A Bottom-Up Algorithm for Solving ♯2SAT

Guillermo De Ita[*a], J. Raymundo Marcial-Romero[†b], and J. A. Hernández-Servín[‡b]

[a] *Benemerita Universidad Autonoma de Puebla, Puebla, México*
[b] *Universidad Autonoma del Estado de México, Toluca, México*

Abstract. Counting models for a two conjunctive formula (2-CF) F, a problem known as ♯2SAT, is a classic ♯P complete problem. Given a 2-CF F as input, its constraint graph G is built. If G is acyclic, then ♯2SAT(F) can be computed efficiently. In this paper, we address the case when G has cycles. When G is cyclic, we propose a decomposition on the constraint graph G that allows the computation of ♯2SAT(F) in incremental way. Let T be a cactus graph of G containing a maximal number of independent cycles, and let $\overline{T} = (E(G) - E(T))$ be a subset of frond edges from G. The clauses in $\overline{T}$ are ordered in connected components $\{K_1, \ldots, K_r\}$. Each $(G \cup K_i)$, $i = 1, \ldots, r$ is a knot (a set of intersected cycles) of the graph. The arrangement of the clauses of $\overline{T}$ allows the decomposition of G in knots and provides a way of computing ♯2SAT(F) in an incremental way. Our procedure has a bottom-up orientation for the computation of ♯2SAT(F). It begins with $F_0 = T$. In each iteration of the procedure, a new clause $C_i \in \overline{T}$ is considered in order to form $F_i = (F_{i-1} \wedge C_i)$ and then to compute ♯2SAT(F_i) based on the computation of ♯2SAT(F_{i-1}).

Keywords: ♯SAT, Counting models, Enumerative Algorithm, Graph Decomposition

1 Introduction

Counting combinatorial objects is a challenging and relevant area of research in Mathematics, Computer Sciences, and Physics. Counting problems, being mathematically relevant by themselves, are closely related to practical problems. Several relevant counting problems are hard time-complexity problems. For example, ♯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 [1, 4, 2].

♯SAT can be reduced to several different problems in approximate reasoning, for example, in the cases of: the generation of explanations to propositional queries, repairing inconsistent databases, estimating the degree of belief in propositional theories, in a truth maintenance systems, in Bayesian inference [9, 1, 2, 10]. The previous problems come from several AI applications such as expert systems, planning, approximate reasoning, etc.

♯SAT is at least as hard as the SAT problem, however in some cases, even when SAT is solved in polynomial time, no computationally efficient method is known for ♯SAT. For instance, 2-SAT (SAT restricted to formulas in (≤ 2)-CF's) can be solved in linear time. However, the corresponding counting problem ♯2SAT is a ♯P-complete problem. Even though ♯2SAT is ♯P-Complete, there are instances of 2-CF which can be solved in polynomial time [2, 10]. For example, if the graph which represents the input formula is acyclic, then ♯2SAT can be solved in polynomial time.

Recently, new upper bounds expressed in base of the number of variables, and derived from exact deterministic exponential algorithms for ♯2SAT have been found

[*] deita@cs.buap.mx
[†] jrmarcialr@uaemex.mx
[‡] xoseahernandez@uaemex.mx

by Dahllöf [13], Fürer [7], and Angelsmark [9]. In other cases, other parameters have been used in the determination of the computational complexity of ♯2SAT. For example, by considering the number of clauses [6], or considering the dual problem of counting falsifying assignments [5], or considering a relative problem to ♯2SAT as is the counting of the number of independent sets [11]. Given that ♯2SAT is a ♯P-complete problem, all of the above proposals have an exponential-time computational complexity.

There are different methods for solving ♯2SAT, for example, procedures based on the Davis & Putnam decomposition and its adaptations [9, 13, 7], as well as to use a k-tree decomposition on the constraint graph of the input formula [3]. In this work, we propose a novel procedure with a bottom-up orientation that we consider adequate to design a dynamic programming method for solving ♯2SAT.

In this article, some of the procedures presented in [4, 5, 8] for ♯2SAT are considered. We show some conditions on the constraint graph of the formula which allow the efficient computation of ♯2SAT. We also consider the general case of a 2-CF, showing that ♯2SAT can be adequately parameterized according to the number of intersected cycles appearing in the constraint graph of the 2-CF.

In summary, the main contributions in this work that are different to the other or our previous works, are:

- We propose a novel decomposition of a constraint graph G_F of a 2-CF F. G_F is descomposed in a cactus graph A_F containing a maximal number of independent cycles and a set of edeges $\overline{T} = (E(G) - E(T))$ ordered in connected components.
- The computation of ♯2SAT(F) has a bottom-up orientation. It begins with $F_0 = T$. In each iteration of the procedure, a new clause $C_i \in \overline{T}$ is considered in order to form $F_i = (F_{i-1} \wedge C_i)$ and then to compute ♯2SAT(F_i) based on the computation of ♯2SAT(F_{i-1}).
- We show that each new subformula F_i has less knots (intersected cycles) than those from its father formula. And only if the subformula F_i has new knots, then recursive calls to the original procedure has to be performed.

In section 2, we present the preliminaries to understand the rest of the paper. In section 3, we present a graph decomposition on the constraint graph of the input 2-CF. In section 4, we show how to compute ♯2SAT(F) based on the graph decomposition of the constraint graph and our main result. Finally, the conclusions of the paper are established.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A *literal* denoted as l is, a variable x_i or a denied 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* c is a disjunction of different literals. For $k \in N$, a k-clause is a clause with exactly k literals, and $(\leq k)$-clause is a clause with at most k literals. Sometimes, we consider a clause as a set of literals.

A variable $x \in X$ appears in a clause c if either x or $\overline{x}$ is an element of c.

A *conjunctive normal form* (or conjunctive form), denoted as CF, is a conjunction of clauses, and *k-CF* is a CF containing only *k-clauses*. A CF F with n variables is a *n-ary* Boolean function $F{:}\{0,1\}^n \rightarrow \{0, 1\}$. We also consider a CF as a set of clauses.

We denote with Y any of the basic logic elements that we are using, such as a literal, a clause, or a CF. Then, $\upsilon(Y)$ expresses the set of variables involved in the object Y. For example, for the clause $c = \{\overline{x}_2, x_3\}, \upsilon(c) = \{x_2, x_3\}$. Meanwhile, $Lit(Y)$ denotes the set of literals involved in object Y. For example, if $X = \upsilon(F)$,

then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$. We also use $\neg Y$ as the negation operator on the object Y. We denote to $\{1, 2, 3, \ldots, n\}$ by $[[n]]$, and the cardinality of a set A by $|A|$.

An assignment s for F is a Boolean mapping $s : \upsilon(F) \to \{0, 1\}$. An assignment s can be also considered as a set of non-complementary pairs of literals. $l \in s$ if and only if s assigns true to l and false to $\overline{l}$. Considering a clause c and an assignment s as set of literals, c is satisfied by s if and only if $(c \cap s) \neq \emptyset$, and if for all $x^{\varepsilon} \in c$, $x^{1-\varepsilon} \in s$ then s falsifies c.

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 of F is an assignment for $\upsilon(F)$ that satisfies F. s is a partial assignment for the formula F when s has determined a logical value only to variables of a proper subset of F. Given a CF F, the SAT problem consists of determining if F has a model. Meanwhile, $SAT(F)$ denotes the set of models of F. The ♯SAT problem consists of counting the number of models of F defined over $\upsilon(F)$. ♯2SAT denotes ♯SAT for formulas in 2-CF.

2.1 The constraint graph of a 2-CF

There are some graphical representations of a conjunctive form (see e.g. [12]), we use here the signed primal graph of a 2-CF. Let F be a 2-CF with set of variables $\upsilon(F)$. The constraint graph (signed primal graph) of F is denoted by $G_F = (V(F), E(F))$, with $V(F) = \upsilon(F)$ and $E(F) = \{\{\upsilon(x), \upsilon(y)\} : \{x, y\} \in F\}$, that is, the vertices of G_F are the variables of F, and for each clause $\{x, y\}$ in F there is an edge $\{\upsilon(x), \upsilon(y)\} \in E(F)$. We say that a 2-CF F is a path, cycle, a tree, or a forest, if its constraint graph G_F represents a path, cycle, a tree, or a forest, respectively.

Each edge $c = \{\upsilon(x), \upsilon(y)\} \in E$ has an ordered pair (s_1, s_2) of signs associated to the endpoints of the edge. Such signs are used as labels of the edge that connect the vertices of the variables appearing in the clause. For example, the clause $\{y^0, z^1\}$ determines the labelled edge: "$y \overset{-+}{-} z$" which is equivalent to the edge "$z \overset{+-}{-} y$". The signs $s_1, s_2 \in \{+, -\}$ are related to the variables y and z, respectively.

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 constraint graph with labelled edges on $S \times S$. Let x and y be vertices in V. If $e = \{x, y\}$ is an edge and $\psi(e) = (s, s')$, then $s(s')$ is called the adjacent sign to $x(y)$.

Notice that a constraint 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 so parallel edges. Furthermore, an unitary clause defines a loop in the constraint graph. A polynomial time algorithm to process parallel edges and loops to solve ♯2SAT has been shown in [4, 8].

Let ρ : 2-CF $\to G_F$ be the function whose domain is the space of non strict Boolean formulae in 2-CF and codomain the set of multi-graphs. It is clear that ρ is a bijection. So any 2-CF formula has a unique signed constraint graph (up to isomorphism) associated via ρ and viceversa, any signed constraint graph G_F has a unique formula associated via ρ^{-1}.

3 A graph decomposition of the input formula

For the sake of simplicity, we used ♯2SAT(G) where G is the constraint graph of a formula F, to denote ♯2SAT(F). Let G be a forest, since

$$\sharp 2\text{SAT}(G) = \prod_{i=1}^{k} \sharp 2\text{SAT}(G_i)$$

where $G_i, i = 1, \ldots, k$ are the connected components of G [2], then the total time complexity for computing $\sharp 2\textsc{Sat}(G)$, denoted as $T(\sharp 2\textsc{Sat}(G))$, is given by the maximum rule:

$$T(\sharp 2\textsc{Sat}(G)) = \max\{T(\sharp 2\textsc{Sat}(G_i)) : G_i \text{ is a connected component of } G\}.$$

Thus, from here on, we consider as an input graph only a connected component. Let $G_F = (V(F), E(F), \{+, -\})$ be a signed connected graph of an input formula F in 2-CF, with $n = |V|$ and $m = |E|$.

A depth-first search (abbreviated as *dfs*) is applied over G_F. The *dfs* starts with the node $v_r \in V$ of minimum degree; when a new node is visited, we select first the nodes with minimum degree, if there are several, as a second criterion, we select the one with minimum value in its label. The result of applying a *dfs* on G is a depth-first graph G', which we will denote as $G' = dfs(G_F)$ and a spanning tree T_G with v_r as the root node.

The *dfs* allows us to detect if G has or has not cycles and the parity of such cycles in time $O(m + n)$. The edges in T_G are called *tree edges*. An edge $e \in E(G)\backslash E(T_G)$ is called a *frond edge*, and we call to its respective clause $c_e \in F$ a *frond clause*. Let $e \in E(G)\backslash E(T_G)$ be a frond edge, the union of the path in T_G between the endpoints of e with the edge e itself forms a simple cycle C_e, such cycle is called a basic cycle of G with respect to T_G.

Let $\mathcal{C} = \{e_1, e_2, ..., e_k\}$ be the set of frond edges found during the depth-first search. If two distinct simple cycles C_{e_i} and $C_{e_j}, i \neq j$ from $\mathcal{C}$ have common edges then we say that both cycles are *intersected*, and then, $C_{e_i} \triangle C_{e_j}$ form a new cycle, where $\triangle$ denotes the symmetric difference operation between the set of edges of both cycles. If two simple cycles are non-intersected we say that they are *independent*. Notice that $k \leq m - n + 1$ is the dimension of the $\mathcal{Z}_2$-vector space with the symmetric difference on the edge sets as addition, and $\mathcal{C}$ is a base in that $\mathcal{Z}_2$-vector space.

Decomposition plays an important role in graph theory. There are various decompositions of graphs such as decomposition by clique separators, tree decomposition or clique decomposition. All of them have been shown to be a useful tools in the design of efficient graph algorithms. There are even beautiful general results stating that a variety of NP-complete graph problems can be solved in linear time for graphs of bounded treewidth and bounded clique-width, respectively [3, 7].

We propose a decomposition of G, by A_F that is a cactus graph containing a maximal number of independent cycles from G, and $\bar{\mathcal{C}} = E(G_F) - E(A_F) = \{e_1, e_2, \ldots, e_t\} \subseteq \mathcal{C}$ be the set of frond edges forming intersecting cycles with the cycles in A_F.

The decomposition of G allows us to consider basic cases for the computation of $\sharp 2\textsc{Sat}(F)$. Some basic cases could be consider e.g. that the input instance F is such that its constraint graph A_F has not intersected cycles ($\bar{\mathcal{C}} = \emptyset$), or the case when $|\upsilon(A_F)| \leq d$ for some constant d, in some works such constant is for example $d < 8$ [9, 13, 7]. When A_F has not intersecting cycles, we have designed a procedure (see [4, 8]) for computing $\sharp 2\textsc{Sat}(F)$ in linear time on the size of A_F.

Let us consider the case when G has intersected cycles. For this case, let A_F be a cactus graph with a maximal number of independent cycles from G and let $\bar{\mathcal{C}} = E(G_F) - E(A_F)$. Let κ be a *knot* that is a set of intersecting cycles. This means that each pair of cycles in κ are intersected in G. The cardinality of a knot $|\kappa|$ is the number of cycles forming such knot. Notice that any knot has at least one edge e that is common to all cycle in that knot.

The knots of the graph are identified by traversing through the frond edges, and also the cardinality of each knot can be computed at the same time. We call *knotting number* of a graph to the maximum cardinality of any knot in the graph. We denote the *knotting number* of a graph G as $\varpi(G)$, then $\varpi(G) = \max\{|\kappa| : \kappa \text{ is a knot of } G\}$.

The edges in $\bar{\mathcal{C}}$ are ordered in connected components $\{K_1, \ldots, K_r\}$. Each $(A_F \cup K_i), i = 1, \ldots, r$ forms a knot of the graph. Note that $\{K_1, \ldots, K_r\}$ is a partition of $\bar{\mathcal{C}}$ and then, $\bar{\mathcal{C}} = \cup_{j=1}^{r}(K_j)$, and $(K_i) \cap (K_j) = \emptyset, i \neq j, i, j = 1, \ldots, r$.

Our procedure begins considering $F_0 = A_F$. The order on the clauses of $\bar{\mathcal{C}}$ allows the decomposition of G_F in knots, at the same time that provide a way of computing ♯2SAT(F) in a bottom-up orientation, based on the computation of each ♯2SAT$(G_0 \cup K_i), i = 1, \ldots, r$, as we show in the following section.

4 Incremental Computation of ♯2SAT

Contrary to the Davis & Putnam counting procedure, which has a top-down orientation for computing ♯2SAT, as well as some of its adaptations [9, 13, 7], we propose here, a novel procedure with a bottom-up orientation, which is adequate for solving the ♯2SAT problem applying dynamic programming.

Given a 2-CF F as input, our proposal begins forming a decomposition on G (the constraint graph of F). The decomposition of G is given by A_F that is a cactus graph containing a maximal number of independent cycles from G, and a set $\{K_1, \ldots, K_r\}$ of connected components of $\bar{\mathcal{C}}$, such that each $(A_F \cup K_i), i = 1, \ldots, r$ forms a knot of the graph.

In each iteration of our proposal, a new frond edge $e_i \in \bar{\mathcal{C}}, i = 1, \ldots, t$ is considered in order to form the corresponding formula $F_i = (F_{i-1} \wedge c_{e_i})$, where c_{e_i} is the frond clause of e_i and then ♯2SAT(F_i) is computed. Notice that ♯2SAT(F) = ♯2SAT$(A_F \bigwedge_{e_i \in \bar{\mathcal{C}}} c_{e_i})$.

One relevant tool in our proposal is the application of the unit resolution propagation as a part to perform into the computation of ♯2SAT(F_i). In this section, we present first the procedure $Eval(F_i, c_{e_i})$, that is the application of unit resolution propagation on F_i in order to recognize which models of F_i do not satisfy (c_{e_i}).

DEFINITION 4.1
Let H be a CF and $x^\varepsilon, \varepsilon \in \{0, 1\}$ a literal of H. The reduction of H by x^ε, also called forcing x^ε and denoted by $H[x^\varepsilon]$, is the formula generated from H by the following two rules:

a) removing from H the clauses containing x^ε (subsumption rule),

b) removing $x^{1-\varepsilon}$ from the remaining clauses (unit resolution rule).

A reduction is also sometimes called a *unit reduction*. The reduction by a set of literals can be inductively established as follows: let $s = \{x_1^{\varepsilon_1}, x_2^{\varepsilon_2}, \ldots, x_k^{\varepsilon_k}\}$ be a partial assignment of $\upsilon(H)$. The reduction of H by s is defined by suceessively applying definition 4.1 for $x_i^{\varepsilon_i}$, $i = 1, \ldots, k$. It means that the reduction of H by $x_1^{\varepsilon_1}$ forms the formula $H[x_1^{\varepsilon_1}]$, following of a reduction of $H[x_1^{\varepsilon_1}]$ by $x_2^{\varepsilon_2}$, forming as a result the formula $H[x_1^{\varepsilon_1}, x_2^{\varepsilon_2}]$, and so on. The process continues until $H[s] = H[x_1^{\varepsilon_1}, ..., x_k^{\varepsilon_k}]$ is reached. In case that $s = \emptyset$, then $H[s] = H$.

EXAMPLE 4.1
Let $H = \{\{x_1^1, x_2^0\}, \{x_1^1, x_2^1\}, \{x_1^1, x_3^1\}, \{x_1^0, x_3^1\}, \{x_2^0, x_4^1\}, \{x_2^0, x_4^0\}$, $\{x_2^1, x_5^1\}, \{x_3^1, x_5^0\}\}$. Then, $H[x_2^0] = \{\{x_1^1\}, \{x_1^1, x_3^1\}, \{x_1^0, x_3^1\}, \{x_5^1\}, \{x_3^1, x_5^0\}\}$, and for $s = \{x_2^1, x_3^0\}$ we have that $H[s] = \{\{x_1^1\}, \{x_1^1\}, \{x_1^0\}, \{x_4^1\}, \{x_4^0\}, \{x_5^0\}\}$.

Let F be a CF and s 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 unitary clauses can be set to true at the same time. Thus, $F[s]$ does not have models.

During the computation of $F[s]$ new unitary clauses can be generated. Thus, the partial assignment s is extended by adding the unitary clauses found, that is,

$s = s \cup \{u\}$ where $\{u\}$ is a unitary clause. So, $F[s]$ can be again reduced using the new unitary clauses. To this process of reducing an initial formula F via unit clauses is also called *unit resolution propagation*. We call to this iterative process $\text{EVAL}(F, s)$.

The application of definition 4.1 on a 2-CF formula F, could remove variables which have to be considered in the models of F. Let us present an example.

EXAMPLE 4.2
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 $\text{EVAL}(F, s)$, form a set denoted by $\text{ELIMVARS}(s)$. In fact, it can be checked that $|\text{ELIMVARS}(s)| = |\upsilon(F)| - |\upsilon(F')| - |s'|$, where s' is the assignment of s extended by the unit clauses processed during the iterative process $\text{EVAL}(F, s)$. Meanwhile, F' is the subformula from F resulting of the application of $\text{EVAL}(F, s)$. We will denote to those results of $\text{EVAL}(F, s)$, the subformula F' and the extended assignment s', as: $(F', s') = \text{EVAL}(F, s)$. For the sake of simplicity, when we want to reference to only one element of the pair, we use $F' = \text{EVAL}(F, s)$, or $s' = \text{EVAL}(F, s)$.

LEMMA 4.1
Let F be a 2-CF and s a partial assignment on $\upsilon(F)$, if $(F', s') = \text{EVAL}(F, s)$, then

$$\sharp\text{SAT}(F) = \sharp\text{SAT}(F') \times 2^{|\text{ELIMVARS}(s(F))|}. \tag{1}$$

Every model of A_F had already determined truth values for all variable $\upsilon(F)$ of the total formula F. Given a frond clause $c = \{x_j^{\varepsilon_1}, x_k^{\varepsilon_2}\}$ in $\bar{\mathcal{C}}$, for any model s of A_F such that $x_j^{1-\varepsilon_1} \in s$ and $x_k^{1-\varepsilon_2} \in s$, then s is not a model of $A_F \wedge c$, because s falsifies c. Thus, if we know $\text{SAT}(A_F)$ then

$$\sharp\text{SAT}(A_F \wedge c) = \sharp\text{SAT}(A_F) - |\{s \in SAT(A_F) : s \text{ falsifies } c\}|. \tag{2}$$

Let $Y = \{s \in SAT(A_F) : s \text{ falsifies } c\}$, where c is a clause in $\bar{\mathcal{C}}$. A procedure to compute $|Y|$ starts forming a partial assignment $s_c = \overline{c} = \{x_j^{1-\varepsilon_1}, x_k^{\varepsilon_2}\}$, and then Evaluate s_c on A_F, that is, $F' = A_F[s_c]$. Finally, $|Y| = \sharp\text{SAT}(F')$. Eq. (2) can be rewritten, as: $\sharp\text{SAT}(A_F \wedge c) = \sharp\text{SAT}(A_F) - \sharp\text{SAT}(A_F[\overline{s_c}])|$.

We iterate equation (2) for computing new frond clauses in F. This means that

$$\sharp 2\text{SAT}(A_F \bigwedge_{e_i \in \bar{\mathcal{C}}} c_{e_i})$$

is computed in an iterative way, forming in each iteration a new formula $F' = \text{EVAL}(F_{i-1}, s_i)$, which is the resulting formula of the Evaluation of the partial assignment s_i on the formula $F_{i-1} = (A_F \wedge_{j=1}^{i-1} c_{e_j})$. In our case, $s_i = \overline{c}_i$ denotes the partial assignment on $\upsilon(F)$ falsifying c_i, that is, if $c_i = (x_1^{\varepsilon_1}, x_2^{\varepsilon_2})$ then $s_i = \{x_1^{1-\varepsilon_1}, x_2^{1-\varepsilon_2}\}$.

We present now, the pseudo-code for the incremental computation of

$$\sharp 2\text{SAT}(A_F \bigwedge_{e_i \in \bar{\mathcal{C}}} c_{e_i}).$$

Thus, assuming $A_F = (A_F \wedge c_{e_0})$, our procedure computes $\sharp 2\text{SAT}(F)$ based on the following equation:

$$\sharp 2\text{SAT}(F) = \sharp 2\text{SAT}(A_F) - \sum_{i=1}^{t} \sharp 2\text{SAT}((A_F \bigwedge_{j=1}^{i-1} c_{e_j})[\overline{c}_{e_i}]) \tag{3}$$

Algorithm 1 Counting Models

```
1: procedure CountModels(F)
2:     Decompose F in a cactus formula A_F and the frond edges C̄
3:     F_0 ← A_F                          ▷ Cactus formula is a base case for this recursive
4:     A_0 ← ♯2Sat(F_0)                   ▷ process, which is computed in linear time
5:     for all e_i ∈ C̄ do {
6:         F' ← Eval(F_{i-1}, c̄_{e_i})
7:         A_i ← A_{i-1} − CountModels(F')
8:         F_i ← (F_{i-1} ∧ c_{e_i})
9:         }
10:    return (A_i)                       ▷ ♯2Sat(F) = A_i
```

In fact, if the constraint graph of each resulting subformula $F_i = (A_F \bigwedge_{j=1}^{i-1} c_{e_j})[\overline{c}_{e_i}], i = 1, \ldots, t$ has not intersecting cycles, or $|\upsilon(F_i) \leq d|$, then each $\sharp 2\text{SAT}(F_i)$ is computed in linear time on the size of F_i, which in the worst case is of order $O(m)$, and then the sum $\sum_{i=0}^{t-1} \sharp 2\text{SAT}(F_{i+1})$ will be computed in a complexity time of $O(m \cdot t) \leq O(m(m-n))$ and all the computation of $\sharp 2\text{SAT}(F)$ will have, in the worst case, a polynomial time complexity of order $O(m \cdot (m-n))$, that is a quadratic-time complexity on the number of clauses of the input formula.

In this case, we can consider the computation of $\sharp 2\text{SAT}(F)$ by $t+1$ linear-time logical subformulas F_i (where $F_0 = A_F$ and the computation of each $\sharp 2\text{SAT}(F_i), i = 1, \ldots, t$ is done in linear-time complexity). Notice that in this case, the initial graph G_F could have knots but each one of the constraint graphs of each $F_i, i = 1, \ldots, t$ may have not knots.

If the constraint graph of any $F_i, i = 1, \ldots, t$ has knots (intersecting cycles) then the same equation (3) is applied recursively in order to compute $\sharp 2\text{SAT}(F_i)$. For example, if only two recursive calls are needed to compute $\sharp 2\text{SAT}(F)$ (one for F and one for some $F_i, i = 1, \ldots, t-1$) then the computation of $\sharp 2\text{SAT}(F)$ is done by two recursive calls of the procedure CountModels, that includes linear-time and quadratic-time polynomial functions, where each one of its constraint graphs has at most one set of intersecting cycles.

In this way, we can consider the raise of the degree of the polynomial time-function for computing $\sharp 2\text{SAT}(F)$ in accordance to the cardinality of the knots κ_i appearing in the constraint graphs of the subformulas G_{F_i}, that also represents the maximum number of recursive calls to CountModels until subgraphs holding the basic cases for the procedure are obtained.

Of course, an ordering on the set of frond edges $\bar{\mathcal{C}}$ is crucial for accelerating the computation of $\sharp 2\text{SAT}(F)$. For this reason $\bar{\mathcal{C}}$ is ordered in connected components $\mathcal{K} = \{K_1, \ldots, K_r\}$, where each $(A_F \cup K_i)$ is a knot of the graph.

The order on the clauses of $\bar{\mathcal{C}}$ allows the decomposition of G_F in knots, at the same time that provides a way of computing $\sharp 2\text{SAT}(F)$ in a bottom-up orientation, based on the computation of each $\sharp 2\text{SAT}(F_i \cup K_{i+1}), i = 1, \ldots, r$. Then, the recursive procedure CountModels for computing $\sharp 2\text{SAT}(F)$ depends on the knot of the maximum cardinality, more than the number of knots in F.

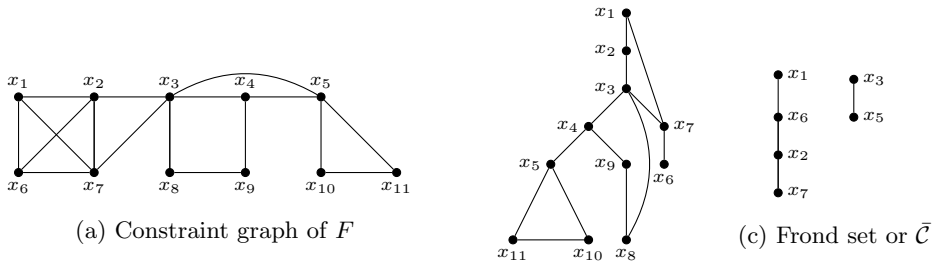
In fact, for two edges e_1, e_2 in $K_j \in \mathcal{K}$ usually they form a same subgraph when they are Evaluated on the graph A_F, as it will be shown in the following example.

Example 4.3

Let us consider as an example a 2-CF positive monote. This means that all variable in $\upsilon(F)$ appear in unnegated form in F.

Let $F = \Big\{\{x_1^1, x_2^1\}, \{x_1^1, x_6^1\}, \{x_1^1, x_7^1\}, \{x_2^1, x_3^1\}, \{x_2^1, x_6^1\}, \{x_2^1, x_7^1\}, \{x_3^1, x_4^1\}, \{x_3^1, x_7^1\}, \{x_3^1, x_8^1\}, \{x_3^1, x_5^1\}, \{x_4^1, x_5^1\}, \{x_4^1, x_9^1\}, \{x_5^1, x_{10}^1\}, \{x_5^1, x_{11}^1\}, \{x_6^1, x_7^1\}, \{x_8^1, x_9^1\},$

$\{x_{10}^1, x_{11}^1\}\Big\}$ *be a 2-CF whose constraint graph is shown in Figure (1a).*



(a) Constraint graph of F

(b) $A_F = F_0$

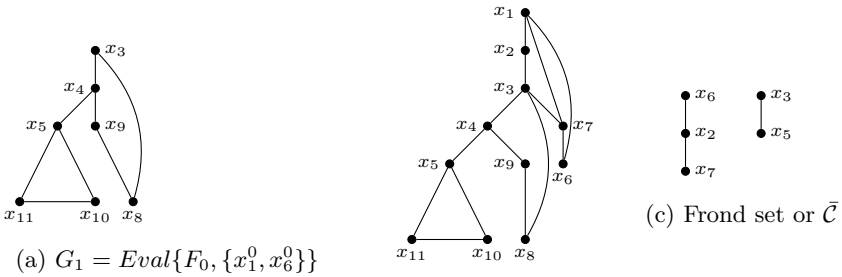
(c) Frond set or $\bar{\mathcal{C}}$

Figure 1: The cactus graph A_F (1b) built using a *dfs* traversal on the graph (1a), its frond set is given by graph (1c).

(a) Knot 1

(b) Knot 2

Figure 2: Knots formed from the constraint graph of Figure (1a). The dotted lines indicate the elements of the frond set.



(a) $G_1 = Eval\{F_0, \{x_1^0, x_6^0\}\}$

(b) F_1

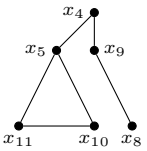
(c) Frond set or $\bar{\mathcal{C}}$

Figure 3: The constraint graphs after Evaluation of $\text{EVAL}(F_0, \{x_2^0, x_6^0\})$, F_1 and the new frond set $\bar{\mathcal{C}}$.

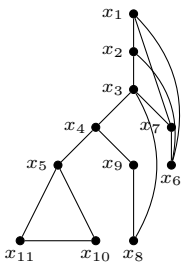
For monotone formulas, ♯2SAT *continues being a ♯P-complete problem. Furthermore,* ♯2SAT *for a positive monotone formula is directly related to the problem of counting independent sets on the constraint graph of the formula.*

For our input formula, the cactus graph $A_F = F_0$ and its frond set $\bar{\mathcal{C}}$ are shown in figures (1b) and (1c) respectively. The constraint graph has two knots which are shown in figures 2a and 2b.

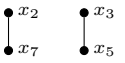
Let $A_0 = \sharp\text{SAT}(F_0) = 176$, since there are two knots, the algorithm processed the first one, lets say Knot 1 (2a). Select the first edge $\{x_1^1, x_6^1\}$ from the frond set, so $A_1 = A_0 - \text{COUNTMODELS}(Eval\{F_0, \{x_1^0, x_6^0\}\})$. The constraint graph of $\text{EVAL}\{F_0, \{x_1^0, x_6^0\}\}$ is presented in Figure (3a). Hence $A_1 = A_0 - \sharp\text{SAT}(G_1)$=176-26=150. $F_1 = F_0 \cup \{x_1^1, x_6^1\}$ is shown in Figure (3b) and the new frond set in Fig-



(a) $G_2 = Eval\{F_1, \{x_2^0, x_6^0\}\}$

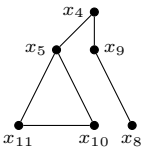


(b) F_2

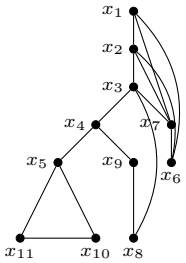


(c) Frond set or $\bar{\mathcal{C}}$

Figure 4: The constraint graphs after Evaluation of $\textsc{Eval}(F_1, \{x_2^0, x_6^0\})$, F_2 and the new $\bar{\mathcal{C}}$.



(a) $G_3 = Eval\{F_2, \{x_2^0, x_7^0\}\}$



(b) F_3

(c) Frond set or $\bar{\mathcal{C}}$

Figure 5: The constraint graphs after Evaluation of $\textsc{Eval}(F_2, \{x_1^0, x_7^0\})$, F_3 and the new frond set $\bar{\mathcal{C}}$.

ure (3c). Notice that for this class of instances, the resulting graph from $\textsc{Eval}\{F_i, \{x_j^0, x_k^0\}\}$ *is the same graph obtained of the reduction:* $(F_i - (N[x_j] \cup N[x_k]))$.

The process is repeated, now select a second edge from $\bar{\mathcal{C}}$ *let said* $\{x_6^1, x_2^1\}$. *Hence* $G_2 = \textsc{Eval}(\{F_1, \{x_6^0, x_2^0\}\})$, *Figure (4a). So* $\textsc{CountModels}(G_2) = 18$ *and* $A_2 = A_1 - \textsc{CountModels}(G_2) = 150 - 18 = 132$. *So* $F_2 = F_1 \cup \{x_2^1, x_6^1\}$ *and* $\bar{\mathcal{C}} = \{\{x_2^1, x_7^1\}, \{x_3^1, x_5^1\}\}$, *Figures (4b) and (4c) respectively.*

The last edge of the first knot is selected $\{x_2, x_7\}$ *which allows to compute* $G_3 = \textsc{Eval}(F_2, \{x_1^0, x_7^0\})$, *Figure (5a). This time* $G_3 = G_2$ *so* $\textsc{CountModels}(G_3) = 18$ *and* $A_3 = A_2 - \textsc{CountModels}(G_3) = 132 - 18 = 114$. *So* $F_3 = F_2 \cup \{x_2^1, x_7^1\}$ *and* $\bar{\mathcal{C}} = \{\{x_3^1, x_5^1\}\}$, *Figures (5b) and (5c) respectively. The last clause* $\{x_3^1, x_5^1\}$ *from* $\bar{\mathcal{C}}$, *which forms the second knot of* F, *is selected. Then* $G_4 = Eval(F_3, \{x_3^0, x_5^0\})$ *is computed, Figure (6).*

Therefore $\textsc{CountModels}(G_4) = 6$. *So,* $A_4 = A_3 - \textsc{CountModels}(G_4) = 114 - 6 = 108 = \sharp\textsc{Sat}(F)$ *the initial formula.*

Notice that for the above example, the computation of $\textsc{CountModels}(G_i)$, $i = 1, 2, 3, 4$ has been done in linear time, since each G_i is a cactus graph. Hence if the resulting Evaluation of the constraint graph G_i with a clause from the frond set is not cactus, an unfold of G_i has to be done forming its own frond set which results in an exponential time in the knotting number of G_F in the worst case.

5 Conclusions

The ♯2SAT problem is a classical ♯P-complete problem. However, there are several instances of 2-CF's for which ♯2SAT can be solved efficiently. For example, if G_F (the

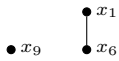


Figure 6: The constraint graphs G_4 after Evaluation of $\text{EVAL}(F_3, \{\overline{x}_3, \overline{x}_5\}\})$.

constraint graph of F) is a cactus graph, then $\sharp 2\text{SAT}(F)$ is solved in linear time.

Otherwise, let G_0 be a cactus graph from G_F containing a maximal number of independent cycles from G_F, and let $\bar{\mathcal{C}} = E(G_F) - E(G_0)$ be the set of edges forming intersected cycles in G_F. The clauses in $\bar{\mathcal{C}}$ are ordered in connected components $\{K_1, \ldots, K_r\}$. Each $(G_0 \cup K_i), i = 1, \ldots, r$ is a knot of the graph.

We have presented an incremental procedure for computing $\sharp 2\text{SAT}(F)$. The procedure begins computing $\sharp 2\text{SAT}(G_0)$, and in each iteration, a new clause $c_i \in \bar{\mathcal{C}}$ is considered in order to form $G_i = (G_{i-1} \cup C_i)$, and then compute $\sharp 2\text{SAT}(G_i) = \sharp 2\text{SAT}(G_{i-1}) - \sharp 2\text{SAT}((G_{i-1})[\overline{c}_i])$.

The order on the clauses of $\bar{\mathcal{C}}$ allows the decomposition of G_F in knots, at the same time that provide a way of computing $\sharp 2\text{SAT}(\text{F})$ in a bottom-up orientation, based on the computation of each $\sharp 2\text{SAT}(G_0 \cup K_i), i = 1, \ldots, r$.

References

[1] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11:1–2, 2000.

[2] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273 – 302, 1996.

[3] E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511 – 529, 2008. Third Haifa Workshop on Interdisciplinary Applications of Graph Theory, Combinatorics & Algorithm.

[4] Guillermo De Ita, Pedro Bello López, and Meliza Contreras González. New polynomial classes for ♯2sat established via graph-topological structure. *Engineering Letters*, 15:250–258, 11 2007.

[5] Guillermo De Ita, J Marcial-Romero, Fernando Zacarias, M.C. González, and P.B. López. Counting falsifying assignments of a 2-cf via recurrence equations. *Engineering Letters*, 23:82–86, 04 2015.

[6] Junping Zhou, Minghao Yin, and Chunguang Zhou. New worst-case upper bound for ♯2-sat and ♯3-sat with the number of clauses as the parameter. volume 1, 01 2010.

[7] Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-sat solutions and colorings with applications. In Ming-Yang Kao and Xiang-Yang Li, editors, *Algorithmic Aspects in Information and Management*, pages 47–57, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] Marco López, J Marcial-Romero, Guillermo De Ita, and Rosa Valdovinos. *A Fast and Efficient Method for ♯2SAT via Graph Transformations: 16th Mexican International Conference on Artificial Intelligence, MICAI 2017, Enseneda, Mexico, October 23-28, 2017, Proceedings, Part I*, pages 95–106. 01 2018.

[9] Ola Angelsmark and Peter Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, pages 81–95, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[10] Russ Bubley. Randomized algorithms: Approximation, generation and counting. In *Distinguished Dissertations*, 2001.

[11] Serge Gaspers and Edward J. Lee. Faster graph coloring in polynomial space. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics*, pages 371–383, Cham, 2017. Springer International Publishing.

[12] Stefan Szeider. On fixed-parameter tractable parameterizations of sat. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 188–202, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[13] Vilhelm Dahllf, Peter Jonsson, and Magnus Wahlstrm. Counting models for 2sat and 3sat formulae. *Theoretical Computer Science*, 332(1):265 – 291, 2005.