A Note for Parametric Complexity of #2SAT

Guillermo De Ita Luna, Fernando Zacarías Flóres, Alejandro Rangel Huerta

Faculty of Computer Sciences, BUAP
deita@cs.buap.mx, fzflores@yahoo.com.mx, arangel@cs.buap.mx

Abstract. We present some results about the parametric complexity for counting the number of truth assignments for two Conjunctive Forms (2-CF's), such problem is denoted as #2SAT. It is common to analyze the computational complexity for #2SAT regarding the number of variables or the number of clauses on the input formula F
We consider here, the time complexity analysis for #2SAT based on a positive integer parameter k, where k represents the number of satisfy assignments of F. We establish that for some values of k, the question: Is $\#2SAT(F) = k$?, can be answered in polynomial time with respect to the size of the formula F.
Keywords: #SAT, Efficient Algorithms, Enumerative Combinatorics, Parametric Complexity.

1 Introduction

To count combinatorial objects on graphs is an interesting and important area of research in Mathematics, Physics, and Computer Sciences. Counting problems, being mathematically relevant by themselves, are closely related to practical problems. For instance, reliability network issues are often equivalent to counting problems. Computing the probability that a graph remains connected, given the probabilities of failure over each edge, is essentially equivalent to counting the number of ways that the edges could fail without losing connectivity [11, 12].

The problem of counting models for a Boolean formula (#SAT problem) can be reduced to several problems in approximate reasoning. For example, in the cases of: estimating the degree of belief in propositional theories, the generation of explanations to propositional queries, repairing inconsistent databases, Bayesian inference and a truth maintenance systems [1, 2, 8, 10–12]. The previous problems come from several AI applications such as planning, expert systems, approximate reasoning, etc.

The combinatory problem that we are going to address is the computation of the number of satisfy assignments for Boolean formulas in two Conjunctive Forms, this problem is denoted as #2SAT problem. #2SAT is a classical $\#P$-complete problem even for the restricted cases of monotone and Horn formulas.

Among the class of #P-complete problems, #2SAT is considered a fundamental instance because of its relevance application in deduction issues and also, because it would allow to establish a boundary between efficient and intractable counting problems. As a result, given a 2-CF F with n variables and m clauses,

several methods with lower time upper bound than the trivial $O(2^n)$ or $O(2^m)$ have been proposed to solve #2SAT.

Alternatively, given a positive integer k and a 2-CF F, we want to know if the number of models in F is k. We show that for some specific values of the parameter k, the main question: is $\#2SAT(F) = k$?, can be solved in polynomial time with respect to the size of the formula F.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *boolean variables*. A *literal* is either a variable x_i or a negated variable $\overline{x_i}$. As usual, for each $x_i \in X$, $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$.

A *clause* is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in I\!N$, a *k-clause* is a clause consisting of exactly k literals and, a *$(\leq k)$-clause* is a clause with at most k literals. A variable $x \in X$ *appears* in a clause c if either x or $\overline{x}$ is an element of c.

A *conjunctive form* (CF) F is a conjunction of clauses (we also consider a CF as a set of clauses). We say that F is a monotone CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses and, $(\leq k)$-CF denotes a CF containing clauses with at most k literals. 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 $v(X)$ to express the variables involved in the object X, where X could be a literal, a clause or a CF. For instance, for the clause $c = \{\overline{x_1}, x_2\}$, $v(c) = \{x_1, x_2\}$. $Lit(F)$ is the set of literals appearing in F, i.e. if $X = v(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$. We denote $\{1, 2, ..., n\}$ by $[\![n]\!]$ and the cardinality of a set A by $|A|$.

An assignment s for F is a boolean function $s : v(F) \to \{0, 1\}$. An *assignment* can also be considered a set of non complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l *true* and $\overline{l}$ *false*. Considering a clause c and assignment s as a set of literals, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$, and if for all $l \in c$, $\overline{l} \in s$, then s falsifies c.

Assuming $n = |v(F)|$, then there are 2^n possible assignments defined over $v(F)$. We denote the set of all 2^n assignments defined over $v(F)$ by $S(F)$.

If $F_1 \subset F$ is a formula consisting of some clauses from F, and $v(F_1) \subset v(F)$, an assignment over $v(F_1)$ is a *partial* assignment over $v(F)$. Assuming $n = \mid v(F) \mid$ and $n_1 = \mid v(F_1) \mid$, any assignment over $v(F_1)$ has 2^{n-n_1} extensions as assignments over $v(F)$.

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 M of F is an assignment for $v(F)$ that satisfies F. The SAT problem consists of determining if 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. And let #UNSAT(F) $= 2^n - \#SAT(F)$.

A 2-CF formula F can be represented by an undirected graph. Let F be a 2-CF, the *constrained graph* of F is the undirected graph $G_F = (V(F), E(F))$, 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 $\{\upsilon(x), \upsilon(y)\} \in E(F)$.

The *neighborhood* for $x \in V(F)$ is $N(x) = \{y \in V(F) : \{x, y\} \in E(F)\}$ and its *closed neighborhood* is $N(x) \cup \{x\}$ denoted as $N[x]$. The degree of a variable x, denoted by $\delta(x)$, is $|N(x)|$, and the degree of F is $\Delta(F) = max\{\delta(x) : x \in V(F)\}$.

3 Parametric Time Complexity for #2SAT

An alternative to compute $\#SAT(F)$, for a given 2-CF F, is to determine the set of connected components of its constrained graph G_F. In [10], it has been proved that the set of connected components of a constrained graph can be determined in linear time with respect to the number of clauses in the formula. Thus,

$$\#2SAT(F) = \#2SAT(G_F) = \prod_{i=1}^{k} \#2SAT(G_i) \tag{1}$$

where $\{G_1, \ldots, G_k\}$ is the set of connected components of G_F [10].

The set of different connected components of G_F conforms a partition of F. Of course, if the number of models in any connected component of F is zero, then $\#SAT(F) = 0$. So, from now on, we will design an algorithm which solves #2SAT just for one connected component. And its application to 2CF- formulas with several connected components is inferred from (1). We say that a 2-CF F is a *path*, a *cycle*, or a *tree* if its corresponding constrained graph G_F is a path, a cycle, or a tree, respectively.

Let $F = \{C_1, C_2, \ldots, C_m\}$ be a strict 2-CF (all clause has length 2) and let $n = |\upsilon(F)|$. We consider here, the size of F as $(n + m)$. If F is not a empty formula then any clause $C_i \in F, i = 1, \ldots, m$ is falsified by 2^{n-2} assignments from the 2^n possible assignments of $S(F)$, in fact, by those assignments where both literals from C_i take the false value. Thus, #SAT(F) $\leq 2^{n-2}$.

Let k be a positive integer parameter such that $k \leq 2^{n-2}$. Our main concern is to determine if #SAT(F) $= k$. For this question there is an immediate answer for some values of k. For example, if $k = 0$ or $k = 1$ the *Transitive Closure* procedure could be applied to answer such question, and its is known that such procedure has a linear time complexity over the size of the 2-CF F (see [7]).

In fact, if the value k can be upper bounded by a polynomial function on n, e.g. $k \leq n^3$, then in [3], an exact algorithm, called *Count_Models*, is shown for determining if #SAT(F) $\leq n^3$. Such algorithm runs on polynomial time complexity on the size of F. Thus, *Count_Models* can be applied for answering if #SAT(F) $= k$ when $k \leq n^3$.

Then, the hard cases to answer if $\#\mathrm{SAT}(F) = k$, appear when the parameter k is a higher value than n^3. However, $\#\mathrm{SAT}(F)$ could not be so close to 2^{n-2} while $m > 1$.

A way that the value $\#\mathrm{SAT}(F)$ might be close to 2^{n-2} is when F has few clauses. But in this case, almost all exact procedures can compute $\#\mathrm{SAT}(F)$ in polynomial time. For example, if $m \leq n$ then G_F is an acyclic graph, or a simple cycle. In the both previous cases, the procedures shown in [4] compute $\#\mathrm{SAT}(F)$ in linear time on the size of F.

From now on, we assume $m > n$. Let $nc = m - n + 1$ be the number of fundamental cycles in the graph G_F. Let v_r be any node of G_F chosen to start a depth-first search. We obtain a spanning tree T_F with v_r as the root node and a set of fundamental cycles $\mathcal{C} = \{C_1, C_2, ..., C_{nc}\}$, where each back edge $c_i \in E$ marks the beginning and the end of a fundamental cycle.

Given any pair of fundamental cycles C_i and C_j of $\mathcal{C}$, $i \neq j$, If C_i and C_j share edges, we call them *intersecting* cycles; otherwise, they are called *independent* cycles. Let A_F be the depth-first search graph formed by the spanning tree T_F and the set of fundamental cycles $\mathcal{C}$.

If the fundamental cycles appearing in A_F are independent one from the other. Then there is a procedure in [4], called *Count_Models_for_Disjoint_cycles*, which computes $\#\mathrm{SAT}(F)$ in polynomial time on the size of F, without regarding the value of nc.

Finally, the hard cases for determining if $\#\mathrm{SAT}(F) = k$ are the cases when there is intersected cycles in G_F. Let ic be the number of intersected cycles in G_F. Notice that $ic \leq nc = m - n + 1$ and that ic is therefore smaller than m.

Definition 1. *Let F be a 2-CF and x a literal of F. The reduction of F by x, also called forcing x and denoted by $F[x]$, is the formula generated from F by the following two rules*

a) removing from F the clauses containing x (subsumption rule),
b) removing $\overline{x}$ from the remaining clauses (unit resolution rule).

A reduction is also sometimes called a *unit reduction*. Now, the reduction by a set of literals can be inductively established as follows: let $s = \{l_1, l_2, \ldots, l_t\}$ be a set of literals, the reduction of F by s is defined as successively applying definition (1) for l_i, $i = 1, \ldots, t$. That is, the reduction of F by l_1 gives the formula $F[l_1]$, following a reduction of $F[l_1]$ by l_2, giving as a result the formula $F[l_1, l_2]$ and so on. The process continues until $F[s] = F[l_1, ..., l_k]$ is reached. If $s = \emptyset$ then $F[s] = F$.

Example 1. Let $F = \{\{x_1, \overline{x}_2\}, \{x_1, x_2\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{\overline{x}_2, x_4\}, \{\overline{x}_2, \overline{x}_4\}, \{x_2, x_5\}, \{x_3, \overline{x}_5\}\}$. Then, $F[\overline{x}_2] = \{\{x_1\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{x_5\}, \{x_3, \overline{x}_5\}\}$. And if $s = \{x_2, \overline{x}_3\}$ we have that $F[s] = \{\{x_1\}, \{x_1\}, \{\overline{x}_1\}, \{x_4\}, \{\overline{x}_4\}, \{\overline{x}_5\}\}$.

Furthermore, during the computation of $F[s]$ new unitary clauses can be generated. The partial assignment s could be extended by adding the literals coming from the new unitary clauses found, that is, $s = s \cup \{u\}$ where $\{u\}$ is

Algorithm 1 Unit_Propagation(F, s)

Input: F a 2-CF formula and s - a partial assignment of F.
Let s' be a global variable initiated with the initial value s.
$F' = F[s]$ /* forcing s */
for each new unitary clause $\{u\}$ in F' **do**
  $\{s' = s' \cup \{u\}$
  $F' = Unit\_Propagation(F', \{u\})\}$
end for
return F'

a unitary clause. So, $F[s]$ can be reduced again using the new unitary clauses. The above mentioned iterative process is generalized in the following procedure.

For simplicity, we will abbreviate $Unit\_Propagation(F, s)$ as $UP(F, s)$. Given a 2-CF F and an initial assignment s, we obtain after the application of $UP(F, s)$ a subformula F' and an extended assignment s' from s. s' is s extended by the literals appearing in the unitary clauses found during the iterative process $UP(F, s)$. Let $T(s) = s'$ be such extended assignment built by the procedure $UP(F, s)$.

Let F be a 2-CF and let s be a partial assignment of F. If a pair of contradictory unitary clauses is obtained while $F[s]$ is being computed then $\#SAT(F[s]) = 0$, because under no circumstances a pair of complementary unit clauses can be set to true at the same time, and then, $F[s]$ does not have models.

The application of definition (1 a) in a 2-CF formula F, could remove variables which have to be considered in the models of F. Let us present an example.

Example 2. Let $F = \{\{x\}, \{x, y\}\}$, in other words $F = x \wedge (x \vee y)$, it follows that $F[x] = \emptyset$. It can be noticed that $\{y\} \notin F[x]$, however y can take any logical value in the models of F.

The variables which are removed from F during the application of $UP(F, s)$ form a set, which will be denoted by $Elim\_Vars(s)$. This set is constructed during the computation of $UP(F, s)$. In fact, it can be reviewed that $|Elim\_Vars(s)| = |\upsilon(F)| - |\upsilon(UP(F, s)| - |T(s)|$.

It is time to show that counting the number of models of a 2-CF formula F can be simplified by counting the number of models of sub-formulas of F.

Theorem 1. *Let F be a 2-CF and x a variable in $\upsilon(F)$. Let $F_1 = UP(F, x)$ and $F_2 = UP(F, \overline{x})$ then*

$$\#SAT(F) = \#SAT(F_1) * 2^{|Elim\_Vars(x)|} + \#SAT(F_2) * 2^{|Elim\_Vars(\overline{x})|}$$

Proof. We can split the number #SAT(F) between the models where $x \in \upsilon(F)$ is true and the models where x is false. The number of models where x is true are counted by $\#\text{SAT}(F_1) * 2^{|Elim\_Vars(x)|}$ where $F_1 = UP(F, x)$. While the number of models where x is false are counted by $\#\text{SAT}(F_2) * 2^{|Elim\_Vars(\overline{x})|}$, F_2 being the subformula $F_2 = UP(F, \overline{x})$. As $T(x)$ and $T(\overline{x})$ haven't common elements since $x \in T(x)$ and $\overline{x} \in T(\overline{x})$ then in the above sum we don't consider common models between $SAT(F_1)$ and $SAT(F_2)$.

Let us assume G_F has ic intersected cycles. Let $x \in v(F)$ be the variable appearing in a maximum number of intersected cycles of G_F. Let $F_1 = UP(F, x)$ and $F_2 = UP(F, \overline{x})$, then the application of the theorem (1) defines a splitting rule for computing #SAT(F). And the iterative application of that splitting rule over the resulting subformulas determines a branch and bound algorithm whose base case is when the subformula does not have intersected cycles. Let us show how this algorithm works with an example.

Example 3. Let F be a monotone 2-CF, $F = \{\{x_1, x_2\}, \{x_1, x_9\}, \{x_2, x_3\}, \{x_2, x_8\}, \{x_3, x_4\}, \{x_3, x_{10}\}, \{x_4, x_5\}, \{x_4, x_7\}, \{x_5, x_6\}, \{x_6, x_7\}, \{x_6, x_9\}, \{x_8, x_7\}, \{x_8, x_9\}\}$. F contains 4 intersected fundamental cycles. Then, $F_1 = UP(F, x_7) = \{\{x_1, x_2\}, \{x_1, x_9\}, \{x_2, x_3\}, \{x_2, x_8\}, \{x_3, x_4\}, \{x_3, x_{10}\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_9\}, \{x_8, x_9\}\}$ which includes two intersected cycles and $F_2 = UP(F, \overline{x}_7) = \{\{x_1, x_2\}, \{x_1, x_9\}, \{x_2, x_3\}, \{x_3, x_{10}\}$ which is an acyclic graph. We can see the processing of F in figure 1. Notice that the splitting rule has to be performed again to process F_1, since F_1 has a pair of intersected cycles. While F_2 represents a basic instance because it has not intersected cycles and to count its models is done by the $Count\_Models\_for\_Disjoint\_cycles$ procedure in polynomial time on the size of F_2.

Notice that all clauses of F where the variable x appears, are not more clauses neither F_1 nor F_2. And as x appears in at least one pair of intersected cycles from G_F then G_{F_1} and G_{F_2} have at least two intersected cycles less than G_F. Consequently, the following recurrence equation holds:

$$r(ic) = 2 * r(ic - 2) = 2^t * r(ic - 2 * t)$$

.

Such recurrence ends when $ic - 2 * t = 1$, that is, when $t = (ic - 1)/2$. Therefore, the time complexity will be $O(2^{(ic-1)/2})$. In the case that G_F contains a small number of intersected cycles, e.g. $ic \leq n/2$, then the above branch and bound procedure would have a time complexity upper bounded by $O(2^{n/4}) = O(1.1892^n)$.

To check for the conditions to apply the previous algorithm is easy to do, since we apply a depth first search over G_F and we count the number of intersected cycles ic in G_F in order to determine if $ic \leq n/2$. All those previous tasks can be performed in linear time on the size of F. While the search for the variable x appearing in a maximum number of intersected cycles in G_F that require a time of order ic^2, that is polynomial on the size of F.

In fact, we can accelerate the computation of #SAT(F), if for example, all variable with one occurrence in the formula is processed before applying the splitting rule. Another way to accelerate the computation of #SAT(F) is when we consider the application of two sequential splitting rules at the same time. For example to select a clause $c = \{l_1, l_2\} \in F$ where l_1 is the literal which appears in a maximum number of intersected cycles and l_2 has a maximum degree into the set $N(v(l_1))$.

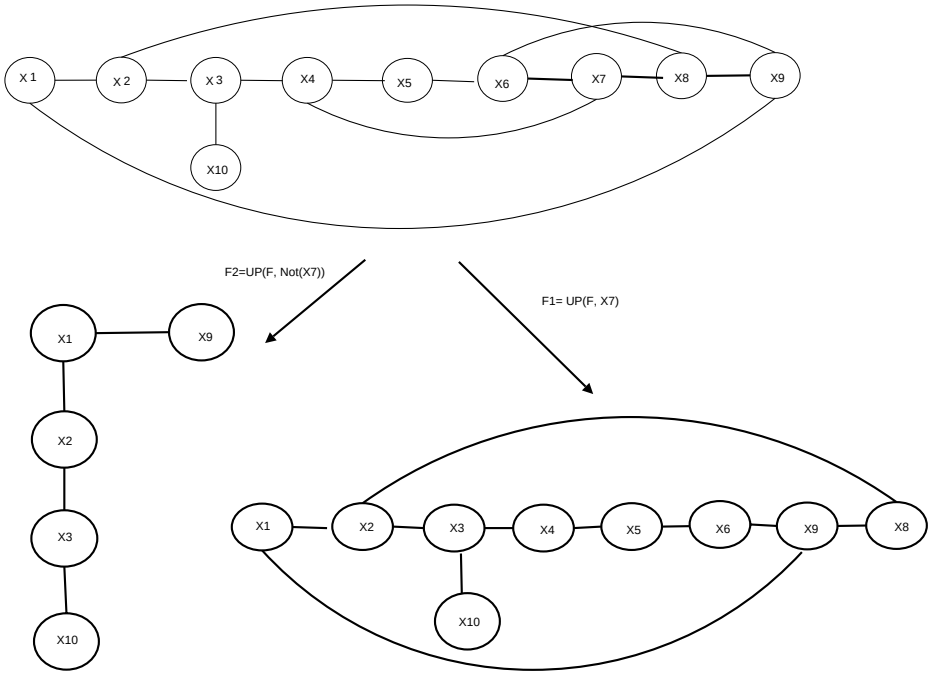

Fig.1 Applying the splitting rule on a monotone 2-CF.

The application of two consecutive splitting rules, first with l_1 and next with l_2, generate three new branches in the enumerative tree instead of four. As the branch for $UP(F, [\overline{l}_1, \overline{l}_2])$ generates an unsatisfiable subformula (since it falsify to c) then such branch is discarded from the enumerative tree.

3.1 The General case

If there are a great number of intersected cycles, that means that $m >> n$, then there is a bigger number of clauses than the number of variables.

Let $F = \{C_1, C_2, \ldots, C_m\}$ be a stricter 2-CF involving n variables. For any clause $C_i \in F$, there are exactly $2^{n-|C_i|}$ assignments from $S(F)$ falsifying C_i. Let $C_i, C_j \in F, i \neq j, i, j = 1, \ldots, m$. Each clause $C_i, C_j \in F$ is falsified by 2^{n-2} assignments from the 2^n possible assignments from $S(F)$. Both clauses suppress generally fewer than $2^{n-2} + 2^{n-2}$ of assignments from $S(F)$ because they have common assignments falsifying both clauses. I.e. $\#UNSAT(C_i \vee C_j) = \#UNSAT(C_i) + \#UNSAT(C_j) - \#UNSAT(C_i \wedge C_j)$.

For example, if $C_i \cap C_j = \{l\}$ and there are not complementary pair of literals in $C_i \cup C_j$ then there are a common partial assignment formed by $C_i \cup C_j$ which falsifies both clauses. As, $|C_i \cup C_j| = 3$, the clauses C_i and C_j are falsified by a total number of $2^{n-2} + 2^{n-2} - 2^{n-3}$ assignments from $S(F)$.

If C_i and C_j contain a common variable in opposite forms, that is direct in C_i and complemented in C_j, the unsatisfied assignments of each clause form disjoint sets of assignments. Consequently, both clauses suppress exactly 2^{n-2} + 2^{n-2} assignments from $S(F)$.

The property between two clauses having at least one common variable in opposite form is called *independence*. Two clauses are independent if they have at least one common variable in opposite form [5]. Otherwise, we say that both clauses are *dependent*.

In general, the inclusion-exclusion formula can be applied for computing #UNSAT(F). Several algorithms have been designed as finer or shorter versions of the application of the inclusion-exclusion formula in order to compute #UNSAT(F) [5, 9].

Each clause is falsified by a determined number of assignments from $S(F)$, but while more and more clauses are considered, then more and more assignments are discarded from $S(F)$. Therefore, it is expected that #SAT(F) = $2^n - \#UNSAT(F)$ would be a smaller value than 2^{n-2}, in which case, it might be that #SAT(F) could be upper bounded by a polynomial value on n. And then, as we have seen, #SAT(F) can be computed in polynomial time on the size of F.

Thus, the real hard cases for computing #SAT(F) are when there are a considerable number of intersected cycles in G_F, e.g. $ic \geq n/2$, and when it is expected that #SAT(F) would be greater than, for example, n^3.

In that latter case, we can apply the algorithm which provides the leader upper bound on the time complexity for computing #2SAT. For instance, in ([13]) a branch and bound procedure for computing #2SAT with an upper bound of $O(1.2377^n)$ was shown.

In fact, one of the important tasks when branch and bound procedures are performed for computing #2SAT, is how to select a pivotal variable (or pivotal clause) for applying the branching rule.

We have shown, in the previous subsection, an adequate way to select the pivotal variable x for applying the splitting rule. With the intention of reducing the number of intersected cycles in each application of the splitting rule, it is adequate to select the variable x that appears in the maximum number of intersected cycles. This guarantees that all cycles where x is part of them, will be reduced in the subformulas associated to the new branches. And then, this will accelerate the processing of intersected cycles appearing in the current subformulas.

The results presented here, allow us to establish in a more precise way the boundary between efficient and intractable instances for the #2SAT Problem.

4 Conclusions

Given a 2-CF F with n variables, m clauses, and a positive integer parameter k, the question: Is #SAT(F) = k?, can be answered in an efficient way for values of k; $k \leq n^3$ or $k \geq 2^{n-2}$. When k is outside of those interval values, is convenient

to apply a depth first search on G_F (the constrained graph of F), and to count the number of intersected fundamental cycles (ic) appearing on G_F.

If $ic \leq n/2$ then the branch and bound algorithm presented here, computes $\#SAT(F)$ with a time complexity of $O(2^{ic/2})$, which in a worst case has an upper bound of $O(1.1892^n)$. Otherwise, we can apply the leader upper bound time complexity algorithm which computes $\#\mathrm{SAT}(F)$ with a running time of $O(1.2377^n)$. Therefore, in any case, to answer the question: Is $\#\mathrm{SAT}(F) = k?$, requires until now, an exponential time complexity on the number of intersected cycles in G_F or an exponential time complexity on the number of variables in F.

References

1. Darwiche A., On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, *Jour. of Applied Non-classical Logics*, (2001), pp. 11-34.
2. Dahllöf V., Jonsson P., Wahlström M., Counting models for 2SAT and 3SAT formulae., *Theoretical Computer Sciences*, 332(1-3), (2005), pp. 265-291.
3. De Ita G., Marcial-Romero R., Hernández J.A., A threshold for a Polynomial Solution of #2SAT, *Fundamenta Informaticae*, to appear.
4. De Ita G., Bello P., Contreras M., New Polynomial Classes for #2SAT Established Via Graph-Topological Structure, *Engineering Letters*, Vol. 15;2, Int. Assoc. of Engineers, www.engineeringletters.com, (2007), pp. 250-258.
5. Dubois Olivier, Counting the number of solutions for instances of satisfiability, *Theor. Comp. Sc.* 81, (1991), 49-64.
6. Fernau H., On Parameterized Enumeration, *LNCS 2387*, (2002), pp. 564-573.
7. Gusfield D., Pitt L., A bounded approximation for the minimum cost 2-Sat problem, *Algorithmica* 8, (1992), 103-117.
8. Khardon R., Roth D., Reasoning with Models, *Artificial Intelligence*, Vol. 87, No. 1, (1996), pp. 187-213.
9. Lozinskii E., Counting propositional models, *Inf. Proc. Letters* 41, (1992), pp. 327-332.
10. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), pp. 273-302.
11. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distinguished dissertations Springer, 2001.
12. Vadhan Salil P., The Complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), pp. 398-427.
13. Wahlström M., A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances, *LNCS 5018*, (2008), pp. 202-213.