A Polynomial Graphical Reduction to Speed Up the Counting of Models for Boolean Formulas

Guillermo De Ita Luna*, Meliza Contreras González+

*Universidad Autónoma de Puebla, +Universidad Angelópolis
deita@cs.buap.mx, *mel_22281@hotmail.com*

Abstract. In this paper, we focus on exact, deterministic algorithms for computing the number of models in Boolean formulas in Two Conjuntive Form (2-CF), denoted as #2-SAT problem.
We present a series of linear procedures which when they are integrated into a main program, allow us to compute in polynomial time the number of models of a formula F in 2-CF when the constraint graph G_F holds the following condition: G_F can be reduced to one free tree joined with a set of fundamental cycles, and such that those cyles are non-intersected (any pair of cycles do not share edges) or, they are intersected in just one edge. The resulting method for counting models in a 2-CF could be used to impact directly in the reduction of the complexity time of the algorithms for other counting problems.
Keywords: #SAT Problem, exact combinatorial algorithms.

1. Introduction

As is well known, #SAT problem is a classical #P-complete problem, and an interesting area of research has been the identification of restricted cases for which the #SAT problem can be solved efficiently.

#SAT is a special concern to Artificial Intelligence (AI), 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 [2, 5, 7]. The previous problems come from several AI applications such as planning, expert systems, data-mining, approximate reasoning, etc.

#SAT looks harder than the SAT problem, 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. On the other hand, 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) [3, 5, 6].

The dichotomy theorem [1] for #SAT just guarantees the tractability by affine formulas F (F is affine if the number of models of F is obtained via a linear system over a finite field) and the problem is #P-complete otherwise. We determine here new classes of formulas in 2-CF which are not affine and where the #2-SAT problem is solved in polynomial time. Our proposal is based on the topological structure of the constraint graph of the formula.

2. Notation and 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 is usual, for each $x_i \in X$, $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$.

A *clause* is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in I\!N$, a *k-clause* is a clause consisting of exactly k literals and, a *$(\leq k)$-clause* is a clause with k literals at most. 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) 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.

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

We use $\upsilon(X)$ to indicate the set of variables involved by the object X, where X could be a literal, a clause or a Boolean Formula.I.e. for the clause $c = \{\overline{x_1}, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$. And we will denote $[\![n]\!] = \{1, 2, ..., n\}$.

Let F be a Boolean formula in Conjuctive Form (CF), F is *satisfied* by an assignment s if each clause in F is satisfied by s. F is *contradicted* by s if any clause in F is contradicted by s. A model of F is an assigment over $\upsilon(F)$ that satisfies 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 $\upsilon(F)$. We will also denote $\mu_{\upsilon(F)}(F)$ by #SAT(F). When $\upsilon(F)$ will clear from the context, we will explicitly omit it as a subscript. #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.

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 $V = \upsilon(\Sigma)$ and $E = \{(\upsilon(x), \upsilon(y)) : (x, y) \in \Sigma\}$, that is, the vertices of G_Σ are the variables of Σ and for each clause (x, y) in Σ there is an edge $(\upsilon(x), \upsilon(y)) \in E$. Given a 2-CF Σ, a *connected component* of G_Σ is a maximal subgraph such that for every pair of vertices x, y, there is a path in G_Σ from x to y. We say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ.

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

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

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

In order to compute $\mu(\Sigma)$, first we should determine the set of connected components of Σ, and this procedure is done in linear time [5]. The different connected components of G_Σ conform the partition of Σ in its connected components. Then, compute $\mu(\Sigma)$ is translated to compute $\mu_{\upsilon(G)}(G)$ for each connected component G of Σ. From now on, when we mention a formula Σ, we suppose that Σ is a connected component graph. We say that a 2-CF Σ is a *cycle*, a *chain* or a *tree* if its corresponding graph G_Σ is a cycle, a chain or a tree, respectively.

3. Basic Procedures for computing #2-SAT

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

As Σ has m clauses then $|\upsilon(\Sigma)| = n = m + 1$. We 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 $\upsilon(\Sigma)$. The 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:

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

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).

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,* $\#SAT(\Sigma) = \mu(\Sigma) = \mu_5 = \alpha_5 + \beta_5 = 10 + 7 = 17$.

There are other procedures for computing $\mu(\Sigma)$ when Σ is a $(2, 2\mu)$-CF [5, 6], 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, as we have done in algorithms proposed previously [3, 4].

Case B:
Let G_Σ be a free tree and Σ its Boolean formula associated which has n variables and m clauses. 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. 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, 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., $u_1, u_2, ..., u_k$ are the child nodes of v, then as we have already visited all the child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been defined based on (2). $(\alpha_{v_i}, \beta_{v_i})$ is obtained by apply (2) over $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_{u_j}, \beta_{u_j})$. This step is iterated until computes all the values $(\alpha_{v_j}, \beta_{v_j})$, $j = 1, ..., k$. And 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$(\alpha_v + \beta_v)$.

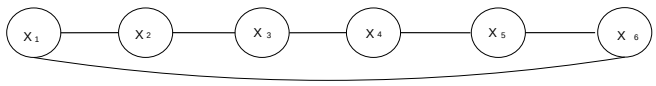
This procedure returns the number of models for Σ in time $O(n + m)$ which is the necessary time for traversing G_Σ in depth-first.

Case C:
Let G_Σ be 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

$| \upsilon(c_i) \cap \upsilon(c_{i+1}) | = 1$, 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: $y_0, y_1, ..., y_{m-1}, y_0$. We can apply the linear procedure described in (A) for computing $\mu(\Sigma')$. Every model of Σ' had determined logical values for the variables: y_{m-1} and y_0 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 $Y = \Sigma' \cup \{(y_{m-1}^{1-\epsilon_m}) \wedge (y_m^{1-\delta_m})\}$, $\mu(Y)$ is computed as a chain with two unitary clauses, then:

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

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 which represent a cycle* G_Σ*=(V,E). Note that* $|\upsilon(c_i) \cap \upsilon(c_{i+1})| = 1$*, i=1,...,5. Let* $G' = (V, E')$ *where* $E = E' \cup \{c_6\}$ *this is, the new graph* G' *is* Σ *minus the edge* c_6*. As* G' *is a chain graph with 6 nodes then* $\#SAT(G') = \alpha_6 + \beta_6 = 13 + 8 = 21$*. While the computing of* $\mu(Y)$ *is given by the series: (0,1)→(1,0)→(1,1)→(2,1)→ (3,2)→(5,3),obtaining as last pair associated to* $\mu(Y)$ *the pair* $(0, 3)$*. The pair associated to the last node of* Σ *is* $(13, 8) - (0, 3) = (13, 5)$*. And then* $\#SAT(\Sigma) = 13 + 5 = 18$.



$$(\alpha_1, \beta_1) \to (\alpha_2, \beta_2) \to (\alpha_3, \beta_3) \to (\alpha_4, \beta_4) \to (\alpha_5, \beta_5) \to (\alpha_6, \beta_6)$$
$$(1,1) \to (2,1) \to (3,2) \to (5,3) \to (8,5) \to (13,8)$$
$$(0,1) \to (1,0) \to (1,1) \to (2,1) \to (3,2) \to (5,3)$$
$$\Rightarrow (13,8) - (0,3) = (13,5)$$

Fig. 1. Computing $\#SAT(\Sigma)$ when G_Σ is a cycle

Case D:

Let G_Σ be a constraint graph with independent cycles. Note that the combination of the previous procedures for free trees and cyles 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 a complexity time of $O(m + n)$.

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. A Polynomial Graphical Reduction for Graphs with Intersected Cycles

When the constraint graph G_Σ of a Boolean formula Σ has intersected cyles to compute the number of models of Σ is done using *computing threads*. A main thread, denoted by Lp, is associated to the spanning tree of the graph, this thread is always active until all the process finishes.

When a fundamental cycle is processed we have to start new computing threads, one new thread for each one of the active threads so, the resulting number of active threads is twice than the previous one. The general procedure for counting the number of models in G_Σ is similar to the case (C) of previous chapter, we traversing G_Σ in depth-first search. A pair (α_i, β_i) is associated to each node v_i when it is visited by first time. At the beginning of the procedure, the main thread Lp is open, and new threads will be opened according to the start nodes of a cycle.

When the traversing by a cycle is finished then its respective threads (associated to this cycle) are closed and the last pair obtained (α, β) is changed to the pair $(0, \beta)$ which has to be substracted to the corresponding pairs of the remaining theads. This process is illustrated in Figure 3.

The procedure works over graphs G of degree 3. For example, consider the graph G on Figure 2, the depth-first search over G builds the graph on Figure 3, which is processed as we mention previously. But, if the graph G were transforming to the equivalent graph on Figure 4, then the number of computing threads is smaller.

The main purposes of the reduction between equivalent graphs is to build a graph where two intersected cycles has a common start node, and then each new thread carry on the values for both cycles at the same time, avoiding with this, to duplicate the number of computing threads. As we can see in Figure 4.

This polynomial graphical reduction is sketching in the following procedure.

Procedure Graph_Reduction
Input: The graph G built by the depth-first search, and where $\Delta(G) = 3$
Output: A new graph with cycles with an initial common node
Procedure:

1. if there are no intersected cycles then apply the case D
2. if the cycles start with a common node then exit
3. Let C_1 and C_2 be the intersected cycles
4. If both cycles do not have a common edge then exit
5. Begin the traversing by the back edge of C_1 and continue traversing by the nodes of degree 2 of the cycle C_1
6. when arrive to the other node of the back edge (the node of degree 3)

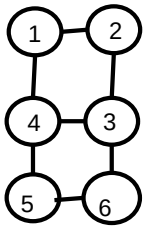
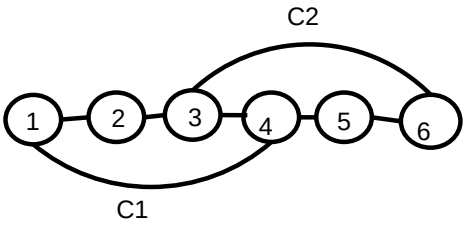
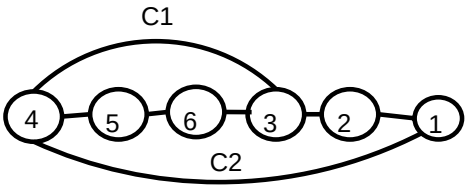


Fig. 2. An initial graph G



$Nodes:$	$Node_1$	$Node_2$	$Node_3$	$Node_4$	$Node_5$	$Node_6$
$Lp:$	$(1,1)$	$\rightarrow (2,1)$	$\rightarrow (3,2)$	$\rightarrow (5,3)$		
$C1:$	$(0,1)$	$\rightarrow (1,0)$	$\rightarrow (1,1)$	$\rightarrow (2,1)$ *closed*		
$C2 \rightarrow Lp:$			$(0,2)$	$\rightarrow (2,0)$		
$C2 \rightarrow C1:$			$(0,1)$	$\rightarrow (1,0)$ *closed*		
			$Lp:$	$(5,3)-(0,1)=(5,2)$	$\rightarrow (7,5) \rightarrow$	$(12,7)$
			$C2:$	$(2,0)-(0,1)=(2,0)$	$\rightarrow (2,2) \rightarrow$	$(4,2)$ *closed*
					$\Rightarrow (12,7)-(0,2)=$	$(12,5)$

Fig. 3. Computing $\#SAT(G_\Sigma)$ with intersected cycles



$Nodes:$	$Node_4$	$Node_5$	$Node_6$	$Node_3$	$Node_2$	$Node_1$
$Lp:$	$(1,1)$	$\rightarrow (2,1)$	$\rightarrow (3,2)$	$\rightarrow (5,3)$		
$C1\&C2:$	$(0,1)$	$\rightarrow (1,0)$	$\rightarrow (1,1)$	$\rightarrow (2,1)$ $C1$ *is closed*		
			$Lp:$	$(5,3)-(0,1)=(5,2)$	$\rightarrow (7,5) \rightarrow$	$(12,7)$
			$C2:$	$(2,1)-(0,1)=(2,0)$	$\rightarrow (2,2) \rightarrow$	$(4,2)$ *closed*
					$\Rightarrow (12,7)-(0,2)=$	$(12,5)$

Fig. 4. Computing $\#SAT(G_\Sigma)$ with intersected cycles and with a common start node

7. traversing by the back edge of the cycle C_2
8. continue traversing by the nodes of degree 2 until arrive to the first node of the total traversing
9. added the lack edges of the original graph G

This polynomial procedure builds a new graph where the original intersected cycles are rewriter with a common start node.

5. Conclusions

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, or if there exists intersected cycles they have at most one egde common, in all those cases, we show that $\#SAT(\Sigma)$ is computed in polynomial time over the length of the formula Σ.

This new polynomial classes of 2-CF contains to the class $(2, 2\mu)$-CF, and it does not have restrictions over the number of occurrences of a variable in the given formula, although $(2, 3\mu)$-SAT is a $\#P$-complete problem. Then, this new polynomial classes for #2-SAT 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.

References

1. Creignou N., Hermann M., Complexity of Generalized Satisfiability Counting Problems, *Information and Computation* 125, (1996), 1-12.
2. 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.
3. De Ita G., Polynomial Classes of Boolean Formulas for Computing the Degree of Belief, *Advances in Artificial Intelligence LNAI 3315*, 2004, 430-440.
4. De Ita G., Tovar M., Applying Counting Models of Boolean Formulas to Propositional Inference, *Advances in Computer Science and Engineering*, Vol 19, 2006, pp. 159-170.
5. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), 273-302.
6. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.
7. Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.