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

Guillermo De Ita, Carlos Guillén

Fac. Cs. de la Computación, Universidad Autónoma de Puebla
Puebla, Pue., 72000 (México)
{deita,cguillen}@cs.buap.mx

Abstract Assuming a Knowledge Base (KB) Σ expressed by a two Conjunctive Form (2-CF), we determine a set of recurrence equations which allow us to design an incremental technique for counting models of Σ by a simple traversal of its constraint graph G_Σ. We show that if the constraint graph of Σ has no intersecting cycles then it is possible to count efficiently the number of models of Σ.

One of the advantages of our technique, furthermore that it has a polynomial time complexity, is that applying it backwards, we can determine the charge (the number of true and false logical values) that each variable has into the set of models of the input 2-CF.

The charge of the variables allow us to design an efficient scheme of reasoning. Given the KB Σ such that G_Σ has no intersecting cycles, and new information F (a literal or a binary clause), the degree of belief in F with respect to Σ, denoted as $P_{F|\Sigma}$, is computed efficiently even in the case $\Sigma \nvDash F$.

Keywords: #SAT Problem, Automated Reasoning, Degree of Belief, Efficient Counting.

1 Introduction

A widely accepted framework for reasoning in intelligent systems is the knowledge-based system approach [14, 11]. The idea is to keep the knowledge in some *representation language* with a well defined meaning assigned to those sentence. The sentence are stored in a Knowledge Base (KB) Σ which is combined with a reasoning mechanism, used to determine what can be inferred from the KB. For example, to decide whether a KB Σ implies a sentence α (denoted as $\Sigma \models \alpha$), even in the propositional case, is a co-NP-Hard problem [11].

Many other forms of reasoning which have been developed to avoid, at least partly, the computational difficulties for solving $\Sigma \models \alpha$, have also shown to be hard to compute [1, 3, 11, 12, 16, 18]. For example, the goal of Artificial Intelligence to provide a logic model of a human agent capable of reasoning, in the presence of incomplete and changing information, has proven to be very difficult to achieve.

There are various methods used in approximate reasoning such as; computing the degree of belief, Bayesian belief networks, knowledge compilation, constraint satisfaction, that use approximation to avoid computational difficulties and reduce them to count the number of models over propositional domains [1, 12, 15, 16, 18]. As it has been pointed out in [3, 13, 16], an important problem to explore is the computational complexity of the reasoning methods.

The #SAT problem 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, Horn propositional formulas or two conjunctive forms (problem denoted as #2SAT) [16].

Algorithms for #2SAT with a better time upper bound than the trivial $O(2^n)$, n being the number of variables of the 2-CF, have been searched for a long time ago. Nowadays, one of the exact algorithm with a leader upper bound time complexity of $O(1.2377^n)$ has been proposed by Wahlström [20], this upper bound has been obtained as a result of improvement the analysis through compound measures realized previously in the algorithm proposed by Dahllöf et al. [2] (where this analysis appears under the name of piecewise linear measures).

The previous best upper bound of $O(Poly(n) * 1.2461^n)$ for the worst case time complexity, was obtained by Fürer et al. [8] who realized a more detailed version of the compound measures analysis realized by Dahllöf et al. Wahlström realized a further improvement of the analysis through compound measures, for introducing multi-variate compound measures which are a combination of compound measures with the multi-variate recurrences of Eppstein [6].

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 also a query F in 2-CF. In our case, we perform reasoning as the computation of the proportion of the number of models of Σ which continue being model after a new formula F is processed, for processing here, we mean to compute the degree of belief.

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 and denoted as $P_{F|\Sigma}$. Assigning an equal degree of belief to all basic 'situations' consistent with the KB Σ, it means that all possible models of Σ are given equal weight, we search on logical structures derived from Σ which can be helpful for computing the fraction of models that are consistent with a propositional query F.

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 truth value 'True', $Prob$ is used to denote the probability and $Prob(\Sigma \equiv \top)$ means the proportion of the number of assignments over 2^n which satisfy to the Boolean formula Σ.

The degree of belief of the agent in a propositional formula F (with respect to its knowledge base Σ) is the fraction of models of Σ that are consistent with the formula F, that is, the conditional probability of F with respect to Σ, which is denoted as $P_{F|\Sigma} = Prob((\Sigma \wedge F) \equiv \top | \Sigma \equiv \top) = \frac{\#SAT(\Sigma \wedge F)}{\#SAT(\Sigma)}$.

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*, i.e. there exists a polynomial time complexity algorithm for computing $P_{F|\Sigma}$, and for this, the development of smart algorithms for solving the #SAT is key.

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 at most twice) [16, 17]. Here, we extend such class by considering the topological structure of the constraint graph induced by the formula, and we show in section 3.1 that the #2SAT problem is solved in polynomial time for a 2-CF whose constraint graph has no intersecting cycles.

We focus toward the identification of structures on the constraint 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, as the constraint graph of the formula is traversed.

Also, we propose a procedure for determining the charge of each variable in a 2-CF Σ, the charge is the number of true and false logical values that a variable has into the set of models of Σ. To Know 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 [15].

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 this 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 a priori probability that each element of the domain theory requires revision [12].

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 this article, we present an important application in which to know the charges of a 2-CF is useful. We address the problem of computing the degree of belief of an intelligent agent on new information and we show how the computation of the degree of belief can be done efficiently for some classes of 2-CF's.

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 $k \in \mathbb{N}$, 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 variables that appear 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 $X = \upsilon(F) = \{x_1, \ldots, x_n\}$, then $Lit(F) = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$. We denote $\{1, 2, ..., n\}$ by $[\![n]\!]$.

An assignment s for F is a boolean function $s : \upsilon(F) \rightarrow \{0, 1\}$. An *assignment* can be also considered as a set of literals with no complementary pair. 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 and $SAT(F)$ denotes the set of models of F. The #SAT problem (or #SAT(F))consists of counting the number of models of F defined over $\upsilon(F)$. #2-SAT denotes #SAT for formulas in 2-CF. We denote #SAT(F) by $\mu_V(F)$ for a set of variables $V \supseteq \upsilon(F)$, or just by $\mu(F)$ when $V = \upsilon(F)$.

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

Each edge has associated an ordered pair (s_1, s_2) of signs, assigned as labels. Let $S = \{+, -\}$ be a set of signs, and let ψ be a function with domain $E(\Sigma)$ and range $S \times S$. $\psi(e)$ gives the signs of the edge $e \in E(\Sigma)$. For example, for the clause $\{\overline{x} \vee y\} \in \Sigma$ and assuming that its respective edge in G_Σ, is e then $(s_1, s_2) = \psi(e)$ is the pair of signs related to the signs of the literals x and y respectively, then $s_1 = -$ and $s_2 = +$ and the edge is denoted as: $x \overset{-\,+}{\rule{1.2em}{0.4pt}} y$ which is equivalent to $y \overset{+\,-}{\rule{1.2em}{0.4pt}} x$. s_1 (s_2) is called the *adjacent sign* of x (y).

3   Linear procedures for #2SAT

In order to compute $\#SAT(\Sigma)$, first we should determine the set of connected components of G_Σ and that can be done in linear time. And, $\#SAT(G_\Sigma) = \prod_{i=1}^{k} \#SAT(G_i)$, $\{G_1, \ldots, G_k\}$ being the connected components of G_Σ [16]. 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*, or a *tree* if its corresponding constraint graph G_Σ is a path, a cycle, or a tree, respectively.

For each variable $x \in \upsilon(\Sigma)$, Σ a 2-CF, 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 takes when $\#SAT(\Sigma)$ is being computed.

In previous papers, we have presented a set of recurrence equations to compute $\#SAT(\Sigma)$ at the same time that G_Σ is traversing [4, 5, 9]. For completeness purposes, we present in this section some linear procedures for computing #2-SAT, being the new contributions of this section; the soundness proof of our first procedure and the last general procedure to show in section 3.1.

Procedure A: If Σ is a path:
Let Σ be a path (or a linear chain). Σ can be written (ordering clauses and variables, if it were necessary) as: $\Sigma = \{c_1, ..., c_m\} = \left\{ \{y_0^{\epsilon_1}, y_1^{\delta_1}\}, \{y_1^{\epsilon_2}, y_2^{\delta_2}\}, ..., \{y_{m-1}^{\epsilon_m}, y_m^{\delta_m}\} \right\}$, where $\delta_i, \epsilon_i \in \{0, 1\}$, $i \in [\![m]\!]$ and $|\upsilon(c_j) \cap \upsilon(c_{j+1})| = 1$, $j \in [\![m-1]\!]$. As Σ has m clauses then $|\upsilon(\Sigma)| = n = m + 1$.

Let f_i be a family of clauses from Σ built as follows: $f_0 = \emptyset$; $f_i = \{c_j\}_{j \le i}$, $i \in [\![m]\!]$. Notice that $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisfies } f_i\}$, $A_i = \{s \in SAT(f_i) : y_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{y_i} \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

The first pair (α_0, β_0) is (1,1) since for any logical value to y_0, f_0 is satisfied. We compute (α_i, β_i) associated with each variable y_i, $i = 1, .., m$, according to the signs: ϵ_i, δ_i of the literals in the clause c_i, by the following recurrence equations:

$$(\alpha_i, \beta_i) = \begin{cases} (\beta_{i-1}, \mu_{i-1}) & \text{if } (\epsilon_i, \delta_i) = (0, 0) \\ (\mu_{i-1}, \beta_{i-1}) & \text{if } (\epsilon_i, \delta_i) = (0, 1) \\ (\alpha_{i-1}, \mu_{i-1}) & \text{if } (\epsilon_i, \delta_i) = (1, 0) \\ (\mu_{i-1}, \alpha_{i-1}) & \text{if } (\epsilon_i, \delta_i) = (1, 1) \end{cases} \tag{1}$$

As $\Sigma = f_m$ then $\#SAT(\Sigma) = \mu_m = \alpha_m + \beta_m$. We denote with $' \to '$ the application of one of the four rules in the recurrence (1).

Example 1 *Let $\Sigma = \{(x_0, x_1), (\overline{x}_1, \overline{x}_2), (\overline{x_2}, \overline{x}_3), (x_3, \overline{x}_4), (\overline{x}_4, x_5)\}$ be a path, the series $(\alpha_i, \beta_i), i \in [\![5]\!]$, is computed according to the signs of each clause, as: $(\alpha_0, \beta_0) = (1, 1) \to (2, 1) \to (1, 3) \to (3, 4) \to (3, 7) \to (10, 7)$ and then, $\#SAT(\Sigma) = \mu_5 = \alpha_5 + \beta_5 = 10 + 7 = 17$ (see fig. 1).*
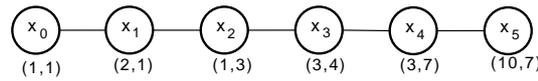

Fig.1 Counting models over paths

As the procedure (A) is a common base for the following procedures to present, we consider relevant to show its soundness proof.

Lemma 1 Soundness of the procedure (A)
Given a path formula Σ, the result of applying recurrence (1) in accordance with the linear traversing of the path, computes $\#SAT(\Sigma)$

Proof.
Let G_Σ be a path with m edges: $y_0 y_1, y_1 y_2, ..., y_{m-1} y_m$. Let f_i be the family of clauses from Σ built previously, that is, $f_0 = \emptyset$, $f_i = \{c_j\}_{j \le i}$, $i \in [\![m]\!]$. $\#SAT(\Sigma)$ is computed in incremental way, i.e. $\#SAT(f_i)$ is computed for each $i = 0, ..., m$.

$A_i = \{s \in SAT(f_i) : y_i \in s\}$ and $B_i = \{s \in SAT(f_i) : \overline{y_i} \in s\}$ conform a partition of $SAT(f_i)$ since $A_i \cap B_i = \emptyset$ and $|SAT(f_i)| = \alpha_i + \beta_i$ with $\alpha_i = |A_i|$ and $\beta_i = |B_i|$. The first pair (α_0, β_0) is (1,1) since for any logical value to y_0, f_0 is satisfied. Notice that $f_i - f_{i-1} = c_i = \{y_{i-1}^{\epsilon_i}, y_i^{\delta_i}\}$. Let us consider the case: $\epsilon_i = \delta_i = 1$. We extend the set of models in $SAT(f_{i-1})$ with the correct value for the variable y_i in order to form the members of $SAT(f_i)$. All model in $SAT(f_{i-1})$ containing y_{i-1} can be

extended with y_i or $\overline{y}_i$ for satisfying c_i and there are α_{i-1} models of this type. A model in $SAT(f_{i-1})$ containing $\overline{y}_{i-1}$ satisfies c_i only if it is extended with y_i, and there are β_{i-1} of such models. According to both cases; there will be $\alpha_i = \alpha_{i-1} + \beta_{i-1}$ models in $\mathrm{SAT}(f_i)$ containing y_i and $\beta_i = \alpha_{i-1}$ containing $\overline{y}_i$. This is how the first line of recurrence (1) is obtained. The other three lines of the same recurrence are obtained in similar way, according to the combination of the signs ϵ_i, δ_i of the clause $c_i = \{y_{i-1}^{\epsilon_i}, y_i^{\delta_i}\}$. $\square$

When we count models over a constraint graph, 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.
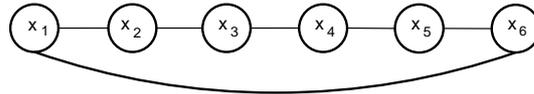
Procedure B: If Σ is a simple cycle:
Let $G_\Sigma = (V, E)$ be a constraint graph of Σ which is a simple cycle with m nodes, $|V| = |E| = m$. Ordering the clauses of Σ in such way that $\mid v(c_i) \cap v(c_{i+1}) \mid = 1$, and $y_{i_1} = y_{i_2}$ whenever $i_1 \equiv i_2 \mod m$, hence $y_0 = y_m$, then $\Sigma = \left\{c_i = \{y_{i-1}^{\epsilon_i}, y_i^{\delta_i}\}\right\}_{i=1}^m$, where $\delta_i, \epsilon_i \in \{0, 1\}$.

Decomposing Σ as $\Sigma = \Sigma' \cup c_m$, where $\Sigma' = \{c_1, ..., c_{m-1}\}$ is a path and $c_m = (y_{m-1}^{\epsilon_m}, y_m^{\delta_m})$ is the edge which forms with $G_{\Sigma'}$ the simple cycle: $y_0, y_1, ..., y_{m-1}, y_0$. The procedure (A) is applied for computing $\#SAT(\Sigma')$ through a thread $(\alpha_i, \beta_i), i = 0, \ldots, m-1$.

Every model of Σ' had determined logical values for the variables: y_{m-1} and y_m since those variables appear in $v(\Sigma')$. Any model s of Σ' satisfies c_m if and only if $(y_{m-1}^{1-\epsilon_m} \notin s$ and $y_m^{1-\delta_m} \notin s)$, that is, $SAT(\Sigma' \cup c_m) \subseteq SAT(\Sigma')$, and $SAT(\Sigma' \cup c_m) = SAT(\Sigma') - \{s \in SAT(\Sigma') : s \text{ falsifies } c_m\}$. Let $Y = \Sigma' \cup \{(y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m})\}$, $\#SAT(Y)$ could be also computed by a new thread $(\alpha_i', \beta_i'), i = 0, \ldots, m-1$ since Y represents a path with two unitary clauses at the extremes of the path. Then,

$$\#SAT(\Sigma) = \mu(\Sigma') - \mu(Y) =$$
$$= \mu(\Sigma') - \mu(\Sigma' \wedge (y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m})) \qquad (2)$$

Notice that the thread for computing $\mu(\Sigma' \wedge (y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m}))$ starts with (0,1) if $\delta_m = 1$ or with (1,0) if $\delta_m = 0$. By other hand, at the end of the thread (α_i', β_i'), the last pair $(\alpha_{m-1}', \beta_{m-1}')$ changes to $(0, \beta_{m-1})$ if $\epsilon_m = 1$ or it changes to $(\alpha_{m-1}, 0)$ if $\epsilon_m = 0$.

$$(\alpha_i, \beta_i): (1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (5, 3) \rightarrow (8, 5) \rightarrow (13, 8)$$
$$(\alpha_i', \beta_i'): (0, 1) \rightarrow (1, 0) \rightarrow (1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (5, 3)$$
$$\Rightarrow (13, 8) - (0, 3) = (13, 5)$$

Fig.2 Counting models over cycles

Example 2 *Let $\Sigma = \{c_i\}_{i=1}^6 = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ be a monotone 2-CF. Its constraint graph $G_\Sigma = (V,E)$ is a simple cycle. Let $G_{\Sigma'} = (V, E')$ where $E = E' \cup \{c_6\}$ that is, the new graph $G_{\Sigma'}$ is Σ minus the edge c_6. As G_Σ is a path with 6 nodes then $\#SAT(G') = \alpha_6 + \beta_6 = 13 + 8 = 21$. While the computation of $\mu(Y)$ is given by the series: $(0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3)$, obtaining as last pair associated with $\mu(Y)$ the pair $(0, 3)$. The pair associated with the last node of Σ is $(13, 8) - (0, 3) = (13, 5)$. And then $\#SAT(\Sigma) = 13 + 5 = 18$. Figure 2 illustrates the processing of such cycle.*

Notice that the class of graphs induced by $(2, 2\mu)$-CF's are exactly paths or simple cycles. Then, this class of formulas can be processed by our procedures (A) and (B)

There are other procedures for computing $\#SAT(\Sigma)$ when Σ is a $(2, 2\mu)$-CF [16, 17], but these proposals do not distinguish the number of models in which a variable x takes value 1 and when it takes value 0 (the charge of the variable), situation important when we want to compute the degree of belief of an intelligent agent on new information, as we will show in section 4.

3.1   Computation of #2SAT for formulas without intersecting cycles

Let $G_\Sigma = (V, E)$ be a connected graph of an input formula Σ in 2-CF, n and m being the number of variables and m the number of clauses of Σ, respectively. 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. Note that a pair of independent cycles can share a common node (but not edges), e.g. see figure 3.

Let A_Σ be the depth-first search graph of G_Σ formed by the spanning tree T_G and the set of k fundamental cycles $\mathcal{C}$. Notice that A_Σ is obtained in linear time $O(n + m)$ over the size of the graph G_Σ, while to determine if there exists intersecting cycles in A_Σ is done in polynomial time $O(k^2)$ over the number of fundamental cycles, since it consists in verify if the result of the intersection between all pair of fundamental cycles is the emptyset, i.e. for $i, j = 1, \ldots, k, i \neq j$ check if $(E(C_i) \cap E(C_j)) = \emptyset$.

We transform G_Σ to a Directed Acyclic Graph (DAG), denoted by D_Σ, assigning an orientation to each edge $\{u, v\}$ in G_Σ by directing: $u \to v$ if v is an ancestor node of u in T_G (see figures 3 and 4). For a node v in D_Σ, let $\delta_{in}(v) = |\{w : w \to v\}|$ and $\delta_{out}(v) = |\{w : v \to w\}|$ be its input and output degree, respectively. 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 $u \to v$.

There are two basic structures in D_Σ: *Branches* and *Rings*. A branch B_s of D_Σ is a directed sequence of nodes which starts in a node v_s where $\delta_{in}(v_s){=}0$ and $\delta_{out}(v_s){=}1$. The internal nodes of the sequence are reached from v_s through a path formed by nodes v_i where $\delta_{in}(v_i) = 1 = \delta_{out}(v_i)$. The branch ends in the first node v_o such that $\delta_{in}(v_o) > 1$ or $\delta_{out}(v_o) > 1$, then v_o is reached through the linear path started in v_s.

A ring R_s of D_Σ is the subgraph formed by the edges and nodes that are part of a fundamental cycle in T_G. So, for each cycle $C_i \in C$, there is a ring R_i, $i{=}1, ...k$. The start-node of R_s is the node v_s that is part of the back edge $c_i = \{v_s, v_o\}$ and $\delta_{out}(v_s){=}2$, while the second node v_o of the back edge is called the end-node of the ring.

The topological order among nodes is extended to branches and rings of D_Σ according with the topological order number associated with the end-node of each substructure. This ordering is a guide for processing the substructures of D_Σ. For example, the order for visiting the edges of the DAG in fig. 4, is: $x_8 \to x_6; x_9 \to x_6; x_6 \to x_5; x_9 \to x_5; x_5 \to x_4; x_5 \to x_2; x_4 \to x_2; x_7 \to x_3; x_3 \to x_2; x_2 \to x_1; x_7 \to x_1$.
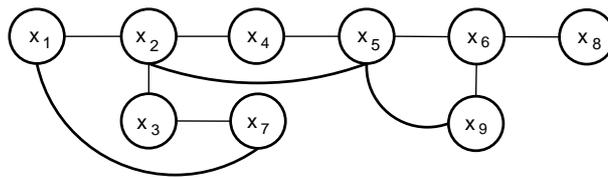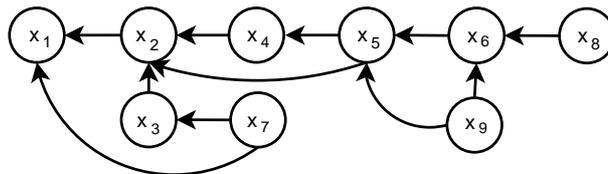

Fig.3 The constraint graph of a 2-CF

Fig.4 The DAG D_Σ associated with the previous constraint graph, being x_1 the root node

Theorem 1 *If the constraint graph of a formula Σ in 2-CF has no intersecting cycles then #2SAT(Σ) can be computed in polynomial time.*

We present a constructive proof of this theorem through the design of a polynomial time algorithm which computes #2SAT for this class of formulas. We have presented in [5] a procedure for counting the number of models of a 2-CF (the #2SAT problem) where its constraint graph has no intersecting cycles. We sketch here a more refined and simple algorithm than the previous one for solving #2SAT to the same class of constraint graphs.

Our new procedure, called $Count\_Models$, goes traversing for each substructure of the DAG D_Σ (the corresponding DAG of the constraint graph of Σ) and at the same time, the initial charge (α_x, β_x) for each variable $x \in \upsilon(\Sigma)$ is computed. At the beginning of the procedure, each node $x \in D_\Sigma$ is associated with the pair (α_x, β_x) which is initialized to (0,0).

Procedure $Count\_Models(\Sigma)$
Input: D_Σ: a constraint directed graph of a 2-CF Σ
Output: The charge (α_y, β_y) associated with the root node y, where $\#SAT(\Sigma) = \alpha_y + \beta_y$

$\#SAT(\Sigma)$ is computed traversing each substructure in D_Σ according to the topological order of each substructure. A main thread, denoted by Lp, is associated with the spanning tree of D_Σ. This thread is always active until $Count\_Models$ finishes. When a node v is visited, the pair (α_v, β_v) is computed depending on the substructure in which this node appears, as follows:

1. When a branch B_s is evaluated, the basic procedure (Case A) is applied. The evaluation of the branch starts in its initial node v_s, associating the pair $(\alpha_s, \beta_s)=(1,1)$ when v_s does not have a stored value yet. After evaluating the branch, its initial node and its internal nodes are removed from D_G as well as its incident edges. The end-node v_o is the only node of the branch preserved maintaining in (α_o, β_o) the final value obtained after processing the whole branch.

2. When a ring R_s is evaluated, the basic procedure for processing simple cycles (Procedure B) is applied. The processing of R_s starts with the start-node v_s and two computing threads are created with initial pairs; (α_s, β_s) and (α'_s, β'_s). If v_s has already a value stored in (α_s, β_s) (v_s has been visited before) then $(\alpha'_s, \beta'_s) = (0, \beta_s)$ if v_s occurs in non-negative way in the back edge or $(\alpha'_s, \beta'_s) = (\alpha_s, 0)$ otherwise. If v_s has not been visited previously then $(\alpha_s, \beta_s)=(1,1)$ and $(\alpha'_s, \beta'_s) = (0, 1)$ if v_s occurs in non-negative way in the back edge or $(\alpha'_s, \beta'_s) = (1, 0)$ otherwise. After evaluating the ring R_s, all of its nodes as well as its associated edges are removed from D_G preserving only the end-node v_o of R_s. The pair associated to v_o is $(\alpha_o, \beta_o - \beta'_o)$ if v_o occurs in non-negative way in the back edge or $(\alpha_o - \alpha'_o, \beta_o)$ otherwise, such as it occurs in the procedure (B).

   When a node v is considered, if it has been visited before then a pair $(\alpha_{v_1}, \beta_{v_1})$ has already been associated with v. When v is visited again, a new pair $(\alpha_{v_2}, \beta_{v_2})$ is obtained. The final pair (α_v, β_v) associated with v is computed as: $\alpha_v = \alpha_{v_1} * \alpha_{v_2}$ and $\beta_v = \beta_{v_1} * \beta_{v_2}$, since two different processing lines have been met in a common node.

3. The two previous steps are applied repeatedly, since the internal nodes of rings and branches are removed from D_G and new branches could appear. The topological order number assigned to the end-node of the emergent branches determines the order for processing such new branches.

4. When the root node y of D_Σ is visited then returns its computed charge: (α_y, β_y).

Notice that when a branch and a ring have the same topological number, the branch is evaluated before the ring. For illustrating this procedure, we apply it on the DAG of the figure 4, assuming a monotone 2-CF, that is, all variable appear in nonnegative way and then the last rule of recurrence (1) is always applied. In figure 5, we denote as an subtracted operator the application of the formula (2).
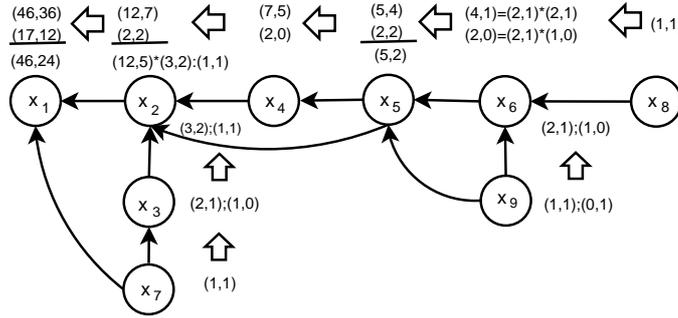
(46,36)   (12,7)   (7,5)   (5,4)   (4,1)=(2,1)*(2,1)
(17,12)   (2,2)   (2,0)   (2,2)   (2,0)=(2,1)*(1,0)   (1,1)
(46,24)   (12,5)*(3,2):(1,1)   (5,2)

x_1 ← x_2 ← x_4 ← x_5 ← x_6 ← x_8
          (3,2);(1,1)   (2,1);(1,0)

          x_3 (2,1);(1,0) x_9 (1,1);(0,1)

          x_7 (1,1)

Fig.5 Computing the initial charge of the variables

Notice that all the procedures involved in *Count_Models* such as: depth first search, topological sorting, processing rings and branches, and so on, all of them are linear-time procedures in the size of the graph. Thus, this main procedure has a time complexity of $O(n + m)$, n being the number of variables and m the number of clauses of Σ. While to recognize if a constraint graph has no intersecting cycles is done also in polynomial time complexity.

In [4, 10], 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.

4   Computation of the Charges of a 2-CF

We present here, novel procedures for computing the charges of the variables of a 2-CF whose constraint graph has physical topologies like: paths, trees and independent cycles.

Suppose that *Count_Models* has been applied for computing #SAT(Σ), and the initial charges for all variables of Σ have already been computed. We have introduced in [9] the concept of final charge (or just the charge) (a_x, b_x) of a variable $x \in \upsilon(\Sigma)$ as the number of true and false logical values respectively, that x takes into the set of models of Σ, i.e. #SAT(Σ) = $a_x + b_x$. Notice that the initial charge (α_y, β_y) for the last evaluated variable y in *Count_Models* (the root node of the spanning tree T_G) is also its final charge since #SAT(Σ) = $\alpha_y + \beta_y$.

In that previous work, we design a matrix method for computing the charge of any variable which appears in a path graph. In this section, we propose a new method, based on new recurrence equations, which acts as the inverse operations done with the equations (1), (2), and following an inverse topological order followed by the procedure *Count_Models*.

This new method for computing the charges of the variables allow us to compute the charges not only for variables over paths as it was shown in [9], but also for cycles, trees and in general, for constraint graphs without intersecting cycles. In this chapter we show how to apply the inverse action realized in each step of the procedure *Count_Models* in order to propagate the final charge to all variables in Σ.

Let $A_1, \ldots, A_n$ be the sequence of initial charges obtained by the procedure *Count_Models*. Now, we build a new sequence of pairs which represent the final charges (or just the charges) $B_n, \ldots, B_1$, being B_i the charge of the variable $x_i \in \upsilon(\Sigma)$, and which is computed as:

$$\begin{aligned} B_n =&\ A_n \\ B_{n-i} =&\ balance(A_{n-i}, B_{n-i+1}),\ i = 1, ..., n-1 \end{aligned} \tag{3}$$

$balance(A, B)$ is a binary operator between two pairs, e.g. if $x \overset{s_1\ s_2}{\mapsto} y$ is an edge of the DAG D_Σ and assuming $A = (\alpha_x, \beta_x)$ be the initial charge of the variable x, $B = (a_y, b_y)$ be the final charge of the variable y, then *balance* produces a new pair (a_x, b_x) which will be the final charge for x, i.e. #SAT(Σ) = $a_x + b_x$.

Let $\mu_x = \alpha_x + \beta_x$ and $\mu_y = a_y + b_y$. Let $P_1 = \frac{\alpha_x}{\mu_x}$ and $P_0 = \frac{\beta_x}{\mu_x}$ be the proportion of the number of 1's and 0's in the initial charge of the variable x. The charge (a_x, b_x) is computed, as:

$$
\begin{aligned}
a_x &= a_y \cdot P_1 + b_y; \, b_x = \mu_y - a_x \quad &\text{if}(s_1, s_2) = (+, +) \\
b_x &= b_y \cdot P_0 + a_y; \, a_x = \mu_y - b_x \quad &\text{if}(s_1, s_2) = (-, -) \\
b_x &= b_y \cdot P_0 + a_y; \, a_x = \mu_y - b_x \quad &\text{if}(s_1, s_2) = (+, -) \\
a_x &= a_y \cdot P_1 + b_y; \, b_x = \mu_y - a_x \quad &\text{if}(s_1, s_2) = (-, +)
\end{aligned}
\tag{4}
$$

Note that the essence of the rules in *balance* consists in applying the inverse operation utilized via recurrence (1) during the computation of #SAT(Σ), 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.

Furthermore, in the case of the bifurcation from a father node to a list of child nodes the application of the recurrence (4) remains valid since each branch has its respective pair of signs.

A special case is the propagation of the charge into the nodes of a ring since for this substructure there were used two computing threads in the procedure *Count_Models*. For explaining the computation of these charges, let us consider a ring R_s with start-node v_s, end-node v_o and where the two computing threads; $L_s : (\alpha_s, \beta_s), (\alpha_{s+1}, \beta_{s+1}), \ldots, (\alpha_o, \beta_o)$ and $L_s' : (\alpha_s', \beta_s'), (\alpha_{s+1}', \beta_{s+1}'), \ldots, (\alpha_o', \beta_o')$ were utilized. The equations in (4) are lightly modified considering a new auxiliary thread which has to do the inverse action that L_s' did when the ring was processed. Let us represent such auxiliary thread as $temp$: $(t\alpha_s, t\beta_s), (t\alpha_{s+1}, t\beta_{s+1}), \ldots, (t\alpha_o, t\beta_o)$.

The order for computing the charges to the variables in $\upsilon(R_s)$ is inverse to the order followed during the processing of R_s. For example, if $x \overset{s_1 \, s_2}{\rightarrow} y$ is an edge of R_s, and assuming the values defined previously; $A = (\alpha_x, \beta_x)$ - the initial charge of x, $B = (a_y, b_y)$ - the final charge of y, $\mu_x = \alpha_x + \beta_x$, $\mu_y = a_y + b_y$, $P_1 = \frac{\alpha_x}{\mu_x}$ and $P_0 = \frac{\beta_x}{\mu_x}$ are the proportion of the number of 1's and 0's in the initial charge of x, and $(t\alpha_y, t\beta_y)$ is the corresponding pair associated with y into the auxiliary thread $temp$. Then, the charge (a_x, b_x) is computed, as:

$$
\begin{aligned}
a_x &= (a_y + t\alpha_y) \cdot P_1 + (b_y - t\beta_y); \, b_x = \mu_y - a_x \quad &\text{if}(s_1, s_2) = (+, +) \\
b_x &= (b_y + t\beta_y) \cdot P_0 + (a_y - t\alpha_y); \, a_x = \mu_y - b_x \quad &\text{if}(s_1, s_2) = (-, -) \\
b_x &= (b_y + t\beta_y) \cdot P_0 + (a_y - t\alpha_y); \, a_x = \mu_y - b_x \quad &\text{if}(s_1, s_2) = (+, -) \\
a_x &= (a_y + t\alpha_y) \cdot P_1 + (b_y - t\beta_y); \, b_x = \mu_y - a_x \quad &\text{if}(s_1, s_2) = (-, +)
\end{aligned}
\tag{5}
$$

We explain now how to compute the temporal thread $temp$. First, an auxiliary new thread, that we call r_s, is built based on L_s' and following the inverse order used when L_s' was built. The first pair of r_s is: $(r\alpha_o, r\beta_o) = (1, 1)$, and for the following edges of R_s (excepting the back edge), the rule applied during the computation of L_s' is now used for computing its respective pair into the thread $temp$. E.g. if $u \overset{s_1 \, s_2}{\rightarrow} v$ is an edge of R_s and the initial charge for the variables u and v was computed in L_s', as: $(\alpha_u', \beta_u') \overset{s_1 \, s_2}{\rightarrow} (\alpha_v', \beta_v')$. Then $(r\alpha_v, r\beta_v)$ is known and we apply the corresponding rule of (1) with the signs (s_2, s_1) for computing $(r\alpha_u, r\beta_u)$. i.e. $(r\alpha_v, r\beta_v) \overset{s_2 \, s_1}{\rightarrow} (r\alpha_u, r\beta_u)$.

Ordering the pairs of the threads L_s' and r_s from the end-node to the start-node of the ring R_s; $L_s' : (\alpha_o', \beta_o'), (\alpha_{o-1}', \beta_{o-1}'), \ldots, (\alpha_s', \beta_s')$ and $r_s : (r\alpha_o, r\beta_o), (r\alpha_{o-1}, r\beta_{o-1}), \ldots, (r\alpha_s, r\beta_s)$. The thread $temp$ is built in the order $(t\alpha_o, t\beta_o), (t\alpha_{o-1}, t\beta_{o-1}), \ldots, (t\alpha_s, t\beta_s)$ and each pair is computed from the previous threads L_s' and r_s as: $temp : (0, 0), (\beta_o', \beta_{o-1}' * r\beta_o), (\beta_{o-1}' * r\beta_{o-1}, \beta_{o-2}' * r\beta_{o-1}), \ldots, (0, 0)$.

Notice that the computation of all computing thread, as well as 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 constraint graph of Σ does not contain intersecting cycles, then we compute all the charges of the variables of Σ in polynomial time.

If a 2-CF Σ codify the KB of an intelligent agent who has to take a decision according with a set of options $Q = \{C_1, \ldots, C_k\}$. We can search for the options inferred from Σ, but if we assume that $\Sigma \nvDash C_i$ for all $i = 1, \ldots, k$ then the deduction methods do not provide to the agent enough information to select the closest option to be consistent with Σ.

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 (although such options are not deduced from his KB) and to select the option assuring the maximum degree of belief.

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.g. statistical information, is to use the computation of a degree of belief, as an effort to avoid the computationally hard task of deductive inference [16].

Let F be a query which is a unitary or binary clause and let Σ be a satisfiable 2-CF which represents a KB, then $\#SAT(\Sigma) > 0$ and the degree of belief of F with respect to Σ, denoted as $P_{F|\Sigma} = \frac{\#SAT(\Sigma \wedge F)}{\#SAT(\Sigma)}$, is well-defined. We assume that Σ has been processed by the above procedures and that 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, i.e. checking for $i = 1, \ldots, k, \Sigma \models l_i$, recognizing equal value to all literals consistent with Σ.

But $P_{l|\Sigma}$ provides more information than $\Sigma \models l_i$ gives. $P_{l|\Sigma}$ gives the proportion of the original models of Σ that remains models for $(\Sigma \wedge l)$. This 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. We show now how to compute $P_{l|\Sigma}$ and for this, we have two cases:

1. $l \in Lit(\Sigma)$: As every variable $x_i \in \upsilon(\Sigma)$ has associated its respective charge (α_i, β_i), we use $\upsilon(l)$ 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 [7]. Let us consider here, a more general case.

   Let $F = (\bigwedge_{j=1}^{k} l_j)$ be a conjunction of literals such that each variable in F does not appear in $\upsilon(\Sigma)$. Let $| \upsilon(\Sigma) | = n$. 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:

   $$P_{F|\Sigma} = \frac{\mu(\Sigma) \cdot \mu(F)}{2^k \cdot \mu(\Sigma)} = \frac{\mu(\Sigma)}{2^k \cdot \mu(\Sigma)} = \frac{1}{2^k} \qquad (6)$$

   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. The fraction of models of Σ which are consistent with $\{l\}$ are $1/2$ and the other half are consistent with $\{\bar{l}\}$.

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 constraint 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;

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

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 $A = \{l \in F | v(l) \notin v(\Sigma)\}$ be the literals in F whose variables do not appear in $v(\Sigma)$ and let $F' = F - A$ be the literals in F whose variables appear in $v(\Sigma)$, let $t = | A |$.

We compute $\mu(\Sigma \wedge F)$ by extending the models of Σ with the new variables $v(A)$ 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:

$$P_{F|\Sigma} = \frac{\mu(\Sigma \wedge F)}{2^t \cdot \mu(\Sigma)} = \frac{\mu(\Sigma) \cdot 2^t - \mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} = 1 - \frac{\mu(\Sigma \wedge \overline{F'})}{2^t \cdot \mu(\Sigma)} \tag{7}$$

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 $v(\Sigma) \cup v(\overline{F})$ 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, i.e. $\Sigma[l]$. 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, i.e. $\Sigma' = \Sigma[s]$.

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 do not appear more in the new formula 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 y is not member of $v(\Sigma')$, 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 variables of $v(\Sigma)$ which disappear from $v(\Sigma')$ 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.

The analysis presented in this subsection on the different cases for computing $P_{c|\Sigma}$, Σ being a 2-CF and c being a unary or binary clause, it allows us to show that if we know the charges of the variables in Σ, then for all the previous cases (with exception of the last one), $P_{c|\Sigma}$ is computed in a constant time complexity. Although, just for the case where $\upsilon(c) \subset \upsilon(\Sigma)$ and $\Sigma \nvDash c$ the computation of $P_{c|\Sigma}$ could require almost the same time that computing $\#SAT(\Sigma)$.

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 $\#SAT(\Sigma)$ 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) which were modified from G_{Σ} to $G_{\Sigma'}$.

Example 3 *Let $\Sigma_1 = \{c_i\}_{i=1}^{7} = \{\{x_1, x_2\}, \{x_2, \overline{x}_3\}, \{\overline{x}_3, x_4\}, \{\overline{x}_4, \overline{x}_5\}, \{x_1, \overline{x}_4\}, \{\overline{x}_3, \overline{x}_5\}, \{x_2, \overline{x}_6\}\}$ 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 (7), we have that $P_{c|\Sigma_1} = 1 - \frac{4}{15} = \frac{11}{15}$.*

5   Conclusions

We present novel procedures for computing the charge of the variables of a 2-CF Σ, the charge is the number of true and false logical values that a variable has into the set of models of Σ. We show that the charge of the variables of Σ can be computed efficiently when the constraint graph of Σ has physical topologies like: paths, trees or when its cycles are independent.

The charge of the variables is helpful for determining the relative value for all elements in Σ, which is an essential problem in some applications of reasoning. For example, we present a model-based approach for approximate reasoning based on the computation of the degree of belief of an intelligent agent.

If the Knowledge Base Σ of an intelligent agent is given by a 2-CF such that G_{Σ} has no intersecting cycles, and if we know the charge of the variables of Σ, then the computation of the degree of belief in new information F (a literal or a binary clause), denoted as $P_{F|\Sigma}$, is done efficiently even if $\Sigma \nvDash F$.

References

[1] Bacchus F., Grove A.J., Halpern J.Y., Keller D., From statistical knowledge bases to degrees of belief, *Artificial Intelligence 87*, (1996), pp. 75-143.

[2] Dahllöf V., Jonsonn P., Wahlström M., Counting models for 2SAT and 3SAT formulae., *Theoretical Computer Sciences 332*, 332(1-3): 265-291, 2005.

[3] 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.

[4] De Ita G., Bello P., Contreras M., Efficient counting of models for Boolean Formulas Represented by Embedded Cycles, CEUR WS Proceedings, vol. 286, 2007.

[5] De Ita G, Bello P., Contreras M., New Polynomial Classes for #2SAT Established Via Graph-Topological Structure, *Jour. Engineering Letters*, Vol. 15, No. 2, (2007), pp.250-258.

[6] Eppstein D., Quasiconvex analysis of backtracking algorithms, Procs. of the 15th annual ACM-SIAM symp. on Discrete algorithms (SODA 2004), pp. 788-797.

[7] 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.

[8] Fürer M., Kasiviswanathan S.P., Algorithms for Counting 2-SAT Solutions and Colorings with Applications, *LNCS 4508*, AAIM 2007.

[9] Guillén C., De Ita G., López-López A., Efficient Computation of the Degree of Belief in a Propositional Theory, CEUR WS Proceedings, Vol. 408, 2008.

[10] Guillén C., López-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.

[11] Khardon R., Roth D., Reasoning with Models, *Artificial Intelligence*, Vol. 87, No. 1, (1996), pp. 187-213.

[12] Koppel M., Feldman R., Maria Segre A., Bias-Driven Revision of Logical Domain Theories, *Jour. of Artificial Intelligence Research 1*, (1994), 159-208.

[13] Liberatore P., Schaerf M., The Complexity of Model Checking for Belief Revision and Update, Procc. Thirteenth Nat. Conf. on Art. Intellegence (AAAI96), 1996.

[14] McCarthy J., Programs with common sense, *Readings in Knowledge Representation*, R. Brachman and H. levesque, 1985.

[15] 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.

[16] Roth D., On the hardness of approximate reasoning, *Artificial Intelligence* 82, (1996), pp.273-302.

[17] Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.

[18] Selman B., *Tractable Default Reasoning*, PhD. thesis, Department of Computer Science, Univ. of Toronto, 1990.

[19] Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.

[20] Wahlström M., A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances, *LNCS 5018*, Springer-Verlag (2008), pp. 202-213.