The Computation of #2SAT by a Fixed-Parameter Tractable Algorithm

Guillermo de Ita, Pedro Bello, and Miguel Rodriguez

Benémerita Universidad Autónoma de Puebla, Mexico
{deita,pbello,mrodriguez}@cs.buap.mx

Abstract. The counting of models for two conjunctive forms (2-CF), problem know as #2SAT, is a classic #P problem. We determine different structural patterns on the underlying graph of a 2-CF F that allows the efficient computation of #2SAT(F). We show that if the constrained graph G_F of a 2-CF F is acyclic or it has cycles without common edges with other cycles, then #2SAT(F) is computed efficiently.
Finally, if G_F has entwined cycles (set of cycles with common edges), we have designed a deterministic fixed-parameter algorithm for #2SAT, that is, an algorithm whose time complexity is $O(2^k \cdot poly(|F|))$, where $poly(|F|)$ is a polynomial function on the size of F, and k denotes the maximum 'edge-entwined' of G_F, that is, k is a parameter equal to the maximum number of fundamental cycles of G_F with a common edge.

Keywords: #SAT Problem, · Counting models,· Efficient Enumerative Algorithms,· Parameterized Complexity

1 Introduction

#SAT (the problem of counting models for a Boolean formula) is of special concern to Artificial Intelligence (AI), and it has a direct relationship to Automated Theorem Proving, as well as to approximate reasoning [5, 6, 18]. #SAT can be reduced to several different problems in approximate reasoning.

For example, in the cases of; the estimation of the degree of belief in propositional theories, the generation of explanations to propositional queries, the reparation of inconsistent databases, the approximation in Bayesian inference [1, 5, 6, 18, 19]. In general approach used to compute the degree of belief, 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, Prob is used to denote the probability, and $\mu(\Sigma)$ denotes the number of models that Σ has.

#SAT is as hard as the SAT problem, but in many cases, even when SAT is solved in polynomial time, there is not any efficient method known for #SAT. For example, 2SAT problem (SAT restricted to consider two conjunctive forms), it can be solved in linear time. However, the corresponding counting problem #2SAT is a #P-complete problem.

The counting of combinatorial objects on graphs is an interesting and important area of research in Mathematics, Physics, and Computer Sciences. For example, the maximum polynomial class recognized for #2SAT is the class $(\leq 2, 2\mu)$-CF (two conjunctive forms where each variable appears twice at most) [18, 19].

Earlier works on #2-SAT include papers by Dubois [8], Zhang [22] and Littman [12]. More recently, new upper bounds for exact deterministic algorithms for #2-SAT have been found by Angelsmark [1], Dahllöf [2], Fürer [11], and Wahlstr$\ddot{o}$m [21].

All the above proposals are part of the class of exponential algorithms, and several of them follow the branch and backtracking technique begun in the pioneering algorithm by Davis & Putnam (D&P). Notable improvements include Kullmann's work on complexity measures [15], and Eppstein's work on solving multivariate recurrences through quasiconvex analysis [9]. Still, one limitation that remains in Eppstein's framework is that it is difficult to introduce (non-trivial) restrictions on the applicability of a possible recursion [21].

Parameterized complexity theory relaxes the classical notion of tractability and allows us to solve some classically hard problems in a reasonable time. It turned out that many intractable problems can be solved efficiently "by the slice", that is, in time $O(f(k) \cdot n^c)$, where f is any function of some parameter k, n is the size of the instance, and c is a constant independent from k.

In this case, the problem is called fixed-parameter tractable (FPT). If a problem is in the class FPT, then large instances can be solved efficiently, although it can be an exponential-time that depends on the fixed-parameter [20].

Bacchus et al. [3] consider structural restrictions on the input formula and propose an algorithm for #SAT whose complexity is polynomial in its number of variables and exponential in the 'branch-width' of the underlying constraint graph. Gottlob et al. [13] provide a similar result in terms of the 'tree-width' of the constrained graph. Fischer et al. [10] extend this to a similar result in terms of 'cluster-width' [4].

Then, although it exists parameterized algorithms to solve #SAT, they request to build a k-tree decomposition or the k-branch decomposition of the constrained graph of the input formula (a problem that is known to be NP-complete if we want the minimum value for k). Furthermore, the time complexity on the fixed-parameter algorithms comes from being doubly exponential in k [16] to be of order 4^k [10].

All of the previous procedures are parameterized algorithms, where a parameter k, bounded more by the topologies that lye in the formula than in the number of variables (or clauses) of the formula, determines the exponential complexity for solving the #2SAT problem. Indeed, for those previous algorithms to find an optimal value for the parameter k for any input formula request to solve a NP-complete problem.

We focus toward the identification of underlying structures on the constrained graph G_F of the input formula F, that allow us the efficient computation of the number of models of F. We extend here the procedures presented in [6, 7], where different structural patterns on the constrained graph are identified in order to

compute #2SAT in an incremental and efficient way, at the same time that its constrained graph is being traversing systematically.

The new result presented is a novel deterministic fixed-parameter tractable algorithm for #2SAT, based on our previous procedures and that works on the constrained graph G_F of the input formula F. The time function for the parameter k in our algorithm is upper bounded by 2^k. Furthermore, our proposal does not build the tree-width of the formula, instead we show how to compute the parameter k in an efficient way. Our parameter k represents the 'edge-entwined' of the constrained graph.

While these improved bounds are only guaranteed to hold if dedicated algorithms are used, these parameters can still be seen as ways in which an instance can be easy, hopefully in ways that influence the behaviour of the "popular" (i.e. commonly used) algorithms as well [21].

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 $x_i \in X$, $x_i^0 = x_i$ and $x_i^1 = x_i$. A clause is a disjunction of different literals (we also consider a clause as a set of literals). For $k \in N$, 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 can also consider a CF as a set of clauses). We say that F is a positive 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 $\nu(Y)$ to express the set of variables involved in the object Y, where Y could be a literal, a clause or a conjunctive formula. For instance, for the clause $c = \{\overline{x}_1, x_2\}, \nu(c) = \{x_1, x_2\}$. And $Lit(F)$ is the set of literals that appears in a CF F, i.e. if $X = \nu(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$. We denote the cardinality of a set A, by $|A|$. We also denote $\{1, 2, \ldots, n\}$ by $[[n]]$.

An assignment s for F is a Boolean function $s : \nu(F) \to \{0, 1\}$. An assignment can also be considered as a set of non complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l true and $\bar{l}$ false. Considering a clause c and an assignment s both as a set of literals, c is satisfied by s if and only if $(c \cap s) \neq \emptyset$, and if for all $l \in c$, $\bar{l} \in s$ then s falsifies c. If $F_1 \subset F$ is a formula consisting of some clauses of F, then $\nu(F1) \subset \nu(F)$, and an assignment over $\nu(F_1)$ is a partial assignment over $\nu(F)$. Assuming $n = |\nu(F)|$ and $n_1 = |\nu(F_1)|$, any assignment over $\nu(F_1)$ has 2^{n-n_1} extensions as assignments over $\nu(F)$.

Let F be a Boolean formula in Conjunctive 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 assignment for $\nu(F)$ that

satisfies F. Given a CF F, the SAT problem consists of determining if F has a model. The #SAT problem consists of counting the number of models of F defined over $\nu(F)$. #2SAT denotes #SAT for formulas in 2-CF.

2.1 The Constrained graph of a 2-CF

There are some graphical representations of a conjunctive form (see e.g. [10, 20]). We use here the signed primal graph of a two conjuctive form. Let F be a 2-CF, its signed constrained undirected graph (constrained graph) is denoted by $G_F = (V(F), E(F))$, with $V(F) = \nu(F)$ and $E(F) = \{\{\nu(x), \nu(y)\} : \{x, y\} \in F\}$, that is, the vertices of G_F are the variables of F, and for each clause $\{x, y\}$ in F there is an edge $\{\nu(x), \nu(y)\} \in E(F)$. Each edge $c = \{\nu(x), \nu(y)\} \in E$ is associated with an ordered pair (s_1, s_2) of signs, assigned as labels of the edge that connect the variables appearing in the clause. The signs s_1 and s_2 are related to the signs of the literals x and y respectively. For example, the clause $\{\overline{x} \vee y\}$ determines the labelled edge: "$x \overset{-+}{—} y$" which is equivalent to the edge "$y \overset{+-}{—} x$".

Let $S = \{+, -\}$ be a set of signs. A graph with labelled edges on a set S is a pair (G, ψ), where $G = (V, E)$ is a graph, and ψ is a function with domain E and range S. $\psi(e)$ is called the label of the edge $e \in E$. Let $G = (V, E, \psi)$ be a signed graph with labelled edges on SxS. Let x and y be nodes in V . If $e = \{x, y\}$ is an edge and $\psi(e) = (s, s')$, then $s(s')$ is called the adjacent sign to $x(y)$. From now on, the term for describing the topology of a constrained graph G_F is inherited to its original 2-CF formula F.

Two vertices v and w of a constrained graph G_F are called *adjacent* if there is an edge $\{v, w\} \in E(G_F)$, joining them. The *neighbourhood* of $x \in V(G_F)$ is $N(x) = \{y \in V : \{x, y\} \in E(G_F)\}$ and its *closed neighbourhood* is $N(x) \cup \{x\}$ which is denoted by $N[x]$. Note that v is not in $N(v)$, but is in $N[v]$. The degree of a vertex $x \in V(G_F)$, denoted by $\delta(x)$, is $|N(x)|$. If A is a set of vertices from a graph G, $N(A)$ is the set of neighbour vertices from any vertex of A, that is, $N(A) = \cup_{x \in A} N(x)$, while $N[A] = N(A) \cup A$. The maximum degree of G_F or just the degree of G_F is $\Delta(G_F) = max\{\delta(x) : x \in V(G_F)\}$.

Notice that a constrained graph of a 2-CF can be a multigraph since two fixed variables can be involved in more than one clause of the formula forming parallel edges. Furthermore, an unitary clause is represented by a loop in the constrained graph (an edge to join a vertex to itself).

3 Processing Constrained Graphs with Cycles

Let $G_F = (V, E)$ be a simple cycle with m nodes, that is, all the variables in $v(F)$ appear twice, $m = n = |V| = |E|$. F can be decomposed as $F = F' \cup c_m$, where $F' = \{c_1, ..., c_{m-1}\}$ is a path and $c_m = (x_{m-1}^{\epsilon_m}, x_1^{\delta_m})$ is the edge that conforms with $G_{F'}$ a simple cycle: $x_1, x_2, ..., x_{m-1}, x_1$. We will call $G_{F'}$ the internal path of the cycle and c_m the *back edge* of the cycle.

Lemma 1. *If* $F = \{C_i\}_{i=1}^{m-1} \cup \{\{x_a^{\epsilon_a}, x_b^{\delta_b}\}\}$ *is such that*

– $x_a^{\epsilon_a}$ *or* $x_a^{1-\epsilon_a} \in C_j$ *for some* $j = 1, \ldots m-1$ *and*
– $x_b^{\delta_b}$ *or* $x_b^{1-\delta_b} \in C_k$ *for some* $k = 1, \ldots m-1$ *then*

$$\#SAT(\{C_i\}_{i=1}^{m-1}) \geq \#SAT(F)$$

Proof. Let s be a satisfying assignment of $\{c_i\}_{i=1}^{m-1}$. Then either $x_a^{\epsilon_a}$ *or* $x_a^{1-\epsilon_a} \in s$ and also $x_b^{\delta_b}$ *or* $x_b^{1-\delta_b} \in s$. If $x_a^{\epsilon_a}$ & $x_b^{\delta_b} \in s$, then s is also a satisfying assignment of F. Similarly, if $x_a^{\epsilon_a}$ & $x_b^{1-\delta_b} \in s$ or $x_a^{1-\epsilon_a}$ & $x_b^{\delta_b} \in s$ results in s is also a satisfying assignment of F. If $x_a^{1-\epsilon_a}$ & $x_b^{1-\delta_b} \in s$ then the clause $\{x_a^{\epsilon_a}, x_b^{\delta_b}\}$ does not hold, so this assignment is not a model of F.

So that a model of $F \cup \{\{x_a^{\epsilon_a}, x_b^{\delta_b}\}\}$ is not a model of $F \cup \{\{x_a^{1-\epsilon_a}\}, \{x_b^{1-\delta_b}\}\}$ since $\overline{(x_a^{\epsilon_a} \vee x_b^{\delta_b})} = x_a^{1-\epsilon_a} \wedge x_b^{1-\delta_b}$.

Let $Y = \{c_i\}_{i=1}^{m-1} \cup \{\{x_a^{1-\epsilon_a}\}, \{x_b^{1-\delta_b}\}\}$, then $\#SAT(Y)$ can be computed as a path with two unitary clauses on the extremes of the path.

In order to process cycles of a constrained graph, we use *computing threads* or just *threads*. A computing thread is a sequence of pairs $(\alpha_i, \beta_i), i = 1, \ldots, m$ used for computing the number of models over a path of m nodes. Any simple cycle C request two computing threads. The main thread denoted by L_p processes the internal path on the cycle, and its subordinated thread denoted by L_C computes #SAT(Y). When the last node x_b on the cycle is visited by the linear search, the last pair $(\alpha_b, \beta_b)_C$ associated to L_C is substracted from the the pair $(\alpha_b, \beta_b)_p$ associated with the thread L_P, as it is shown in Figure (1).

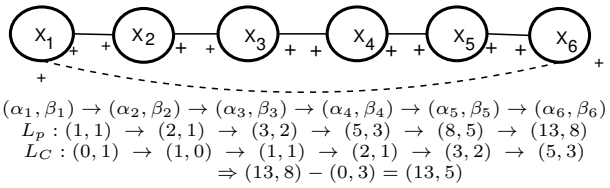


Fig. 1. Computing $\#SAT(F)$ when G_F is a simple cycle

Example 1. Let $F = \{c_i\}_{i=1}^{6} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ be a monotone 2-CF which represents a cycle: G_F=(V, E). Let $G' = (V, E')$ where $E = E' \cup \{c_6\}$, that is, the new graph G' is F minus the edge c_6. we have that $\#SAT(F) = \#SAT(F') - \#SAT(F' \wedge \overline{x}_6 \wedge \overline{x}_1) = 21 - 3 = 18$. This example is illustrated in Figure 1.

Let $G_F = (V, E, \{+, -\})$ be a signed connected graph of an input formula F in 2-CF, with $n = |V|$ and $m = |E|$. We will use the notation #2SAT(F) or #2SAT(G_F) to denote the number of models for a 2-CF F expressed as a set of clauses or through its constrained graph G_F.

If a depth-first search (dfs) is applied over G_F, starting the search, for example, with the node $v_r \in V$ of minimum degree, and selecting among different potential nodes to visit the node with minimum degree first and with minimum value in its label as a second criterion, we obtain an unique depth-first graph G, which we will denote as $G = dfs(G_F)$. This dfs also builds an unique spanning tree T_G with v_r as the root node.

In time $O(m+n)$, the dfs allows us to detect if G has cycles or not, as well as the edges forming each cycle. The edges in T_G are called *tree edges*, whereas the edges in $E(G) \backslash E(T_G)$ are called *back edges*. Let $e \in E(G) \backslash E(T_G)$ be a back edge, the union of the path in T_G between the endpoints of e with the edge e itself forms a simple cycle, such cycle is called a fundamental (or basic) cycle of G with respect to T_G. Each back edge $e = \{x, y\}$ holds the maximum path contained in the basic cycle that it is part of. Assuming that x is visited first than y during the dfs, we say that x is the start-node and y is the end-node of the back edge $e = \{x, y\}$.

If two distinct cycles C_i and C_j from an graph G have common edges then we say that both cycles are *intersected*, that is, $C_i \triangle C_j$ form a new cycle, where $\triangle$ denotes the symmetric difference operation between the set of edges in both cycles. In fact, $C_i \triangle C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$ forms a composed cycle. If two cycles are non-intersected we say that they are *independent*. I.e. two independent cycles (C_i, C_j) hold $(E(C_i) \cap E(C_j)) = \emptyset$.

A cycle basis of a graph is a family of cycles which spans all cycles of the graph. The cycles can be considered as vectors indexed by edges. The entry for an edge is one if the edge belongs to the cycle and is zero otherwise. Addition of cycles corresponds to vector addition modulo 2 (symmetric difference of the underlying edge sets) denoted as Z_2-vector space. In this way, the cycles of a graph form a vector space and a cycle basis is simply a basis of this vector space [14].

According with our particular depth-first search $G = dfs(G_F)$ on G_F, let $\mathcal{C} = \{C_1, ..., C_t\}$ be the set of fundamental cycles found during such depth-first search. Notice that $t = m - n + 1$ is the dimension of the Z_2-vector space with the symmetric difference on the edge sets as addition, and $\mathcal{C}$ is a cycle basis for the constrained graph G_F.

3.1 Processing Intersected Cycles

In order to show how to process a set of intersected cycles, we show first how to process a pair of them. Assume C_i, C_j are two basic cycles such that $E(C_i) \cap E(C_j) \neq \emptyset$, and let $e_i = x \rightarrow y$ and $e_j = u \rightarrow v$ be its corresponding back edges. Let G_F be the constrained graph of the formula F containing both cycles C_i and C_j.

A main thread denoted by L_p will be associated with the computation of #2SAT(F). L_p is always active during all the computation of #2SAT(F).

While the threads L_{C_i} and L_{C_j} will be used for computing the number of assignments to be substracted from L_p when the cycles C_i and C_j are found.

We say that both threads L_{C_i} and L_{C_j} are subordinated threads of L_p and that L_p dominates L_{C_i} and L_{C_j}.

We apply our deterministic dfs on G_F, and let us assume that the start-node x of C_i is always visited first than the start-node u of C_j. When the node x is visited, the thread L_{C_i} becomes active. And when the node u is visited, two new threads ($L_{C_j \mapsto p}$ and $L_{C_j \mapsto C_i}$) are created. $L_{C_j \mapsto p}$ carries the value that will be associated with the thread L_p and $L_{C_j \mapsto C_i}$ carries the value that will be associated with the thread L_{C_i}.

Assuming $(\alpha_x, \beta_x)_i$ as the pair of a dominant thread L_i at a start-node x, then the initial pair $(\alpha_x, \beta_x)_e$ for its subordinated thread $L_{e \mapsto i}$ is computed as:

$$(\alpha_x, \beta_x)_e = \begin{cases} (0, \beta_x)_i & \text{if } + \text{ is the sign of } x \in e \\ (\alpha_x, 0)_i & \text{if } - \text{ is the sign of } x \in e \end{cases}$$

In fact, when a start-node of a new cycle is visited, then the number of current threads will be duplicated, building a new subordinate thread for each previous current active thread. When the end-node y of the cycle C_e is visited, the pair $(\alpha_y, \beta_y)_e$ of any subordinated thread L_e is computed as:

$$(\alpha_y, \beta_y)_e = \begin{cases} (0, \beta_y)_e & \text{if } + \text{ is the sign of } y \in e \\ (\alpha_y, 0)_e & \text{if } - \text{ is the sign of } y \in e \end{cases}$$

and the subordinated thread L_e is closed, substracting the resulting pair $(\alpha_y, \beta_y)_e$ from the corresponding pair of its dominant thread.

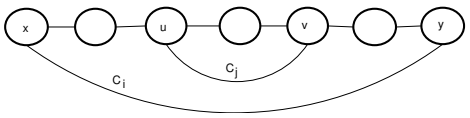


Fig. 2. Case with embedded intersected cycles

In the first case, if C_j is embedded into C_i, i.e. $E(C_j) - E(C_i) = \{e_j\}$, then the end-node v of C_j is visited first than the end-node y of C_i (see Figure 2 and Table 1).

In the first case, as v is visited before y, then the cycle C_j finishes before visit all node of C_i, and the two threads ($L_{C_j \mapsto p}$ and $L_{C_j \mapsto C_i}$) are closed before the threads L_P and L_{C_i} finish.

The last pair associated to the threads $L_{C_j \mapsto p}$ and $L_{C_j \mapsto C_i}$ at the node v are substracted from its corresponding pairs on its dominant threads L_p and L_{C_i}, respectively. After this, the processing of the two remaining threads L_p and L_{C_i} continue, as it is shown in Table 1.

For the second pattern, we assume that end-node y of C_i is visited before the end-node v of C_j, then the cyle C_i begins and finishes before visit all node of the cycle C_j (see Figure 3 and Table 2). The threads (L_{C_i} and $L_{C_j \mapsto C_i}$) are closed when the node y is visited and the last pair of those threads are substracted

Table 1. Computing embedded intersected cycles

Node	x	...	u	...	v	...	y
L_p :	$(\alpha_x, \beta_x)_p$	...	$(\alpha_u, \beta_u)_p$	...	$(\alpha_v, \beta_v)_p$ $-$ $(0, \beta_v)_{C_j,p}$ $=$ $(\alpha_v, \beta_v)_p$	...	$(\alpha_y, \beta_y)_p$ $-$ $(0, \beta_y)_{C_i}$ $=$ $(\alpha_y, \beta_y)_p$
L_{c_i} :	$(0, \beta_x)_{C_i}$	...	$(\alpha_u, \beta_u)_{C_i}$	...	$(\alpha_v, \beta_v)_{C_i}$ $-$ $(\alpha_v, 0)_{C_j,C_i}$ $=$ $(\alpha_v, \beta_v)_{C_i}$	...	$(\alpha_y, \beta_y)_{C_i} \Rightarrow$ $(0, \beta_y)_{C_i}$ closed
$L_{c_j \to p}$:			$(\alpha_u, 0)_{C_j,p}$	...	$(\alpha_v, \beta_v)_{C_j,p} \Rightarrow$ $(0, \beta_v)_{C_j,p}$ closed		
$L_{c_j \to c_i}$:			$(0, \beta_u)_{C_j,C_i}$	...	$(\alpha_v, \beta_v)_{C_j,C_i} \Rightarrow$ $(\alpha_v, 0)_{C_j,C_i}$ closed		

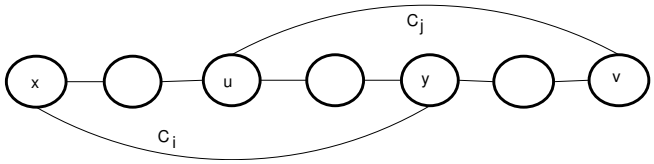


Fig. 3. Case with intersected cycles (not embedded)

from its corresponding pairs on the threads L_p and $L_{C_j \to p}$, respectively. After this, the processing of the two remaining threads L_p and $L_{C_j \to p}$ continue, until visit all node of the cycle C_j.

Sometimes, the pair associated to a computing thread L_x results to be (0,0); in such cases L_x is closed (stops being active) because it will not affect the value of any other active thread. A relevant point to consider in our procedure, is to recognize which threads have to keep being active and which have to be closed when an end-node of a cycle is visited,i.e. which of the two patterns for intersected cycles has to be applied.

4 Computing #2SAT for General Graphs

We present, the general case for computing #2SAT. Let F be the input 2-CF and G its constrained graph, we assume that $G_F = dfs(G)$ has intersected cycles. Let T_F be the unique spanning tree formed during the dfs and let $\mathcal{C} = \{C_1, C_2, ..., C_t\}$ be the unique set of fundamental cycles found during such depth-first search.

For each edge $e \in E(G_F)$, we define $\kappa(e)$ as the number of basic cycles where $\kappa(e) = |\{C \in \mathcal{C} : e \in E(C)\}|$. Notice that $\kappa(e) = 1, \forall e \in (E(G_F) - E(T_F))$. $\kappa(e)$ denotes the number of entwined cycles of G where the edge e is a common

Table 2. Computing intersected cycles (not embedded)

Node	x	...	u	...	y	...	v
L_p :	$(\alpha_x, \beta_x)_p$	...	$(\alpha_u, \beta_u)_p$	...	$(\alpha_y, \beta_y)_p$ $-$ $(0, \beta_y)_{C_i}$ $=$ $(\alpha_y, \beta_y)_p$	...	$(\alpha_v, \beta_v)_p$ $-$ $(0, \beta_v)_{C_j,p}$ $=$ $(\alpha_v, \beta_v)_p$
L_{c_i} :	$(0, \beta_x)_{C_i}$	...	$(\alpha_u, \beta_u)_{C_i}$	...	$(\alpha_y, \beta_y)_{C_i}$ $\Rightarrow$ $(0, \beta_y)_{C_i}$ closed		
$L_{c_j \to p}$:			$(\alpha_u, 0)_{C_j,p}$	...	$(\alpha_y, \beta_y)_{C_j,p}$ $-$ $(\alpha_y, 0)_{C_j,C_i}$ $=$ $(\alpha_y, \beta_y)_{C_j,p}$	...	$(\alpha_v, \beta_v)_{C_j,p}$ $\Rightarrow$ $(0, \beta_v)_{C_j,p}$ closed
$L_{c_j \to c_i}$:			$(0, \beta_u)_{C_j,C_i}$	...	$(\alpha_y, \beta_y)_{C_j,C_i}$ $\Rightarrow$ $(\alpha_y, 0)_{C_j,C_i}$ closed		

edge among them. We call the *edge-entwined* of a graph, denoted as $\varpi(G_F)$, the maximum value of $\kappa(e), \forall e \in E(G_F)$. I.e. $\varpi(G_F) = max\{\kappa(e) : e \in E(G_F)\}$.

Therefore, $\kappa(e), \forall e \in E(G_F)$ is a function denoting the degree in which all cycles with the same edge e are entwined. As a result, we also obtain an unique value for the *edge-entwined* $\varpi(G_F)$ of the constrained graph G_F.

In a previous work [6] (the *Count_Models* procedure), we have presented a polynomial time algorithm for computing #2SAT when the constrained graph of the input formula has non-intersected cycles. As in the *Count_Models* procedure, we build an orientation on each edge $\{u, v\}$ in G_F by directing: $u \to v$, if v is an ancestor node of u in T_F. G_F is traversing in post-order according to the orientation given to the edges. A main thread denoted by L_p is associated with the computation of #SAT(T_F). L_p keeps always active during all the computation of #2SAT(F).

The computation of #2SAT(F) is done as it was done in the *Count_Models* procedure, and only a light extension has to be done when a start-node of a back edge $e = x \to y$ is visited. The following cases consider how to process the new cycle C_e that has been found while G_F is being traversing.

1. If C_e is an independent cycle then we process it as a Ring, as it was explained in the *Count_Models* procedure.
2. If C_e is part of a set of intersected cycles, then the number of current threads will be duplicated, building a new subordinate thread for each previous current active thread, it was shown in section 3.1.

When a start-node of a back edge $e = \{x \to y\}$ of a cycle is visited, the number of current thread is duplicated. In the worst case, we will have at most 2^k active threads, where $k = \kappa(e)$ is the degree of entwined among cycles containing the edge e and e is an adjacent edge to the current node that is being visited. Thus, in our procedure, the maximum number of active computing threads at

any moment will be $2^{\varpi(G_F)}$, where $\varpi(G_F)$ is the edge-entwined number of the constrained graph G_F.

We apply this algorithm to the constrained graph G_F shown in figure 4. We split the computation of #2SAT(F) according to the two parts of the graph show in figures 5 and 6.

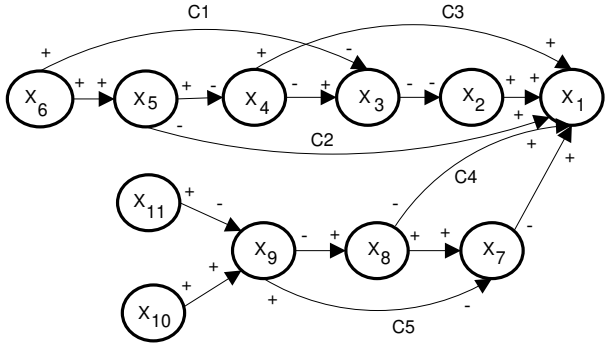
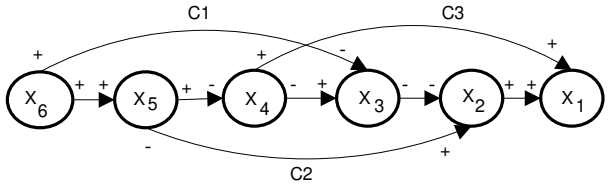


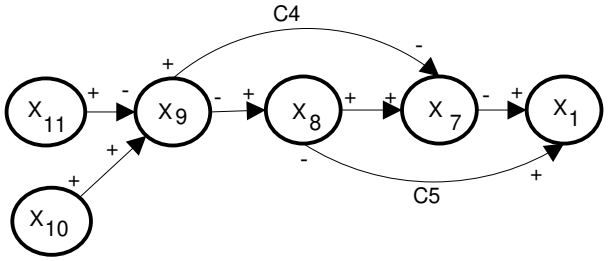
Fig. 4. A constrained graph with intersected cycles



$$
\begin{array}{rlllllll}
Nodes: & x_6 & x_5 & x_4 & x_3 & & x_2 & x_1 \\
Lp: & (1,1) \rightarrow & (2,1) \rightarrow (2,3) & \rightarrow (5,3)-(2,0)=(3,3) & \rightarrow (3,6) & \rightarrow (9,3) & & \\
C1: & (0,1) \rightarrow & (1,0) \rightarrow (1,1) & \rightarrow (2,1) \Rightarrow (2,0) & & & & \\
C2 \rightarrow Lp: & & (2,0) \rightarrow (2,2) & \rightarrow (4,2)-(2,0)=(2,2) & & & & \\
 & & & (2,2)-(2,0)=(0,2) & \rightarrow (2,2) & \rightarrow (4,2) \Rightarrow (0,2) & & \\
C2 \rightarrow C1: & & (1,0) \rightarrow (1,1) & \rightarrow (2,1) \Rightarrow (2,0) & & & & \\
C3 \rightarrow Lp: & & & (0,3) \rightarrow (3,3)-(2,0)=(1,3) & & & & \\
 & & & (1,3)-(1,0)=(1,2) & \rightarrow (2,3) & \rightarrow (5,2) \Rightarrow (0,2) & & \\
C3 \rightarrow C1: & & & (0,1) \rightarrow (1,1) \Rightarrow (1,0) & & & & \\
C3 \rightarrow C2: & & & (0,2) \rightarrow (2,2)-(2,0)=(0,2) & & & & \\
 & & & (0,2)-(1,0)=(0,2) & \rightarrow (2,2) & \rightarrow (4,2)-(0,2)=(4,0) & & \\
C3 \rightarrow C2 \rightarrow C1: & & & (0,1) \rightarrow (1,1) \Rightarrow (1,0) & & & & \\
 & \Rightarrow (9,3)- & (0,2)- & (0,2)= & (9,0) & & &
\end{array}
$$

Fig. 5. First part of the Computation of $\#SAT(G_F)$

The last pair obtained in the first part of the graph is $(\alpha_1, \beta_1) = (9, 0)$ and the pair for the second part of the graph is $(\alpha_1, \beta_1) = (6, 0)$. Both pairs corresponds with the value for the same node x_1, then the last pair associated to x_1 is $(54, 0) = (9 * 6, 0 * 0)$, and #2SAT(F) = 54 models.



$Nodes:$ x_{11} x_{10} x_9 x_8 x_7 x_1

$Lp:$ $(1,1) \rightarrow (1,2) * (2,1) = (2,2) \rightarrow (4,2) \rightarrow (6,4) - (4,0) = (2,4) \rightarrow (6,4)$

$C4:$ $(0,2) \rightarrow (2,2) \rightarrow (4,2) \Rightarrow (4,0)$

$C5 \rightarrow Lp:$ $(4,0) \rightarrow (4,4) - (2,0) = (2,4)$

$(2,4) - (4,0) = (0,4) \rightarrow (4,4) \Rightarrow (0,4)$

$C5 \rightarrow C4:$ $(2,0) \rightarrow (2,2) \Rightarrow (2,0)$

$\Rightarrow (6,4) - (0,4) = (6,0)$

Fig. 6. Second part of the computation of $\#SAT(G_F)$

4.1 Time Complexity Analysis

The $Count\_Models$ has a time complexity of $O(n+m)$, n being the number of nodes and m the number of edges of the constrained graph of an formula F.

The marking of edges forming part of fundamental cycles during the dfs can be done at most in time $O(m \cdot n)$ since a fundamental cycle has length at most n. Consequently, the computation of $\kappa(e), \forall e \in E(G_F)$, is done in time $O(m \cdot n)$. The identification of the value *edge-entwined* of a graph $\varpi(G)$, representing the value of the parameter k for our fixed-parameter algorithm, also involves at most $O(m \cdot n)$ basic operations.

Now, we analyze the time-complexity for processing intersected cycles. Let κ be a set of k intersected cycles with common edges.

In each beginning of a cycle of κ, the number of current thread has to be duplicated so that in the worst case, we will have at most 2^k active threads. And we have shown that $O(poly(n,m))$ operations are enought to travers by the nodes $x \in V(G_F)$ at the same time that its pair (α_x, β_x) is computed.

Given that the size of the formula is $(n+m)$, we obtain that the complexity time for our proposal is of order $O(2^{\varpi(G_F)} \cdot p(n,m))$, where $p(n,m)$ represents a polynomial function based on n and m, and $\varpi(G_F)$ is a non-negative integer parameter representing the edge-entwined number of the constrained graph G_F of the input F.

Then, the complexity-time for the general case is in the worst case, upper bounded by $O(2^{\varpi}(G_F) \cdot p(n,m))$. Then, one can afford (mildly) simple exponential behaviour of our algorithm in terms of the parameter $k = \varpi(G_F)$ - the edge-entwined number of the constrained graph, as long as the overall running time is polynomial on the size of the 2-CF when considering the parameter as a fixed constant, and this shows that #2SAT is in the class FPT (the class of all fixed-parameter tractable problems).

The computation of the edge-entwined number $k = \varpi(G_F)$ of a graph G_F is polynomial on the zise of the graph and its value is usually smaller than the

number of nodes (n) of the graph. Also, the computation of $\varpi(G_F)$ does not require the tree-decomposition of the graph. Furthermore, the complexity time of $O(2^k \cdot p(n,m))$ is usually smaller than the exponential upper bounds based on n or m, expressed in classical works for the computation of $\#2SAT$.

The generated results can be compared with the results showed in [2], where a n parameter was used in the order of complexity. In contrast in this work a k parameter was used that improves the complexity for some instances.

On the other hand, if the value for $\varpi(G_F)$ is increased adding intersected cycles to G_F, then the value #2SAT(F) will decrease, for example, #2SAT(F) $\leq n^5$, $n = |\upsilon(F)|$, then #2SAT(F) could be computed in polynomial time for the algorithm proposed in [7].

In order to formalize the results obtained, an formal argumentation can be used. For example, semantic argumentation can be characterized as a two-value model, in this way, the $STABLE$ argumentation model proposes interesting logic properties [17].

5 Conclusion

#SAT problem for the class of Boolean formulas in 2-CF is a classical #P-complete problem. However, there are several instances of 2-CF's for which #2SAT can be solved efficiently.

We have shown here different polynomial-time procedures for counting models of Boolean formulas for subclasses of 2-CF's. For example, for formulas whose contrained graph is acyclic, its corresponding number of models is computed in linear time. Furthermore, if the cycles in a constrained graph G_F can be arranged as independent cycles, then we can count the number of models of F efficiently.

Finally, according with the parameter $k = \varpi(G_F)$ - the 'edge-entwined' of the constrained graph of the 2-CF F, we have designed a deterministic fixed-parameter tractable algorithm for #2SAT, that is, an algorithm whose time complexity is $O(2^k \cdot poly(|F|))$, where $poly(|F|)$ is a polynomial function on the size of F and k is equal to the maximum number of fundamental cycles with a common edge on the constrained graph of the input formula.

References

1. Angelsmark O., Jonsson P.: Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems, *In ICCP: Int. Conf. on Constraint Programming*, Springer, Berlin,1991.
2. Dahllöf, V., Jonsonn, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae, *J. Theoretical Computer Sciences*,**332**(1-3), 2005, 265–291.
3. Bacchus F., Dalmao S., Pitassi T.: Algorithms and complexity results for #SAT and Bayesian inference, *Proc. of FOCS-03: 44th Annual Symposium on Foundations of Computer Science*, USA,2003.
4. Gomes C. P., Sabharwal A., Selman B.: Model Counting,in: *Handbook of Satisfability*, Chap. 20, Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsch Eds., IOS Press, 2008.

5. Darwiche A.: On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, *J. of Applied Non-classical Logics*, **11**(1-2), 2001, 11–34.
6. De Ita G., Bello P., Contreras M.: New Polynomial Classes for #2SAT Established via Graph-Topological Structure, *Engineering Letters*, **15** (2), 2007, 250–258.
7. De Ita G., Marcial-Romero R., Hernández A. : A Threshold for a Polynomial Solution of #2SAT, *Fundamenta Informaticae*, **113**(1), 2011, 63–77.
8. Dubois, O.: Counting the number of solutions for instances of satisfiability, *J. Theoretical Computer Sciences*,**81**(1), 1991, 49–64.
9. Eppstein D.: Quasiconvex analysis of backtracking algorithms, *Proc. of the 15th annual ACM-SIAM symp. on Discrete algorithms (SODA-2004)*,ACM, New York, USA,2004.
10. Fischer E., Makowsky J.A., Ravve E.V.: Counting truth assignments of formulas of bounded tree-width or clique-width, *Discrete Applied Mathematics*, **156**(4), 2008, 511–529.
11. Fürer, M., Prasad, S. K.: *Algorithms for Counting 2-SAT Solutions and Coloring with Applications*, Technical Report 33, Electronic Colloqium on Comp. Complexity,2005.
12. Littman M. L., Pitassi T., Impagliazzo R.: *On the Complexity of counting satisfying assignments*, Technical Report Unpublished manuscript.
13. Gottlob G., Scarcello F., Sideri M.: Fixed-parameter complexity in AI and nonmonotonic reasoning, *Artificial Intelligence*, **138**(1-2),2002, 55–86.
14. Kavitha T., Liebchen C., Mehlhorn K., Michail D., Rizzi R., Ueckerdt T., Zweig K.: Cycle bases in graphs: Characterization, algorithms, complexity, and applications, *Computer Science Rev.*, **3**(4), 2009, 199–243.
15. Kullmann O.: New methods for 3-SAT decision and worst-case analysis, *Theoretical Computer Science*, **223**, 1999, 1–72.
16. Makowsky J.A.: Algorithmic uses of the Feferman-Vaught theorem, *Ann. Pure Appl. Logic* **126**, 2004, 1–3.
17. Nieves J.C, Osorio M.:Extending Well-Founded Semantics with Clark's Completion for Disjunctive Logic Programs, *Scientific Programming*, 2018, Article ID 4157030, 10 pages.
18. Roth D.: On the hardness of approximate reasoning, *J. Artificial Intelligence*, **82**, 1996, 273–302.
19. Russ B.: Randomized Algorithms: Approximation, Generation, and Counting, *Distinguished dissertations*, Springer,2001.
20. Szeider Stefan.: On Fixed-Parameter Tractable Parameterizations of SAT, *Theory and Applications of Satisfiability Testing*(E. Giunchiglia, A. Tacchella, Ed.), LNCS 2919, Springer-Verlag,Berlin, 2004.
21. Wahlström M.: *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*, Dissertation No. 1079, Linkping, 2007.
22. Zhang, W.: Number of models and satisfiability of set of clauses, *J. Theoretical Computer Sciences*, **155**(1), 1996, 277–288.