Efficient Counting of Models for Boolean Formulas Represented by Embedded Cycles

Guillermo De Ita Luna, Pedro Bello López, Meliza Contreras González

Faculty of Computer Sciences, Universidad Autónoma de Puebla
deita@cs.buap.mx, *pbello@cs.buap.mx*, *mel_22281@hotmail.com*

Abstract. To compute the number of models of a 2-CF (conjunction of clauses with two literals at most), the well-known problem as #2-SAT, is a classical #P-complete problem. We show here an extensive class of instances of 2CF's where to compute the number of models can be done in polynomial time.
Given a 2-CF Σ, we show that if its constrained graph G_Σ is acyclic or if its set of cycles can be arranged as a set of independent and embedded cycles then #2-SAT(Σ) can be computed in polynomial time. We present a procedure for detecting if a set of cycles can be embedded one into other. Furthermore, we design a polynomial reduction for given two restricted intersected cycles redraw those as embedded cycles.
The resulting method for counting models for a 2-CF could be used to impact directly in the reduction of the complexity time of the algorithms for other counting problems. For example, for counting independent sets, counting colouring of graphs, counting cover vertex, etc.
Keywords: #SAT Problem, Exact Combinatorial Algorithms, Embedded Cycles.

1. Introduction

#SAT is a classical #P-complete problem even for formulas which are restricted to conjunction of clauses with two literals at most, this last problem denoted as #2-SAT. #SAT is a very hard counting problem, for example, Hunt, et. al. has shown that #SAT to planar 3-CF formulae (its constrained graph is planar) is #P-complete [6].

#2-SAT continues being a #P-complete problem if we consider only monotone formulas or Horn formulas [9]. Even more, #SAT restricted to formulas in the class $(2, 3\mu)$-CF (the class of conjunctions of 2-clauses where each variable appears three times at most) is also #P-complete [8].

On the other hand, the maximum polynomial class recognized for #2SAT is the class $(\leq 2, 2\mu)$-CF (conjunction of binary or unitary clauses where each variable appears two times at most) [3, 8, 9].

#SAT as well as its decision counterpart - the SAT problem, are a special concern to Artificial Intelligence (AI), and it has a direct relationship to Automated Theorem Proving as well as in Approximate Reasoning. For example,

#SAT has applications in estimating the degree of reliability in a communication network, for computing the degree of belief in propositional theories, in Bayesian inference, in a truth maintenance systems, for repairing inconsistent databases [2, 8, 10]. The previous problems come from several AI applications such as planning, expert systems, data-mining, approximate reasoning, etc.

For example, if we assign an equal degree of belief to all basic 'situations' that appear in a knowledge base Σ of an intelligent agent, then we can compute the probability that Σ will be satisified. If Σ involves n variables, the probabilty to satisfies Σ, is: $P_\Sigma = Prob(\Sigma \equiv \top) = \frac{\#SAT(\Sigma)}{2^n}$, where $\top$ stands for the Truth value and $Prob$ is used to denote the probability. Thus, to compute the degree of belief of the agent in a propositional formula F with respect to its knowledge base Σ, is the fraction of models of Σ that are consistent with the formula F, that is, the conditional probability of F with respect to Σ, which is denoted by $P_{F|\Sigma} = Prob((\Sigma \wedge F) \equiv \top | \Sigma \equiv \top) = \frac{\#SAT(\Sigma \wedge F)}{\#SAT(\Sigma)}$.

One important goal of research is to recognize the class of formulas for Σ and F where the degree of belief $P_{F|\Sigma}$ can be done efficiently, and for this, to develop smart algorithms for solving #SAT is key.

By other way, many combinatorial problems ask about embeddings of graphs into other objects [7]. For instance, the polynomial time solvable *graph planarity* problem ask whether a given graph G, it can be embedded in the plane in such a way that no two edges intersect (except at a common endpoint).

In our case, we are interested in a particular subclass of planar graphs, those graphs whose set of cycles can be arranged as independent cycles as well as embedded cycles.

We have designed some procedures for counting models for 2-CF [3–5]. In this article, we extend those procedures. Specially, we consider the previous reduction presented in [5] for translating a pair of intersected cycles with exactly one common edge to a pair of embedded cycles.The main contribution in this article is to present a way to detect if a set of cycles can be embedded one into other, and to show how to compute #SAT for formulas whose constrained graphs have such topology.

2. Notation and Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x or a negated variable $\overline{x}$. As is usual, for each $x \in X$, $x^0 = \overline{x}$ and $x^1 = x$. We use $\upsilon(l)$ to indicate the variable involved by the literal l.

A *clause* is a disjunction of different literals (sometimes, we also consider a clause as a set of literals). For $k \in I\!N$, a *k-clause* is a clause consisting of exactly k literals and, a *$(\leq k)$-clause* is a clause with k literals at most. A unary clause has just one literal and a binary clause has exactly two literals. The empty clause signals a contradiction. A clause is *tautological* if it contains a complementary pair of literals. From now on, we will consider just non-tautological and non-contradictory clauses. A variable $x \in X$ *appears* in a clause c if either x or $\overline{x}$ is an element of c. Let $\upsilon(c) = \{x \in X : x \text{ appears in } c\}$.

A *Conjunctive Form* (CF) is a conjunction of clauses (we also consider a CF as a set of clauses). We say that Σ 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 $(k, s\mu)$-CF, $(\leq k, s\mu)$-CF is a k-CF, $(\leq k) - CF$, such that each variable appears no more than s times. For any CF Σ, let $\upsilon(\Sigma) = \{x \in X : x \text{ appears in any clause of } \Sigma\}$.

An assignment s for Σ is a function $s : \upsilon(\Sigma) \to \{0, 1\}$. An *assignment* can be also considered as a set of no complementary pairs of literals. If $l \in s$, being s an assignment, then s makes l *true* and makes $\overline{l}$ *false*. A clause c is *satisfied* by s if and only if $c \cap s \neq \emptyset$. If for all $l \in c$, $\overline{l} \in s$ then s falsifies c. A CF F is *satisfied* by an assignment s if each clause in F is satisfied by s and s falsifies c if c is not satisfied by s and F is contradicted if it is not satisfied.

Let $SAT(\Sigma)$ be the set of models that Σ has over $\upsilon(\Sigma)$. Σ is a *contradiction* or *unsatisfiable* if $SAT(\Sigma) = \emptyset$. Let $\mu_{\upsilon(\Sigma)}(\Sigma) = |SAT(\Sigma)|$ be the cardinality of $SAT(\Sigma)$ and $[\![n]\!]$ denotes the set $\{1, 2, ..., n\}$.

Given Σ a CF, the SAT problem consists in determining if Σ has a model. The #SAT consists of counting the number of models of F defined over $\upsilon(\Sigma)$. We will also denote $\mu_{\upsilon(\Sigma)}(\Sigma)$ by $\#SAT(\Sigma)$. When $\upsilon(\Sigma)$ will clear from the context, we will omit it as a subscript.

The Graph Representation of a 2-CF

Let Σ be a 2-CF, the *constrained graph* of Σ is the undirected graph $G_\Sigma = (V, E)$, with $V = \upsilon(\Sigma)$ and $E = \{(\upsilon(x), \upsilon(y)) : (x, y) \in \Sigma\}$, that is, the vertices of G_Σ are the variables of Σ and for each clause (x, y) in Σ there is an edge $(\upsilon(x), \upsilon(y)) \in E$. The degree of a node $v \in V$ is the number of incident edges to v. We say that a 2-CF Σ is a *cycle*, a *path* or a *tree* if G_Σ is a cycle, a path or a tree, respectively.

When G_Σ contains several cycles, they could be independent or intersected. We say that to cycles C_i, C_j are independent one to the other if they don't have common edges. If a cycle $C \in G_\Sigma$ is independent with all the other cycles in G, we say that C is an independent cycle in G_Σ.

Given a 2-CF Σ, a *connected component* of G_Σ is a maximal subgraph such that for every pair of vertices x, y, there is a path in G_Σ from x to y. We say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ.

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

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

In order to compute $\mu(\Sigma)$, first we should determine the set of connected components of Σ, and this procedure is done in linear time [9]. Then, compute $\mu(\Sigma)$ is translated to compute $\mu_{\upsilon(G)}(G)$ for each connected component G of Σ. From now on, when we mention a formula Σ, we suppose that Σ is a connected component.

3. Basic Procedures for computing #2-SAT

For completeness purposes, we present in this chapter some procedures designed for us in previous articles [3,4,5] although the new contributions will be developed until chapter 4.

Case A:
First, let us consider $G_\Sigma = (V, E)$ **be a path** (also called a chain). Let us write down its associated formula Σ, without a loss of generality (ordering the clauses and its literals, if it were necessary), as: $\Sigma = \{c_1, ..., c_m\}$
$= \left\{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_2}, x_3^{\delta_2}\}, \ldots, \{x_{m-1}^{\epsilon_m}, x_m^{\delta_m}\}\right\}$, where $|\upsilon(c_i) \cap \upsilon(c_{i+1})| = 1$, $i \in [\![m-1]\!]$, and $\delta_i, \epsilon_i \in \{0, 1\}$, $i = 1, ..., m$.

As Σ has m clauses then $|\upsilon(\Sigma)| = n = m+1$. We will compute $\mu(\Sigma)$ in base to build a series (α_i, β_i), $i = 1, ..., m$, where each pair of the series is associated to the variable x_i of $\upsilon(\Sigma)$. The value α_i indicates the number of times that the variable x_i is 'true' and β_i indicates the number of times that the variable x_i takes value 'false' over the set of models of Σ.

Let f_i be a family of clauses of Σ built as follows: $f_0 = \emptyset$,$f_i = \{c_j\}_{j \leq i}$, $i \in [\![m]\!]$. Note that $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisfies } f_i\}$, $A_i = \{s \in SAT(f_i) : x_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{x}_i \in s\}$. Let $\alpha_i = |A_i|$; $\beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$. From the total number of models in $\mu_i, i \in [\![m]\!]$, there are α_i of which x_i takes the logical value 'true' and β_i models where x_i takes the logical value 'false'.

In general, we compute the values for (α_i, β_i) associated to each node x_i, $i = 2, .., m$, according to the signs (ϵ_i, δ_i) of the literals in the clause c_i, by the next recurrence equation:

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

Notice that, as $\Sigma = f_m$ then $\mu(\Sigma) = \mu_m = \alpha_m + \beta_m$. We denote with $' \rightarrow'$ the application of one of the four rules of the recurrence (2).

If Σ is a path, we apply (2) in order to compute $\mu(\Sigma)$. The procedure has a linear time complexity over the number of variables of Σ, since (2) is applied while we are traversing the path, from the initial node to the final node.

Example 1 *Let Υ be a* monotone 2-*CF with m clauses and where G_Υ is a path. I.e $\Upsilon = \{(x_1, x_2), (x_2, x_3), \ldots, (x_{m-1}, x_m)\}$. Then, at the beginning of the recurrence (2), $(\alpha_1, \beta_1) = (1, 1)$ and $(\alpha_2, \beta_2) = (2, 1)$ since $(\epsilon_1, \delta_1) = (1, 1)$, and in general, as $(\epsilon_i, \delta_i) = (1, 1)$, for $i \in [\![m]\!]$, then the rule: $(\alpha_i, \beta_i) = (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1})$ is always applied while we are scanning each node of the path, thus the Fibonacci numbers appear!.*

$\mu_2 \quad = \alpha_2 + \beta_2 = \alpha_1 + \beta_1 + \alpha_1 = 3,$
$\mu_3 \quad = \alpha_3 + \beta_3 = \mu_2 + \alpha_2 = 5,$
$(\mu_i)_{i \geq 3} = \alpha_i + \beta_i \ = \mu_{i-1} + \mu_{i-2}$

Applying the Fibonacci series for a monotone formula F, we obtain the values $(\alpha_i, \beta_i) : (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \rightarrow (8,5) \rightarrow (13,8)$, ..., and this last series coincides with the Fibonacci numbers: $(F_2, F_1) \rightarrow (F_3, F_2) \rightarrow (F_4, F_3) \rightarrow (F_5, F_4) \rightarrow (F_6, F_5) \rightarrow (F_7, F_6)$... Then, we infer that $(\alpha_i, \beta_i) = (F_{i+2}, F_{i+1})$ and then $\mu_i = F_{i+2} + F_{i+1} = F_{i+3}, i = 1, ..., m$. E.g. for $m = 5$, we have $\#SAT(F) = \mu_5$ Thus, the following theorem follows:

Theorem 1 *Let Σ be a monotone 2-CF with n nodes such that G_Σ is a path, then: $\#SAT(\Sigma) = F_{n+2}$*

Case B:

Let G_Σ be a simple cycle with m nodes, that is, all the variables in $\upsilon(\Sigma)$ appear twice, $|V| = m = n = |E|$. Ordering the clauses in Σ in such a way that $\mid \upsilon(c_i) \cap \upsilon(c_{i+1}) \mid = 1$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $x_1 = x_m$, then $\Sigma = \left\{ c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\} \right\}_{i=1}^{m}$, where $\delta_i, \epsilon_i \in \{0,1\}$. Decomposing Σ as $\Sigma = \Sigma' \cup c_m$, where $\Sigma' = \{c_1, ..., c_{m-1}\}$ is a path and $c_m = (x_{m-1}^{\epsilon_m}, x_1^{\delta_m})$ is the edge which conforms with $G_{\Sigma'}$ the simple cycle: $x_1, x_2, ..., x_{m-1}, x_1$. We can apply the linear procedure described in the case A for computing $\mu(\Sigma')$.

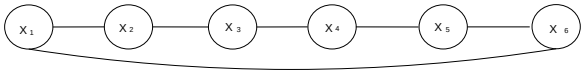
Every model of Σ' had determined logical values for the variables: x_{m-1} and x_1 since those variables appear in $\upsilon(\Sigma')$. Any model s of Σ' satisfies c_m if and only if ($x_{m-1}^{1-\epsilon_m} \notin s$ and $x_m^{1-\delta_m} \notin s$), that is, $SAT(\Sigma' \cup c_m) \subseteq SAT(\Sigma')$, and $SAT(\Sigma' \cup c_m) = SAT(\Sigma') - \{s \in SAT(\Sigma') : s \text{ falsifies } c_m\}$. Let $X = \Sigma' \cup \{(x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})\}$, then $\mu(X)$ can be computed as a path with two unary clauses, that is:

$$\begin{aligned} \#SAT(\Sigma) = \mu(\Sigma) &= \mu(\Sigma') - \mu(X) \\ &= \mu(\Sigma') - \mu(\Sigma' \wedge (x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})) \end{aligned} \tag{3}$$

For example, let us consider Σ be a monotone 2-CF with m clauses such that G_Σ is a simple cycle. $\Sigma = \left\{ c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\} \right\}_{i=1}^{m}$, where $\delta_i = \epsilon_i = 1$, $\upsilon(c_i) \cap \upsilon(c_{i+1}) = \{x_i\}$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $x_1 = x_m$. Let $\Sigma' = \{c_1, ..., c_{m-1}\}$, then: $\mu(\Sigma') = \mu_{m-1} = F_{m+2}$ for theorem 1. As $\epsilon_m = \delta_m = 1$ then $\mu(X) = \mu(\Sigma' \wedge (\overline{x}_1) \wedge (\overline{x}_{m-1}))$ which is computed by the series: $(\alpha_1, \beta_1) = (0,1) = (F_0, F_1)$ since $(\overline{x}_1) \in X$, $(\alpha_2, \beta_2) = (1,0) = (F_1, F_0)$; $(\alpha_3, \beta_3) = (1,1) = (F_2, F_1)$; $(\alpha_4, \beta_4) = (2,1) = (F_3, F_2)$; and in general $(\alpha_i, \beta_i) = (F_i, F_{i-1})$, then for the variable x_{m-1}, $(\alpha_{m-1}, \beta_{m-1}) = (F_{m-1}, F_{m-2})$, then $\mu(X) = \beta_{m-1} = F_{m-2}$ since $(\overline{x}_m) \in X$. Finally, $\#SAT(\Sigma) = \mu(\Sigma) = \mu(\Sigma') - \mu(X) = F_{m+2} - F_{m-2}$. Thus, the following theorem holds.

Theorem 2 *Let Σ be a monotone 2-CF with m clauses and where G_Σ is a simple cycle, then: $\#SAT(\Sigma) = F_{m+2} - F_{m-2}$.*

Example 2 *Let $\Sigma = \{c_i\}_{i=1}^{6} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_5, x_6\}, \{x_6, x_1\}\}$ be a monotone 2-CF which represents the cycle G_Σ=(V,E), see Figure 1. Let $G' = (V, E')$ where $E = E' \cup \{c_6\}$, that is, the new graph G' is Σ minus the edge c_6. Applying theorem 2, $\#SAT(\Sigma) = F_7 + F_5 = 13 + 5 = 18$*



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

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

Case C:
Let $G_\Sigma = (V, E)$ be a tree. As G_Σ Notice that since G_Σ is a tree, all of its edges are *tree edges* and, there are no *back edges*. We denote with (α_v, β_v) the associated pair to a node v ($v \in G_\Sigma$). We compute $\#SAT(\Sigma)$ while we are traversing G_Σ in depth-first search.

Algorithm Count_Models_for_trees(A_Σ)
Input: A_Σ the tree defined by a depth-first search over G_Σ
Output: The number of models of Σ
Procedure: Traversing A_Σ in depth-first, and when a node $v \in A_\Sigma$ is left, assign:

1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in A_Σ.
2. If v is a father node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, then as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been defined based on (2). $(\alpha_{v_i}, \beta_{v_i})$ is obtained by applying (2) over $(\alpha_{i-1}, \beta_{i-1}) = (\alpha_{u_j}, \beta_{u_j})$. This step is iterated until computes the values $(\alpha_{v_j}, \beta_{v_j})$, $j = 1, ..., k$. Finally, let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$.
3. If v is the root node of A_Σ then returns$(\alpha_v + \beta_v)$.

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

Notice that the combination of the procedure for trees and the processing of cycles (equation 3) can be applied for computing $\#SAT(\Sigma)$ if G_Σ is a graph where the depth-first search generates a tree and a set of independent fundamental cycles.

Thus, the procedures presented in this section, for computing $\#SAT(\Sigma)$ being Σ a cycle, a path, a tree, or a tree union independent cycles, each one runs in linear time over the size of the graph.

The class of Boolean formulas Σ depth-first search builds a tree and a set of fundamental independent cycles conforms a new polynomial instances for #2-SAT. This class of instances is a superclass of $(2, 2\mu)$-CF, and it has not restrictions over the number of occurrences of a variable in the formulas, although $(2, 3\mu)$-CF is a $\#P$-complete problem.

4. Processing Embedded Cycles

Given a constrained graph G_Σ of a Boolean formula Σ, if G_Σ contains some intersected cycles, we look at if those cycles can be considered as embedded cycles, see figure 2. It is possible that some cycles can be expressed as embedded cycles although they are not initially recognized as such.

In order to recognize when two cycles C_i and C_j can be expressed as embedded cycles, we use the or-exclusive operation. Given two intersected cycles C_i, C_j of a graph we say that C_i can be embedded into C_j, if:
a)$V(C_i) \subset V(C_j)$: the set of nodes of C_i is a subset of the nodes of C_j.
b)$|E(C_i) - E(C_j)| = 1$: there is only one edge from C_i which is not edge of C_j.
c)$C_i \oplus C_j = C_k$: being C_k a new cycle distinct to C_i and C_j and $\oplus$ is the or-exclusive operation between the edges of the cycles.

If the cycles C_i and C_j hold the previous three conditions, we say that C_i is an internal embedded cycle of C_j while C_j is an external embedded cycle of C_i. Notice that the embedded property can be extended to a set of embedded cycles. Let $D = (C_1, C_2, \ldots, C_k)$ be a tuple of cycles of G_Σ such that C_i is embedded in C_{i+1}, $i = 1, \ldots, k-1$, then we say that C_1 is the most internal embedded cycle of D while C_k is the most external cycle of D.

Given a tuple of embedded cycles $D = (C_1, C_2, \ldots, C_k)$, we present in this section how to compute $\#SAT(D)$. First, the order of processing is from the most internal embedded cycle to the most external embedded cycle. The beginning of the processing of each cycle starts for traversing a cycle in depth-first order, visiting the nodes of the cycle as a path from its initial node v_0 to its end node v_f.

Three computing threads are used for carry on the values of the number of models for each node which is visted during the depht-first search. So, a computing thread is a sequence of pairs $(\alpha_i, \beta_i), i = 1, ...m$ used for computing the number of models over a path of m nodes. A main thread, denoted by Lp, is associated to the spanning tree of the graph, this thread is always active during all the counting process.

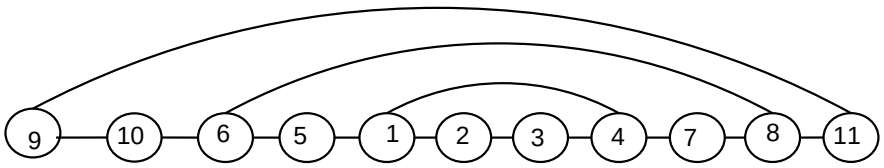


Fig. 2. An initial graph G_e with embedded cycles

Case 1: Processing an Internal Embedded Cycle
Consider that we are processing the most internal cycle C_1 of a tuple $D = (C_1, C_2, \ldots, C_k)$ of embedded cycles. We starts the depht-first search in the first node of the path defined by the cycle C_1 and assigning the values $(\alpha_0^1, \beta_0^1) = (1, 1)$ to the main computing thread, the second thread starts with the values $(\alpha_0^2, \beta_0^2) = (1, 0)$ indicating that this thread carry on the number of models of C_1 where the first variable x_0 does not appear while the third thread starts with the pair $(\alpha_0^3, \beta_0^3) = (0, 1)$ indicating that this thread carry on the number of models of C_1 where the first variable x_0 appears.

We traverse by C_1 as a path from its initial node v_0 to its end node v_f. Each time that a new node of the path in C_1 is visited, the recurrence (2) is applied, obtaining: $(\alpha_i, \beta_i) \rightarrow (\alpha_{i+1}, \beta_{i+1})$. When the search arrives to the end node of the cycle, we have obtained the pairs (α_f^1, β_f^1), (α_f^2, β_f^2) and (α_f^3, β_f^3) corresponding with the final pairs of the three different computing threads. Finally, for processing the back edge in C_1 we use the temporal variables α_{C1} and β_{C1} defined as: $\alpha_{C1} = \alpha_f^1$, $\beta_{C1} = \beta_f^1 - \beta_f^3$, and the numbers of models for the internal cycle C_1, is: $\#SAT(C_1) = \alpha_{C1} + \beta_{C1}$.

Case 2: Processing an External Embedded Cycle
If a current cycle C_i is embedded into other C_j, the pairs (α_f^2, β_f^2) and (α_f^3, β_f^3) corresponding to the threads for processing C_i are used for processing the external cycle C_j. Furthermore, after to process C_i, all the cycle is contracted into a single node Cr_{sf} where s is the number of the initial node and f is the number of the final node of the path in C_i. Then, the new node Cr_{ij} is now part of the path contained in the cycle C_j (see figure 4).

We use the three threads for processing the external cycle C_j, as it was explained previously. The depth-first search is applied over the path defined by C_j and the recurrence (2) is applied over each node of the path until arrive to the node Cr_{sf}. When a computing thread arrive to Cr_{sf}, each current pair $(\alpha_x^i, \beta_x^i), i = 1, 2, 3$ is updated according to the following recurrence.

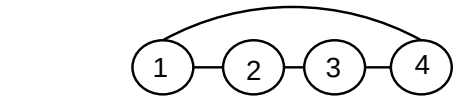
$$\begin{aligned} \alpha_{x+1}^i &= \alpha_f^2 * \alpha_x^i + \alpha_f^3 * \beta_x^i \\ \beta_{x+1}^i &= \beta_f^2 * \alpha_x^i + \beta_f^3 * \beta_x^i, \text{for } i = 1, 2, 3. \end{aligned} \tag{4}$$

Obtaining then the new pairs $(\alpha_{x+1}^i, \beta_{x+1}^i)$, for $i = 1, 2, 3$. We will denote the application of the recurrence (4) as $(\alpha_x, \beta_x) \curvearrowright (\alpha_{x+1}, \beta_{x+1})$.

How it exists an implicit back edge into the contracted node Cr_{sf} then we have to update only the pair $(\alpha_{x+1}^1, \beta_{x+1}^1)$ as $(\alpha_{x+1}^1, \beta_{x+1}^1) = (\alpha_{x+1}^1, \beta_{x+1}^1 - \beta_x^1 * \beta_f^3)$. We will denote the processing of a back edge as $\dashv$, then $(\alpha_x, \beta_x) \dashv (\alpha_{x+1}, \beta_{x+1})$ meaning the application of the formula $(\alpha_{x+1}, \beta_{x+1}) = (\alpha_{x+1}, \beta_{x+1} - \beta_x * \beta_f^2)$.

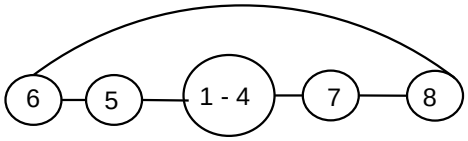
The processing of the current cycle continues as in the Case 1 until arrives to the final node v_f of the external cycle C_j. Obtaining new current values for (α_f^1, β_f^1),(α_f^2, β_f^2),(α_f^3, β_f^3). We illustrate in figures 2-5 how to process a set of embedded cycles.

Example 3 *Let G_e be a graph with embedded cycles (see Figure 2), applying the case 1 for processing the internal cycle from node 1 to node 4, we obtain the new node Cr_{14}. In Figure 4, appears a new graph G_2 which contains a cycle from node 6 to node 8, this cycle has a Cr_{14} as an internal node. Applying the case 2 the graph G_3 is obtained (see Figure 5). This graph has the node Cr_{68} which comes from the contracted cycle G_2. And again, the case 2 is applied obtaining the value for $\#SAT(G_e)$.*



$$
\begin{array}{rl}
Nodes: & Node_1 \quad Node_2 \quad Node_3 \quad Node_4 \\
Lp: & (1,1) \rightarrow (2,1) \rightarrow (3,2) \rightarrow (5,3) \quad = (\alpha_4^1, \beta_4^1) \\
L1 \notin S: & (1,0) \rightarrow (1,1) \rightarrow (2,1) \rightarrow (3,2) \quad = (\alpha_4^2, \beta_4^2) \\
L2 \in S: & (0,1) \rightarrow (1,0) \rightarrow (1,1) \rightarrow (2,1) \quad = (\alpha_4^3, \beta_4^3) \\
 & \Rightarrow (5,3) - (0,2) = (5,2)
\end{array}
$$

Fig. 3. Processing the most internal cycle

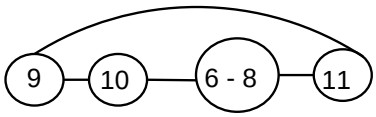


$$
\begin{array}{rl}
G_2 & Node_6 \quad Node_5 \quad Node_{1-4} \qquad\qquad\qquad\qquad Node_7 \quad Node_8 \\
Lp: & (1,1) \rightarrow (2,1) \rightarrow (3,2) \curvearrowright (13,8) \dashv (13,6) \rightarrow (19,13) \rightarrow (32,19) = (\alpha_8^1, \beta_8^1) \\
L1 \notin S: & (1,0) \rightarrow (1,1) \rightarrow (2,1) \curvearrowright (8,5) \quad \dashv (8,4) \quad \rightarrow (12,8) \quad \rightarrow (20,12) = (\alpha_8^2, \beta_8^2) \\
L2 \in S: & (0,1) \rightarrow (1,0) \rightarrow (1,1) \curvearrowright (5,3) \quad \dashv (5,2) \quad \rightarrow (7,5) \quad \rightarrow (12,7) \quad = (\alpha_8^3, \beta_8^3) \\
 & \Rightarrow (32,19) - (0,7) = (32,12)
\end{array}
$$

Fig. 4. (G_2) with the node Cr_{14} which represents an embedded cycle

4.1. Reducing Intersected Cycles to Embedded Cycles

Suppose that we want to process the graph G on Figure 6, the depth-first search over G builds the graph on Figure 7, which has to be processed (count the number of models) as it is illustrated in the same figure. But, if the graph G were reduced to the equivalent graph on Figure 8, then the number of computing threads is smaller. Furthermore, we can apply the procedure for processing embedded



$$
\begin{array}{lllll}
G_3 & Node_9 & Node_{10} & Node_{6-8} & Node_{11} \\
Lp: & (1,1) \rightarrow (2,1) \rightarrow (3,2) \curvearrowright (84,50) \dashv (84,36) \rightarrow & (120,84) & = (\alpha_{11}^1, \beta_{11}^1) \\
L1 \notin S: & (1,0) \rightarrow (1,1) \rightarrow (2,1) \curvearrowright (52,31) \dashv (52,24) \rightarrow & (76,52) & = (\alpha_{11}^2, \beta_{11}^2) \\
L2 \in S: & (0,1) \rightarrow (1,0) \rightarrow (1,1) \curvearrowright (32,19) \dashv (32,12) \rightarrow & (44,32) & = (\alpha_{11}^3, \beta_{11}^3) \\
\Rightarrow (120,84) - (0,32) = (120,52) & & & &
\end{array}
$$

Fig. 5. (G_3) with two implicit embedded cycles

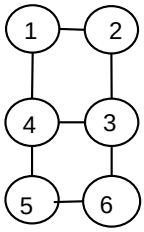
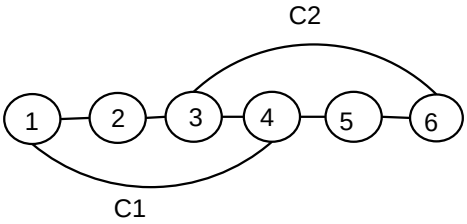


Fig. 6. An initial graph G



$$
\begin{array}{rlllll}
Nodes: & Node_1 & Node_2 & Node_3 & Node_4 & Node_5 \quad Node_6 \\
Lp: & (1,1) \rightarrow & (2,1) \rightarrow & (3,2) \rightarrow & (5,3) & \\
C1: & (0,1) \rightarrow & (1,0) \rightarrow & (1,1) \rightarrow & (2,1)\ closed & \\
C2 \rightarrow Lp: & & & (0,2) \rightarrow & (2,0) & \\
C2 \rightarrow C1: & & & (0,1) \rightarrow & (1,0)\ closed & \\
 & & & Lp: & (5,3) - (0,1) = (5,2) \rightarrow & (7,5) \rightarrow (12,7) \\
 & & & C2: & (2,0) - (0,1) = (2,0) \rightarrow & (2,2) \rightarrow (4,2)\ closed \\
 & & & & \Rightarrow (12,7) - (0,2) = & (12,5)
\end{array}
$$

Fig. 7. Computing $\#SAT(G_\Sigma)$ with intersected cycles

cycles as it was explained previously. We design a polynomial reduction whose main purpose is to translate some intersected cycles as set of embedded cycles. This polynomial graphical reduction is sketching in the following procedure.

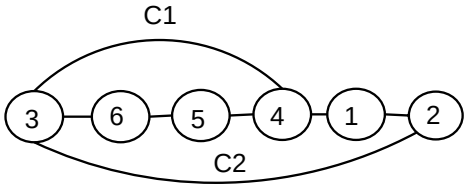
Procedure Graph_Reduction

Input: The graph $T_G = dfs(G)$ and the set $C = \{C_1, C_2, ..., C_k\}$ containing the fundamental cycles of the graph G.

Output: An equivalent graph to T_G where some of its fundamental cycles were redrawn as embedded cycles.

Procedure:

1. for $i = 1$ to k
   (a) If not (C_i is intersected with other cycle $C_j \in C$ and C_i and C_j are not embedded and $|E(C_i) \cap E(C_j)| = 1$) then continue
   (b) /* in other case apply reduction * /
      i. Choose the node v_s which is incident to the back edge of C_i and to the common edge e between C_i and C_j
      ii. Redraw C_j starting with the node v_s and its edge which is not in $E(C_i)$. The cycle C_j is redrawed being the last redrawing edge: e (the common edge between C_i and C_j).
      iii. From the last node v_f of C_j, now we redrawing C_i using its incident edge which is not e. All the nodes and edges of C_i are redrawing being the back edge of C_i the last redrawing edge.
      The original common edge e is translated now as a back edge.
   (c) Continue



$$
\begin{array}{llllll}
G_2 & Node_3 & Node_6 & Node_5 & Node_4 & \\
Lp: & (1,1) \rightarrow (2,1) \rightarrow & (3,2) \rightarrow & (5,3) = (\alpha_4^1, \beta_4^1) & & \\
L1 \notin S: & (1,0) \rightarrow (1,1) \rightarrow & (2,1) \rightarrow & (3,2) = (\alpha_4^2, \beta_4^2) & & \\
L2 \in S: & (0,1) \rightarrow (1,0) \rightarrow & (1,1) \rightarrow & (2,1) = (\alpha_4^3, \beta_4^3) & & \\
\Rightarrow (5,3) - (0,1) = (5,2) & & & & & \\
G_3 & Node_{3-4} & & & Node_1 & Node_2 \\
Lp: & (1,1) \curvearrowright (5,3) \dashv (5,2) \rightarrow & & & (7,5) \rightarrow & (12,7) = (\alpha_2^1, \beta_2^1) \\
L1 \notin S: & (1,0) \curvearrowright (3,2) \dashv (3,2) \rightarrow & & & (5,3) \rightarrow & (8,5) = (\alpha_2^2, \beta_2^2) \\
L2 \in S: & (0,1) \curvearrowright (2,1) \dashv (2,0) \rightarrow & & & (2,2) \rightarrow & (4,2) = (\alpha_2^3, \beta_2^3) \\
\Rightarrow (12,7) - (0,2) = (12,5) & & & & &
\end{array}
$$

Fig. 8. Applying the method for embedded cycles for computing $\#SAT(G)$

This reduction is helping for pre-processing cycles of G, since we have translated the maximum number of intersected cycles as embedded cycles before to apply the algorithm for counting models over embedded cycles.

5. Conclusions

We present different efficient procedures to compute #SAT for subclasses of 2-CF. Let Σ be a 2-CF where G_Σ (the constrained undirected graph of Σ) is acyclic or, if G_Σ contains cyles they can be drawn as independent cycles and embedded cycles, in all those cases, we show that $\#SAT(\Sigma)$ is computed in polynomial time over the length of the formula Σ.

We also show how to detect if a set of cycles can be embedded one into other. Furthermore, we design a polynomial reduction for given two restricted intersected cycles redraw those as embedded cycles.

The efficient methods presented here can be adapted for solving other counting problems and then, to incide directly over the complexity time of those problems.

References

1. Creignou N., Hermann M., Complexity of Generalized Satisfiability Counting Problems, *Information and Computation* 125, (1996), 1-12.
2. Darwiche Adnan, On the Tractability of Counting Theory Models and its Application to Belief Revision and Truth Maintenance, *Jour. of Applied Non-classical Logics*, 11(1-2), (2001), 11-34.
3. De Ita G., Polynomial Classes of Boolean Formulas for Computing the Degree of Belief, *Advances in Artificial Intelligence LNAI 3315*, 2004, 430-440.
4. De Ita G., Tovar M., Applying Counting Models of Boolean Formulas to Propositional Inference, *Advances in Computer Science and Engineering*, Vol 19, 2006.
5. De Ita G., Contreras M., A Polynomial Graphical Reduction to Speed Up the Counting of Models for Boolean Formulas, Proc. of the Workshop in Logic, Language and Computation, CEUR Workshop Proceedings, Vol.220, 2006.
6. Harry B. Hunt, Madhav V. Marathe, Venkatesh Radhakrishnan, Richard E. Stearn, The complexity of planar counting problems, *SIAM Journal on Computing*, 27(4):1142-1167, 1998.
7. Johnson David S., The NP-Completeness Column: An Ongoing Guide, J. Algorithms 3, 1982, pp. 89-99.
8. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), 273-302.
9. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distingished dissertations Springer, 2001.
10. Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, (2001), 398-427.