Recognizing Structural Patterns on Graphs for the Efficient Computation of #2SAT

Guillermo De Ita, Pedro Bello, and Meliza Contreras

Benemérita Universidad Autónoma de Puebla, Faculty of Computer Science
Av. San Claudio y 14 sur, Puebla, México
{deita,pbello,mcontreras}@cs.buap.mx

Abstract. To count models for two conjunctive forms (#2SAT problem) is a classic #P problem. We determine different structural patterns on the underlying graph of a 2-CF F allowing the efficient computation of #2SAT(F).

We show that if the constrained graph of a formula is acyclic or the cycles on the graph can be arranged as independent and embedded cycles, then the number of models of F can be counted efficiently.

Keywords: #SAT Problem, Counting models, Structural Patterns, Graph Topologies.

1 Introduction

#SAT is of special concern to Artificial Intelligence (AI), and it has a direct relationship to Automated Theorem Proving, as well as to approximate reasoning [3,4,9].

The problem of counting models for a Boolean formula (#SAT problem) can be reduced to several different problems in approximate reasoning. For example, for estimating the degree of reliability in a communication network, computing degree of belief in propositional theories, for the generation of explanations to propositional queries, in Bayesian inference, in a truth maintenance systems, for repairing inconsistent databases [1,3,5,9,10]. The previous problems come from several AI applications such as planning, expert systems, approximate reasoning, etc.

#SAT is at least as hard as the SAT problem, but in many cases, even when SAT is solved in polynomial time, no computationally efficient method is known for #SAT. For example, 2-SAT problem (SAT restricted to consider (≤ 2)-CF's), it can be solved in linear time. However, the corresponding counting problem #2-SAT is a #P-complete problem. Earlier works on #2-SAT include papers by Dubois [6], Zhang [11] and Littman [8]. More recently, new upper bounds for exact deterministic algorithms for #2-SAT have been found by Dahllöf [2], Fürer [7], Angelsmark [1] and Jonsson [2]. And given that #2SAT is a #P-complete problem, all the above proposals are part of the class of exponential algorithms.

The maximum polynomial class recognized for #2SAT is the class ($\leq 2, 2\mu$)-CF (conjunction of binary or unary clauses where each variable appears twice at

J.A. Carrasco-Ochoa et al. (Eds.): MCPR 2013, LNCS 7914, pp. 274–283, 2013.

most) [9,10]. Here, we extend such class for considering the topological structure of the undirected graph induced by the restrictions (clauses) of the formula. We extend here some of the procedures presented in [5,4] for the #2-SAT problem and show how to apply them to compute the number of models in a propositional theory. Furthermore, we show different structural patterns on the constrained graph of the formula which allow the efficient computation of the number of models for some classes of 2-CF's.

2 Notation

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 (sometimes, 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 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 $\nu(X)$ to express the set of variables involved in the object X, where X could be a literal, a clause or a Boolean 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 which appear 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 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 be also considered as a set of non complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l true and $\overline{l}$ false. Considering a clause c and assignment s as a set of literals, c is satisfied by s if and only if $c \cap s \neq 0$, and if for all $l \in c$, $\overline{l} \in s$ 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 F a CF, 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 $\nu(F)$. #2-SAT denotes #SAT for formulas in 2-CF.

3 Computing #2SAT for Acyclic Formulas

Let F be a 2-CF F, its signed constrained undirected 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 connecting 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 "$y \overset{+-}{—} x$".

A graph with labelled edges on a set S is a pair (G, ψ), where $G = (V, E)$ is a graph, and φ is a function with domain E and range S. $\psi(e)$ is called the label of the edge $e \in E$. Let $S = \{+, -\}$ be a set of signs . 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 $x(y)$. We say that a 2-CF F is a path, cycle, or a tree if its signed constrained graph G_F is a path, cycle, or a tree, respectively.

3.1 If the 2-CF Represents a Path

If G_F is a path, then $F = \{C_1, C_2, \ldots, C_m\} = \{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_1}, x_3^{\delta_1}\}, \ldots, \{x_m^{\epsilon_m}, x_{m+1}^{\delta_m}\}\}$, where $\delta_i, \epsilon_i \in \{0, 1\}$, $i \in [\![m]\!]$. Let f_i be a family of clauses of the formula F, built as follows: $f_1 = \emptyset; f_i = \{C_j\}_{j<i}$, $i \in [\![m]\!]$. Notice that $n = |\upsilon(F)| = m + 1$, $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. 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 $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

For every node $x \in G_F$ a pair (α_x, β_x) is computed, where α_x indicates how many times the variable x is 'true' and β_x indicates the number of times that the variable x can take value 'false' into the set of models of F. The first pair is $(\alpha_1, \beta_1) = (1, 1)$ since x_1 can be true or false in order to satisfy f_1. The pairs (α_x, β_x) associated to each node x_i,$i = 2, \ldots, m$ are computed according to the signs (ϵ_i, δ_i) of the literals in the clause c_i by the following recurrence equation:

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

Note that, as $F = f_m$ then $\#SAT(F) = \mu_m = \alpha_m + \beta_m$. We denote with $\rightarrow$ the application of one of the four rules of the recurrence (1).

Example 1. *Let* $F = \{(x_1, x_2), (\overline{x}_2, \overline{x}_3), (\overline{x_3}, \overline{x}_4), (x_4, \overline{x}_5), (\overline{x}_5, x_6)\}$ *be a path. The series* $(\alpha_i, \beta_i), i \in [\![6]\!]$, *is computed as:* $(\alpha_1, \beta_1) = (1, 1) \rightarrow (\alpha_2, \beta_2) = (2, 1)$ *since* $(\epsilon_1, \delta_1) = (1, 1)$, *and the rule 4 has to be applied. In general, applying the corresponding rule of the recurrence (1) according to the signs expressed by* $(\epsilon_i, \delta_i), i = 3, ..., 6$, *we have* $(2, 1) \rightarrow (1, 3) \rightarrow (3, 4) \rightarrow (3, 7) \rightarrow (\alpha_6, \beta_6) = (10, 7)$, *and then,* $\#SAT(F) = \mu_6 = \alpha_6 + \beta_6 = 10 + 7 = 17$.

3.2 If the 2-CF Represents a Tree

Let F be a 2-CF where its associated constrained graph G_F is a tree. We denote with (α_v, β_v) the pair associated with the node v ($v \in G_F$). We compute $\#SAT(F)$ while we are traversing by G_F in post-order.

Algorithm Count_Models_for_trees(G_F)
Input: G_F - a tree graph.
Output: The number of models of F
Procedure:
Traversing G_F in post-order, and when a node $v \in G_F$ is left, assign:

1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in G_F.
2. If v is a parent node with a list of child nodes associated, i.e., $u_1, u_2, ..., u_k$ are the child nodes of v, as we have already visited all child nodes, then each pair $(\alpha_{u_j}, \beta_{u_j})$ $j = 1, ..., k$ has been determined based on recurrence (1). Then, let $\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ and $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$. Notice that this step includes the case when v has just one child node.
3. If v is the root node of G_F then return$(\alpha_v + \beta_v)$.

This procedure returns the number of models for F in time $O(n + m)$ which is the necessary time for traversing G_F in post-order.

Example 2. *If* $F = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ *is a monotone 2-CF, we consider the post-order search starting in the node* x_1*. The number of models at each level of the tree is shown in Figure 1. The procedure* $Count\_Models\_for\_trees$ *returns for* $\alpha_{x_1} = 41$*,* $\beta_{x_1} = 36$ *and the total number of models is:* $\#SAT(F) = 41 + 36 = 77$*.*

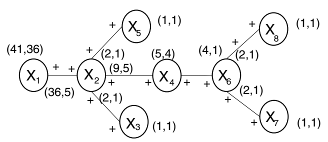


Fig. 1. Counting models over a tree

4 Processing 2-CF's Containing Cycles

Let G_F be a simple cycle with m nodes, that is, all the variables in $\upsilon(F)$ appear twice, $|V| = m = n = |E|$. Ordering the clauses in F in such a way that $| \upsilon(c_i) \cap \upsilon(c_{i+1}) |= 1$, and $c_{i_1} = c_{i_2}$ whenever $i_1 \equiv i_2 \bmod m$, hence $x_1 = x_m$, then $F = \left\{ c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\} \right\}_{i=1}^{m}$, where $\delta_i, \epsilon_i \in \{0, 1\}$. Decomposing F 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 which conforms with $G_{F'}$ the simple cycle: $x_1, x_2, ..., x_{m-1}, x_1$. We will call to

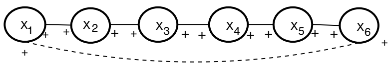
$G_{F'}$ the internal path of the cycle and to c_m the back clause of the cycle. We can apply the linear procedure described above in equation 1 for computing $\#SAT(F')$.

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

$$\#SAT(F) = \#SAT(F' \wedge C_m) = \#SAT(F') - \#SAT(F' \wedge (x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})) \tag{2}$$

Example 1. 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 a cycle: G_Σ=(V,E). Let $G' = (V, E')$ where $E = E' \cup \{c_6\}$, that is, the new graph G' is Σ minus the edge c_6. Applying equation (2), we have that $\#SAT(\Sigma) = \#SAT(F') - \#SAT(F' \wedge \overline{x}_6 \wedge \overline{x}_1) = 21 - 3 = 18$. This example is illustrated in figure 2.

When we count models over any constrained graph G_F, we use *computing 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. A main thread, denoted by L_p, is associated to a spanning tree of G_F, this thread is always active until the process of counting finishes completely. While the thread used for computing the pair associated with $\#SAT(F' \wedge (x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m}))$ is denoted by L_e.



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

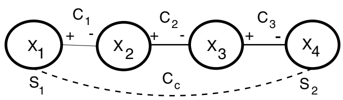
Fig. 2. Computing $\#SAT(F)$ when G_F is a cycle

4.1 Cycles on Alternating Signed Paths

Let $G_F = (V, E, \{+, -\})$ be a signed connected graph of an input formula F in 2-CF. Let v_r be the node of minimum degree in G_F which is chosen to start a depth-first search. We obtain a spanning tree T_G with v_r as the root node and a set of fundamental cycles $C = \{C_1, C_2, ..., C_k\}$, and where each back edge $c_i \in E$ marks the beginning and the end of a fundamental cycle $C_i \in C$.

The edges in T_G are called *tree edges*. A *back edge* $e \in E$ is an edge of G_F which is not part of the spanning tree T_G but e is incident to two nodes of T_G. Each back edge holds the maximum path contained in the fundamental cycle which is part of. We will call to such maximum path, the *internal path* of a fundamental cycle. Given any pair of fundamental cycles C_i and C_j in G_F, if C_i and C_j share edges, we call them *intersecting* cycles; otherwise, they are called *independent* cycles.

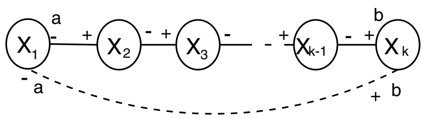
In some cases, the value $\#\mathrm{SAT}(C_i)$ for a fundamental cycle $C_i \in G_F$ can be computed previously to the computation of the total graph G_F in order to determine if the cycle C_i can be reduced to a path or any other simple structure. For example, let us assume a cycle C_i whose internal path is formed by nodes with alternating signs on its edges, while the signs on the back edge e determine the different cases to analyze. Let us order the nodes into the internal path of the cycle as: $x_1 - x_2 - \ldots - x_k$, and we consider to x_1 as the initial node and x_k as the final node of the cycle. Notice that $e = \{x_1, x_k\}$ is the back edge.



$$(\alpha_1, \beta_1) \to (\alpha_2, \beta_2) \to (\alpha_3, \beta_3) \to (\alpha_4, \beta_4)$$
$$L_p : (1,1) \;\to\; (1,2) \;\to\; (1,3) \;\to\; (1,4)$$

Fig. 3. G_F and the computing over the path

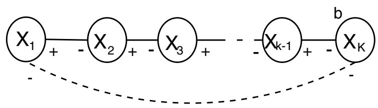
Figure 3 shows a cycle with an internal path whose nodes have alternating signs. Let $\psi(e) = (a, b')$ be the signs on the back edge. Assuming that the variable x_1 appears only with sign a and the variable x_k appears only with sign b, that means that the signs of the back edge coincides with the signs of its endpoints in the internal path. For this case, the back edge (its corresponding clause) can be eliminated from the cycle because the final pair obtained in the secondary thread L_e is $(0,0)$, as it is shown in figure 4.



$$(\alpha, \beta) \to (\alpha + \beta, \beta) \to (\alpha + 2\beta, \beta) \ \ldots \to (\alpha + k \cdot \beta, \beta) \Rightarrow (\alpha + k \cdot \beta, \beta)$$
$$(\alpha, 0) \to (\alpha, 0) \qquad \to (\alpha, 0) \qquad \ldots \to (\alpha, 0) \qquad \Rightarrow -(0,0)$$

Fig. 4. When a back edge does not substract models to the path

The previous example shows that pre-processing the cycles appearing in a current constrained graph is relevant. In some cases, the clause corresponding with the back edge can be eliminated or the value $\#SAT(C_i)$ for a cycle C_i can be computed using symbolic values without knowing the real values of the pairs (α_j, β_j) on the internal path of the cycle, as it is shown in figure 5.



$$(\alpha, \beta) \rightarrow (\alpha, \alpha + \beta) \rightarrow \ldots \rightarrow (\alpha, k \cdot \alpha + \beta) \Rightarrow (\alpha, k \cdot \alpha + \beta) - (\alpha, 0) = (0, k \cdot \alpha + \beta)$$
$$(\alpha, 0) \rightarrow (\alpha, \alpha) \rightarrow \ldots \rightarrow (\alpha, k \cdot \alpha) \Rightarrow -(\alpha, 0)$$

Fig. 5. Computing $\#SAT(C_i)$ using symbolic values on the pairs (α_j, β_j)

5 Processing Embedded Cycles

Let $G_F = (V, E)$ be a connected constrained graph of a 2-CF F. Given two intersecting plane cycles C_i, C_j of a graph, C_i is 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.
In this case, C_i is an internal embedded cycle of C_j and C_j is an external embedded cycle of C_i.

Let us consider a graph G_F formed by a set $D = (C_1, C_2, \ldots, C_k)$ of embedded cycles, such that C_i is embedded in C_{i+1}, $i = 1, \ldots, k-1$. C_1 is the most internal embedded cycle and C_k is the most external cycle of D. For processing this class of graphs, we determine a processing order given by traversing the graph from the most internal to the most external embedded cycle.

For a graph G_F formed by a set of embedded cycles, Lp will be associated with the path formed by the nodes of G_F. Three computing threads are used for processing a current cycle, and for processing all the set of embedded cycles we require at most six computing threads.

Case 1: Processing the Most Internal Embedded Cycle
Let $e_b = \{v_s, v_f\}$ be the back edge which embraces the most internal cycle C_1 of G_F. We use three computing threads with initial values: $(\alpha_s^1, \beta_s^1) = (1, 1)$, $(\alpha_s^2, \beta_s^2) = (1, 0)$ - this thread carry on the number of models of C_1 where the

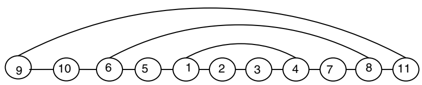
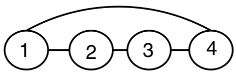


Fig. 6. An initial graph G_F with embedded cycles

variable x_s does not appear, and $(\alpha_s^3, \beta_s^3) = (0, 1)$ - this thread carry on the number of models of C_1 where the first variable x_0 appears.

We traverse by the internal path of C_1 from its initial node v_s to its end node v_f and the last visited edge is its back edge e_b. Each time that a new node on the path is visited, recurrence (1) is applied, obtaining: $(\alpha_i^j, \beta_i^j) \rightarrow (\alpha_{i+1}^j, \beta_{i+1}^j), j = 1, \ldots, 3$. When the search arrives to v_f, we have obtained the pairs (α_f^1, β_f^1), (α_f^2, β_f^2) and (α_f^3, β_f^3). The last edge processed is e_b and for this, we use two temporal variables α_{C1} and β_{C1} defined as: $\alpha_{C1} = \alpha_f^1$, $\beta_{C1} = \beta_f^1 - \beta_f^3$ for the monotone case or according to the signs associated with e_b this last rule is tuned. And, the numbers of models for the internal cycle C_1, is $\#SAT(C_1) = \alpha_{C1} + \beta_{C1}$.



$$
\begin{array}{rlllll}
Nodes: & Node_1 & Node_2 & Node_3 & Node_4 & \\
Lp: & (1,1) \rightarrow & (2,1) \rightarrow & (3,2) \rightarrow & (5,3) & = (\alpha_4^1, \beta_4^1) \\
L1 \notin S: & (1,0) \rightarrow & (1,1) \rightarrow & (2,1) \rightarrow & (3,2) & = (\alpha_4^2, \beta_4^2) \\
L2 \in S: & (0,1) \rightarrow & (1,0) \rightarrow & (1,1) \rightarrow & (2,1) & = (\alpha_4^3, \beta_4^3) \\
 & & \Rightarrow (\alpha_{C1}, \beta_{C1}) = (5, 3-1) = (5, 2) & & &
\end{array}
$$

Fig. 7. Processing the most internal cycle

Case 2: Processing an External Embedded Cycle

Let $C_j = C_{i+1}$ be the following external embedded cycle of the last processed cycle C_i. After processing an internal embedded cycle 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, Cr_{ij} is now a new fat node on the path formed by the nodes: $(V(C_j) - V(C_i)) \cup Cr_{ij}$.

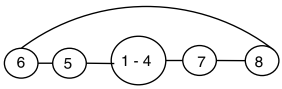
We use new three computing threads for processing the external cycle C_j according to the previous case 1. C_j is traversing as a path and appliying recurrence (1) over each node of the path until arrive to the fat node Cr_{sf}. When the computing threads cross by Cr_{sf}, each current pair $(\alpha_x^i, \beta_x^i), i = 1, 2, 3$ is updated according to the following recurrence.

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

Obtaining the new pairs $(\alpha_{x+1}^i, \beta_{x+1}^i)$, for $i = 1, 2, 3$. We will denote the application of the recurrence (3) as $(\alpha_x, \beta_x) \odot (\alpha_{x+1}, \beta_{x+1})$. As it exists an implicit back edge into the contracted fat node Cr_{sf} then we have to update 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 by $\hookleftarrow$, then $(\alpha_x, \beta_x) \hookleftarrow (\alpha_{x+1}, \beta_{x+1})$ meaning the application of the formula $(\alpha_{x+1}, \beta_{x+1}) = (\alpha_{x+1}, \beta_{x+1} - \beta_x * \beta_f^2)$.

We obtain new current values for the last node of the cycle C_j:(α_f^1, β_f^1),(α_f^2, β_f^2), (α_f^3, β_f^3) and the cycle C_j is contracted into a new fat node C_{kf} where k was the number of the initial node and f was the number of the final node processed in

C_j. In this way, we process any embedded cycle until arrives to the most external cycle of G_F.



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

Fig. 8. Processing an external embedded cycle

Notice that our proposal for counting models on a set D of embedded cycles has a linear time complexity over the number of edges $|E(D)|$. Then, we have shown that for some restricted cases of a 2-CF F, particularly when G_F contains only independent and embedded cycles, #2-SAT(F) can be computed in polynomial time.

6 Conclusion

#SAT problem for the class of Boolean formulas in 2-CF is a classical #P-complete problem. However, 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.

Given a formula F in 2-CF, we show that if the cycles in its constrained graph G_F can be arranged as independent and embedded cycles, then we can count efficiently the number of models of F.

Thus, the unique graph topology for the constrained graph G_F of a 2-CF F where the computation of #2SAT(F) continues being intractable is when G_F has intersected cycles and they can not be arranged as embedded cycles.

References

1. Angelsmark, O., Jonsson, P.: Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 81–95. Springer, Heidelberg (2003)
2. Dahllöf, V., Jonsonn, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulae. J. Theoretical Computer Sciences 332(1-3), 265–291 (2005)
3. 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), 11–34 (2001)

4. De Ita, G., Tovar, M.: Applying Counting Models of Boolean Formulas to Propositional Inference. J. Advances in Computer Science and Engineering Researching in Computing Science 19, 159–170 (2006)
5. De Ita, G., Bello, P., Contreras, M.: New Polynomial Classes for #2SAT Established via Graph-Topological Structure. Engineering Letters 15(2), 250–258 (2007)
6. Dubois, O.: Counting the number of solutions for instances of satisfiability. J. Theoretical Computer Sciences 81(1), 49–64 (1991)
7. Fürer, M., Prasad, S.K.: Algorithms for Counting 2-SAT Solutions and Coloring with Applications. Technical Report No. 33, Electronic Colloqium on Comp. Complexity (2005)
8. Littman, M.L., Pitassi, T., Impagliazzo, R.: On the Complexity of counting satisfying assignments. Technical Report Unpublished manuscript
9. Roth, D.: On the hardness of approximate reasoning. J. Artificial Intelligence 82, 273–302 (1996)
10. Russ, B.: Randomized Algorithms: Approximation, Generation, and Counting, Distingished dissertations. Springer (2001)
11. Zhang, W.: Number of models and satisfiability of set of clauses. J. Theoretical Computer Sciences 155(1), 277–288 (1996)