Counting Falsifying Assignments of a 2-CF via Recurrence Equations

Guillermo De Ita Luna, J. Raymundo Marcial Romero, Fernando Zacarias Flores,
Meliza Contreras González and Pedro Bello López

***Abstract*—We consider the problem $\#2UNSAT$ (counting the number of falsifying assignments for two conjunctive forms).**

Because $\#2SAT(F) = 2^n - \#2UNSAT(F)$, results about $\#2UNSAT$ can dually be established for solving $\#2SAT$. We establish the kind of two conjunctive formulas with a minimum number of falsifying assignment. As a result, we determine the formulas with a maximum number of models among the set of all 2-CF formulas with a same number of variables.

We have shown that for any 2-CF F_n with n variables, $\#UNSAT(F_n) \geq \#SAT(F_n)$ with the exception of totally dependent formulas. Thus, if $\#UNSAT(F_n) \leq p(n)$ for a polynomial on n, then $\#SAT(F_n) \leq p(n)$ too, and then $\#SAT(F_n)$ can be computed in polynomial time on the size of the formula.

***Index Terms*—$\#SAT$, Binary Patterns, Enumerative Combinatorics.**

I. Introduction

THERE are many real-life problems that can be abstracted as counting combinatorial objects on graphs. For instance, reliability network issues are often equivalent to connected component issues on graphs, e.g. the probability that a graph remains connected is given by the probabilities of failure over each edge, which is essentially the same as counting the number of ways that the edges could fail without losing connectivity [9],[10].

The combinatorial problems that we are going to address are the computation of the number of models and falsifying assignment for Boolean formulas in two Conjunctive Forms (2-CF), denoted as $\#2SAT$ and $\#2UNSAT$ respectively, based on string patterns formed by their set of falsifying assignments [11]. Both problems are classical $\#P$-complete problems even for the restricted cases of monotone and Horn formulas.

Among the class of $\#P-$complete problems,$\#SAT$ is considered a fundamental instance due to both its application in deduction issues and its relevance in establishing a boundary between efficient and intractable counting problems.

Previous studies have developed levels of complexity for counting $\#SAT$ and due to the duality between $\#SAT$ and $\#UNSAT$, bounds can be set for the latter .

Several algorithms have been designed for computing $\#UNSAT(F)$ as finer or shorter versions of the application of the inclusion-exclusion formula [6], [5], [7]. However, there are no concise algorithms or procedures which establish the cases in which $\#UNSAT(F)$ can be computed in polynomial time considering its constrained graph.

Since $\#SAT(F) = 2^n - \#UNSAT(F)$, analogous results can be proved for both. We investigate about the minimum number of falsifying assignments that a 2-CF could have. In this paper, we show some cases where $\#SAT(F)$ can be upper bounded by considering a binary pattern analysis over its set of falsifying assignments.

Manuscript received December 15, 2014; revised January 2, 2015.

G. De Ita, F. Zacarias,M. Contreras, P. Bello are with the Faculty of Computer Science, Benemerita Universidad Autónoma de Puebla, Puebla, 52570 México e-mail: {deita,fzacarias,mcontreras,pbello}@cs.buap.mx

J. R. Marcial is with the Department of Computation Faculty of Engineering, Universidad Autónoma del Estado de México, Toluca, 50110 México e-mail: rmarcial@fi.uaemex.mx

II. 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 $\neg x_i$.

A *clause* is a disjunction of different and non complementary literals (sometimes, we also consider a clause as a set of literals, e.g. $x_1 \vee x_2 = \{x_1, x_2\}$). For a positive integer k, a *k-clause* is a clause consisting of exactly k literals. A variable $x \in X$ *appears* in a clause c if either x or $\neg x$ is an element of c.

A *conjunctive form* (CF) F is a conjunction of non-tautological clauses. We say that F is a monotone positive CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses. $(\leq k)$-CF denotes a CF containing clauses with at most k literals. A 2-CF formula F is said to be strict only if each clause of F is binary.

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

An *assignment* s for F is a boolean function $s : \upsilon(F) \to \{0, 1\}$. An *assignment* s can also be considered as a set of non-complementary pairs of literals, e.g., if $l \in s$, then $\neg l \notin s$, in other words s turns l *true* and $\neg l$ *false*. Let c be a clause and s an assignment, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$. On the other hand, if for all $l \in c$, $\neg l \in s$, then s falsifies c. If $n = |\upsilon(F)|$, then there are 2^n possible assignments defined over $\upsilon(F)$. Let $S(F)$ be the set of 2^n assignments defined 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 falsified by s. A model of F is an assignment for $\upsilon(F)$ that satisfies F. A falsifying assignment of F is an assignment for $\upsilon(F)$ that contradicts F. The SAT problem consists of determining whether F has a model. $SAT(F)$ denotes the set of models of F, then $SAT(F) \subseteq S(F)$. Let $UNSAT(F) = S(F) - SAT(F)$, i.e. $UNSAT(F)$ is the set of assignments from $S(F)$ that falsify F.

The $\#SAT$ problem (or $\#SAT(F)$) consists of counting the number of models of F defined over $\upsilon(F)$, $\#2SAT$ denotes $\#SAT$ for formulas in 2-CF. So $\#UNSAT(F) = 2^n - \#SAT(F)$.

A 2-CF F can be represented by an undirected graph, called the *constrained graph* of F, and determined as: $G_F = (V(F), E(F))$, where $V(F) = \upsilon(F)$ and $E(F) = \{\{\upsilon(x), \upsilon(y)\} : \{x, y\} \in F\}$. I.e. the vertices of G_F are the variables of F, and for each clause $\{x, y\}$ in F there is an edge $\{\upsilon(x), \upsilon(y)\} \in E(F)$.

In order to compute $\#SAT(F)$, we can consider first the connected components of the constrained graph G_F of F. Such connected components $\{G_1, \ldots, G_k\}$ of G_F can be determined in linear time with respect to the number of clauses in the formula, and then $\#SAT(F) = \#SAT(G_F) = \prod_{i=1}^{k} \#SAT(G_i)$.

The set of connected components of G_F conforms a partition of F. So, from now on, we will work with a connected component graph. We say that a 2-CF F is a *path*, a *cycle*, or a *tree* if its corresponding constrained graph G_F represents a path, a cycle, or a tree, respectively.

III. Computing #2UNSAT via Recurrrence equations

Let $F = \{C_1, C_2, \ldots, C_m\}$ be a strict 2-CF (a conjunction of binary clauses) and let $n = |\upsilon(F)|$. The size of F is $n + m$. Let k be a positive integer parameter such that $k < 2^n$. The values of k where $\#SAT(F) = k$ can be determined in polynomial time have been proved for the following cases [8].

If $k = 0$ or $k = 1$ the *Transitive Closure* procedure presented in [1] can be applied. Such procedure has a linear time complexity on the size of the 2-CF.

If k is upper bounded by a polynomial function on n, e.g. $k \leq p(n)$, then in [2], an exact algorithm was shown for determining when $\#SAT(F) \leq p(n)$. Such algorithm has a polynomial time complexity on the size of F.

So, the cases where a polynomial time complexity algorithm has not been found to answer whether $\#SAT(F) = k$ are given when $k > p(n)$. We further bounded those cases by the following theorem (proved in [4]).

Theorem 1. [4] Let $F = \{C_1, C_2, \ldots, C_m\}$ be a 2-CF and $n = |\upsilon(F)|$. If $m \leq n$ then $\#SAT(F)$ is computed efficiently.

In this section we present the binary patterns used to represent the set of falsifying assignments of a 2-CF. Those patterns allow us to design efficient procedures for computing $\#2UNSAT$. We firstly present some previous results to give a context to this paper.

Lemma 1. [4] Let $F = \{C_1, \ldots, C_m\}$ be a 2-CF and $n = |\upsilon(F)|$. Then $\#UNSAT(F)$ has at least 2^{n-2} elements.

Let $F = \{C_1, \ldots, C_m\}$ be a 2-CF and $n = |\upsilon(F)|$. Assume an enumeration over the variables of $\upsilon(F)$, e.g. $x_1, x_2, \ldots, x_n$. For each $C_i = \{x_j, x_k\}$, let A_i be a set of binary strings such that the length of each string is n. The values at the j-th and k-th positions of each string, $1 \leq j, k \leq n$ represent the truth value of x_j and x_k that falsifies C_i. E.g., if $x_j \in C_i$ then the j-th element of A_i is set to 0. On the other hand, If $\neg x_j \in C_i$ then the j-th element of A_i is set to 1. The same argument applies to x_k.

We will use the symbol $*$ to represent the elements that can take any truth value in the set A_i, e.g. if $F = \{C_1, \ldots, C_m\}$ is a 2-CF, $n = |\upsilon(F)|$, $C_1 = \{x_1, x_2\}$ and $C_2 = \{x_2, x_3\}$ then we will write $A_1 = 00**\ldots*$ and $A_2 = *00*\ldots*$. This abuse of notation will allow us to present a concise and clear representation in the rest of the paper, for considering the string A_i as a binary pattern that represents the falsifying assignments of the clause C_i. We call *A Falsifying String*.

It is known [5] that for any pair of clauses C_i and C_j, it holds that $\#UNSAT(C_i \cup C_j) = \#UNSAT(C_i) + \#UNSAT(C_j) - \#UNSAT(C_i \cap C_j)$. The following lemmas show when the number of models can be reduced.

Lemma 2. [4] Let F be a 2-CF, $n = |\upsilon(F)|$. If $C_i \in F$ and $C_j \in F$, $i \neq j$ have no complementary pairs of literals and they share a literal (e.g. $C_i \cap C_j \neq \emptyset$), then there are exactly $2^{n-1} - 2^{n-3}$ assignments from $S(F)$ falsifying $C_i \cup C_j$.

Lemma 3. [4] Let F be a 2-CF, $n = |\upsilon(F)|$. If $C_i \in F$ and $C_j \in F$, $i \neq j$ contain complementary literals, that is $x_k \in C_i$ and $\neg x_k \in C_j$, the unsatisfied set of assignments A_i and A_j form a disjoint set of assignments. Consequently, both clauses suppress exactly $2^{n-2} + 2^{n-2} = 2^{n-1}$ assignments from $S(F)$.

Definition III.1. [5] Given two binary clauses C_i, C_j, If they have at least one complementary literal, it is said that they are independent. Otherwise, we say that both clauses are *dependent*.

Definition III.2. Let $F = \{C_1, C_2, \cdots C_m\}$ be a 2-CF. F is called independent if for each pair of clauses $C_i, C_j \in F, i \neq j$, it holds that $x_k \in C_i$ and $\neg x_k \in C_j$, otherwise F is called dependent.

Let C be a clause and let x be any variable, we have that

$$C = (C \vee \neg x) \wedge (C \vee x) \tag{1}$$

Furthermore, this reduction preserves the number of falsifying assignments, since $\#UNSAT(C) = 2^{n-(|C|+1)} + 2^{n-(|C|+1)} = \#UNSAT((C \vee \neg x) \wedge (C \vee x))$, because $(C \vee \neg x) \wedge (C \vee x)$ are two independent clauses.

Given a pair of dependent clauses C_1 and C_2. Let us assume that there exist literals in C_1 which are not in C_2, and let $x_1, x_2, ..., x_p$ be these literals. There exists a reduction to transform C_2 (or C_1) to be independent with C_1 (or C_2), we will call this transformation the *independence reduction*, and this works in the following way. By (1) one can write: $C_1 \wedge C_2 = C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1)$. Now C_1 and $(C_2 \vee \neg x_1)$ are independent. Applying (1) to $(C_2 \vee x_1)$: $C_1 \wedge C_2 = C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge (C_2 \vee x_1 \vee x_2)$ The first three clauses are independent. Repeating the independence reduction until x_p, we have that $C_1 \wedge C_2$ can be written as:

$C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge ... \wedge$
$(C_2 \vee x_1 \vee x_2 \vee ... \vee \neg x_p) \wedge (C_2 \vee x_1 \vee x_2 \vee ... \vee x_p)$.

The last clause contains all literals of C_1, so it can be eliminated, and then

$$C_1 \wedge C_2 = C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge \ldots \wedge (C_2 \vee x_1 \vee x_2 \vee \ldots \vee \neg x_p) \quad (2)$$

Clauses on the right hand side of (2) are independent by construction.

For any independent formula $F = \{C_1, \ldots, C_m\}$ involving n variables, we have that: $\#UNSAT(F) = \sum_{i=1}^{m} 2^{n-|C_i|}$. In particular, if F is a k-CF then $\#UNSAT(F) = \sum_{i=1}^{m} 2^{n-k}$. And as F is unsatisfiable when $\#UNSAT(F) = 2^n$, then all k-CF with at least 2^k independent clauses will be unsatisfiable.

On the other hand, a pair of dependent binary clauses can be one of the following two types:

a) with one common literal.
b) without any variable in common.

Case (a): Let C_i, C_j be two binary dependent clauses involved in a formula with n variables such that $x_k \in C_i$ and $x_k \in C_j$. The number of falsifying assignments for this pair of clauses is: $2^{n-1} - 2^{n-2}$, given by one application of the independence reduction:

Case (b): let $C_i = \{x_{i1}, x_{i2}\}$ and $C_j = \{x_{j1}, x_{j2}\}$ such that all the variables involved in C_i and C_j are different. The number of falsifying assignments for this pair of clauses is: $2^{n-2} + 2^{n-3} + 2^{n-4}$, according to a double application of the independence reduction.

Example III.1. Let $C_i, C_j \in F$, where $C_i = (x_1 \vee x_2)$, $C_j = (x_3 \vee x_4)$, the transformation to obtain independent clauses is applied over C_j as shown:
$(x_1 \vee x_2) \wedge (x_3 \vee x_4) = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee x_4)$. Since the last three clauses are independent we have: $\#UNSAT(C_i \wedge C_j) = 2^{n-2} + 2^{n-3} + 2^{n-4}$.

Lemma 4. Let $F(n, m)$ be a 2-CF with n variables and m clauses. Let $C = \{x, y\}$ be a binary clause not in F. It holds that $\#UNSAT(F \wedge C) \geq \#UNSAT(F)$.

Proof:
It is well known that: $A \cap B \subseteq A$, $A \cap B \subseteq B$ hence:

$$(UNSAT(F) \cap UNSAT(C)) \subseteq UNSAT(C)$$

which means that $-\#(UNSAT(F) \cap UNSAT(C)) \geq -\#UNSAT(C)$. Adding the number of falsifying assignments of F and C to both sides of the inequality: $\#UNSAT(F) + \#UNSAT(C) - (\#UNSAT(F) \cap \#UNSAT(C)) \geq \#UNSAT(F) + \#UNSAT(C) - \#UNSAT(C)$.
The left hand size in the inequality is $\#UNSAT(F \wedge C)$, while the right hand size is $\#UNSAT(F)$.

■

Definition III.3. A formula $F = \{C_1, C_2, \cdots C_m\}$ in which $C_i = \{x_1, l_i\}, \forall C_i \in F$ and each l_i is a literal which comes from a different variable, such formula is called totally dependent. The formula is also totally dependent if each $C_i = \{\neg x_1, l_i\}, \forall C_i \in F$ and l_i involves a different variable with any other $l_j, i \neq j = 1, \ldots, m$. We will call such F a *totally dependent positive* formula in the first case, and a *totally dependent negative* formula in the second case.

Notice that in a totally dependent formula with m clauses there are $m + 1$ different variables involved.

Theorem 2. Let $F_d(n, n-1)$ be a totally dependent formula with n variables and $n-1$ clauses, then for any 2-CF $F(n, m_1)$, with $m_1 \geq n-1$, it holds that $\#UNSAT(F_d) \leq \#UNSAT(F)$. In other words, a totally dependent formula has the minimum number of falsifying assignments into the set of all 2-CF with the same number of variables and at least the same number of clauses.

Proof: Let $F(n, m_1)$ be a 2-CF such that $m_1 > n$ (the number of clauses is greater than the number of variables). By the result of lemma 4, clauses in F can be chosen until a subformula $F_1(n, n-1) \subseteq F(n, m_1)$ is obtained such that the n variables are considered and $\#UNSAT(F_1(n, n-1)) \leq \#UNSAT(F(n, m_1))$. It remains to be shown that $\#UNSAT(F_1(n, n-1)) \geq \#UNSAT(F_d(n, n-1))$.
The proof is by induction on the number of clauses in F_d.

i) For $m = 1$, it holds, since $\#UNSAT(F_d(2, 1)) = 1 \leq \#UNSAT(F_1(2, 1))$.
ii) If $m = 2$, $F_d(3, 2) = \{(x_1, x_2), (x_1, x_3)\}$ where $\#UNSAT(F_d(3, 2)) = 3$. It can also be easily verified for $F_1(3, 2)$.
iii) We asume that the theorem holds for $m-1$ clauses, that is: $\#UNSAT(F_d(m, m-1)) \leq \#UNSAT(F_1(m, m-1))$.
iv) We show that it holds for formulas with m clauses. Let $F_1 = \{C_1, .., C_m\}$ be a 2-CF with m clauses and $m+1$ variables. Let C_m be the only clause where x_{m+1} appears and for any other clause $C_i \in F_1, i = 1, \ldots, m-1$, x_{m+1} does not appear (hence G_{F_1} is an acyclic graph). As $\{C_1, \ldots, C_{m-1}\}$ has less than m clauses, we apply the inductive hypothesis and there exists a totally dependent formula $F_d(m, m-1)$ such that $\#UNSAT(F_d(m, m-1)) \leq \#UNSAT(\{C_1, \ldots, C_{m-1}\})$. Let us analyze the different cases for $(F_d(m, m-1) \wedge C_m)$.

   a) If $C_m = \{x_1, x_{m+1}\}$, then $(F_d(m, m-1) \wedge C_m)$ continues being totally dependent and the theorem holds. Notice that $\#UNSAT(F_d(m+1, m)) = (\#UNSAT(F_d(m, m-1))) * 2 + 1$.
   b) If $C_m = \{\overline{x}_1, x_{m+1}\}$ then C_m is independent with any other clause of $F_d(m, m-1)$ and $\#UNSAT(F_d(m+1, m)) = \#UNSAT(F_d(m, m-1) \wedge C_m) = (\#UNSAT(F_d(m, m-1))) * 2 + 2^{m-1}$
   c) If $C_m = \{x_i, x_{m+1}\}$, where $v(x_i) \neq v(x_1)$. By one application of the independence reduction over C_m we obtain $C'_m = \{\neg x_1 \vee x_i \vee x_{m+1}\}$ independent with any other clause in $F_d(m, m-1)$ and $\#UNSAT(F_d(m, m-1) \wedge C_m) = \#UNSAT(F_d(m, m-1) \wedge C'_m)$, then: $\#UNSAT(F_d(m+1, m)) = \#UNSAT(F_d(m, m-1) \wedge C_m) = (\#UNSAT(F_d(m, m-1))) * 2 + 2^{m-2}$

   As $\#UNSAT(F_d(m+1, m)) = (\#UNSAT(F_d(m, m-1)) * 2 + 1 \leq (\#UNSAT(F_d(m, m-1)) * 2 + 2^{m-2} \leq (\#UNSAT(F_d(m, m-1)) * 2 + 2^{m-1}$,

for $m > 2$. Then, $\#UNSAT(F_d(m+1,m)) \leq \#UNSAT(F_1(m+1,m))$.

■

A totally dependent formula $F_d(n, n-1)$ can be represented by a tree of height 2, where the root (level 1) represents the shared literals and each child of the root (level 2) represents the non common literals. In Figure 1 we show a totally dependent positive formula. Notice that $\#UNSAT(F_d(n,n-1))$ can be computed from the value $\#UNSAT(F_d(n-1,n-2))$ according to the following recurrence: $\#UNSAT(F_d(n,n-1)) = \#UNSAT(F_d(n-1,n-2)) \cdot 2 + 1$, and developing such recurrence until the base case $\#UNSAT(F_d(2,1)) = 1$, we obtain that $\#UNSAT(F_d(n,n-1)) = 2^{n-1} - 1$.

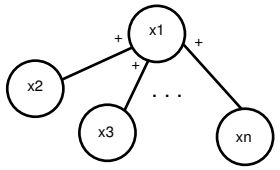

Fig. 1. A totally dependent positive formula

Lemma 5. For any 2-CF F_n with n variables which is not a totally dependent formula, it holds that $\#UNSAT(F_n) \geq \#SAT(F_n)$.

Proof: As only the totally dependent formula holds that $\#UNSAT(F) = 2^{n-1} - 1$. For any other formula F_n with n variables, we have that $\#UNSAT(F_n) \geq 2^{n-1}$, so the lemma is derived from the fact that $\#UNSAT(F_n) + \#SAT(F_n) = 2^n$. ■

Then, according to this last lemma, an upper bound for the value $\#SAT(F_n)$ is the same value of $\#UNSAT(F_n)$.

IV. An Incremental Computation for #2UNSAT

Let $F_d(n, n-1)$ be a totally dependent formula and let $C = \{x_i, x_j\}$ be a clause such that $v(C) \subset v(F_d)$, and $v(x_1) \notin v(C)$. This means that $(F_d \wedge C)$ has a triangle (see Figure 2)(the cycle: $x_1 - x_i - x_j - x_1$). It is not hard to compute $\#UNSAT(F_d \wedge C)$ because with just one application of the independence reduction on C and $C_i \in F_d$, the clause $C' = \{\neg x_1, x_i, x_j\}$ is independent with each clause in F_d, and as the independence reduction preserves the number of unsatisfied assignments, $\#UNSAT(F_d \wedge C) = \#UNSAT(F_d \wedge C') = \#UNSAT(F_d) + \#UNSAT(C') = 2^{n-1} - 1 + 2^{n-3}$.

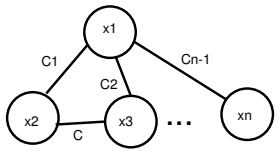

Fig. 2. A triangle in a totally dependent formula

In fact, the above procedure gives us an easy way to compute $\#UNSAT(F \wedge C)$ when the value $\#UNSAT(F)$ has already been computed. The procedure consists of applying the independence reduction on C and the clauses in F involving the variables $v(C)$ until a new clause C' is built which will be independent with all clauses $C_i \in F$ where $v(C) \cap v(C_i) \neq \emptyset$ and then $\#UNSAT(F \wedge C) = \#UNSAT(F \wedge C') = \#UNSAT(F) + \#UNSAT(C')$.

In [2], it was proved that if the constrained graph G_F of the formula is a path, then $\#UNSAT(F)$ is computed applying the following recurrence $(\alpha_i, \beta_i) =$

$$\begin{cases} (\mu_{i-1} + (2^{i-2} - \alpha_{i-1}), \mu_{i-1} &) \text{ if } (\epsilon_i, \delta_i) = (0,0) \\ (\mu_{i-1} & , \mu_{i-1} + (2^{i-2} - \alpha_{i-1})) \text{ if } (\epsilon_i, \delta_i) = (0,1) \\ (\mu_{i-1} + (2^{i-2} - \beta_{i-1}), \mu_{i-1} &) \text{ if } (\epsilon_i, \delta_i) = (1,0) \\ (\mu_{i-1} & , \mu_{i-1} + (2^{i-2} - \beta_{i-1})) \text{ if } (\epsilon_i, \delta_i) = (1,1) \end{cases} \quad (3)$$

Let $P = \{C_i\}_{i=1}^{5}$ be the path in Figure 3, then $F_1 = P \cup \{C_C\}$. $\#UNSAT(P) = 53$ and if we want to compute $(P \wedge C_C)$, C_C has to be made independent with each clause from P which is contained by the endpoints of the clause C_C. In Table 1, we analyze the falsifying strings of each clause in F_1 [3].

TABLE I
The falsifying strings for the clauses in F_1

	x_1	x_2	x_3	x_4	x_5	x_6
C_1	0	1	*	*	*	*
C_2	*	0	1	*	*	*
C_3	*	*	0	1	*	*
C_4	*	*	*	0	0	*
C_5	*	*	*	*	0	0
C_C	*	0	*	0	*	*

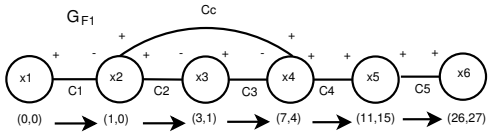

Fig. 3. Computing #UNSAT for the path in the graph.

By one application of the independence reduction between C_C and C_2, the new string for C'_C is formed as: $*000**$ and C'_C is independent with C_3. And as only C_2 and C_3 are the clauses embraced by the back edge in the cycle formed by $x_2 - x_3 - x_4 - x_2$, then $\#UNSAT(F_1) = \#UNSAT(P \wedge C_C) = \#UNSAT(P \wedge C'_C) = \#UNSAT(P) + \#UNSAT(C'_C) = 53 + 2^{n-|C'_C|} = 53 + 8 = 61$.

By duality, we have that $\#SAT(F_1) = 2^n - \#UNSAT(F_1) = 2^6 - 61 = 3$.

In the case that $v(C) \cap v(F) = \emptyset$, C must be tranformed, via independence reductions until a new clause C' is formed from C and F such that C' is independent with any other clause $C_i \in F$.

Definition IV.1. The constrained graph $G_F = (v(F), E)$ of a 2-CF formula $F = \{C_1, \ldots, C_m\}$ is called a tree if the following holds:

1) There exist j clauses $1 < j < n$ such that $\bigcap C_j \neq \emptyset$.
2) For any $F' \subset F$ whose constrained graph is a path, there are no transitive clauses from F' in F, and F has no cycles.

Lemma 6. If G_F is a path, then $F = \{C_1, C_2, \ldots, C_m\} = \{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_2}, x_3^{\delta_2}\}, \ldots,$

$\{x_m^{\epsilon_m}, x_{m+1}^{\delta_m}\}\}$, where $\delta_i, \epsilon_i \in \{0,1\}$, $i \in [\![m]\!]$. Let f_i be a family of clauses of the formula F built as follows: $f_1 = \emptyset; f_i = \{C_j\}_{j<i}$, $i \in [\![m]\!]$. Notice that $n = |v(F)| = m+1$, $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisfies } f_i\}$, $A_i = \{s \in SAT(f_i) : x_i \in s\}$, $B_i = \{s \in SAT(f_i) : \neg x_i \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

For every node $x \in G_F$, a pair (α_x, β_x) is computed, where α_x indicates how many times the variable x is true and β_x indicates the number of times that the variable x is false in the set of models of F. The first pair is $(\alpha_1, \beta_1) = (1,1)$ since x_1 can be true or false in order to satisfy f_1. The pairs (α_x, β_x) associated to each node x_i,$i = 2, \ldots, m$ are computed according to the signs (ϵ_i, δ_i) of the literals in the clause c_i. Let $F = \{C_1, \ldots, C_m\}$ be a 2-CF where G_F is a tree. For each father node x_p with two branches, one of size i (it has i clauses), and the other of size j (j clauses are involved), the recurrence for updating the charge associated to the node x_p: (α_p, β_p), is given as:

$$\begin{aligned} \alpha_p &= \alpha_i \cdot (2^j - \alpha_j) + \alpha_j \cdot (2^i - \alpha_i) + \alpha_i \cdot \alpha_j \\ \beta_p &= \beta_i \cdot (2^j - \beta_j) + \beta_j \cdot (2^i - \beta_i) + \beta_i \cdot \beta_j \end{aligned} \quad (4)$$

Proof: If two different branches, one of size i and the other of size j meet in the same father node, then there are two charges (α_i, β_i) and (α_j, β_j) associated at the same variable x_p (the father node). (α_i, β_i) initially denotes the number of falsifying assignments for the branch i, and then it updates considering all the assignments for the other branches minus the corresponding number of falsifying assignments ($2^j - \alpha_j$ or $2^j - \beta_j$ assignments). A similar argument is given when (α_j, β_j) is considered. Respectively we have to add the common falsifying assignments for the variable x_p; $\alpha_i \cdot \alpha_j$ for the value α_p and $\beta_i \cdot \beta_j$ for β_p. ■

It is obvious that acyclic graphs can be represented by trees. Let G_T be a tree, Figure 4 shows the result after applying equations (4) and (3) on G_T. In this instance, $\#2UNSAT(G_T) = 103 + 116 = 219$. Then, we obtain from the duality principle that $\#2SAT(G_T) = 2^7 - \#2UNSAT(G_T) = 256 - 219 = 37$. Thus, we have the following theorem.

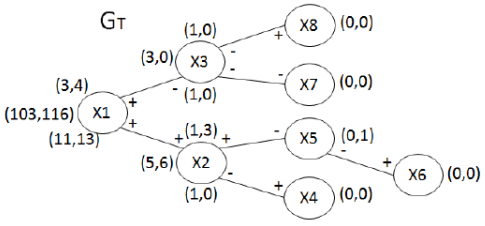

Fig. 4. Applying the formulas for $\#UNSAT$ on a Tree

Theorem 3. Let F be a 2-CF where G_F is an acyclic graph. $\#2UNSAT(F)$ is computed in linear time on the number of nodes in G_F.

Proof: A postorder traversal of the tree computes $\#UNSAT(F)$ while $\#2UNSAT(F)$ is calculated by applying the recurrences (3) and (4). ■

Remember that the hard cases to answer whether $\#SAT(F) = k$, is when $k > p(n)$ for a polynomial $p(n)$ where $n = |v(F)|$. Discarding the minimum dependent trees, we have that $2^n = \#UNSAT(F_n) + \#SAT(F_n) \geq 2^{n-1} + k$ then the value k is upper bounded by $2^n - 2^{n-1} = 2^{n-1}$. So, the hard cases for answering whether $\#SAT(F) = k$, are those where $p(n) < k < 2^{n-1}$. Thus, we can work over the computation of $\#UNSAT(F)$ as an upper bound for the value of $\#SAT(F)$.

V. Conclusion

We have determined the kind of 2-CF formulas with a minimum number of falsifying assignments from the set of all 2-CF formulas with the same number of variables and at least the same number of clauses. As a result of the duality property between $\#2SAT$ and $\#2UNSAT$, we determine the formulas with a maximum number of models from the set of all 2-CF formulas with the same number of variables and at least the same number of clauses.

We have shown that for any 2-CF F_n with n variables, $\#UNSAT(F_n) \geq \#SAT(F_n)$ with the exception of the totally dependent formulas. If $\#UNSAT(F_n) \leq p(n)$ for a polynomial on n, then $\#SAT(F_n) \leq p(n)$ too, and then $\#SAT(F_n)$ can be computed in polynomial time on the size of the formula.

References

[1] V. Dahllöf,P. Jonsson and M. Wahlström, "Counting models for 2SAT and 3SAT formulae", *Theoretical Computer Sciences*, vol. 332, no. 3, pp. 265-291, Feb. 2005.

[2] G De Ita, R. Marcial Romero and J.A. Hernández," A threshold for a Polynomial Solution of #2SAT", *Fundamenta Informaticae*, vol. 113, no.1, pp. 63-77, Jan. 2011.

[3] G. De Ita, P. Bello and M. Contreras, "New Polynomial Classes for #2SAT Established Via Graph-Topological Structure", *Engineering Letters*, vol. 15, pp. 250-258,Nov. 2007.

[4] G. De Ita and J.R. Marcial Romero, "Computing #2SAT and #2UNSAT by Binary Patterns", *Lecture Notes in Computer Sciences*, Vol. 7329, pp. 273-282, Jun. 2012.

[5] O. Dubois, "Counting the number of solutions for instances of satisfiability", *Theoretical Computer Sciences.*, vol. 81, pp. 49-64, Apr. 1991.

[6] N. Linial and N. Nisan," Approximate inclusion-exclusion". *Combinatorica*, vol. 10, no. 4, pp. 349-365, Dec.1993.

[7] E. Lozinskii E," Counting propositional models", *Information Processing Letters* , vol. 41, no. 4, pp. 327-332, Apr. 1992.

[8] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol 82, pp. 273-302, Apr. 1996.

[9] S.P. Vadhan," The Complexity of Counting in Sparse, Regular, and Planar Graphs", *SIAM Journal on Computing*, vol. 31, no.2, pp. 398-427, Sep. 2000.

[10] M. Wahlström, "A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances", *Lecture Notes in Computer Sciences* , vol. 5018, pp. 202-213, May. 2008.

[11] J. Zhou, M. Yin and C. Zhou, "New Worst-Case Upper Bound for #2-SAT and #3-SAT with the Number of Clauses as the Parameter", in *Proceedings of the AAAI 2010*, pp. 217-222.