Applying Counting Models of Boolean Formulas to Propositional Inference

Guillermo De Ita Luna[1]* , Mireya Tovar[2]

[1] Universidad Politécnica de Puebla, deita@ccc.inaoep.mx
[2] Universidad Autónoma de Puebla, mtovar@cs.buap.mx

Abstract. We address the problem of designing efficient procedures for counting models of Boolean formulas and, in this task, we establish new polynomial classes for #2SAT determined via the topological structure of the underlying graph of the formulas.
Although #2SAT is a classical #P-complete problem, we show that, if the depth-first search of the constraint graph of a formula generates a free tree and a set of fundamental independent cycles, this is, there are not common edges neither common nodes among such fundamental cycles, then to count the number of different models of the formula can be computed in polynomial time, in fact, in linear time.
The new polynomial class of 2-CF's brings us a new paradigm for solving #SAT, and our method to count models could be used to impact directly in the reduction of the complexity time of the algorithms for other counting problems, i.e. for counting independent sets, counting colouring of graphs, counting cover nodes, etc. We present just one of the applications of this counting results for realizing the incremental recompilation of an initial knowledge base Σ with a new formula F, in an inductive and efficient way.
Keywords: #SAT Problem, Counting Models, Incremental Recompilation of Knowledge, Propositional Inference.

1 Introduction

As is well known, the propositional *Satisfiability* problem (SAT problem) is a classical NP-complete problem, and an intensive area of research has been the identification of restricted cases for which the SAT problem, as well as its optimization and counting version: MaxSAT and #SAT problems respectively, can be solved efficiently.

SAT and #SAT are a special concern to the Artificial Intelligence (AI) field, and they have a direct relationship to Automated Theorem Proving as well as in approximate reasoning. For example, #SAT has applications in the estimating of the degree of reliability in a communication network, for computing the degree of belief in propositional theories, in Bayesian inference, in a truth maintenance systems, for repairing inconsistent databases [1, 3, 13, 15]. The previous problems

* The researcher is partly supported by the Autonomous University of Puebla - BUAP

come from several AI applications such as planning, expert systems, data-mining, approximate reasoning, etc.

The #SAT problem is at least as hard as the SAT problem, but in many cases, even when SAT is solved in polynomial time, no computationally efficient method is known for #SAT. For example, the 2-SAT problem (SAT restricted to consider formulas where each clause has two literals at most), it can be solved in linear time. However, the corresponding counting problem #2-SAT is a #P-complete problem. #2-SAT continues being a #P-complete problem if we consider only monotone formulas or Horn formulas [14]. Even more, #SAT restricted to formulas in the class $(2, 3\mu)$-CF (the class of conjunctions of 2-clauses where each variable appears three times at most) is also #P-complete [13], while their respective SAT versions are solved efficiently.

The maximum polynomial class recognized for #2SAT is the class $(\leq 2, 2\mu)$-CF (conjunction of binary or unitary clauses where each variable appears two times at most) [13, 14]. Here, we extend such a class for considering the topological structure of the undirected graph induced by the restrictions (clauses) of the formula.

We extend here some of the procedures presented in [4, 5] for the #2-SAT Problem, keeping the polynomial time of the algorithms we determine class of 2-CF where #2-SAT is tractable. In fact, we determine a new polynomial class for #2-SAT and show that this new class is not restricted by the number of occurrences per variable of the given formula, but rather, by the topological structure of its constraint graph.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x or a negated variable $\overline{x}$. As is usual, for each $x \in X$, $x^0 = \overline{x}$ and $x^1 = x$. We use $v(l)$ to indicate the variable involved by the literal l.

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 k literals at most. A unitary clause has just one literal and a binary clause has exactly two literals. The empty clause signals a contradiction. A clause is *tautological* if it contains a complementary pair of literals. From now on, we will consider just non-tautological and non-contradictory clauses. A variable $x \in X$ *appears* in a clause c if either x or $\overline{x}$ is an element of c. Let $v(c) = \{x \in X : x \text{ appears in } c\}$.

A *Conjunctive Form* (CF) 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 $(k, s\mu)$-CF ($(\leq k, s\mu)$-CF) is a k-CF ($(\leq k)$-CF) such that each variable appears no more than s times. In this sense we have a hierarchy given by the number of occurrences by variable, where $(k, s\mu)$-CF is a restriction of $(k, (s+1)\mu)$-CF, and a hierarchy given by the number of literals by clause, where $(\leq k, s\mu)$-CF is a restriction of $(\leq (k+1), s\mu)$-CF. For any CF F, let $v(F) = \{x \in X : x$ appears in any clause of $F\}$.

An assignment s for F is a function $s : v(F) \rightarrow \{0,1\}$. An *assignment* can be also considered as a set of no complementary pairs of literals. If $l \in s$, being s an assignment, then s makes l *true* and makes $\overline{l}$ *false*. A clause 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.

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 assigment over $v(F)$ that satisfies F.

Let $SAT(F)$ be the set of models that F has over $v(F)$. F is a *contradiction* or *unsatisfiable* if $SAT(F) = \emptyset$. Let $\mu_{v(F)}(F) = |SAT(F)|$ be the cardinality of $SAT(F)$. Given F a CF, the SAT problem consists of determining if F has a model. The #SAT consists of counting the number of models of F defined over $v(F)$. We will also denote $\mu_{v(F)}(F)$ by #SAT(F). When $v(F)$ will clear from the context, we will explicitly omit it as a subscript.

Let #$LANG$-SAT be the notation for the #SAT problem for propositional formulas in the class $LANG$-CF, i.e. #2-SAT denotes #SAT for formulas in 2-CF, while #$(2, 2\mu)$-SAT denotes #SAT for formulas in the class $(2, 2\mu)$-CF. FP denotes the class of functions calculable in deterministic polynomial time, while #P is the class of functions calculable in nondeterministic polynomial time. The #SAT problem is a classical #P-complete problem, similarly for its restrictions #2-SAT, #$(2, 3\mu)$-SAT and for $(2, 3\mu)$-MON and $(2, 3\mu)$-HORN, monotone and Horn formulas, respectively.

The Graph Representation of a 2-CF

Let Σ be a 2-CF, the *constraint graph* of Σ is the undirected graph $G_\Sigma = (V, E)$, with $V = v(\Sigma)$ and $E = \{(v(x), v(y)) : (x, y) \in \Sigma\}$, that is, the vertices of G_Σ are the variables of Σ and for each clause (x, y) in Σ there is an edge $(v(x), v(y)) \in E$. Given a 2-CF Σ, a *connected component* of G_Σ is a maximal subgraph such that for every pair of vertices x, y, there is a path in G_Σ from x to y. We say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ. We will denote $[\![n]\!] = \{1, 2, ..., n\}$.

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

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

$$\mu_{v(\Sigma)}(\Sigma) = [\mu_{v(G_1)}(G_1)] \cdot \ldots \cdot [\mu_{v(G_r)}(G_r)] \qquad (1)$$

In order to compute $\mu(\Sigma)$, first we should determine the set of connected components of Σ, and this procedure is done in linear time [14]. The different connected components of G_Σ conform the partition of Σ in its connected components. Then, compute $\mu(\Sigma)$ is translated to compute $\mu_{v(G)}(G)$ for each connected component G of Σ. From now on, when we mention a formula Σ, we suppose that Σ is a connected component. We say that a 2-CF Σ is a *cycle*, a *chain* or a *free tree* if G_Σ is a cycle, a chain or a free tree, respectively.

3 Linear Procedures for subclasses of #2-SAT

Our purpose is to identify the restrictions over the class of ($\leq$ 2)-CF's under which the hard problem #2-SAT either remains hard or become easy. We suppose that G_Σ is the *constraint graph* of a connected component type given by Σ a ($\leq$ 2)-CF. We present the different typical simple graphs for G_Σ and design linear procedures to compute $\#SAT(\Sigma)$.

3.1. If G_Σ is acyclic

First, let us consider that $G_\Sigma = (V, E)$ is **a linear chain.** We write the associated formula Σ, as: $\Sigma = \{c_1, ..., c_m\} = \left\{\{y_0^{\epsilon_1}, y_1^{\delta_1}\}, \{y_1^{\epsilon_2}, y_2^{\delta_2}\}, \ldots, \{y_{m-1}^{\epsilon_m}, y_m^{\delta_m}\}\right\}$, without a loss of generality (ordering the clauses and its literals, if it were necessary) in such a way that $|\upsilon(c_i) \cap \upsilon(c_{i+1})| = 1$, $i \in [\![m-1]\!]$, and $\delta_i, \epsilon_i \in \{0,1\}$, $i = 1, ..., m$.

As Σ has m clauses then $|\upsilon(\Sigma)| = n = m + 1$. We will compute $\mu(\Sigma)$ in base to build a series (α_i, β_i), $i = 0, ..., m$,where each pair is associated to the variable y_i of $\upsilon(\Sigma)$. Te value α_i indicates the number of times that the variable y_i is 'true' and β_i indicates the number of times that the variable y_i takes value 'false' over the set of models of Σ. Let f_i a family of clauses of Σ builds as follows: $f_i = \{c_j\}_{j \leq i}$, $i \in [\![m]\!]$. Note that $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisifies } f_i\}$, $A_i = \{s \in SAT(f_i) : y_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{y}_i \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$. From the total number of models in $\mu_i, i \in [\![m]\!]$, there are α_i of which y_i takes the logical value 'true' and β_i models where y_i takes the logical value 'false'.

For example, $c_1 = (y_0^{\epsilon_1}, y_1^{\delta_1})$, $f_1 = \{c_1\}$, and $(\alpha_0, \beta_0) = (1,1)$ since y_0 can take one logical value 'true' and one logical value 'false' and with whichever of those values it would satisfy the clause c_1 which is the only clause of Σ where y_0 appears. $SAT(f_1) = \{y_0^{\epsilon_1} y_1^{\delta_1}, y_0^{1-\epsilon_1} y_1^{\delta_1}, y_0^{\epsilon_1} y_1^{1-\delta_1}\}$, and $(\alpha_1, \beta_1) = (2,1)$ if δ_1 were 1 or rather $(\alpha_1, \beta_1) = (1,2)$ if δ_1 were 0.

In general, we compute the values for (α_i, β_i) associated to each node x_i, $i = 1, .., m$, according to the signs (ϵ_i, δ_i) of the literals in the clause c_i, by the next recurrence equations:

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

$$= \begin{cases} (\beta_{i-1}, \mu_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (0,0) \\ (\mu_{i-1}, \beta_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (0,1) \\ (\alpha_{i-1}, \mu_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (1,0) \\ (\mu_{i-1}, \alpha_{i-1}) \text{ if } (\epsilon_i, \delta_i) = (1,1) \end{cases} \tag{2}$$

Note that, as $\Sigma = f_m$ then $\mu(\Sigma) = \mu_m = \alpha_m + \beta_m$. We denote with $' \rightarrow '$ the application of one of the four rules of the recurrence (2), so, the expression

$(2,3) \rightarrow (5,2)$ denotes the application of one of the rules (in this case, the rule 4), over the pair $(\alpha_{i-1}, \beta_{i-1}) = (2,3)$ in order to obtain $(\alpha_i, \beta_i) = (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1}) = (5,3)$.

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 2-CF which conforms a chain, the series* $(\alpha_i, \beta_i), i \in [\![5]\!]$*, is computed as:* $(\alpha_0, \beta_0) = (1,1) \rightarrow (\alpha_1, \beta_1) = (2,1)$ *since* $(\epsilon_1, \delta_1) = (1,1)$*, and the rule 2 have to be applied. In general, applying the corresponding rule of the recurrence (2) according to the signs expressed by* $(\epsilon_i, \delta_i), i = 2, ..., 5$*, we have* $(2,1) \rightarrow (\alpha_2, \beta_2) = (1,3) \rightarrow (\alpha_3, \beta_3) = (3,4) \rightarrow (\alpha_4, \beta_4) = (3,7) \rightarrow (\alpha_5, \beta_5) = (10,7)$*, and then,* $\#SAT(\Sigma) = \mu(\Sigma) = \mu_5 = \alpha_5 + \beta_5 = 10 + 7 = 17$.

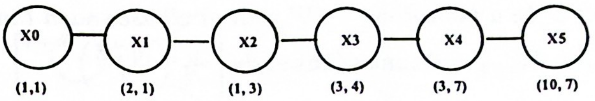

Fig. 1. The constraint graph for the formula of the example 1

If Σ is a chain, we apply (2) in order to compute $\mu(\Sigma)$ and this procedure has a linear time complexity over the number of variables of Σ, since (2) is applied while we are traversing the chain, from the initial node y_0 to the final node y_m.

There are other procedures for computing $\mu(\Sigma)$ when Σ is a $(2, 2\mu)$-CF [13, 14], but these last proposals do not distinguish the number of models in which a variable x takes value 1 of the number of models in which the same variable x takes value 0, situation which is made explicit in our procedure through the pair (α, β) labeled by x. This distinction over the set of models of Σ is essential when we want to extend the computing of $\mu(\Sigma)$ for more complex formulas.

Example 2 *Suppose now a* monotone 2-*CF* Υ *with* m *clauses and where* G_Υ *is a linear chain. I.e* $\Upsilon = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{m-1}, x_m)\}$*. Then, at the beginning of the recurrence (2),* $(\alpha_0, \beta_0) = (1,1)$ *and* $(\alpha_1, \beta_1) = (2,1)$ *since* $(\epsilon_1, \delta_1) = (1,1)$*, and in general, as* $(\epsilon_i, \delta_i) = (1,1)$*, for* $i \in [\![m]\!]$*, then the rule:* $(\alpha_i, \beta_i) = (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1})$ *is always applied while we are scanning each node of the chain, thus the Fibonacci numbers appear!.*

$$\begin{aligned}
\mu_1 &= \alpha_1 + \beta_1 = \alpha_0 + \beta_0 + \alpha_0 = 3, \\
\mu_2 &= \alpha_2 + \beta_2 = \mu_1 + \alpha_1 = 5, \\
(\mu_i)_{i \geq 2} &= \alpha_i + \beta_i \ = \mu_{i-1} + \mu_{i-2}
\end{aligned}$$

The sequence: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...$, in which each number is the sum of the preceding two, is denoted as the Fibonacci series. The numbers in the sequence, known as the Fibonacci numbers, will be denoted by F_i and we formally define them as: $F_0 = 0$; $F_1 = 1$; $F_{i+2} = F_{i+1} + F_i, i \geq 0$. Each Fibonacci number can be bounded up and low by $\phi^{i-2} \geq F_i \geq \phi^{i-1}, i \geq 1$, where $\phi = \frac{1}{2} \cdot (1 + \sqrt(5))$

is known as the 'golden ratio'. Any Fibonacci number F_i can be computed by the equation $F_i = \texttt{ClosestInteger}(\frac{\phi^i}{\sqrt{5}}) = \frac{(1+\sqrt{5})^i}{2 \cdot \sqrt{5}}$.

The Fibonacci series appear in many applications of the mathematics and it presents interesting properties that they are reflected in the nature, for example, it has been shown that some leaves of the trees present a growth in hairspring as the series of Fibonacci. Thus, applying the Fibonacci series for computing the number of models in the formula F of the example 2, we obtain the values $(\alpha_i, \beta_i), i = 0, ..., 5$: $(1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$, and this last series coincides with the Fibonacci numbers: $(F_2, F_1) \rightarrow (F_3, F_2) \rightarrow (F_4, F_3) \rightarrow (F_5, F_4) \rightarrow (F_6, F_5) \rightarrow (F_7, F_6)$. We infer that $(\alpha_i, \beta_i) = (F_{i+2}, F_{i+1})$ and then $\mu_i = F_{i+2} + F_{i+1} = F_{i+3}, i = 0, ..., m$. I.e. for $m = 5$, we have $\mu(F) = \mu_5 = F_7 + F_6 = F_8 = 21$.

Theorem 1 *Let Σ be a monotone 2-CF with m clauses such that G_Σ is a chain, then:* $\#SAT(\Sigma) = F_{m+3} = \texttt{ClosestInteger}\left[\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{m+3}\right]$

Let G_Σ be a free tree: Let Σ be a Boolean formula with n variables and m clauses and where there are no cycles in $G_\Sigma = (V, E)$. Traversing G_Σ in depth-first build a free tree, that we denote as A_Σ, whose root node is any vertex $v \in V$ with degree 1, and where v is used for beginning the depth-first search. The next procedure give us a recursive view of the depth-first search, which we show in order to present the different moments of a node during the search.

```
Procedure dfs($G_\Sigma$, v)

1. Mark $v$ as discovered
2. For each vertex $w$ such that there is an edge $(v, w) \in G_\Sigma$
   (a) IF $w$ is undiscovered then dfs($G_\Sigma$,w)
3. Mark $v$ as finished
```

Note that since G_Σ is a free tree, then all its edges are *tree edges* and, there are no *back edges* in A_Σ. We denote with (α_v, β_v) the associated pair to a node v ($v \in A_\Sigma$). We compute $\mu(\Sigma)$ while we are traversing G_Σ in depth-first, for the next procedure.

Algorithm Count_Models_for_free_trees(A_Σ)
Input: A_Σ the tree defined by the depth-search over G_Σ
Output: The number of models of Σ
Procedure: Traversing A_Σ in depth-first, and when a node $v \in A_\Sigma$ is left (marked as "finished" in *dfs*()), assign:

1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in A_Σ.
2. If v is a father node with an unique child node u, we apply the recurrence (2) considering that $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_u, \beta_u)$ and then $(\alpha_{i-1}, \beta_{i-1}) \rightarrow (\alpha_i, \beta_i) = (\alpha_v, \beta_v)$.

3. If v is a father node with a list of child nodes associated, i.e., $u_1, u_2, \dots, u_k$ are the child nodes of v, then as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, \dots, k$ has been defined based on (2). $(\alpha_{v_i}, \beta_{v_i})$ is obtained by applying (2) over $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_{u_j}, \beta_{u_j})$. This step is iterated until computes the values $(\alpha_{v_j}, \beta_{v_j})$, $j = 1, \dots, k$. Finally, let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$.
4. If v is the root node of A_Σ then returns$(\alpha_v + \beta_v)$.

This procedure returns the number of models for Σ in time $O(n+m)$ which is the necessary time for traversing G_Σ in depth-first.

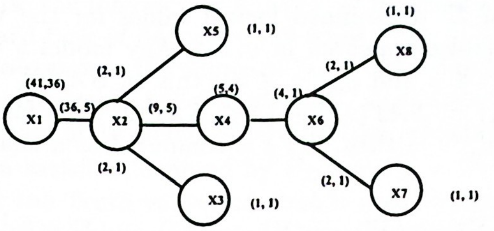

Fig. 2. The free tree graph for the formula of the example 3

Example 3 *If* $\Sigma = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ *is a 2-CF, we consider the depth-first search starting in the node* x_1. *The free tree generated by the depth-search as well as the number of models in each level of the tree is shown in Figure 2. The procedure* $Count\_Models\_for\_free\_trees$ *returns for* $\alpha_{x_1} = 41$, $\beta_{x_1} = 36$ *and the total number of models is:* $\#SAT(\Sigma) = 41 + 36 = 77$.

If there are unitary clauses in Σ, i.e. $U \subseteq \Sigma$ and $U = \{(l_1), (l_2), \dots, (l_k)\}$. Then, when the recurrence (2) is being applying over a node x_i of G_Σ, it has to be checked if $x_i \in v(U)$ or not. If $x_i \notin v(U)$ we only apply the recurrence, but if $x_i \in v(U)$ then $(\alpha_i, \beta_i) = \begin{cases} (0, \beta_i) & \text{if } (\overline{x}_i) \in U, \\ (\alpha_i, 0) & \text{if } (x_i) \in U \end{cases}$

Since a unitary clause uniquely determines the values of its variable is not needed to consider the opposite value for the variable.

Let $U' = \{(l) : v(l) \in v(U)\}$ and let $U'' = U - U'$. If there are no contradictory pairs of unitary clauses in U'' then $\mu(U'') = 1$; otherwise $\mu(U'') = 0$. In base to (1), $\mu(\Sigma \cup U) = \mu(\Sigma \cup U') \cdot \mu(U'')$ since $G_{\Sigma \cup U'}$ and $G_{U''}$ are independent connected components.

3.2. If G_Σ contains cycles

As is known, $\#SAT$ for formulas in the class $(2, 3\mu)$-CF is a #P-complete problem [14], for this class, if we discard the class of formulas Σ where G_Σ is a free tree (because for this subclass #SAT is computed in linear time), remains in the class just cycle formulas. In this section we show that even for cycle formulas, there is a tractable subclass for #2SAT.

First, let us consider that Σ is a $(2,2\mu)$-CF such that $G_\Sigma = (V,E)$ is a simple *cycle* with m nodes, that is, all the variables in $v(\Sigma)$ appear twice, $|V| = m = n = |E|$. Ordering the clauses in Σ in such a way that $| v(c_i) \cap v(c_{i+1}) |= 1$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, 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 chain and $c_m = (y_{m-1}^{\epsilon_m}, y_0^{\delta_m})$ is the edge which conforms with $G_{\Sigma'}$ the simple cycle: $y_0, y_1, ..., y_{m-1}, y_0$. Then, we can apply the linear procedure in (3.1) for computing $\mu(\Sigma')$.

Every model of Σ' determined logical values for the variables: y_{m-1} and y_0 since those variables appeared in $v(\Sigma')$. Any model s of Σ' satisfies c_m if and only if $(y_{m-1}^{1-\epsilon_m} \notin s$ and $y_m^{1-\delta_m} \notin s)$, this 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 $Y = \Sigma' \cup \{(y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m})\}$ then, $\mu(Y)$ is computed as a chain with two unitary clauses, and then:

$$\#SAT(\Sigma) = \mu(\Sigma) = \mu(\Sigma') - \mu(Y) = \mu(\Sigma') - \mu(\Sigma' \wedge (y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m})) \quad (3)$$

For example, let us consider Σ to be a monotone 2-CF with m clauses such that G_Σ is a simple cycle. $\Sigma = \left\{c_i = \{y_{i-1}^{\epsilon_i}, y_i^{\delta_i}\}\right\}_{i=1}^{m}$, where $\delta_i, \epsilon_i = 1$, $| v(c_i) \cap v(c_{i+1}) |= 1$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $y_0 = y_m$. Let $\Sigma' = \{c_1, ..., c_{m-1}\}$, then: $\mu(\Sigma') = \mu_{m-1} = F_{m-1+3} = F_{m+2}$ for theorem 1 and being F_i the i-esimo Fibonacci number. As $\epsilon_m = \delta_m = 1$ and $\mu(Y) = \mu(\Sigma' \wedge (\overline{y}_0) \wedge (\overline{y}_{m-1}))$ is computed by the series: $(\alpha_0, \beta_0) = (0,1) = (F_0, F_1)$ since $(\overline{y}_0) \in Y$, $(\alpha_1, \beta_1) = (1,0) = (F_1, F_0)$; $(\alpha_2, \beta_2) = (1,1) = (F_2, F_1)$; $(\alpha_3, \beta_3) = (2,1) = (F_3, F_2)$; and in general $(\alpha_i, \beta_i) = (F_i, F_{i-1})$, then for the clause c_{m-1}, $(\alpha_{m-1}, \beta_{m-1}) = (F_{m-1}, F_{m-2})$, then $\mu(Y) = \beta_{m-1} = F_{m-2}$ since $(\overline{y}_m) \in Y$. Finally, $\#SAT(\Sigma) = \mu(\Sigma) = \mu(\Sigma') - \mu(Y) = F_{m+2} - F_{m-2}$. On the other hand; $F_{m+2} - F_{m-2} = F_{m+1} + F_m - F_{m-2} = F_{m+1} + F_{m-1} + F_{m-2} - F_{m-2} = F_{m+1} + F_{m-1} = F_{m+2} \cdot F_{m-1}$.

Theorem 2 *Let Σ be a monotone 2-CF with m clauses and where G_Σ is a simple cycle, then: $\#SAT(\Sigma) = F_{m+2} - F_{m-2} = F_{m+1} + F_{m-1}$.*

Note that the combination of the procedure for free trees and the processing of cyles (equation 3) can be applied for computing $\#SAT(\Sigma)$ if G_Σ is a graph where the depth-first search generates a free tree and a set of fundamental cycles, such that any fundamental cycle is independent with any other fundamental cycle, that is, there are no common vertices neither common edges among any pair of fundamental cycles.

Thus, the procedures presented here, for computing $\mu(\Sigma)$ being Σ a Boolean formula in 2-CF and where G_Σ is a cycle, a chain, a free tree, or a free tree union independent cycles, each one runs in linear time over the length of the given formula, and they have the complexity time $O(m+n)$.

The class of Boolean formulas F such that its depth-first search builds a free tree and a set of fundamental independent cycles, such class conforms a new polynomial class for #2-SAT. This new class is a superclass of $(2, 2\mu)$-CF, and it has not restriction over the number of occurrences of a variable over the formulas, although $(2, 3\mu)$-CF is a $\#P$-complete problem.

4 Applying #SAT to Propositional Inference

We now turn to one important application of the results of the previous section into the area of propositional reasoning. A generalization of deductive inference which can be used when a knowledge base is augmented by, e.g., statistical information, is to use the inference of a degree of belief as an effort to avoid the computationally hard task of deductive inference [13].

The approach to compute the degree of belief of an intelligent agent, consists of assigning an equal degree of belief to all basic "situations". In this manner, we can compute the probability that Σ (an initial knowledge base which involves n variables) will be satisfied, denoted by P_Σ, as: $P_\Sigma = Prob(\Sigma \equiv \top) = \frac{\mu(\Sigma)}{2^n}$, where $\top$ stands for the Truth value and *Prob* is used to denote the probability.

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

We want to determine the class of formulas for Σ and F where the degree of belief $P_{F|\Sigma}$ can be done efficiently, in such a way that we can realize the incremental recompilation of knowledge of Σ by F in an efficient way.

Many approaches for incorporating dynamically a single or a sequence of changes into an initial Knowledge Base (KB) have been proposed [3, 7, 9, 11, 12]. Almost all these proposals are plagued by serious complexity-theoretic impediments, even in the Horn case [7, 9]. More fundamentally, these schemes are not inductive, in the sense that they may lose in a single update any positive properties of the structure of the KB.

Thus, in order to perform the computing of the degree of belief $P_{F|\Sigma}$ in polynomial time, we begin considering a knowledge base Σ in 2-CF and where G_Σ is a free tree and if there are cycles in G_Σ, these are independent cycles, since as we have shown in the previous section that #2-SAT for this class of formulas is computed in polynomial time.

We also suppose that the KB Σ is a satisfiable 2-CF and $\mu(\Sigma) > 0$ and then $P_{F|\Sigma} = \frac{\mu(\Sigma \wedge F)}{\mu(\Sigma)}$ is well-defined. We will show here how to compute $\mu(\Sigma \wedge F)$, and for this, we consider the different cases for F.

4.1. Computing the Degree of Belief in Basic Formulas

Let $F = \{(l)\}$, where $v(l) \in v(\Sigma)$. We have shown in chapter 3 how to compute $\mu(\Sigma \cup \{(l)\})$. Notice that if $\Sigma \wedge F$ is unsatisfiable then $P_{F|\Sigma} = 0$.

If $F = \{(l)\}$ and $v(l) \notin v(\Sigma)$, and as we have considered the degree of belief in F given Σ as a conditional probability, then it makes sense to *update* the degree of belief for *updating* the probability space where $P_{F|\Sigma}$ is computed [6].

Incremental Knowledge Process: When new pieces of information that did not originally appear in the sample space must be considered then, we will introduce into the area of updating the degree of belief for making an extension of the original probability space [8].

Let $F = (\bigwedge_{j=1}^{k} l_j)$ a conjunction of literals where there are variables that do not appear in the original knowledge base Σ. Let $A = \{(l) \in F : v(l) \notin v(\Sigma)\}$, $t = \mid A \mid$ and $\mid v(\Sigma) \mid = n$. We consider F as a set of literals (the conjunction is understood between the elements of the set), let $F' = F - A$. There are 2^n assignments defined over $v(\Sigma)$ and 2^{n+t} assignments defined over $v(\Sigma) \cup v(F)$, then we *update* the domain of the probability space over which we compute $P_{F|\Sigma}$, as: $P_{F|\Sigma} = \frac{Prob_{(\Sigma \wedge F)}}{Prob_{\Sigma}} = \frac{\frac{\mu(\Sigma \wedge F)}{2^{n+t}}}{\frac{\mu(\Sigma)}{2^n}} = \frac{\mu(\Sigma \wedge F)}{2^t \cdot \mu(\Sigma)}$. Since G_A and $G_{\Sigma \cup F'}$ are two independent connected components and $\mu(\bigwedge_{l \in A} l) = \prod_{l \in A} \mu(l) = 1$, then:

$$P_{F|\Sigma} = \frac{\mu(\Sigma \wedge F') \cdot \mu(\bigwedge_{l \in A} l)}{2^t \cdot \mu(\Sigma)} = \frac{\mu(\Sigma \wedge F')}{2^t \cdot \mu(\Sigma)} \tag{4}$$

Example 4 *Let* $\Sigma = \{(x_0, x_1), (x_1, x_2), (x_2, x_3), (\overline{x}_3, \overline{x}_4), (\overline{x}_4, x_5)\}$, *and* $F = \{x_0, \overline{x}_3, x_8\}$. $A = \{l \in F : v(l) \notin v(\Sigma)\} = \{x_8\}$, *and* $t = \mid A \mid = 1$. *We can compute* $\mu(\Sigma \wedge F)$ *according of the procedure (3.1) then,* $(\alpha_0, \beta_0) = (1, 0)$, *since* $(x_0) \in F$, *and then* $(\alpha_1, \beta_1) = (1, 1)$. $(\alpha_2, \beta_2) = (\alpha_1 + \beta_1, \alpha_1) = (2, 1) \rightarrow (3, 2) = (\alpha_3, \beta_3)$, *but this last pair must be changed since* $v(x_3)$ *is the label of* (α_3, β_3) *and appears as a negated variable in* F, *then* $(\alpha_3, \beta_3) = (0, \beta_3) = (0, 2)$. *After* $(\alpha_4, \beta_4) = (\beta_3, \alpha_3 + \beta_3) = (2, 2) \rightarrow (4, 2) = (\alpha_5, \beta_5)$. *And* $\mu_5 = 6 = \mu(\Sigma \wedge F) = \mu(\Sigma \wedge F') = \mu(\Sigma \wedge (x_0) \wedge (\overline{x}_3))$, *applying (4) since* x_8 *does not appear in* $v(\Sigma)$. $P_{F|\Sigma} = \frac{\mu(\Sigma \wedge \bigwedge_{j=1}^{k} l_j)}{2^t \cdot \mu(\Sigma)} = \frac{\mu(\Sigma \wedge (x_0) \wedge (\overline{x}_3) \wedge (x_8))}{2^t \cdot \mu(\Sigma)} = \frac{6}{2 \cdot 19} = \frac{6}{38}$.

To compute $\mu(\Sigma \wedge F')$ as well as $\mu(\Sigma)$ is done in linear time by the linear procedure of chapter 3, then (4) is computed in linear time too.

Let now a clause, $F = (\bigvee_{j=1}^{k} l_j)$. Considering F as a set of literals, let $A = \{l \in F | v(l) \notin v(\Sigma)\}$, $t = \mid A \mid$, and let $F' = F - A$. We can compute $\mu(\Sigma \wedge F)$, as:

$$\mu(\Sigma \wedge F) = \mu(\Sigma) \cdot 2^t - \mu(\Sigma \wedge \overline{F}) \tag{5}$$

We are computing $\mu(\Sigma \wedge F)$ by extending the models of Σ for considering the variables which are in $v(F)$ however they are not in $v(\Sigma)$, and we are eliminating the assignments which falsify $\Sigma \cup F$.

As, $F = (\bigvee_{j=1}^{k} l_j)$ then $\overline{F} = (\bigwedge_{j=1}^{k} \overline{l}_j) = (\bigwedge_{x \in F'} \overline{x} \wedge \bigwedge_{x \in A} \overline{x})$ since $v(A) \cap (v(\Sigma) \cup v(F')) = \emptyset$ we could consider G_A as a connected component independent to $G_{\Sigma \cup F'}$, where $\overline{F'} = \bigwedge_{x \in F \wedge v(x) \in v(\Sigma)} \overline{x}$. According to (1), $\mu(\Sigma \wedge \overline{F}) = \mu(\Sigma \wedge \overline{F'}) \cdot \mu(\overline{A}) = \mu(\Sigma \wedge \overline{F'})$ since $\mu(\overline{A}) = 1$, then:

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

Notice that when F is a phrase or a clause there is no restriction on the number of literals that F can contain. Thus, (6) permits us to solve #SAT for formulas $(\Sigma \cup F)$ in a greater hierarchy than $(\leq 2, 3\mu)$-CF, for considering clauses in F with more than 2 literals.

Some methods for choosing among several possible revisions are based on some implict bias, namely a priory probability that each element (literal or clause) of the domain theory requires revision. Opposite to assign the probabilities to each element F of the theory Σ by an expert or simply chosen by default [10], we have shown here a formal and efficient way to determine such probability based on the degree of belief $P_{F|\Sigma}$, with the additional advantages that such probabilities could be adjusted automatically in response to newly-obtained information.

Suppose that Σ is a (≤ 2)-CF where G_Σ is a free tree or it contains a set of cycles such that any pair of such cycles do not share edges neither nodes. The last objective that we approach here, consist on determining the structure of a new formula F (set of clauses) such that $G_{\Sigma \cup F}$ keeps the same conditions of G_Σ, in order that $\#SAT(\Sigma \cup F)$ remains computing in polynomial time complexity and the incremental knowledge process will be inductive.

IF for each $c_i \in F, i = 1, ..., \mid F \mid$

- c_i is a unitary clause, or
- $\upsilon(c_i) - \upsilon(\Sigma) \neq \emptyset$, or
- c_i adds a new fundamental cycle en G_Σ, this is, c_i conforms a new cycle but the set of cycles in $G_\Sigma \cup c_i$ do not share edges neither nodes.

Then $\#SAT(\Sigma \cup F)$ remains computing in polynomial time using the procedures presented in chapter 3.

5 Conclusions

#SAT for the class of Boolean formulas in 2-CF is a classical #P-complete problem. Until now, the maximum subclass of 2-CF where #2SAT is solved efficiently is for the class $(2, 2\mu)$-CF, which are the Boolean formulas in 2-CF where each variable appears twice at most.

We present different linear procedures to compute #SAT for subclasses of 2-CF. Let Σ be a 2-CF where G_Σ (the constraint undirected graph of Σ) is acyclic or, a free tree union independent cycles, we show that $\#SAT(\Sigma)$ is computed in linear time over the length of the formula Σ.

This new polynomial class of 2-CF contains to the class $(2, 2\mu)$-CF, and it does not have restriction over the number of occurrences per variable in the given formula, although $(2, 3\mu)$-SAT is a $\#P$-complete problem. Then, this new class of Boolean formulas brings us a new paradigm for solving #SAT, and would be used to incide directly over the complexity time of the algorithms for other counting problems.

We present one application of our results in the propostional inference area, showing conditions that permit to an intelligent agent, compute the degree of belief in a new formula F given an initial knowledge base Σ and such that it could be done in polynomial time over the length of $(\Sigma \cup F)$.

References

1. Angelsmark O., Jonsson P., Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems, *In ICCP: Int. Conf. on Constraint Programming*, 2003.
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 Adnan, 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), 11-34.
4. De Ita G., Tovar M., Vera E., Guilln C., Designing Efficient Procedures for #2SAT, Proceedings of the short papers of the 12th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-12), 2005, pp. 28-32.
5. De Ita G., Polynomial Classes of Boolean Formulas for Computing the Degree of Belief, Lectures Notes in Artificial Intelligence series No. 3315, 2004, pp. 430-440.
6. Halpern J. Y., Two views of belief: Belief as generalized probability and belief as evidence, *Artificial Intelligence 54*, (1992), 275-317.
7. Eiter T., Gottlob G., The complexity of logic-based abduction, *Journal of the ACM 42(1)*, (1995), 3-42.
8. 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), 347-374.
9. Goran Gogic, Christos H. Papadimitriou, Marta Sideri, Incremental Recompilation of Knowledge, *Journal of Artificial Intelligence Research 8*, (1998), 23-37.
10. Koppel M., Feldman R., Maria Segre A., Bias-Driven Revision of Logical Domain Theories, *Jour. of Artificial Intelligence Research 1*, (1994), 159-208.
11. Liberatore P., Schaerf M., The Complexity of Model Checking for Belief Revision and Update, *Procc. Thirteenth Nat. Conf. on Art. Intellegence (AAAI96)*, 1996.
12. Winslett M., *Updating Logical Databases*, Cambridge University Press., 1990.
13. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), 273-302.
14. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.
15. Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.