always applied, denoted by an arrow with a dotted line in figure 3, while the operator ' $\curvearrowleft$ ' is shown by a line like an substracted operator. For this example, the topological order is: $x_3 \rightarrow x_6; x_9 \rightarrow x_6; x_6 \rightarrow x_4; x_9 \rightarrow x_4; x_4 \rightarrow x_5; x_5 \rightarrow x_2; x_4 \rightarrow x_2; x_7 \rightarrow x_8; x_8 \rightarrow x_2; x_2 \rightarrow x_1; x_7 \rightarrow x_1$.

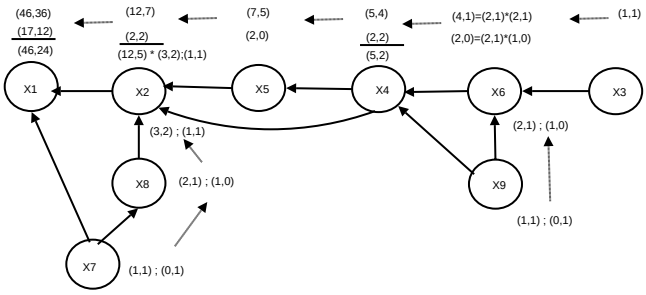

Fig. 3. Computing the initial charge of the variables

In [4], we have extended the previous algorithm in order to process, keeping the polynomial time complexity, graphs with embedded cycles. Although for our purposes, these classes of graphs are not considered in this article.

3.2. Computing the Charges of a 2-CF

In this section, we present a novel method for computing the charge $(\alpha_i, \beta_i), i = 1, \ldots, n$ for all variables $x_i, i = 1, \ldots, n$ of a 2-CF Σ. Once the previous algorithm has been applied for computing $\#\mathrm{SAT}(\Sigma)$ and the inital charges has been computed for all variables in Σ. In fact, the initial charge (α_n, β_n) for x_n is also its final charge, that is, $\#\mathrm{SAT}(\Sigma) = \alpha_n + \beta_n$.

Let $A_1, \ldots, A_n$ be the sequence of initial charges obtained during the application of the procedure $Count\_Models$. Given such sequence, we build a new sequence of pairs which represent the final charges (or just the charges) $B_n, \ldots, B_1$ where B_i is the charge for the variable $x_i \in \upsilon(\Sigma)$, which is computed as:

$$\begin{array}{l} B_n = \quad A_n \\ B_{n-i} = balance(A_{n-i}, B_{n-i+1}),\ i = 1, ..., n-1 \end{array} \tag{2}$$

where $balance(A, B)$ is a binary operator between two pairs, e.g if the edge $x \overset{s_1\ s_2}{\longrightarrow} y$ exists, then $A = (a, a')$ is the initial charge of the variable x and B is the charge of the variable y, then *balance* produces a new pair (α_x, β_x) which is the final charge for x, that is, $\#\mathrm{SAT}(\Sigma) = \alpha_x + \beta_x$.

Let $P_1 = \frac{a}{a+a'}$ and $P_0 = \frac{a'}{a+a'}$ be the proportion of the number of 1's and 0's in the initial charge of the variable x. Let $\mu_y = \alpha_y + \beta_y$. Then the charge (α_x, β_x) is computed, as:

$$\begin{array}{l} \alpha_x = \alpha_y \cdot P_1 + \beta_y;\ \beta_x = \mu_y - \alpha_x \ \text{ if } (s_1, s_2) = (+, +) \\ \beta_x = \beta_y \cdot P_0 + \alpha_y;\ \alpha_x = \mu_y - \beta_x \ \text{ if } (s_1, s_2) = (-, -) \\ \beta_x = \beta_y \cdot P_0 + \alpha_y;\ \alpha_x = \mu_y - \beta_x \ \text{ if } (s_1, s_2) = (+, -) \\ \alpha_x = \alpha_y \cdot P_1 + \beta_y;\ \beta_x = \mu_y - \alpha_x \ \text{ if } (s_1, s_2) = (-, +) \end{array} \tag{3}$$

Note that the essence of the recurrence *balance* consists in applying the inverse of the operation used at each step of the computation of $\#\mathrm{SAT}(\Sigma)$, and

following the inverse order used in the construction of the sequence $A_1, \ldots, A_n$. Thus, it follows an inverse topological order on the nodes of D_Σ.

Futhermore, in the case of the bifurcation from a father node to a list of child nodes (operation $\succ$), the same equation remains valid since each branch has its respective pair of signs. The unique new operation to consider is the case where a cycle was processed, that is, when a back edge is found when computing the charges.

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 in Σ in polynomial time.

4. Propositional Reasoning Based on the Number of Models

An important result is that we can apply the inverse action realized in each step of the procedure $Count\_Models$ in order to propagate the final charge of the last visited variable to all other variables. Then, our polynomial procedures permit us also to know the number of models for the KB Σ and its inverse procedures allow us to propagate that final charge (the number of 1's and 0's into the set of models) associated with all variable in the KB.

Knowing the charge of each variable is 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 [11].

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.

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 [9].

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.

For example, if a 2-CF Σ codify the KB of an intelligent agent which has to take decision according with a set of options $Q = \{C_1, \ldots, C_k\}$. In order to take the best decision, the intelligent agent has to recognize which option maintain the maximum of consistency with his KB. In other words, it is essential to compute the degree of belief that the agent has over each option and to select the option assuring the maximum degree of belief.

In the rest of this paper, we present only one of the applications in which to built logical structure and the charges of a 2-CF are useful. We adresse 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

We assume 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 [12].

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 #SAT(Σ) > 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, 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

$(\alpha_{\upsilon(l)}, \beta_{\upsilon(l)})$. Then, $P_{F|\Sigma} = \frac{\beta_{\upsilon(l)}}{\mu(\Sigma)}$ if l is a negated variable or $P_{l|\Sigma} = \frac{\alpha_{\upsilon(l)}}{\mu(\Sigma)}$ 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 $\upsilon(\Sigma)$. Let $\mid \upsilon(\Sigma) \mid = n$. There are 2^n assignments defined over $\upsilon(\Sigma)$ and 2^{n+k} assignments defined over $\upsilon(\Sigma) \cup \upsilon(F)$, 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:

$$P_{F|\Sigma} = \frac{\mu(\Sigma) \cdot \mu(F)}{2^k \cdot \mu(\Sigma)} = \frac{\mu(\Sigma)}{2^k \cdot \mu(\Sigma)} = \frac{1}{2^k} \tag{4}$$

   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 $\upsilon(l)$, half of those extension assignments have $\upsilon(l)$ true and the other half have $\upsilon(l)$ 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 $\{\bar{l}\}$.

Note 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, the maximum of the time for computing $P_{l|\Sigma}$ is spent for determining the position of $\upsilon(l)$ in the array *vars_pairs* which can be done in a constant time.

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 computing $P_{c|\Sigma}$:

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;
   $$P_{c|\Sigma} = \begin{cases} \frac{2 \cdot \alpha_{\upsilon(x)} + \beta_{\upsilon(x)}}{\mu(\Sigma)} & \text{if } x \text{ appears as unnegated variable,} \\ \frac{2 \cdot \beta_{\upsilon(x)} + \alpha_{\upsilon(x)}}{\mu(\Sigma)} & \text{otherwise} \end{cases}$$

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:

$$P_{F|\Sigma} = \frac{\mu(\Sigma \wedge F)}{2^t \cdot \mu(\Sigma)} = \frac{\mu(\Sigma) \cdot 2^t - \mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} = 1 - \frac{\mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} \qquad (5)$$

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: $\Sigma[s] = \Sigma[\overline{l}_1][\overline{l}_2] \ldots [\overline{l}_k]$. For short, we write $\Sigma[\overline{l}_1, \overline{l}_2, \ldots, \overline{l}_k]$ instead of $\Sigma[\overline{l}_1][\overline{l}_2] \ldots [\overline{l}_k]$. We denote with Σ' the resulting formula of applying unit reduction on Σ and s, that is, $\Sigma' = \Sigma[s]$.

Note that a unit resolution rule can generate new unitary clauses. Furthermore, it 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 $\upsilon(c) \subset \upsilon(\Sigma)$ 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 1 *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 (5), we have that* $P_{c|\Sigma_1} = 1 - \frac{4}{15} = \frac{11}{15}$.

5. Conclusions

We have designed an appropriate logical structural representation of a 2-CF knowledge base. Our model-based reasoning system 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 provide

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 elements 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. Dahllöf V., Jonsonn P., Wahlström M., Counting models for 2SAT and 3SAT formulae., Theoretical Computer Sciences 332,332(1-3): 265-291, 2005.
2. 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.
3. De Ita G., Bello P., Contreras M., Efficient counting of models for Boolean Formulas Represented by Embedded Cycles, CEUR WS Proceedings, vol. 286, 2008.
4. 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.
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. Khardon R., Roth D., Reasoning with Models, *Artificial Intelligence*, Vol. 87, No. 1, (1996), pp. 187-213.
9. Koppel M., Feldman R., Maria Segre A., Bias-Driven Revision of Logical Domain Theories, *Jour. of Artificial Intelligence Research 1*, (1994), 159-208.
10. Liberatore P., Schaerf M., The Complexity of Model Checking for Belief Revision and Update, Procc. Thirteenth Nat. Conf. on Art. Intellegence (AAAI96), 1996.
11. 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.
12. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence* 82, (1996), pp.273-302.
13. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.
14. Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.