Building a Minimal Spanning Tree for the #2SAT Problem

Guillermo De Ita, Meliza Contreras, Pedro Bello

Faculty of Computer Science, Universidad Autónoma de Puebla
{deita,mcontreras,pbello}@cs.buap.mx

Abstract. Due to #2SAT is a #P-complete problem, different efficient alternatives have been proposed for approximate solutions to #2SAT. We exploit the existent relation between counting models for two conjunctive forms (2-CF's) and Fibonacci numbers that allow us to count the number of models of the Boolean formula in an incremental way.
We design a polynomial time algorithm for given a 2-CF Σ, to build its constrained graph G_Σ and a spanning tree A_Σ such that $\#SAT(A_\Sigma)$ has a minimal number of models into the set of all spanning tree of G_Σ.

Keywords: Counting the Number of Models, Enumerative Combinatorics, Minimal Spanning Tree.

1. Introduction

Counting combinational objects over graphs has been an interesting and important area of research in Mathematics, Physics, and Computer Sciences. The counting problems, being mathematically interesting by themselves, are closely related to important practical problems. For instance, reliability issues are often equivalent to counting problems. Computing the probability that a graph remains connected given the probabilities of failure over each edge is essentially equivalent to counting the number of ways in which those edges could fail without losing connectivity [2], [8].

Due to #2SAT is a #P-complete problem [5], [3] different efficient methods have been developed for counting, although approximately, the number of models for Boolean formulas in two Conjunctive Forms (2-CF) and since #2SAT is a key problem to clarify the frontier between efficient counting and intractable counting procedures [4].

Let Σ be a 2-CF and G_Σ its connected constrained graph. The combinatory problem that we address here is to approximate the number of models for Σ through to build in polynomial time, a spanning tree A_Σ from G_Σ and at the same time to compute the value #SAT(A_Σ) holding:

1. $\#SAT(A_\Sigma) \geq \#SAT(\Sigma)$
2. $\#SAT(A_\Sigma)$ is minimal into the set of all spanning tree of G_Σ

©M. Martínez, A. Alarcón (Eds.)
Advances in Computer Science and Engineering.
Research Computing Science 45, 2010, pp. 15-25

Received 30/04/10
Final version 19/05/10
Accepted 14/05/10

There are some observations about the values: $\#\mathrm{SAT}(\Sigma)$ and $\#\mathrm{SAT}(A_\Sigma)$. To identify if $\#\mathrm{SAT}(\Sigma)$ is zero or greater or equal to one can be done in polynomial time since the 2SAT problem is solved in polynomial time. If $\#\mathrm{SAT}(\Sigma) > 1$ then as far as we know, there is not polynomial time algorithm for computing $\#\mathrm{SAT}(\Sigma)$. All spanning tree of G_Σ represent formulas where their number of models is greater than 1. The unique way that a 2-CF Σ represents an unsatisfiable formula is that G_Σ contains cycles.

The techniques for building minimal spanning trees have been developed assuming static weights on the edges of the graph [7]. But when we are considering the #2SAT problem, instead of static weights we have dynamic weights determined by the signs of each edge, as well as the number of partial models associated with the endpoints of the edge.

We address the construction of a minimal spanning tree of a constrained signed graph based on the partial values computed in each node of the constrained graph and in the signs of its edges, determining so a new way to build spanning trees with dynamic weights in the edges of the graph.

2. Preliminaries

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

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

A *conjunctive form* (CF) F is a conjunction of clauses (we also consider a CF as a set of clauses). We say that F is a monotone CF if all of its variables appear in unnegated form. A k-CF is a CF containing only k-clauses and, $(\leq k)$-CF denotes a CF containing clauses with at most k literals. A $k\mu$-CF is a formula in which no variable occurs more than k times. A $(k, j\mu)$-CF ($(\leq k, j\mu)$-CF) is a k-CF ($(\leq k)$-CF) such that each variable appears no more than j times.

We use $\upsilon(X)$ to express the variables involved in the object X, where X could be a literal, a clause or a CF. For instance, for the clause $c = \{\overline{x_1}, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$. $Lit(F)$ is the set of literals appearing in F, i.e. if $X = \upsilon(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$. We denote $\{1, 2, ..., n\}$ by $[\![n]\!]$.

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

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

Let F be a CF, F is *satisfied* by an assignment s if each clause in F is satisfied by s. F is *contradicted* by s if any clause in F is contradicted by s. A model of F is an assignment for $\upsilon(F)$ that satisfies F. The SAT problem consists of determining if F has a model and SAT(F) denotes the set of models of F. The #SAT problem (or #SAT(F))consists of counting the number of models of F defined over $\upsilon(F)$. #2-SAT denotes #SAT for formulas in 2-CF. We also denote #SAT(F) by $\mu_{\upsilon(F)}(F)$ or just $\mu(F)$ when $\upsilon(F)$ is clear from the context.

Let Σ be a 2-CF, the *constrained graph* of Σ is the undirected graph $G_\Sigma = (V(\Sigma), E(\Sigma))$, with $V(\Sigma) = \upsilon(\Sigma)$ and $E(\Sigma) = \{\{\upsilon(x), \upsilon(y)\} : \{x, y\} \in \Sigma\}$, i.e. the vertices of G_Σ are the variables of Σ, and for each clause $\{x, y\}$ in Σ there is an edge $\{\upsilon(x), \upsilon(y)\} \in E(\Sigma)$.

Each edge has associated an ordered pair (s_1, s_2) of signs, assigned as labels. For example, the signs s_1 and s_2 for the clause $\{\overline{x} \vee y\}$ are related to the signs of the literals x and y respectively, then $s_1 = -$ and $s_2 = +$ and the edge is denoted as: $x \overset{-\;+}{\text{——}} y$ which is equivalent to $y \overset{+\;-}{\text{——}} x$.

A graph with labeled 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 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 $S \times S$. 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* of x (y).

Let $G_\Sigma = (V, E)$ be a constrained graph of a 2-CF Σ. Sometimes, $V(G)$ and $E(G)$ are used to emphasize the graph G. We denote the cardinality of a set A by $|A|$.

The neighborhood of a vertex $v \in V$ is the set $N(v) = \{w \in V : \{v, w\} \in E(G)\}$, and the closure neighborhood of v is $N[v] = N(v) \cup \{v\}$. The degree of a node v, denoted as $\delta(v)$, is the number of neighbors that it has, that is $\delta(v) = |N(v)|$. A vertex v is *pendant* if its neighborhood contains only one vertex; an edge e is *pendant* if one of its endpoints is a pendant vertex. The degree of the graph G is $\Delta(G) = max\{\delta(x) : x \in V\}$.

Given a graph $G = (V, E)$, $S = (V', E')$ is a subgraph of G if $V' \subseteq V$ and E' contains edges $\{v, w\} \in E$ such that $v \in V'$ and $w \in V'$. If E' contains every edge $\{v, w\} \in E$ where $v \in V'$ and $w \in V'$ then S is called the *subgraph of G induced by S* and is denoted by $G\|S$. We write $G - S$ to denote the graph $G\|(V - V')$. In the same way, $G - v$ for $v \in V(G)$ denotes the induced subgraph $G\|(V - \{v\})$, and $G - e$ for $e \in E(G)$ is the subgraph of G formed by $V(G)$ and $E(G) - \{e\}$.

A *connected component* of G is a maximal induced subgraph of G, that is, a connected component is not a proper subgraph of any other connected subgraph of G. Notice that, in a connected component, for every pair of its vertices u, v, there is a path from u to v. A tree graph is an acyclic connected graph.

We say that a 2-CF Σ is a *path*, a *cycle*, or a *tree* if its corresponding constrained graph G_Σ is a path, a cycle, or a tree, respectively.*17*

3. The Minimal Spanning Tree of a 2-CF

Given a 2-CF Σ, we say that the set of *connected components* of Σ are the subformulas corresponding to the connected components of G_Σ.

Let Σ be a 2-CF. If $\mathcal{F} = \{G_1, \ldots, G_r\}$ is a partition of Σ (over the set of clauses appearing in Σ), i.e. $\bigcup_{\rho=1}^{r} G_\rho = \Sigma$ and $\forall \rho_1, \rho_2 \in [\![r]\!], [\rho_1 \neq \rho_2 \Rightarrow G_{\rho_1} \cap G_{\rho_2} = \emptyset]$, we say that $\mathcal{F}$ is a *partition in connected components* of Σ if $\mathcal{V} = \{\upsilon(G_1), \ldots, \upsilon(G_r)\}$ is a partition of $\upsilon(\Sigma)$.

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

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

The different connected components of G_Σ constitute the partition of Σ in its connected components, even if G_Σ is disconnected. In order to compute $\#SAT(\Sigma)$, first we should determine the set of connected components of G_Σ and that can be done in linear time [6]. Then, $\#SAT(\Sigma)$ is reduced to compute $\#SAT(G)$ for each connected component G of G_Σ. From now on, when we mention a 2-CF Σ, we assume that Σ is a connected component graph.

In our case, a *minimal spanning tree* of a connected component G_Σ which corresponds with a 2-CF Σ is a tree, denoted by T_Σ, containing all vertices of G_Σ and such that

1. $\#SAT(A_\Sigma) \geq \#SAT(\Sigma)$
2. $\#SAT(A_\Sigma)$ is minimal into the set of all spanning trees of G_Σ

A *spanning tree collection* for G_Σ is a set of trees, one for each connected component of G, so that each tree is a spanning tree for its connected component. A *minimal spanning tree collection* is a spanning tree collection where each tree has a minimal number of models with respect to any other spanning tree of the connected component.

There are different algorithms for finding a minimal spanning tree in an undirected graph although, as far as we know, all of them work assuming a static weight in each edge. In our case, the edges in G_Σ have not associated a static weight insteadthey have associated a pair of signs.

If we have a subtree A_Σ which will be extended by one of a possible set of edges $E = \{e_1, e_2, \ldots, e_k\}$, each edge with one of its end-points in a node of A_Σ and the other end-point in a node not included in A_Σ. The signs of the edges determine how will be the increase of #SAT(A_Σ) to #SAT($A_\Sigma \cup \{e\}$) for just one edge $e \in E$. Then, we have dynamic weights associated to each edge $e \in E$ according to the current configuration of a spanning subtree of G_Σ.

We propose a novel algorithm for building a minimal spanning tree assuming such class of dynamic weigths on the edges of the input graph. But before to introduce our proposal, we present some procedures for computing the number of models of a formula for basic topology graphs [1].

4. Linear procedures for #2SAT

For each variable $x \in \upsilon(\Sigma)$, Σ a 2-CF, a pair (α_x, β_x) called the *initial charge*, is used for indicating the number of logical values: 'true' and 'false' respectively, that x takes when #SAT(Σ) is being computed.

Procedure A: If Σ is a path:
Let Σ be a path (or a linear chain). Σ can be written (ordering clauses and variables, if it were necessary) as: $\Sigma = \{c_1, ..., c_m\} = \left\{\{y_0^{\epsilon_1}, y_1^{\delta_1}\}, \ldots, \{y_{m-1}^{\epsilon_m}, y_m^{\delta_m}\}\right\}$, where $\delta_i, \epsilon_i \in \{0,1\},\ i \in [\![m]\!]$ and $|\upsilon(c_j) \cap \upsilon(c_{j+1})| = 1$, $j \in [\![m-1]\!]$. As Σ has m clauses then $|\upsilon(\Sigma)| = n = m+1$.

Let f_i be a family of clauses from Σ built as follows: $f_0 = \emptyset; f_i = \{c_j\}_{j \leq i}$, $i \in [\![m]\!]$. Notice that $f_i \subset f_{i+1}$, $i \in [\![m-1]\!]$. Let $SAT(f_i) = \{s : s \text{ satisfies } f_i\}$, $A_i = \{s \in SAT(f_i) : y_i \in s\}$, $B_i = \{s \in SAT(f_i) : \overline{y}_i \in s\}$. Let $\alpha_i = |A_i|;\ \beta_i = |B_i|$ and $\mu_i = |SAT(f_i)| = \alpha_i + \beta_i$.

The first pair (α_0, β_0) is (1,1) since for any logical value to y_0, f_0 is satisfied. We compute (α_i, β_i) associated with each variable y_i, $i = 1, .., m$, according to the signs: ϵ_i, δ_i of the literals in the clause c_i, by the following recurrence equations:

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

As $\Sigma = f_m$ then $\#SAT(\Sigma) = \mu_m = \alpha_m + \beta_m$. We denote with $' \rightarrow'$ the application of one of the four rules in the recurrence (2).

Example 1 *Let $\Sigma = \{(x_1, x_2), (x_2, \overline{x}_3), (\overline{x}_3), x_4), (x_4, x_5)\}$ be a path, the series $(\alpha_i, \beta_i), i \in [\![5]\!]$, is computed according to the signs of each clause, as it is illustrated if the figure (1a). A similar path with 5 nodes but with different signs in the edges is shown in figure (1b).*

Notice that, according to figure 1, same signs to the adjacent edges of a same node give a greater value for the number of models than different signs to the adjacent edges at the same node. This principle is the base for building minimal spanning trees, since we are looking for edges which provoke a change of signs when they cross by a node of the graph.

When we count models over a constrained graph, 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.

Procedure B: If Σ is a tree:
Let Σ be a 2-CF where its associated constrained graph G_Σ is a tree. We denote with (α_v, β_v) the pair associated with the node v ($v \in G_\Sigma$). We compute

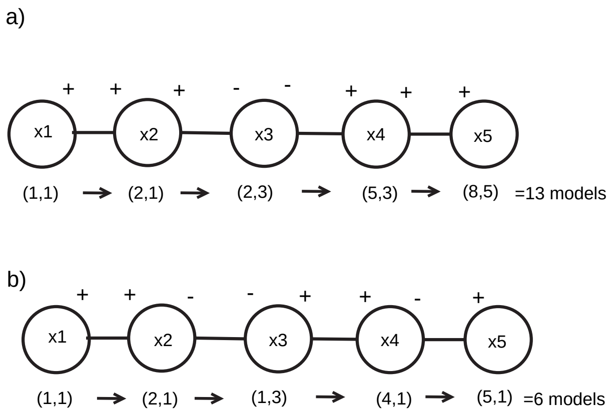

Fig. 1. a) Counting models over paths for monotone formula b)Counting models over paths for non monotone formula

$\#SAT(\Sigma)$ while we are traversing by G_Σ in post-order [7].

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

1. $(\alpha_v, \beta_v) = (1, 1)$ if v is a leaf node in G_Σ.
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 (2). 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_Σ then return$(\alpha_v + \beta_v)$.

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

Example 2 *If* $\Sigma = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ *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 2. The procedure Count_Models_for_trees returns for* $\alpha_{x_1} = 41$*,* $\beta_{x_1} = 36$ *and the total number of models is:* $\#SAT(\Sigma) = 41 + 36 = 77$*.*

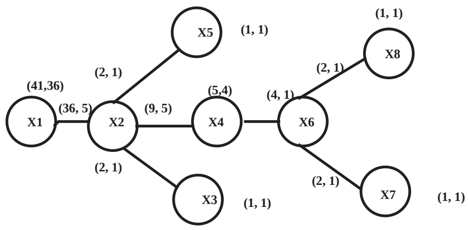

Fig. 2. Counting models over a tree

5. Computation of the Charges of a 2-CF

Once that we know the value $\#\mathrm{SAT}(\Sigma)$, the initial charges for all variables of Σ have already been computed.

The final charge (a_x, b_x) of any variable $x \in \upsilon(\Sigma)$ is the number of true and false logical values respectively, that x takes into the set of models of Σ, i.e. $\#\mathrm{SAT}(\Sigma) = a_x + b_x$. Notice that the initial charge (α_n, β_n) for the last evaluated variable x_n into the above procedures is also its final charge since $\#\mathrm{SAT}(\Sigma) = \alpha_n + \beta_n$.

An important result is that we can apply the inverse action realized in each step of the above procedures in order to propagate the final charge to all variables in Σ, in the following way.

Let $A_1, \ldots, A_n$ be the sequence of initial charges obtained by the above procedure. Now, we build a new sequence of pairs which represent the final charges (or just the charges) $B_n, \ldots, B_1$, being B_i the charge of the variable $x_i \in \upsilon(\Sigma)$, and which is computed as:

$$\begin{array}{l} B_n = \quad A_n \\ B_{n-i} = balance(A_{n-i}, B_{n-i+1}),\ i = 1, ..., n-1 \end{array} \tag{3}$$

$balance(A, B)$ is a binary operator between two pairs, e.g. if $x \overset{s_1\ s_2}{\rightarrow} y$ is an edge of the DAG D_Σ and assuming $A = (\alpha_x, \beta_x)$ be the initial charge of the variable x, $B = (a_y, b_y)$ be the final charge of the variable y, then *balance* produces a new pair (a_x, b_x) which will be the final charge for x, i.e. $\#\mathrm{SAT}(\Sigma) = a_x + b_x$.

Let $\mu_x = \alpha_x + \beta_x$ and $\mu_y = a_y + b_y$. Let $P_1 = \frac{\alpha_x}{\mu_x}$ and $P_0 = \frac{\beta_x}{\mu_x}$ be the proportion of the number of 1's and 0's in the initial charge of the variable x. The charge (a_x, b_x) is computed, as:

$$\begin{array}{l} a_x = a_y \cdot P_1 + b_y;\ b_x = \mu_y - a_x \text{ if}(s_1, s_2) = (+,+) \\ b_x = b_y \cdot P_0 + a_y;\ a_x = \mu_y - b_x \text{ if}(s_1, s_2) = (-,-) \\ b_x = b_y \cdot P_0 + a_y;\ a_x = \mu_y - b_x \text{ if}(s_1, s_2) = (+,-) \\ a_x = a_y \cdot P_1 + b_y;\ b_x = \mu_y - a_x \text{ if}(s_1, s_2) = (-,+) \end{array} \tag{4}$$

Note that the essence of the rules in *balance* consists in applying the inverse operation utilized via recurrence (2) during the computation of $\#\mathrm{SAT}(\Sigma)$, and following the inverse order used in the construction of the sequence $A_1, \ldots, A_n$.

Furthermore, in the case of the bifurcation from a father node to a list of child nodes, the application of the recurrence (4) remains valid since each branch has its respective pair of signs.

A special case to consider is when there are two connected components C_1, C_2 where the charges of their variables have been computed, and a new edge $e = \{x, y\}$ with signs (s_1, s_2) will be utilized for joining both components in just one connected component C. Assuming a charge of (α_x, β_x) for x and (α_y, β_y) for y, we must update such charges, indicated by (α'_x, β'_x) for x and (α'_y, β'_y) for y according with the signs in e, in the following way.

$$\begin{array}{l}
\alpha'_x = \alpha_x * (\alpha_y + \beta_y);\ \beta'_x = \beta_x * \alpha_y \\
\alpha'_y = \alpha_y * (\alpha_x + \beta_x);\ \beta'_y = \beta_y * \alpha_x \text{ if}(s_1, s_2) = (+, +) \\
\beta'_x = \beta_x * (\alpha_y + \beta_y)\,;\ \alpha'_x = \alpha_x * \beta_y \\
\beta'_y = \beta_y * (\alpha_x + \beta_x)\,;\ \alpha'_y = \alpha_y * \beta_x \text{ if}(s_1, s_2) = (-, -) \\
\alpha'_x = \alpha_x * (\alpha_y + \beta_y);\ \beta'_x = \beta_x * \alpha_y \\
\beta'_y = \beta_y * (\alpha_x + \beta_x)\,;\ \alpha'_y = \alpha_y * \beta_x \text{ if}(s_1, s_2) = (+, -) \\
\beta'_x = \beta_x * (\alpha_y + \beta_y)\,;\ \alpha'_x = \alpha_x * \beta_y \\
\alpha'_y = \alpha_y * (\alpha_x + \beta_x);\ \beta'_y = \beta_y * \alpha_x \text{ if}(s_1, s_2) = (-, +)
\end{array} \tag{5}$$

Notice that if some of the initial charges are (1,1) the above recurrence is equivalent with its corresponding case (by the signs) in equation (4).

After computing the new charges for x and y, the charges for all the remaining variables in C have to be updated, for propagating the new values (α'_x, β'_x) to the original variables of C_1 and (α'_y, β'_y) to the original variables of C_2 according to the operator *balance*.

Notice that the computation of the charges of a 2-CF Σ has the same complexity order that the one used for computing $\#\mathrm{SAT}(\Sigma)$. Thus, if the constrained graph of Σ does not contain cycles, then we compute all the charges of the variables of Σ in polynomial time [1].

6. Building the Spanning tree of the Constrained Graph

Let $G_\Sigma = (V, E, \{+, -\})$ be a signed connected graph of an input formula Σ in 2-CF.

Let v_r be a node of G_Σ chosen to start a depth-first search. Each back edge $c_i \in E$ found during the depth-first search marks the beginning and the end of a fundamental cycle of G_Σ.

If the input formula Σ is in fact a tree then the output of the algorithm is the same tree and we just apply the procedure (c) for computing the number of models: $\#\mathrm{SAT}(\Sigma)$.

In the case when there are cycles in G_Σ then we apply the following procedure in order to determine a minimal spanning tree A_F of G_F.

Our proposal works like the well known Kruskal's algorithm. An initial spanning tree $A_\Sigma = (V(G_\Sigma), P\_Edges)$ is formed by all vertices of G_Σ since all vertices are connected components by themselves, and all pendant edge of G are

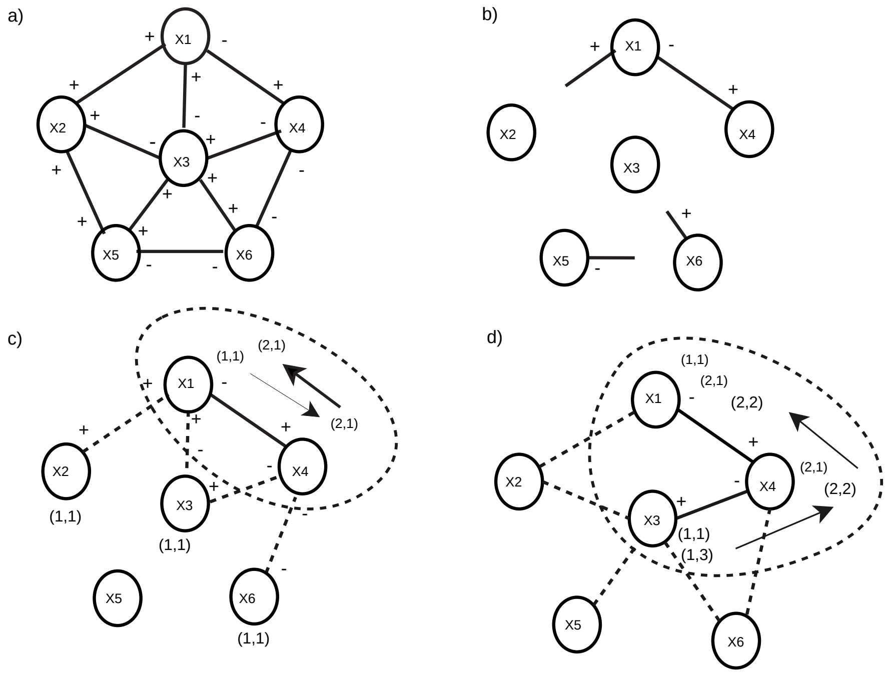

Fig. 3. a) Original Graph b)Selecting of an edge candidate c) Building the first connected component with its respective setback d)Adding a node to the component

edges of the spanning tree (if there are not pendant edges then an emptyset is initially assigned to A_Σ).

In each step of the algorithm $Spanning\_Tree$, the procedure $Count\_Models$ reviews the increment on the number of models when an edge $e \in (E(G_\Sigma) - E(A_\Sigma))$ is considered for being added to the spanning tree.

In order to extend the connected components in A_Σ, the edges in $(E(G_\Sigma) - E(A_\Sigma))$ which conform cycles with A_Σ are deleted from $(E(G_\Sigma)$. And the remaining edges are ordered according to the increment on the number of models in $\#SAT(A_\Sigma)$. If $e \in (E(G_\Sigma)$ infers a minimal increment on #SAT(A_Σ) with respect to the any other edge, e is selected to be added to A_Σ. Notice that the increment on the number of models depends mainly of the signs associated to e as well as the charge of the two-endpoints of e.

There are a set of strategies for detecting the edges in $(E(G_\Sigma) - E(A_\Sigma))$ which infer a minimal increment on the number of models in the spanning tree A_Σ and in fact, when the remaining edges in G have similar values of increment on the number of models such strategies are also applied. Such strategies are:

Algorithm 1 Procedure $Spanning\_Tree(G_\Sigma)$

Input: $G_\Sigma = (V(G), E(G))$ {a constrained signed graph}
Initiate:
Let $P\_Edges = \{e \in E(G_\Sigma) : e$ is a pendant edge $\}$;
$All\_Edges := E(G) - P\_Edges$; {Set of initial edges to test}
$A_\Sigma := (V(G), P\_Edges)$; {all node and pendant edge are connected components of the Tree}
$Cs := \emptyset$; {Set of potential edges which make a change of sign on some vertices}
Iter := 1; {The first iteration}
while ($All\_Edges <> \emptyset$) **do**
  $Count\_Models(All\_Edges, Vect\_Models)$; {count the new number of models generated by each potential edge}
  if ($Vect\_Models\_has\_different\_values$) **then**
    $Sel\_Edge = min\{Vect\_Models\}$; {select the edge which increases a minimum the number of models}
  else
    $Cs = Find(Test, A_\Sigma)$; {looking for edges which could generate a change of sign in any node of A_Σ}
    $Test = complete(Cs)$; {choose edges where its two end-points generate a change of sign on the nodes}
    $Sel\_Edge = First(Test)$; {Select the edge with keeps a potential change of sign of a node}
  end if
  $All\_Edges := All\_Edges - \{Sel\_Edge\}$;
  $E(A_\Sigma) := E(A_\Sigma) \cup \{Sel\_Edge\}$;
  $All\_Edges := All\_Edges - Edges\_Cycles(A_\Sigma, All\_Edges)$; {delete all edge which conform a cycle with the tree A_Σ}
end while

1. If e connects two different connected components of A_Σ where its endpoints v_i and v_j have a change of sign over its incident edges then e is an optimal selection.
2. In general, if two edges e_1 and e_2 generate the same increment on the number of models of A_Σ e_1 is prefered over e_2 if e_1 could bring about a change of signs, in the following steps, on its incident node.
3. When two connected components are joining for forming just one, it is preferable to obtain a path over a tree, as a resulting new connected component.

Thus, we build in polynomial time a spanning tree A_Σ from G_Σ such that

- $\#SAT(A_\Sigma) \geq \#SAT(\Sigma)$
- $\#SAT(A_\Sigma)$ is minimal into the set of all spanning tree of G_Σ

The application of Algorithm 1 to a graph is shown in Figure 3, and finally the minimal spanning tree is shown in Figure 4.

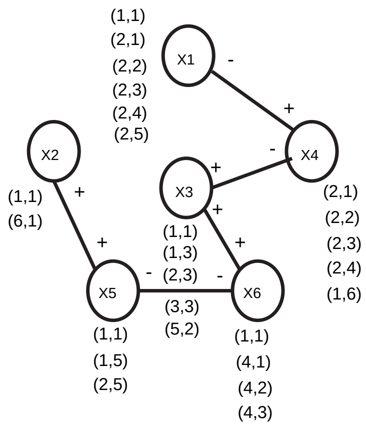

Fig. 4. A Minimal Spanning Tree resulting from the application of the algorithm 1, its number of models is (6,1)=7

7. Conclusions

Although the #2SAT Problem is a #P-complete problem, given a 2-CF Σ there are different methods for computing in an approximate way the value $\#2SAT(\Sigma)$. One of those methods, is based on computing the minimal spanning tree A_Σ.

Although the edges of the constrained graph have not weights, there are a pair of signs associated with each edge. And in this case, the signed edges allow us to determine dynamic weights which are the base for computing a minimal spanning tree of the constrained graph of a Boolean two conjunctive form.

References

1. De Ita G., Bello P., Contreras M., Efficient counting of models for Boolean Formulas Represented by Embedded Cycles, *CEUR WS Proceedings*, 286, (2008).
2. Dyer M., Greenhill C., Some #P-completeness Proofs for Colourings and Independent Sets, Research Report Series, University of Leeds, 1997.
3. Garey M., Johnson D., Computers and Intractability a Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.
4. Greenhill Catherine, The complexity of counting colourings and independent sets in sparse graphs and hypergraphs", *Computational Complexity*, 9(1): 52-72, 2000.
5. Russ B., *Randomized Algorithms: Approximation, Generation, and Counting*, Distinguished dissertations Springer, 2001.
6. Roth D., On the hardness of approximate reasoning, *Artificial Intelligence 82*, (1996), pp. 273-302.
7. Tarjan R., Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, Vol. 1, pp.146-160, 1972.
8. Vadhan Salil P., The Complexity of Counting in Sparse, Regular, and Planar Graphs, *SIAM Journal on Computing*, Vol. 31, No.2, pp. 398-427, 2001.