Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

CONSTRUCCIÓN DE UN ALGORITMO PARA CONTAR MODELOS DE FÓRMULAS EN $2 - FC$

Tesis presentada por Omar Pérez Barrios
para obtener el grado de Maestro en Ciencias de la Computación

Asesor

Dr. Guillermo De Ita Luna

2015

Capítulo 1

Agradecimientos

Debo agradecer de manera especial y sincera al Doctor Guillermo De Ita Luna por aceptarme para realizar esta tesis bajo su dirección. Su apoyo y confianza en mi trabajo y su capacidad para guiar mis ideas ha sido un aporte invaluable, no solamente en el desarrollo de esta tesis, sino tambin en mi formacin. Las ideas propias, siempre enmarcadas en su orientación y rigurosidad, han sido la clave del buen trabajo que hemos realizado juntos, el cual no se puede concebir sin su siempre oportuna participación. Le agradezco tambin el haberme facilitado siempre los medios suficientes para llevar a cabo todas las actividades propuestas durante el desarrollo de esta tesis. Muchas gracias Doctor y espero verlo pronto.

Índice general

Índice de figuras

Capítulo 2

Introducción

2.1. Planteamiento del problema

Dentro del área de ciencias de la computación se han definido gran diversidad de problemas, de los cuales se ha encontrado algoritmos para resolver gran parte de éllos. Sin embargo, conocer un conjunto de pasos para llegar a la solución de un problema no siempre es suficiente, existen algunos problemas en el área de computación en los cuales el número de operaciones que se requieren para llegar a la solución del problema aumenta mucho más rápido de lo que se incrementan los datos de entrada del problema.

El estudio de estas características de un algoritmo son consideradas dentro del área de teoría de la complejidad computacional, la cual tiene sus orígenes en los inicios de la década de los 60's, cuando los primeros usuarios de computadoras electrónicas comenzaron a prestar peculiar interés al desempeño de sus programas.

Encontrar técnicas satisfactorias para resolver problemas computacionales ha eludido a investigadores por años, entre los problemas más desafiantes computacionalmente, destaca el problema de satisfactibilidad de restricciones, introducido por Stephen Arthur Cook [5], tras el cual se han desarrollado diversas líneas de investigación enfocadas en el campo de complejidad computacional.

Dentro del área de los problemas de satisfactibilidad se encuentra el problema de decisión, el cual consiste en determinar si una fórmula Booleana es satisfactible o no, este problema es conocido como: SAT. Se sabe que una función es satisfactible si existe al menos una asignación que la haga verdadera, estas asignaciones son conocidas como modelos de las fórmulas.

A partir del problema de decisión SAT se deriva el problema de conteo asociado: este problema consiste en determinar el número de modelos de una fórmula Booleana, problema conocido comúnmente como $\#SAT$.
Actualmente existen diversos problemas derivados de SAT y $\#SAT$ que no se encuentran completamente resueltos. Por ejemplo, el determinar si existe un algoritmo de complejidad polinomial en tiempo para resolverlos, o bien demostrar que su solución requiere de un tiempo exponencial de cómputo en base al tamaño de sus instancias de entrada.

Hay un gran interés de investigadores de diversas áreas para encontrar diferentes maneras de resolver el problema $\#SAT$, debido a que una solución eficiente tendría un impacto en diversas disciplinas y en distintas aplicaciones.

El objetivo de este trabajo de tesis es diseñar e implementar un método para resolver el problema $\#SAT$, basado en el análisis de fórmulas en su forma normal conjuntiva. El análisis se realizará mediante la representación de estas fórmulas a través de grafos que representan las mismas condiciones y propiedades de la fórmula, y mediante el estudio de las diversas topologías generadas en estos grafos. Y a través del proceso de recorrido de los elementos de los grafos, se propone la aplicación de un conjunto de reglas y procesos que permiten el conteo efectivo de modelos.

El problema de conteo de modelos sobre fórmulas en forma conjuntiva de cláusulas con a lo más 2 literales ($2-CF$), es conocido como $\#2SAT$, y este es un problema $\#P$ clásico. En esta trabajo de tesis se presenta el análisis sobre diferentes topologías de los grafos que representan ($2-CF$) y que permiten el cálculo eficiente de $\#2SAT(F)$.

En este documento de tesis, se demuestra que si la construcción de un grafo de una fórmula F en $2-CF$ es acíclica, o que si esta contienen sólo ciclos simples los cuales son independientes uno de otro, entonces $\#2SAT(F)$ es calculado eficientemente, y para el resto de posibles configuraciones se analizarán y desarrollarán métodos que calculen de forma eficiente el número de modelos para cualquier función en $2-CF$.

2.2. Objetivos Generales y Específicos

En este proyecto de tesis se planteron los siguientes objetivos:

Objetivo General

- Diseñar y desarrollar un método para calcular el número de modelos sobre fórmulas Booleanas y que esto pueda ser ejecutado en tiempo polinomial. Lo que implicaría que se tendría un algoritmo eficiente y robusto.
- Desarrollar un sistema de software que implemente el algoritmo propuesto, permitiendo a través de su interfaz, observar los resultados del proceso y la representación de la información de acuerdo a las características definidas en el método implementado.

Objetivos Específicos

- Analizar la metodología desarrollada para el análisis de la complejidad en tiempo de un algoritmo que resuelve $\#2-SAT$ que es la base del trabajo que se realiza en esta tesis.
- Implementar a través de un sistema de software el análisis actualmente desarrollado sobre conteo de modelos a través del estudio de los diversos patrones estructurales de un grafo que es generado por una función F de la forma $2-CF$.

Capítulo 3

Fundamentos Teórico

3.1. Teoría de la Complejidad Computacional

En el siglo pasado, se observó cómo se desarrolló una ciencia capaz de clasificar problemas computacionales de una forma muy satisfactoria. En 1936 Alan Turing [30] presentó sus modelos de máquinas abstractas, las cuales pueden ser vistas como una computadora capaz de ejecutar algoritmos. Esta percepción de una computadora se toma del hecho de que plantea los principios básicos de los cuales se desarrollaron estos sistemas y también porque todos los modelos de cómputo han demostrado ser equivalentes a las máquinas de Turing.

La primer máquina de Turing fue diseñada para trabajar de forma determinista y no paralela, este modelo puede ser fácilmente reproducido por el cómputo moderno debido a que comparten el mismo principio básico de ejecución. A finales del año 1950 se comenzó a considerar máquinas de Turing no deterministas [27]. Sin embargo, debido a diversos factores que dificultan su implementación, este modelo no tiene el enfoque en el que se basan los equipos de cómputo actuales, tomando como referencia las máquinas de Turing (y algunas variantes de estas) como herramienta, es posible clasificar los problemas computacionales de acuerdo al grado de dificultad (requerimientos de tiempo y espacio) que la resolución computacional del problema requiere.

3.1.1. Conceptos Básicos de la Teoría de la Complejidad

Por conveniencia, la teoría de la complejidad está diseñada especialmente para aplicarse a problemas de decisión. Un problema de decisión PD, se plantea como una pregunta general la cual acepta como respuesta sólo una de dos posibilidades, la respuesta *SI* o la respuesta *NO*. En forma abstracta, un problema de decisión, puede describirse como:

PD : $\langle D, S \rangle$, donde: D corresponde al *Dominio de valores posibles* y $S \subseteq D$ son las *instancias que resuelven el problema*. El planteamiento del PD es:

$$\forall x \in D : PD(x) = \begin{cases} SI, & \text{si } x \in S \\ NO, & \text{en otro caso} \end{cases}$$

En estos términos, un PD puede usarse como medio para reconocer un lenguaje S que es un subconjunto de palabras de un alfabeto D. En la práctica, el PD intenta reconocer algún conjunto específico de objetos matemáticos, tales como: fórmulas lógicas verdaderas, gráficas que tienen una cierta propiedad, etc. Pero esto requiere el codificar adecuadamente las estructuras subyacentes de los objetos para cualquier instancia del problema, de tal forma que finalmente, se pueda construir un predicado que aún sólo con respuestas *SI* o *NO* obtenga cualquier otra información que se desee.

Un *algoritmo* es un procedimiento general que trabaja paso a paso y en un número finito de éstos, resuelve un problema. Un algoritmo resuelve un problema de decisión PD si puede aplicarse a cualquier instancia I del PD y garantiza que siempre produce una solución para esa instancia. Podemos pensar en un algoritmo como un programa de computadora o como la descripción de la función de transición de una máquina de Turing.

La *función de complejidad de tiempo de un algoritmo*, expresa los requerimientos de tiempo que un determinado modelo de computación necesita al ejecutar el algoritmo para resolver cada posible instancia del problema.

Al especificar las cotas de clases, muchas veces se usa la notación de la función de orden O. Si $G(n)$ es una función, $G : \mathbf{N} \to \mathbf{R}$, $\mathbf{O}(\mathrm{G}(n))$ es el conjunto de funciones G' que satisfacen $G'(n) \leq c \cdot G(n)$ para alguna constante real positiva c y para toda $n \geq n_0$, donde n_0 depende de G. Nótese que si n_0 es cero, entonces se cumple para todo $n \in N$.

Con el fin de clasificar el esfuerzo computacional que se requiere al resolver los problemas de decisión, estos problemas se han clasificado en clases de complejidad. Una misma clase de complejidad contiene a todos los problemas que requieren funciones similares de tiempo para ser resueltos. Denotando a un *algoritmo determinista* como AD y a un *algoritmo no-determinista* como AnD, podemos definir las siguientes clases de complejidad:

$$DLOG = \{PD | \exists AD \text{ que resuelve } PD \text{ en tiempo logarítmico}\}$$

$$P = \{PD | \exists AD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$

$$NP = \{PD | \exists AnD \text{ que resuelve } PD \text{ en tiempo polinomial}\}$$

$$EXP = \{PD | \exists AD \text{ que resuelve } PD \text{ en tiempo exponencial}\}$$

Así, el Conjunto de problemas que pueden ser resueltos en tiempo polinomial por una máquina de Turing determinista se define como la clase de complejidad P, mientras que el conjunto de problemas que pueden ser resueltos en tiempo polinomial pero por una máquina de Turing no determinista es denotado como la clase de complejidad NP.

La complejidad NP es la clase de problemas que tienen un algoritmo determinista de resolución que corre en tiempo exponencial, pero para los cuales también existe un algoritmo no determinista que corre en tiempo polinomial.

Las clases de complejidad son ampliamente utilizadas para clasificar una gran diversidad de problemas, y a través de estas clasificaciones se han desarrollado diversas áreas de investigación. Por ejemplo, una línea crucial de investigación es encontrar

problemas que inicialmente se clasifican en la clase NP, y que a travs de un ánalisis profundo, se puede demostrar que están en la clase P.

Un área de experimentación y de gran importancia en la determinación del umbral entre la clase P y NP, lo han proporcionado las fórmulas Booleanas, de las cuales se derivan diversos problemas, tales como el de satisfactibilidad (SAT), máxima satisfactibilidad ($MAXSAT$), el conteo de modelos ($\#SAT$), etc.

3.1.2. Complejidad #P-completo

En teoría de la complejidad computacional, la clase de complejidad #P es el conjunto de los problemas de conteo asociados a los problemas de decisión en el conjunto NP.

Los problemas correspondientes en #P se interesan en saber cuantos elementos satisfacen la pregunta en lugar de si existe algún elemento que la satisfaga. Por ejemplo:

Un problema pertenece a #P si existe una máquina de Turing no-determinista que en tiempo polinomial obtiene para cada instancia I el número de soluciones diferentes que validan a I. Claramente, un problema #P tiene que ser más costoso que su correspondiente problema en NP. Si es fácil contar las respuestas, también lo es el responder si existe alguna, verificando si la cuenta es mayor a cero.

Un problema *#P-completo* es el conjunto de los problemas de conteo que pertenecen a #P tales que todo problema de #P se puede reducir en todos los de #P-completo en tiempo polinomial. Se puede decir que los problemas #P-completo son los problemas más difciles de #P.

Se piensa que no hay algoritmos en tiempo polinomial para resolver problemas #P-completos. Inclusive, no se conocen algoritmos deterministas que puedan dar una solución aproximada con una calidad razonable. Sin embargo, existen algoritmos probabilísticos que dan una buena aproximación a algunos problemas #P-completos con

una muy buena aproximación y existen otros que atacan la solución por partes que puedan ser solucionables o delimitan hasta donde puede ser computable la solución.

3.2. Estado del Arte

Calcular el número de modelos que satisfacen a una fórmula Booleana no es un problema sencillo. Un aspecto importante al analizar este problema y para encontrar una solución, es definir la estructura de las fórmulas sobre las que se realiza el conteo de modelos. En [12][10] se muestra como una fórmula en $2-FC$ es representada como un grafo, y como tal, se pueden identificar topologias del grafo, y cómo a partir de algunas de estas topologias es posible calcular de forma eficiente el número de modelos.

El problema $\#SAT$ consiste en determinar el número de asignaciones que hacen verdadera a una función Booleana. En 1970, Leslie Valiant [33] propuso la clase de complejidad $\#P$ que contendrían a los problemas de conteo.

Es de comentar que tanto los problemas NP así como algunos problemas de decisión conocidos por encontrarse en P, pueden tener una contraparte en términos de conteo, que están en la clase $\#P$.

Por ejemplo $2SAT$ (problema SAT restringido a cláusulas con un máximo de dos literales) se encuentra en P [7], mientras que $3SAT$ se encuentra en la clase $NP-completo$ [13]. Pero sus problemas de conteo derivados: $\#2SAT$ y $\#3SAT$, ambos pertenecen a la clase $\#P$ [22][34].

Mientras los algoritmos para algunos problemas de decisión se encuentran ampliamente estudiados, este no es el caso para sus problemas derivados en la clase $\#P$. Uno de los primeros algoritmos para un problema de conteo fue propuesto en 1963 por John Ryser [25], quién propuso un algoritmo para contar el número de apareamientos perfectos en un grafo bipartito (un apareamiento perfecto de un grafo es un subconjunto de aristas tal que cada vértice del grafo se encuentra conectado exactamente por una arista y un grafo es bipartito si es posible particionar sus vértices en dos conjuntos tales que cada arista del grafo conecta un vértice de cada conjunto),

resultando la complejidad del algoritmo del orden $O(2^n)$.

Por un largo periodo de tiempo el algoritmo de Ryser fue el único algoritmo exacto para un problema $\#P$. Sin embargo, el interés por estudiar la complejidad computacional de los problemas en la clase $\#P$ se ha incrementado recientemente, prueba de ello son la publicaciones de artículos enfocados al problema de conteo relacionados con la clase $\#P$, algunos ejemplos de publicaciones basadas en este problema son los trabajos de Greenhill [15], Zhang [35], DahllÄof [6], Vadhan [32], Goldberg [14] y Valiant [34].

El conteo de modelos para una fórmula Booleana, problema denotado como $\#SAT$, ha sido estudiado por varios investigadores como Lozinskii [3], Dubois [8], Marathe [21] y muchos otros. El problema de conteo de modelos no solo es interesante matemáticamente, sino que tiene importantes aplicaciones, principalmente relacionadas con problemas de Inteligencia Artificial. Por ejemplo, algunas tareas del razonamiento automático requieren del conteo de modelos de las fórmulas de trabajo, así como otros problemas complejos que se derivan del razonamiento son fuente de aplicación para el problema $\#SAT$.

3.3. Preliminares

Como se comentó anteriormente, el cálculo del número de modelos para una fórmula Booleana se realiza sobre funciones de la forma $2 - CF$. A continuación se describirá a detalle los principales conceptos que constituyen la base a partir de la cual se desarrollará nuestra propuesta algorítmica.

Sea $X = \{x_1, ..., x_n\}$ un conjunto de n variables Booleanas. Una literal puede ser tanto una variable x_i o una variable negada $\overline{x_i}$.

Una cláusula es una disyunción de literales distintas (en ocasiones una cláusula es también considerada como un conjunto de literales). Para cada $k \in N$, una k-

cláusula es una cláusula que consiste de exactamente k literales y una ($\leq k$)-cláusula es una cláusula con a lo más k literales.

La Forma Conjuntiva (CF) de una función Booleana F es una conjunción de cláusulas (una función en CF es también considerado un conjunto de cláusulas). Una función F en forma CF se supone como monótona positiva si ninguna de sus variables se encuentra negada. Una función $k - CF$ es una función CF que contiene exactamente k cláusulas, y una función $(\leq k) - CF$ denota una funcion CF con a lo más k cláusulas.

Se usará $v(Y)$ para expresar el conjunto de variables involucradas en el objeto Y, donde Y puede ser una literal, una cláusula o una fórmula Booleana. Por ejemplo, para la cláusula $c = \{x_1, \overline{x_2}\}$, $v(c) = \{x_1, x_2\}$. $Lit(F)$ es el conjunto de literales que aparecen en una fórmula CF F, es decir si $X = v(F)$, entonces $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x_1}, ..., x_n, \overline{x_n}\}$.

Sea F una fórmula Booleana, entonces una asignación s para F es una función Booleana tal que $s : v(F) \rightarrow \{0, 1\}$. Una asignación puede también ser considerada como un conjunto no complementario de literales. Si $l \in s$, siendo s una asignación, entonces s convierte a l en verdadero y a $\bar{l}$ en falso.

Considerando una cláusula c y una asignación s como un conjunto de literales, se dice que c es satsifecha por s si y sólo si $c \cap s \neq \emptyset$ y si para todo $l \in c$ y $\bar{l} \in s$, entonces s falsifica a c.

Sea F una fórmula Booleana en forma conjuntiva (CF), F se satisface por una asignación s si s satisface a cada una de las cláusulas de F. Cuando una asignación s satisface a F, se dice que s es un modelo de F.

s falsifica a F, si alguna cláusula de F es contradecida por s. Se denotará con $M(F)$ el conjunto de modelos que tiene F sobre $v(F)$, y con $Fals(F)$ al conjunto de asignaciones sobre $v(F)$ que falsifican a F.

La cardinalidad de $M(F)$, se denota por $\#SAT(F)$, esto es $\#SAT(F) = |M(F)|$. Sea $P = \{F_1, ..., F_r\}$ una partición de una fórmula Booleana F, esto es, $\{v(F_1), ..., v(F_r)\}$ es una partición de los componentes conexos de F, entonces se cumple que:

$$\#SAT(F) = \prod_{i=1}^{r} \#SAT(F_i) \tag{3.1}$$

Sea F una función en CF, el problema SAT consiste en determinar si F tiene un modelo. El problema $\#SAT$ consiste en determinar el número de modelos de F definidos sobre $v(F)$. $\#2 - SAT$ expresa $\#SAT$ para fórmulas en $2 - CF$.

Capítulo 4

Desarrollo

4.1. Construcción del grafo de restricciones para una fórmula $2 - CF$.

Existen diversas representaciones sobre grafos de una función en forma conjuntiva [28], en este caso se usará el grafo con signos de una $2 - CF$. Sea F una fórmula en $2 - CF$, su grafo de restricciones es expresado por $G_F = (V(F), E(F))$, con $V(F) = v(F)$ y $E(F) = \{\{v(x), v(y)\} : \{x, y\} \in F\}$, esto es, los vértices de G_F son las variables de F, y por cada cláusula $\{x, y\}$ en F existe una arista $\{v(x), v(y)\} \in E(F)$. A cada arista $c = \{v(x), v(y)\} \in E$ se le asocia un par ordenado de signos (s_1, s_2), asignados como etiquetas en la arista que conecta las variables en la cláusula.

Para cada cláusula c en F se genera un nodo en V y definimos una arista $(c, c') \in E$ si y sólo si las cláusulas c y c'tienen una variable en común. Ver figura 4.1. A este tipo de grafo, le llamaremos el grafo de restricciones de la fórmula.

El dual correspondiente a G_F puede ser representado como un grafo cuyos nodos corresponden a variables en $v(F)$ y existe una arista entre cualquier par de variables que aparezcan en la misma cláusula. Ver figura 4.2.

En [1] el grafo dual es llamado grafo fórmula y en [9] se le denomina grafo fundamental. Para calcular el número de modelos de una fórmula F en $2 - FC$ en su

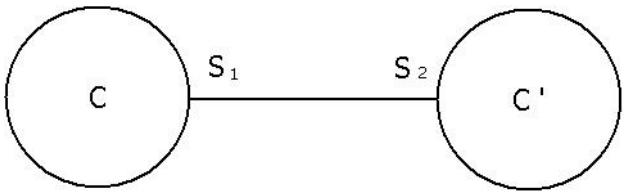

Figura 4.1: Arista entre dos cláusulas con una variable en común.

representación por medio de un grafo, es necesario calcular el número de modelos de cada subgrafo G_F, que corresponda a un componente conexo. A continuación se muestran las diferentes formas de calcular $\#SAT(G_F)$ de acuerdo a la topología que se representa.

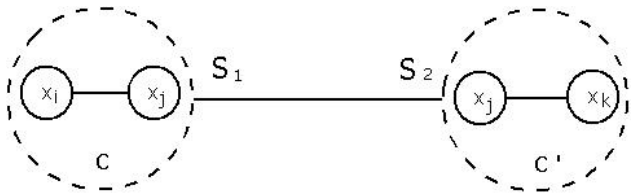

Figura 4.2: Arista entre variables de la misma cláusula.

4.2. Calculo de #2SAT según la topología del grafo

A continuación se presentan distintas topologías sobre un grafo que representa una función en $2-CF$, así como los algoritmos que permiten calcular el número $\#2SAT$ para las fórmulas representadas por esos grafos. También son considerados los casos cuando los grafos tienen aristas paralelas, y cuando las funciones en $2-CF$ tienen cláusulas unitarias.

4.3. Calculo de #2SAT sobre grafos-caminos

Si G_F es un camino entonces F puede escribirse como: $F = \{C_1, C_2, \ldots, C_m\} = \{\{x_1^{\epsilon_1}, x_2^{\delta_1}\}, \{x_2^{\epsilon_2}, x_3^{\delta_2}\}, ..., \{x_m^{\epsilon_m}, x_{m+1}^{\delta_m}\}\}$, donde $\epsilon_i, \delta_i \in \{+, -\}$ para $i \leq N$.

Para cada nodo $x \in G_F$ un par de números (α_x, β_x) es calculado, donde α_x indica cuantas veces la variable x toma valor verdadero y β_x indica el número de veces que la variable x puede tomar valor falso dentro del conjunto de modelos de F. El primer par es $(\alpha_1, \beta_1) = (1, 1)$ porque x_1 puede tomar cualquier valor (verdadero o falso) con el fin de satisfacer f_1.

El par (α_x, β_x) asociado a cada uno de los nodos $x_i, i = 2, \ldots, m$ son calculados de acuerdo a los signos (ϵ_i, δ_i) de las literales en la cláusula c_i de acuerdo a la siguiente ecuación de recurrencia:

$$(\alpha_i, \beta_i) = \begin{cases} (\beta_{i-1} \quad , \alpha_{i-1} + \beta_{i-1}) & \text{si } (\epsilon_i, \delta_i) = (-, -) \\ (\alpha_{i-1} + \beta_{i-1}, \beta_{i-1} \quad) & \text{si } (\epsilon_i, \delta_i) = (-, +) \\ (\alpha_{i-1} \quad , \alpha_{i-1} + \beta_{i-1}) & \text{si } (\epsilon_i, \delta_i) = (+, -) \\ (\alpha_{i-1} + \beta_{i-1}, \alpha_{i-1} \quad) & \text{si } (\epsilon_i, \delta_i) = (+, +) \end{cases} \tag{4.1}$$

De la ecuación de recurrencia anterior dado que $F = f_m$ entonces $\#SAT(F) = \mu_m = \alpha_m + \beta_m$.

Ejemplo 1. Sea $F = \{(x_1, x_2), (\overline{x_2}, \overline{x_3}), (\overline{x_3}, \overline{x_4}), (x_4, \overline{x_5}), (\overline{x_5}, x_6)\}$ una función Booleana cuyo grafo es representado por el camino de la figura 4.3. La serie $(\alpha_i, \beta_i), i \in [|6|]$, es calculada como: $(\alpha_1, \beta_1) = (1, 1) \rightarrow (\alpha_2, \beta_2) = (2, 1)$ donde $(\epsilon_1, \delta_1) = (1, 1)$, porque se aplica la regla 4. En general, aplicando la correspondiente regla de recurrencia (1) de acuerdo a los signos correspondientes a $(\epsilon_i, \delta_i), i = 2, ..., 5$, tenemos $(2, 1) \rightarrow (1, 3) \rightarrow (3, 4) \rightarrow (3, 7) \rightarrow (\alpha_6, \beta_6) = (10, 7)$ y por lo tanto $\#SAT(F) = \mu_6 = \alpha_6 + \beta_6 = 10 + 7 = 17$.

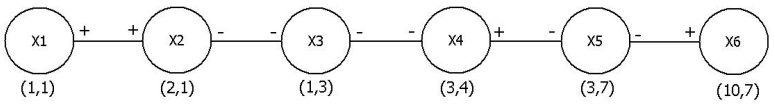

Figura 4.3: Conteo de modelos sobre un camino.

4.4. Procesando aristas paralelas

Considerando el caso de una 2-FC donde hay dos 2-cláusulas que contienen las mismas variables, pero distintos signos entre sus variables, se tiene que el grafo de restricciones tendrá un par de aristas entre los mismos nodos que representan la cláusula, esto significa que tenemos multigrafos, esto es, aristas paralelas entre nodos. En estos caso, la ecuación de recurrencia en (4.2) deberá considerar seis distintos casos.

Tomando en cuenta las dos cláusulas: $c_k = (x_{i-1}^{\epsilon_k}, x_i^{\delta_k})$ y $c_j = (x_{i-1}^{\epsilon_j}, x_i^{\delta_j})$ las cuales involucran las variables: x_{i-1} y x_i. Entonces el cálculo de los valores para (α_i, β_i) asociados al nodo x_i, de acuerdo a los signos (ϵ_k, δ_k) y (ϵ_j, δ_j) se calcula de la siguiente forma:

$$(\alpha_i, \beta_i) = \begin{cases} (\alpha_{i-1}, \alpha_{i-1}) & \text{si } (\epsilon_k, \delta_k) = (+,+) y (\epsilon_j, \delta_j) = (+,-) \\ (\alpha_{i-1} + \beta_{i-1}, 0) & \text{si } (\epsilon_k, \delta_k) = (+,+) y (\epsilon_j, \delta_j) = (-,+) \\ (\beta_{i-1}, \alpha_{i-1}) & \text{si } (\epsilon_k, \delta_k) = (+,+) y (\epsilon_j, \delta_j) = (-,-) \\ (\alpha_{i-1}, \beta_{i-1}) & \text{si } (\epsilon_k, \delta_k) = (+,-) y (\epsilon_j, \delta_j) = (-,+) \\ (0, \alpha_{i-1} + \beta_{i-1}) & \text{si } (\epsilon_k, \delta_k) = (+,-) y (\epsilon_j, \delta_j) = (-,-) \\ (\beta_{i-1}, \beta_{i-1}) & \text{si } (\epsilon_k, \delta_k) = (-,+) y (\epsilon_j, \delta_j) = (-,-) \end{cases} \tag{4.2}$$

Los pares de signos que no se encuentran definidos en (4.2) fueron omitidos debido a que no cambian el número de modelos de la función, tal y como es el caso $(\epsilon_k, \delta_k) = (+,+)$ y $(\epsilon_j, \delta_j) = (+,+)$ donde se puede quitar una de estas cláusulas sin alterar el número de modelos de la función.

También se omiten los casos en que los signos son equivalentes bajo simetría a alguno de los casos ya definidos, por ejemplo, para: $(\epsilon_k, \delta_k) = (+,-)$ y $(\epsilon_j, \delta_j) = (+,+)$ que es equivalente a la regla definida como: $(\epsilon_k, \delta_k) = (+,+)$ y $(\epsilon_j, \delta_j) = (+,-)$.

Cuando en una función en CF existen tres 2-cláusulas que contienen las mismas

variables, entonces el grafo de restricciones tiene tres aristas paralelas. Suponiendo por ejemplo que se tienen las siguientes tres cláusulas: $(x_{i-1}, x_i), (x_{i-1}, \overline{x_i}), (\overline{x_{i-1}}, \overline{x_i})$ (Figura 4.4), entonces $(\alpha_i, \beta_i) = (0, \alpha_{i-1})$ ya que $(\overline{x_{i-1}}, x_i)$ es la única cláusula que no se encuentra considerada, por lo tanto, la única asignación en la que la variable x_{i-1} genera un modelo es preservado y es usado para la asignación falsa de x_i.

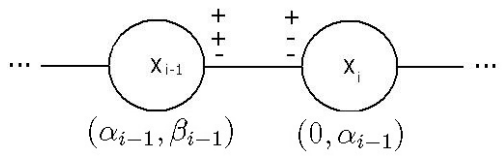

Figura 4.4: Grafo con aristas paralelas.

Procesando cláusulas unitarias: Una cláusula unitaria representa un bucle o un lazo en el grafo de restricciones de una fórmula en $2 - CF$. Si una fórmula F en su forma $2 - CF$ tiene una cláusula unitaria i.e. $U \subseteq F$ y $U = \{(l_1), (l_2), ..., (l_k)\}$. Entonces cuando la recurrencia (4.1) se comienza a aplicar sobre el nodo x_i de G_F, se debe revisar si $x_i \in v(U)$ o no. Si $x_i \notin v(U)$ solo se aplica la recurrencia (4.1), pero si $x_i \in (U)$ entonces:

$$(\alpha_i, \beta_i) = \begin{cases} (0, \beta_i) & \text{si } (\overline{x_i}) \in U \\ (\alpha_i, 0) & \text{si } (x_i) \in U \end{cases} \tag{4.3}$$

4.5. Procesando árboles.

Para el conteo de modelos de una fórmula representada por un grafo acíclico, es necesario recorrer el árbol en post-orden. Sea F una fórmula en 2-CF donde su grafo de restricciones asociado G_F cumple con las propiedades que caracterizan a un árbol, entonces el algoritmo siguiente es aplicado para calcular $\#SAT(F)$ mientras se recorre G_F en post-orden.

Algoritmo Conteo_de_Modelos_para_Árboles (G_F)
Entrada: Un grafo tipo Árbol G_F.
Salida: El número de modelos de F.
Procedimiento:
Recorrer G_F en post_orden, y para todo nodo $v \in G_F$ aplicar:
1. $(\alpha_v, \beta_v) = (1,1)$ si v es un nodo hoja en G_F.
2. Si v es un nodo padre con una lista de nodos hijos asociados
i.e., $u_1, u_2, ..., u_k$ son los nodos hijos de v, como todos los nodos
hijos ya han sido visitados antes de visitar a su nodo padre, entonces cada par $(\alpha_{u_j}, \beta_{u_j})$, $j = 1, ..., k$ se determina en base a la recurrencia lineal en (1). Y se define:
$\alpha_v = \prod_{j=1}^{k} \alpha_{v_j}$ y $\beta_v = \prod_{j=1}^{k} \beta_{v_j}$.
3. Si v es el nodo raíz de G_F entonces regresar $(\alpha_v + \beta_v)$.

Este procedimiento determina el número de modelos para F con un orden de complejidad de $O(n + m)$, el cual es la complejidad computacional necesaria para recorrer G_F en post-orden.

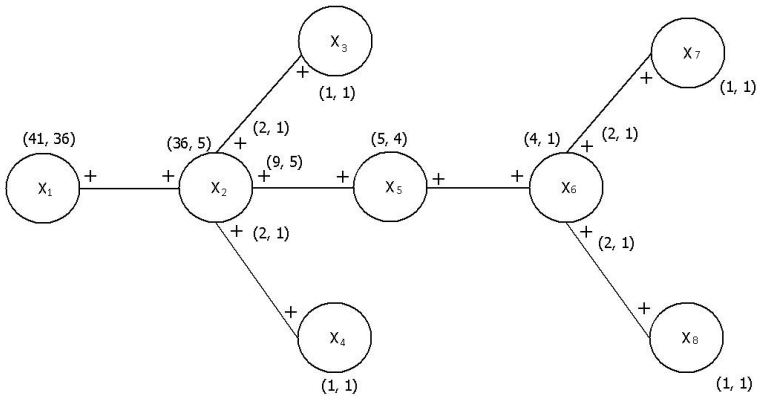

Figura 4.5: Conteo de modelos sobre un árbol.

Ejemplo 2. Sea $F = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_4, x_6), (x_6, x_7), (x_6, x_8)\}$ una función monótona en forma $2-CF$, el cálculo del número de modelos se realiza a través de una búsqueda en post-orden, para este ejemplo en específico se toma como el nodo raíz del árbol a x_1. El número de modelos en cada nivel del árbol es

mostrado en la figura 4.5.

El procedimiento de conteo de modelos para árboles finaliza con los siguientes resultados, para $\alpha_{x_1} = 41$, $\beta_{x_1} = 36$ y el número total de modelos es: $\#SAT(F) = 41 + 36 = 77$.

4.6. Conteo de modelos sobre grafos con ciclos

Sea G_F un ciclo simple con m nodos, esto es que todos las varibles en $v(F)$ se encuentran dos veces en F, $m = n = |V| = |E|$. Ordenando las cláusulas en F de tal manera que $|v(c_i) \cap v(c_{i+1})| = 1$ y $c_{i_2} = c_{i_1}$ siempre que $i \equiv j \bmod m$, por lo tanto, $x_1 = x_m$, entonces $F = \{c_i = \{x_{i-1}^{\epsilon_i}, x_i^{\delta_i}\}\}_{i=1}^{m}$ donde $\epsilon_i, \delta_i \in (1, 0)$.

Descomponiendo F como: $F = F' \cup c_m$ con $F' = \{c_1, ..., c_{m-1}\}$, por lo tanto F' es un camino y $c_m = (x_{m-1}^{\epsilon_m}, x_1^{\delta_m})$ es la arista que conforma junto con $G_{F'}$ el ciclo simple que consta de: $x_1, x_2, ..., x_{m-1}, x_1$. Al grafo $G_{F'}$ se le llamará camino interno del ciclo y a la arista c_m se le llamará la arista de retroceso.

El primer paso para calcular $\#SAT(F)$ es obtener el número de modelos de F' aplicando las reglas de recurrencia definidas en 4.1. Una vez que se han determinado el número de modelos para las variables en F', también se conoce los valores lógicos de las variables en SAT(F‘), incluyendo el de las variables x_{m-1} y x_1, cualquier modelo s de F' satisface a c_m si y solo si ($x_{m-1}^{1-\epsilon_m} \notin s$ y $x_m^{1-\delta_m} \notin s$), esto es $SAT(F' \cup c_m) \subseteq SAT(F')$, y $SAT(F' \cup c_m) = SAT(F') - \{s \in SAT(F') : s$ falsifica $c_m\}$. Sea $Y = F' \cup \{(x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})\}$, entonces $\#SAT(Y)$ es calculado como un camino con dos cláusulas unitarias sobre los extremos del camino, y entonces se cumple que:

$$\#SAT(F) = \#SAT(F' \wedge c_m) = \#SAT(F') - \#SAT(F' \wedge (x_{m-1}^{1-\epsilon_m}) \wedge (x_m^{1-\delta_m})) \quad (4.4)$$

Para contar el número de modelos en un grafo de restricciones con ciclos, se utilizan hilos. Un hilo computacional es una secuencia de pares (α_i, β_i), $i = 1, \ldots, m$ usada para calcular el número de modelos sobre un camino de m nodos. Un ciclo simple C requiere dos hilos computacionales. El hilo principal caracterizado por L_P el cual procesa el camino interno en el ciclo y el segundo hilo denotado por L_C el cual calcula $\#SAT(Y)$. Cuando el último nodo v_m sobre el ciclo es alcanzado, el último par (α_m, β_m) es calculado restando el par asociado a $\#SAT(Y)$ al último par sobre el hilo L_P, (véase la Figura 4.6).

Ejemplo 3. Sea $F = \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_5), (x_5, x_6), (x_6, x_1)\}$ una función monótona en forma $2 - CF$, el cual representa un ciclo: $G_F = (V, E)$. Sea $G' = (V, E')$ donde $E = E' \cup \{c_6\}$, es decir el nuevo grafo G' es F menos la arista c_6. Aplicando la ecuación 4.4, tenemos que $\#SAT(F) = \#SAT(F') - \#SAT(F' \wedge \overline{x_6} \wedge \overline{x_1}) = 21 - 3 = 18$. Este ejemplo es ilustrado en la figura 4.6.

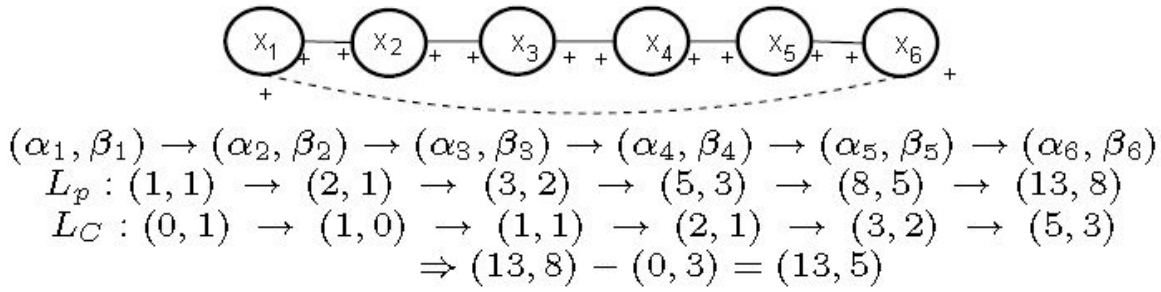

Figura 4.6: Cálculo de $\#SAT(F)$ cuando G_F es un ciclo

4.7. Procedimiento en tiempo polinomial para procesar grafos sin ciclos intersectados.

En la secciones anteriores se presentaron algoritmos de complejidad lineal en tiempo para calcular $\#2SAT(F)$ cuando el grafo de restricciones G_F de la fórmula

de entrada F no tiene ciclos, o bien, en caso de un ciclo simple.

Como procedimiento complementario se presenta a continuación un algoritmo en tiempo polinomial, basado en los procedimientos previos sobre procesamiento de caminos, árboles y ciclos independientes, para determinar $\#2SAT(F)$.

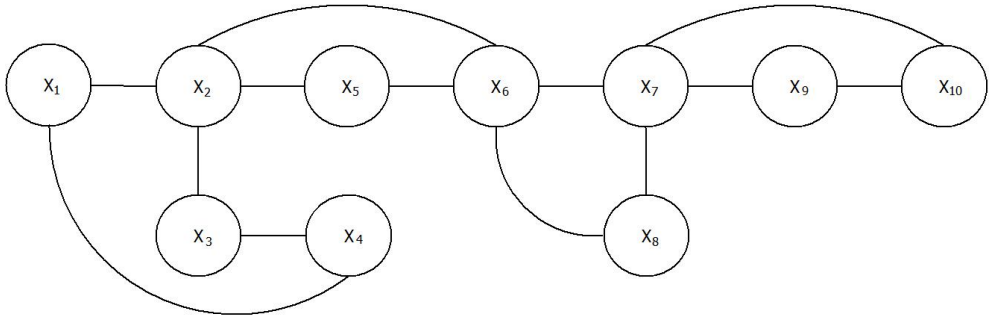

Figura 4.7: Grafo sin ciclos intersectados (Grafo Cactus)

Sea F una fórmula en $2-CF$ y $A_F = dfs(G_F)$ el grafo de búsqueda primero a lo profundo, formado por un árbol de expansión T_G y un conjunto C con los k ciclos fundamentales de A_F . Es importante resaltar que A_F es obtenido con un orden de complejidad de $O(n+m)$ sobre el tamaño de F, mientras que para determinar si existen ciclos intersectados en G_F, esto se puede realizar durante la dfs a través de un contador asociado a cada arista. El contador es incrementado en uno para cada arista que conforma un nuevo ciclo fundamental que es encontrado durante la dfs. Al final de la dfs, si todos los contadores son menores o iguales a uno, entonces G_F no tiene ciclos intersectados.

Dado el grafo G_F este es transformado a un Grafo Acíclico Dirigido (DAG), expresado por D_F, este grafo es construido asignando una orientación a cada arista $\{u, v\}$ en G_F dirigidas de la siguiente manera: $u \to v$ si v es un nodo antecesor de u en T_G.

Para un nodo v en D_F, sea $\delta_{in}(v) = |\{w : w \to v\}|$ y $\delta_{out}(v) = |\{w : v \to w\}|$ sus grados de entrada y salida. Aplicando un proceso de ordenamiento topológico sobre los nodos de D_F, se obtiene un número que indica el orden o asociado con cada nodo

en D_F, y que cumple que $o(u) < o(v)$ para todo $u \to v$.

En el grafo dirigido D_G se tienen dos tipos de estructuras básicas: Ramas y Anillos. Una rama B_s en G_F es una secuencia dirigida de nodos los cuales inician en un nodo v_s donde $\delta_{in}(v_s) = 0$ y $\delta_{out}(v_s) = 1$. Las ramas internas de la secuencia son alcanzables desde v_s a través de un camino formado por nodos v_i donde $\delta_{in}(v_i) = 1 = \delta_{out}(v_i)$. La rama termina en el primer nodo v_0 tal que $\delta_{in}(v_0) > 1$ o $\delta_{out}(v_0) > 1$, a este último nodo se le llama el nodo-final de la rama.

Un anillo R_s en D_F es un subgrafo formado por las aristas y los nodos que son parte de un ciclo fundamental en T_G. Por lo tanto, por cada ciclo $C_i \in C$, existe un anillo R_i, $i = 1, ..., k$. El nodo de inicio de R_s es el nodo v_s que es parte de una arista de retroceso $c_i = \{v_s, v_0\}$ y $\delta_{out}(v_s) = 2$, mientras que el segundo nodo v_0 de la arista de retroceso es llamado el nodo-final del anillo.

El orden topologico entre los nodos es extendido a ramas y anillos de D_G, de acuerdo con el número del orden topológico asociado con el nodo-final de cada subestructura. Este orden es una guía para el procesamiento de subestructuras de D_G. A continuación se muestra el algoritmo que describe el proceso de conteo de modelos sobre el grafo dirigido de una fórmula F en $2 - CF$, lo cual generaliza el conteo de modelos sobre cualquier topología de grafos que no contengan ciclos no intersectados.

Conteo de modelos (F)
Entrada: D_F: un grafo de restricciones dirigido.
Salida: El par (α_y, β_y) asociado con el nodo raíz y, donde $\#SAT(F) = \alpha_y + \beta_y$.

$\#SAT(F)$ es calculado recorriendo cada subestructura en D_F de acuerdo al orden topológico de cada subestructura. Un hilo principal: L_P es asociado con el árbol de expansión de D_F. L_P permanecerá activo hasta que *Conteo_de_Modelos* termina. Cuando un nodo v es visitado, el par(α_v, β_v) es calculado de acuerdo a la subestructura a la que pertenence el nodo, como se muestra a continuación:

Procedimiento:

1. Cuando una rama B_s es evaluada, el procedimiento para caminos es aplicado. La evaluación de la rama empieza en su nodo inicial v_s, asociando el par $(\alpha_s, \beta_s) = (1, 1)$ cuando v_s aún no tiene un valor almacenado. Despues de evaluar la rama, se eliminan el nodo inicial y los nodos internos de D_G, así como sus aristas adyacentes. El nodo-final v_0 es el único nodo de la rama que se conserva y que contendrá el par (α_0, β_0) que representa el valor final obtenido después de procesar la rama entera.

2. Cuando un anillo R_s es evaluado, se aplica el procedimiento básico para procesar ciclos simples. El procesamiento de R_s comienza con el nodo-inicio v_s y se crean dos hilos computacionales con los pares iniciales (α_s, β_s) y (α'_s, β'_s). Si v_s ya contiene un valor almacenado en (α_s, β_s) (v_s ha sido visitado antes) entonces $(\alpha'_s, \beta'_s) = (0, \beta_s)$ si la arista de retroceso tiene signo negativo en v_s, de lo contrario $(\alpha'_s, \beta'_s) = (\alpha_s, 0)$. Si v_s no ha sido visitado previamente entonces $(\alpha_s, \beta_s) = (1, 1)$ y $(\alpha'_s, \beta'_s) = (0, 1)$ si la arista de retroceso tiene signo negativo en v_s, de lo contrario $(\alpha'_s, \beta'_s) = (1, 0)$.

Después de evaluar el anillo R_s, tanto sus nodos como sus aristas incidentes son eliminadas de D_G, manteniendo solo el nodo-final v_0 de R_s. El par asociado a v_0 es $(\alpha_0, \beta_0 - \beta'_0)$ si la arista de retroceso tiene signo negativo en v_0, de lo contrario el par asociado a v_0 es $(\alpha_0 - \alpha'_0, \beta_0)$.

Cada vez que se considera un nodo $v \in V(D_F)$ si este ha sido visitado antes entonces un par $(\alpha_{v_1}, \beta_{v_1})$ ha sido previamente asociado con v. Cuando v es visitado nuevamente, un nuevo par $(\alpha_{v_2}, \beta_{v_2})$ es calculado. El par final (α_v, β_v) asociado con v es calculado como: $\alpha_v = \alpha_{v_1} * \alpha_{v_2}$ y $\beta_v = \beta_{v_1} * \beta_{v_2}$, donde ambas líneas de cálculo deben tener en común un mismo nodo incidente.

3. Los dos pasos previos son aplicados repetidas veces hasta que los nodos internos de anillos y ramificaciones son removidos de D_G. Es de notar que nuevas ramas podrián aparecer. El número de orden topológico asignado al nodo-final de las ramas determina el orden para el procesamiento de nuevas ramas emergentes.

4.Cuando el nodo raíz y de D_F es visitado, entonces se regresa el par (α_y, β_y).

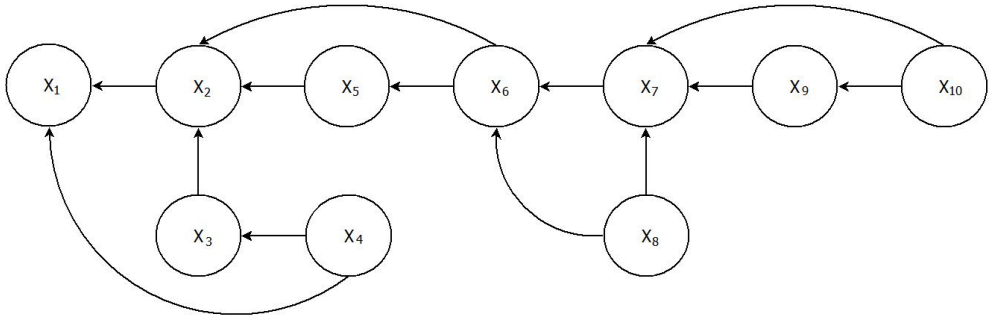

Figura 4.8: Grafo dirigido del grafo en Fig. 3.7

Ejemplo 3. Sea $F = \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_1), (x_2, x_5), (x_5, x_6), (x_6, x_2), (x_6, x_7), (x_7, x_8), (x_8, x_6), (x_7, x_9), (x_9, x_{10}), (x_{10}, x_7)\}$, el grafo asociado a esta fórmula se ilustra en la figura 4.7, de acuerdo con el algoritmo previamente definido para calcular el número de modelos de la fórmula, el primer paso consiste en construir el grafo de restricciones dirigido mostrado en la figura 4.8, una vez creado el grafo de restricciones dirigido D_F se realiza el cálculo del número de modelos de la fórmula de acuerdo a las estructuras existentes en D_F, lo que realiza el algoritmo **conteo de modelos**, y tal y como se muestra en la figura 4.9. Finalmente, el total de modelos de la fórmula se encuentra en el par asociado a la raíz: $(\alpha_1, \beta_1) = (65, 34)$ como se muestra en la figura 4.9. Entonces, $\#2\text{SAT}(F) = 65 + 34 = 99$ modelos.

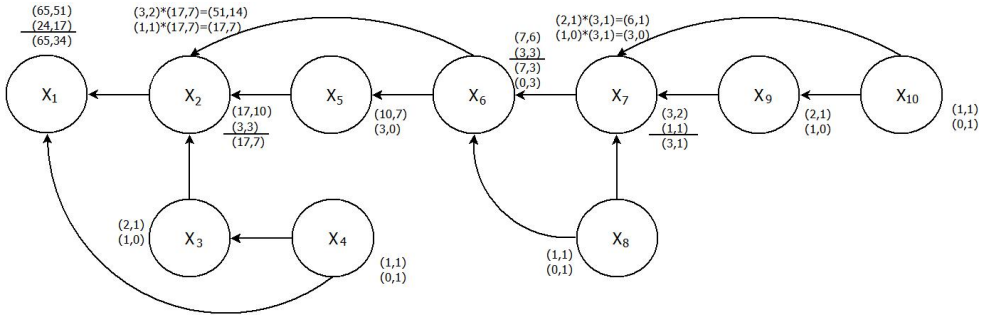

Figura 4.9: Conteo del número de modelos sobre el grafo

Capítulo 5

Una Aplicación de los grafos cactus

5.1. Grafos cactus.

Los elementos conocidos como grafos cactus hicieron su aparición en la literatura a inicios de los 50's con el nombre de árboles Husimi. Su estudio fue iniciado por los artículos de Husimi [20] y Riddel [24] que trataban con el tema de agrupamiento de integrales en la teoría de condensación en mecánica estática[31]. Además de la mecánica estática donde los árboles Husimi y su generalización es usada como modelos simplificados de rejas [23],[26], para este concepto se han encontrado aplicaciones en la teoría de redes eléctricas y de comunicación [37] y en química [19],[38].

Las propiedades matemáticas de los árboles Husimi fueron aclarados poco después de su aparición en una serie de artículos escritos por Harary, Uhlenbeck y Norman [18],[16]y resumidos veinte años después en la referencia "on graph enumeration" por Harary y Palmer[17]. Después los árboles Husimi fueron conocidos en la literatura matemática como grafos cactus. Desde hace más de diez años se ha tenido un gran interés en los grafos cactus dado que se demostró que algunos problemas de asignación de complejidad NP-hard pueden ser resueltos en tiempo polinomial para los grafos cactus [2],[36]. También recientemente se han estudiado ciertas variantes de una clase estrechamente relacionada llamada grafo cactus-bloque[4],[39].

Un grafo cactus en el cual todos sus bloques son ciclos de tamaño m es llamado cactus grafo m-uniforme. Un cactus hexagonal es un cactus 6-uniforme como se muestra en la Fig. 5.1. Un vértice v es llamado un vértice de corte si al quitar v y todas sus aristas incidentes, el resultado es un grafo no conexo.

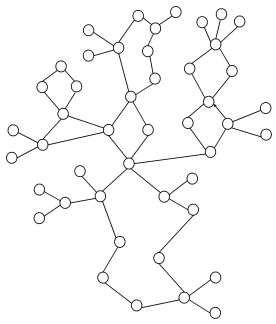

Figura 5.1: Grafo cactus.

5.2. Conteo de modelos sobre cadenas de polifenileno.

Un fenileno es cualquier radical aromático divalente obtenido de una molécula de benceno del cual son removidos dos átomos de hidrogeno. Cualquier número de polímeros en los cuales el bloque de construcción básico es un fenileno es llamado polifenileno. Los polifenilenos componen una importante clase de compuestos que pueden ser utilizados como precursores de algunos materiales científicamente y comercialmente interesantes, tal como lo es el óxido de polifenileno y sulfuro de polifenileno. Los polifenilenos no ramificados suelen ser utilizados en el contexto de conductores orgánicos de baja dimensión, mientras que su contraparte dendrimero mantiene un importante papel en la síntesis de grandes moléculas de grafeno.

Los polifenilenos comparten algunas similitudes estructurales con los compuestos bencenoides. A consecuencia de esta similitud ambas clases de compuestos pueden ser seguidos (y en algunos casos precedidos) por el estudio de los grafos bencenoides.

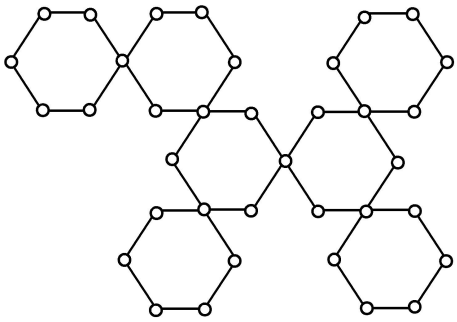

Figura 5.2: Cactus hexagonal.

Los grafos que representan compuestos polifenilenos continúan siendo en gran medida inexplorados, partiendo de esto nace la importancia de investigar y desarrollar los aspectos estructurales y enumerativos de algunos conceptos químicamente relevantes.

Un concepto importante a ser analizado en estos compuestos químicos, es el número de conjuntos independientes para los tres tipos de cadenas uniformes de polifenilenos. Un conjunto $S \subset V(G)$ de vértices de G es un conjunto independiente en G si para cualquier par de vértices $(u, v) \in S$, (u, v) no son adyacentes. En [29] se describen algunas características matemáticas de las cadenas de polifenilenos y algunos posibles desarrollos adicionales. Encontrar el número de conjuntos independientes de un grafo puede ser visto como un caso particular de conteo de modelos ($\#2 - SAT$), en el que todas las aristas del grafo tienen asociados signos positivos ("+") en los extremos delas aristas.

Un polifenileno es un grafo obtenido de un cactus hexagonal a través de la expansión de cada un de sus vértices de corte en una arista. El polifenileno correspondiente a la figura 5.2 es mostrado en la figura 5.3.

Un polifenileno en el cual ningún hexágono tiene más de dos vértices de corte es llamado cadena de polifenileno. Como es fácil de observar, una cadena de polifenileno contiene exactamente dos hexágonos con sólo un vértice de corte. Estos dos hexágonos son llamados vértices terminales y los restantes son llamados hexágonos intermedios. El número de hexágonos en una cadena de polifenilenos se le conoce

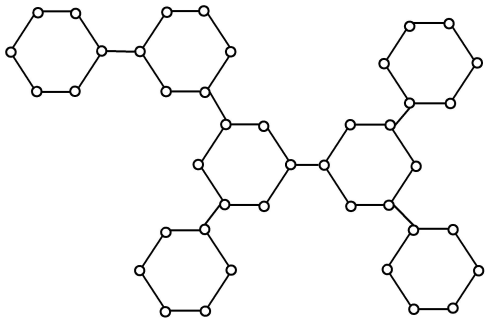

Figura 5.3: Polifenileno.

como su longitud. Una cadena de polifenileno de longitud seis se muestra en la figura 5.4.

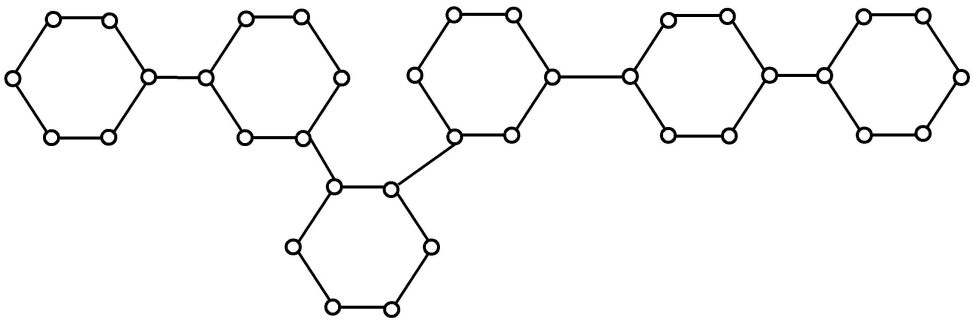

Figura 5.4: Cadena de polifenileno de longitud 6.

Un hexágono interno es llamado **ortho-hexágono**, **meta-hexágono** o **para-hexágono** si sus vértices de corte están a una distancia de 1, 2 o 3 aristas respectivamente. Esta terminología es muy importante en química debido a que es usada para describir la posición relativa de dos átomos en una molécula de benceno. Si todos los hexágonos en una cadena de polifenileno son ortho-hexagonos entonces la cadena es llamada ortho-cadena, la meta-cadena y la para-cadena son definidas de manera análoga.

El conteo de conjuntos independientes es una característica relevante sobre las cadenas ortho - cadenas, meta - cadenas y para - cadenas. El cálculo del número de modelos sobre los grafos que representan estas cadenas puede ser calculado en tiem-

po polinomial utilizando el algoritmo de conteo de modelos presentado en esta tesis. Esto es posible debido a que las topologías que son representadas en estos grafos se conforman por series de ciclos simples, lo que hace que su conteo de modelos (y por tanto, del número de conjuntos independientes) sea calculable en tiempo polinomial a través del algoritmo **Conteo de modelos** presentado en la sección anterior.

Ejemplo 3. Por ejemplo si tenemos la cadena de polifenileno de longitud 3 mostrada en la figura 5.5, y si se desea calcular el número de conjuntos independientes de esta cadena seria equivalente a calcular el número de modelos del grafo, el primer paso consiste en construir el grafo dirigido comenzando desde el vertice X_1 y posteriormente se calcula el total de modelos del grafo utilizando el algoritmo para procesar grafos sin ciclos intersectados como se muestra en la figura 5.6, obteniendo así en el vértice X_1 el par $(\alpha_1, \beta_1) = (3617, 1315)$, y el total de modelos del grafo que es equivalente al número de conjuntos independientes de la cadena de polifenileno es 4932.

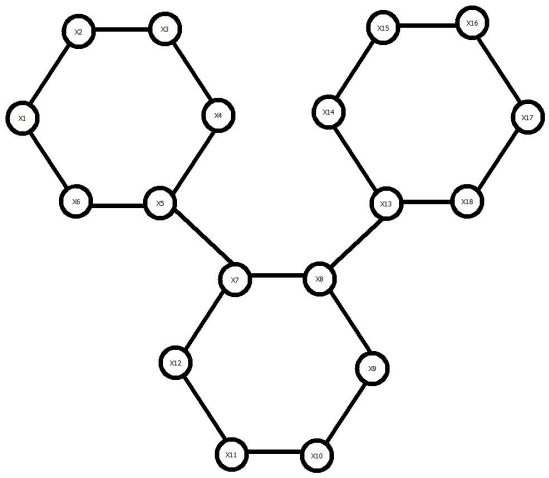

Figura 5.5: Cadena de polifenileno de longitud 3.

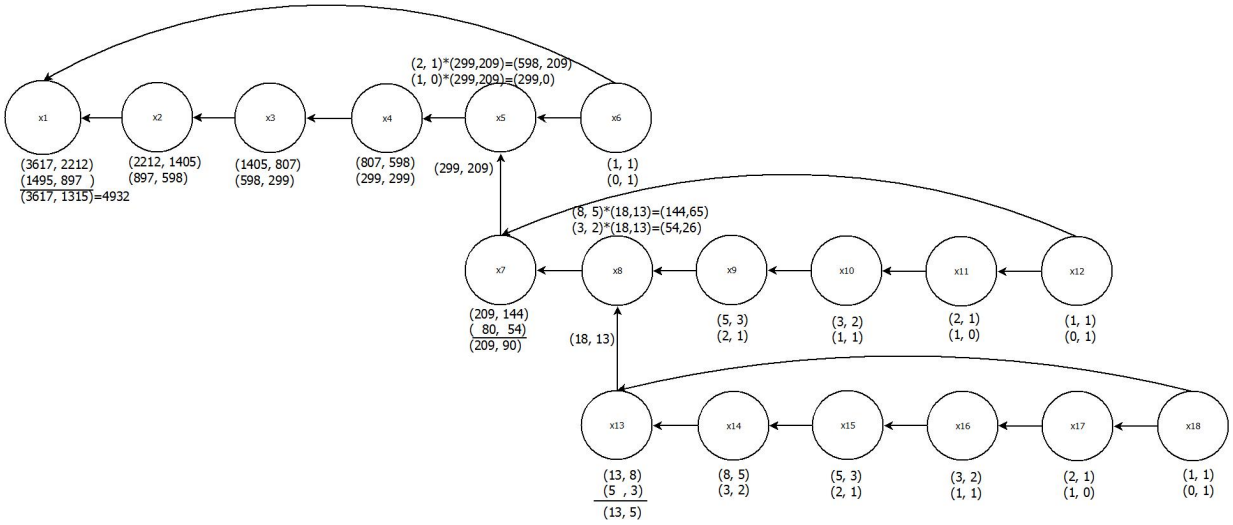

Figura 5.6: Cadena de polifenileno de longitud 3.

Capítulo 6

Conteo de modelos sobre grafos con ciclos anidados.

Sea G_F un grafo compuesto de n ciclos anidados es decir para todo ciclo $C_i \in G_F$ con $i \in \{1, .., n\}$ y $V_i = \{v \in C_i : v \in G_F\}$, si $V_i \cap V_{i+1} \neq \emptyset \Rightarrow V_{i+1} \subset V_i$, y diremos que el ciclo C_{i+1} esta anidado en el ciclo C_i. Presentamos en este capítulo un método para contar el número de modelos $\#SAT(G_F)$ cuando G_F es un grafo con la topología que presenta un conjunto de ciclos anidados.

Si el grafo G_F tiene ciclos anidados, entonces consideraremos que hay tres tipos de vértices en G_F. El primer tipo de vértice es el que representa el inicio de uno o más ciclos anidados, este tipo de vértices es de grado $\delta(v_i) \geq 2$. Si en el grafo hay un solo vértice con grado dos significa que es el vértice de inicio de un ciclo. Otro tipo de vértice en G_F es aquel que constituye el fin de uno o más ciclos, de forma similar al vértice de inicio de ciclo, estos vértices son de grado $\delta(v_f) \geq 2$. Puede haber en G_F un solo vértice de fin de ciclo con grado dos. El resto de vértices pertenecen a una tercera clasificación v_c que se llamarán vértice camino, el grado de estos vértices es $\delta(v_c) = 2$, y en G_F puede haber cualquier cantidad de ellos.

Determinar el número de modelos sobre G_F puede iniciarse con un recorrido en profundidad del grafo tomando el vértice de menor índice como inicial. Se debe ir encontrando el nodo inicial y final de cada ciclo en G_F, además de la arista de retro-

ceso $e_{c_i} \in E$ para cada ciclo $C_i \in G_F$. Si se eliminan todas las aristas de retroceso de G_F el grafo obtenido G' es un camino simple, esto es debido a que G_F consiste únicamente de ciclos anidados. El algoritmo para calcular el número de modelos sobre G_F se describe a continuación:

El número de modelos se calcula mientras se realiza un recorrido sobre el grafo G_F siguiendo el camino de aristas que conforma G_F, es decir recorriendo solo aquellas aristas que no son de retroceso.

En el siguiente algoritmo nuevamente se hace uso de hilos computacionales. El número de estos hilos se incrementa o disminuye de acuerdo a la cantidad de ciclos anidados, como se describe a continuación.

Conteo de modelos sobre ciclos anidados
Entrada: G_F: un grafo con ciclos anidados.
Salida: (α_y, β_y) asociado al nodo final y de G_F, y tal que $\#SAT(F) = \alpha_y + \beta_y$.

$\#SAT(F)$ es calculado mientras se recorre el camino simple formado al ignorar las aristas de retroceso que hay en G_F. Como en los procedimientos anteriores, al vértice inicial de G_F se le asocia el par $(\alpha_1, \beta_1) = (1, 1)$. En este caso, este par de números (α_1, β_1) es asignado al hilo principal representado por L_1. Debido a la naturaleza de ciclos anidados, será necesario incrementar y disminuir la cantidad de hilos mientras se recorre el grafo.

Para cada hilo L_i activo en el vértice que se visita, se irá aplicando las reglas de recurrencia definidas en 4.1, y esto se aplica antes de incrementar o disminuir el número de hilos activos. Al terminar el recorrido a lo profundo sobre G_F, el hilo L_1 tendrá el último par (α_y, β_y) que ha sido calculado y que representará el valor para $\#SAT(F) = \alpha_y + \beta_y$.

Procedimiento:

1. El proceso de conteo se realiza en forma lineal sobre el camino interno de los ciclos, viajando desde el nodo del ciclo más externo hacia los nodos del ciclo más

interno.

2. Si un vértice $v \in G_F$ es el inicio de uno o más ciclos, en este caso se incrementará el número de hilos activos. Si al visitar el nodo v se tienen n hilos de computación activos, entonces se duplica el número actual de hilos, formándose así $2 \cdot n$ hilos activos. Además, si v marca el inicio de m nuevos ciclos que inician en v, entonces el total de ciclos activos será $n \cdot 2^m$.

3. Los pares iniciales asociados a cada nuevo hilo de computación dependerá del signo que tienen la arista de retroceso en el nodo v. Si la arista de retroceso que incide en v tiene signo positivo entonces para cada hilo activo L_i se crea un nuevo hilo $L_(n+i)$ y su par asociado es $(0, \beta)$ siendo β el valor que aparece en el par asociado $(\alpha, \beta) \in L_i$. En caso contrario, si la arista de retroceso que incide en v tiene signo negativo para cada hilo activo L_i se crea un nuevo hilo $L_(n+i)$ con el par asociado $(\alpha, 0)$, siendo α el valor que aparece en el par asociado $(\alpha, \beta) \in L_i$.

4. Si el vértice v marca el fin de un ciclo, en este caso se reduce el número de hilos activos a la mitad. Por cada ciclo que se cierra se divide a la mitad el número de hilos activos. Así, si hay n hilos activos y el vértice v marca el fin de m ciclos, entonces quedarán después de visitar v, $\frac{n}{2^m}$ hilos de computación activos.

5. Antes de eliminar hilos de computación, se realiza una resta entre los valores de los hilos de acuerdo al signo de la arista de retroceso que incide en v. Si el signo de la arista de retroceso que incide en v tiene signo positivo, entonces para cada hilo L_i con $1 <= i <= n/2$, donde n es el número de hilos activos, se le asigna un nuevo par de números a $L_i = (\alpha_i, \beta_i - \beta_{n/2+i})$, donde (α_i, β_i) es el par asociado a L_i y $\beta_{n/2+i} \in L_{n/2+i}$. En caso de que la arista de retroceso que incide en v tenga signo negativo entonces $L_i = (\alpha_i - \alpha_{n/2+i}, \beta_i)$, con $1 <= i <= n/2$.

Ejemplo 5. Sea $F = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_3, x_8\}, \{x_3, \bar{x_8}\}, \{\bar{x_3}, x_7\}, \{x_4, x_5\}, \{x_5, x_7\}, \{x_5, x_6\}, \{x_6, x_7\}, \{x_7, x_8\}, \{\bar{x_1}, \bar{x_8}\}\}$ una fórmula en $2 - CF$, la cual representa un grafo G_F compuesto de 5 ciclos anidados. Para calcular el

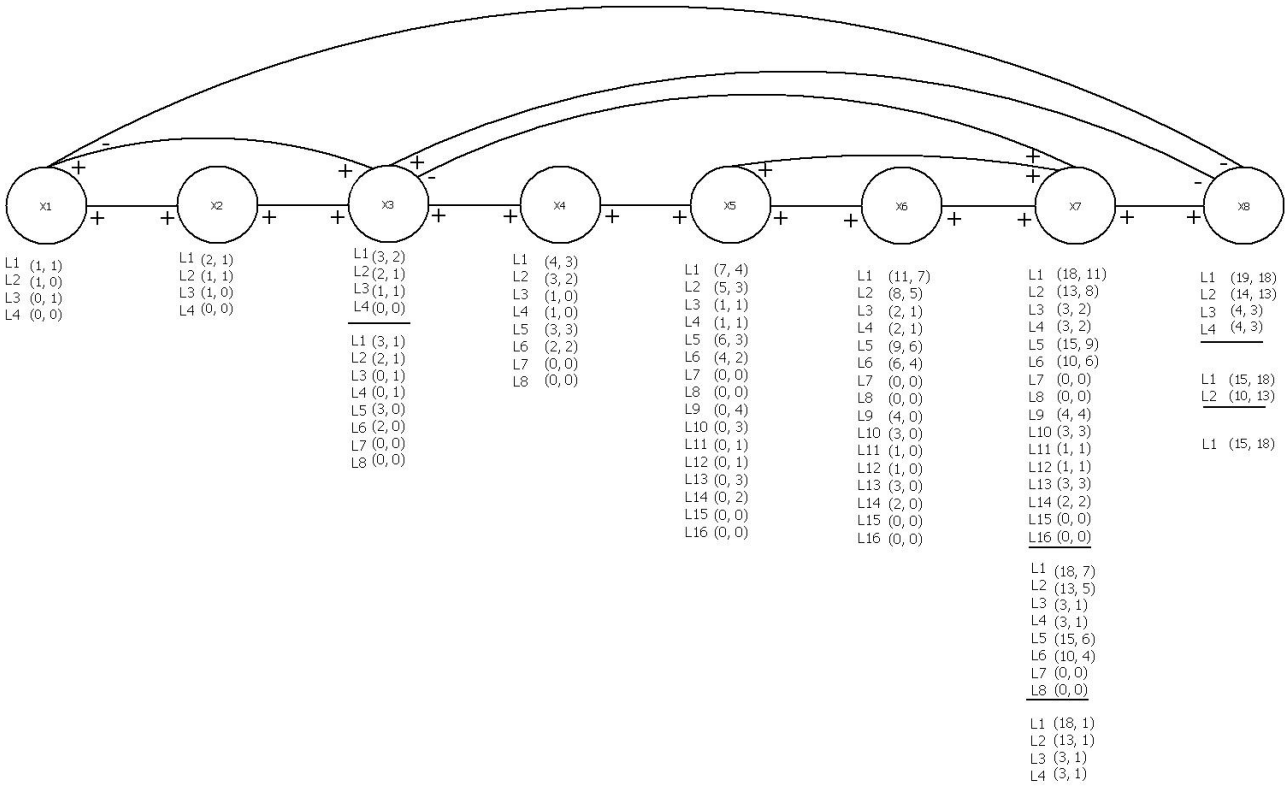

Figura 6.1: Cálculo de #SAT sobre ciclos anidados

número de modelos sobre G_F se recorre el camino que se forma al ignorar las aristas de retroceso, como se ilustra en la figura 6.1. Se comienza el recorrido de G_F en x_1 y se concluye en x_8, en este último vértice después de realizar los cálculos necesarios la suma del par $(\alpha_8, \beta_8) = (15, 18)$ en L_1 representa el total de modelos para G_F, es decir $\#SAT(F) = 15 + 18 = 33$.

6.1. Procesamiento de ciclos anidados utilizando macros.

Calcular el total de modelos sobre un grafo con ciclos anidados utilizando el método anteriormente descrito, implica un número exponencial con respecto al número de hilos de computación que se requieren. Esto nos determina a su vez, una complejidad exponencial con respecto al número de ciclos anidados que hay en el grafo.

Aunque el algoritmo obtiene una solución al problema, es conveniente redefinir el método para reducir su complejidad en tiempo. A partir de estas topologías de ciclos anidados, se desarrolló un nuevo algoritmo que a través de la aplicación de macros, calcula $\#2SAT$.

Un macro es una abreviatura del término "macroinstrucción", un macro es usado para describir un conjunto de instrucciones que pueden ser ejecutadas en cuanto se obtienen los valores que representan las variables que son usadas dentro del macro.

Dado un grafo G_F compuesto por ciclos intersectados, sean C_i y C_j dos ciclos que pertenecen a G_F, definir si C_i se encuentra anidado dentro de C_j en algunos casos puede parecer sencillo y en otros no, por lo tanto es importante definir las características formales que definen si dos ciclos se encuentran anidados.

Con respecto a dos ciclos C_i y C_j, podemos decir que el ciclo C_i se encuentra anidado en C_j si cumple las siguientes condiciones:

a) $V(C_i) \subset V(C_j)$: El conjunto de vértices de C_i es un subconjunto de los vértices de C_j.
b) $|E(C_i) - E(C_j)| = 1$: Existe una sola arista de C_i que no pertenece a C_j.
c) $E(C_i) \oplus E(C_j) = E(C_k)$: Donde C_k es un nuevo ciclo diferente a C_i y C_j y $\oplus$ representa la operación XOR entre las aristas de los ciclos C_i y C_j.

Para el nuevo algoritmo, el grafo G_F compuesto por ciclos anidados se recorrerá en base a los ciclos que lo componen, y se iniciará a partir del ciclo más interno (más embebido) del grafo.

Consideremos que se tiene un conjunto $D = \{C_1, C_2, ..., C_k\}$ compuesto de los ciclos anidados de G_F, y que los elementos de D se han ordenado de tal forma, que para todo $i \neq j, j > i$, entonces C_i o esta embebido en C_j, o ambos ciclos son independientes. El cálculo de $\#2SAT$ se realizará recorriendo los nodos que componen

el ciclo más interno y hacia los nodos del ciclo más externo.

Mientras se realiza el recorrido en profundidad sobre cada ciclo en G_F, se usarán dos o tres hilos computacionales para llevar la cuenta del número de modelos asociado a cada vértice que es visitado. Así, cada hilo computacional contiene un par de variables (y en algunas casos, un macro determinado por un par de ecuaciones lineales) (α, β), usadas para calcular el número de modelos sobre los vértices que componen el camino interno de un ciclo.

Durante el algoritmo se mantendrá siempre activo un hilo principal, denotado por L_p. Al visitar el nodo final del ciclo más externo del grafo, en el hilo principal se tendrá el macro definitivo para obtener el valor de $\#2SAT(G_F)$. El procedimiento para calcular $\#2SAT$ de G_F se realiza mediante el siguiente algoritmo.

Procedimiento
Entrada: G_F un grafo con ciclos anidados.
Salida: El par (α_f, β_f) asociado al nodo final v_f, y tal que $\#SAT(F) = \alpha_f + \beta_f$.

Mientras $|D| > 0$, se toma el ciclo de menor indice $C_i \in D$, el inicio del procesamiento de cada ciclo se realiza mediante un recorrido en profundidad, visitando los vértices del camino simple interno que compone al ciclo, desde su nodo inicial v_0 hasta su nodo final v_f. Durante este recorrido, un nodo del camino puede tener asociado, o un par de valores enteros o un macro, para cada caso, se realizan los siguientes procedimientos:

1. Si v_x es el vértice inicial de uno o más ciclos: En este caso se crea un hilo L_1, se asignan las variables (α, β) al hilo computacional L_1. Una característica que es necesario considerar son las aristas de retroceso que inciden en v_0, para esta situación tenemos dos casos:

    a) Si en v_x incide una arista de retroceso: en este caso se crea un hilo secundario L_2 con las variables $(0, \beta)$ en caso de que el signo de la arista de retroceso sobre v_x sea negativo y $(\alpha, 0)$ en caso de que el signo sobre la arista de retroceso que incide sobre v_x sea positivo.

b) Si en v_x inciden dos o más aristas de retroceso: En este caso si todas las aristas de retroceso que inciden en v_x tienen el mismo signo entonces se crea un hilo secundario L_2 con el par asociado $(0, \beta)$ en caso de que los signos de las aristas de retroceso que inciden en v_x sean negativas, en caso contrario se asigna el par $(\alpha, 0)$. Si los signos de las aristas de retroceso que inciden en v_x son tanto signos positivos como negativos, entonces se crean dos hilos computacionales secundarios L_2 y L_3 con los pares asociados: $(0, \beta)$ y $(\alpha, 0)$, respectivamente.

2. Si v_x es un vértice del camino interno de C_i: En este caso se aplican las reglas de recurrencia definidas en 4.1 sobre el par $(\alpha_{x-1}, \beta_{x-1})$, y esta operación se aplica para cada uno de los hilos activos, obteniendo así un nuevo par (α_x, β_x) en cada uno de estos hilos.

3. Si v_x tiene asociado un macro: En este caso primero se aplica la regla de recurrencia 4.1 para cada hilo activo como se describió en el punto anterior. Para cada hilo activo se procesan los pares de ecuaciones descritas en el par asociado a v_x, obteniéndo así un nuevo par de de ecuaciones lineales, que denotaremos por (α_x, β_x). Este nuevo par de ecuaciones serán usadas posteriormente cuando se visite el siguiente vértice del ciclo.

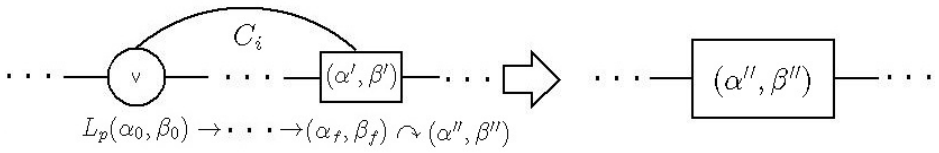

Figura 6.2: Transformación de ciclo por un macro.

Si el nodo visitado involucra el nodo final de un ciclo C_i, entonces de acuerdo al signo de la arista de retroceso que incide en el nodo, se resta el par $(0, \beta_x) \in L_2$ al par $(\alpha_x, \beta_x) \in L_p$. En el caso de que la arista de retroceso tenga signo negativo, entonces se resta el par $(\alpha_x, 0) \in L_2$ al par del hilo principal L_p, obteniéndose así un nuevo par de funciones (α_x, β_x). Y dado que se recorrió por completo el ciclo C_i, este se elimina de D y el ciclo es sustituido en G_F por

un nodo compuesto que tendrá asociado el par (α_x, β_x) que representa un nuevo macro a ser utilizado al procesar el ciclo que contenia a C_i. Finalmente se regresa al paso inicial para seleccionar el nuevo ciclo más interno que se encuentra en D.

4. Si v_x es un vértice de fin de ciclo: Al igual que en cualquier vértice que se visita, se aplica la regla de recurrencia 4.1 para cada hilo activo, si únicamente existen dos hilos activos en v_f entonces se resta $(0, \beta) \in L_j$ al par asociado al hilo principal L_p en caso de que el signo de la arista de retroceso que incide en v_f sea positivo, en caso contrario se resta $(\alpha, 0) \in L_j$ al hilo principal L_p.

   Sea $S = \{s_1, \ldots, s_m\}$ el conjunto de signos de las aristas de retroceso que inciden sobre v_0, ordenados desde la arista del ciclo más interno hacia los ciclos que los van conteniendo, donde m es el número de signos $m = |S|$.

   En caso de que existan solo dos hilos activos entonces $j = 2$, en la circunstancia en que existan tres hilos activos si el signo s_1 es negativo entonces $j = 3$, si por el contrario s_1 es positivo entonces $j = 2$.

   Dado que v_j es el nodo final del ciclo C_i, si en v_j incide únicamente una arista de retroceso entonces se elimina el ciclo y es sustituido por un nodo compuesto que tendrá asociado un par $(\alpha, \beta) \in L_p$ representando un macro. En el caso de que existan más de una arista de retroceso sobre v_j entonces $(\alpha_j, \beta_j) = (\alpha_j, 0)$ si el signo de la arista de retroceso sobre v_f es positivo en caso contrario $(\alpha_j, \beta_j) = (0, \beta_j)$ donde $(\alpha_j, \beta_j) \in L_j$, si $s_1 \neq s_i \forall i > 1$ entonces el hilo secundario L_j es descartado y el hilo secundario restante tendrá el índice dos, de igual modo en C_i se elimina la arista de retroceso al igual que el signo $s_1 \in S$, ahora v_f se denota como un vértice intermedio de C_{i+1}, continuando así el procedimiento desde el vértice v_x sobre el camino interno de C_{i+1} el cual es el ciclo más interno, por lo tanto se incrementa el indice i y $C_i = C_{i+1}$.

Ejemplo 6 Sea $F = \{(x_1, \overline{x_2}), (\overline{x_2}, \overline{x_3}), (\overline{x_3}, x_4), (x_4, x_5), (x_5, \overline{x_6}), (x_6, x_7), (x_7, x_8), (x_8, x_9), (x_1, \overline{x_9}), (\overline{x_2}, x_8), (\overline{x_3}, x_8), (x_4, x_7), (\overline{x_4}, x_6)\}$ una función en forma 2-CF, el

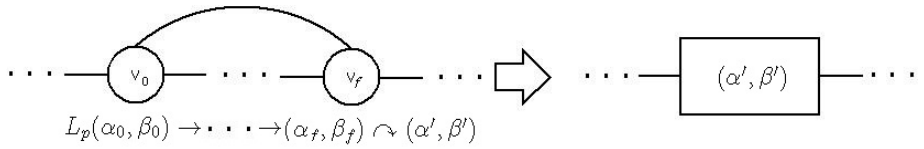

Figura 6.3: Cambio de un ciclo por un macro.

grafo asociado G_F está compuesto únicamente por ciclos anidados donde todos los ciclos pueden ser ordenados en una tupla $D = \{C1, ..., Ck\}$ donde C_i es un ciclo anidado dentro de C_{i+1} con $i = 1, ..., k-1$. Utilizando el algoritmo anteriormente descrito al recorrer el grafo desde su ciclo más interno después de procesar el vértice x_8 en el hilo principal L_p encontraremos el par de funciones $(\alpha, \beta) = (13\alpha, 10\alpha+6\beta)$, este par de funciones son sustituidos por los valores iniciales $\alpha = 1$ y $\beta = 1$, finalmente tenemos que $(\alpha, \beta) = (13, 16)$ por lo tanto $\#SAT(F) = 13 + 16 = 29$. Este ejemplo se ilustra en la figuras 6.4 - 6.7.

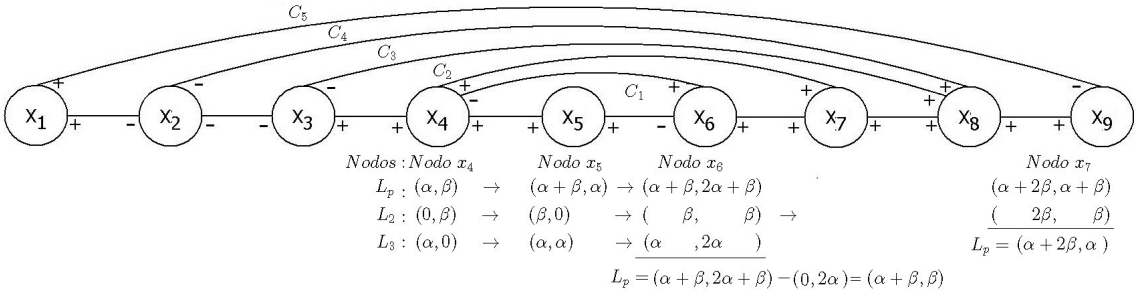

Figura 6.4: Procesando los ciclos mas internos C_1 y C_2.

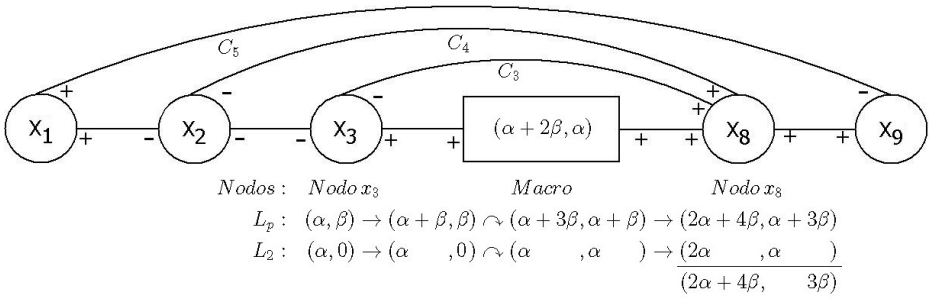

Figura 6.5: Procesando ciclo C_3 que es sustituido por un macro.

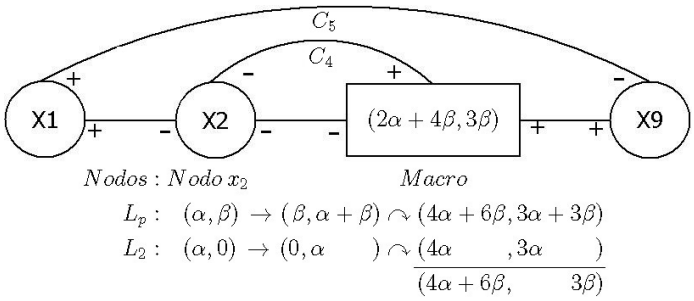

Figura 6.6: Procesando ciclo C_4 que es sustituido por un macro.

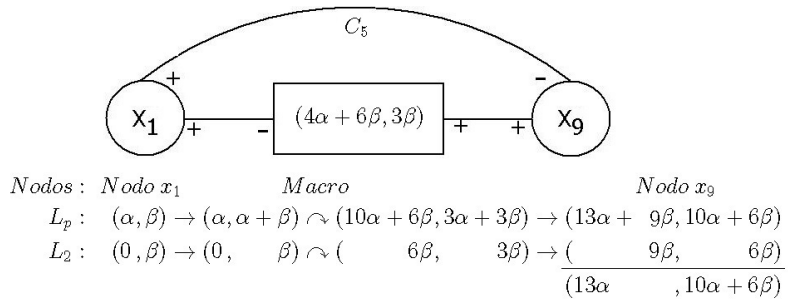

Figura 6.7: Procesando ciclo final y más externo C_5.

El algoritmo propuesto como se explicó trabaja sobre aquellos ciclos que pueden ser ordenados en una tupla $D = \{C1, ..., Ck\}$ donde C_i es un ciclo anidado dentro de C_{i+1} con $i = 1, ..., k-1$, sin embargo para los grafos con ciclos anidados que no pueden ser agrupados de esta manera, en este circunstancia se aplica el algoritmo independientemente a cada conjunto de ciclos que cumplen las condiciones adecuadas, este proceso genera un nuevo grafo donde varios ciclos son sustituidos por macros, el nuevo grafo generado es analizado con el mismo principio que continuara reduciendo el grafo hasta que el grafo completo pueda ser analizado mediante el algoritmo, a continuación se presenta un ejemplo que ilustra lo anteriormente mencionado.

Ejemplo 7 Sea $F = \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_5), (x_5, x_6), (x_1, x_6), (x_1, x_4), (x_2, x_4), (x_4, x_6)\}$ cuyo grafo G_F asociado se ilustra en la figura 6.8 como se puede observar el ciclo C_2 no es un ciclo anidado de C_3 ni viceversa, sin embargo ambos ciclos se encuentran anidados dentro de C_4 en este caso se procesa el conjunto de ciclos C_1 y C_2 por separado del ciclo C_3, el resultado de este procedimiento es un

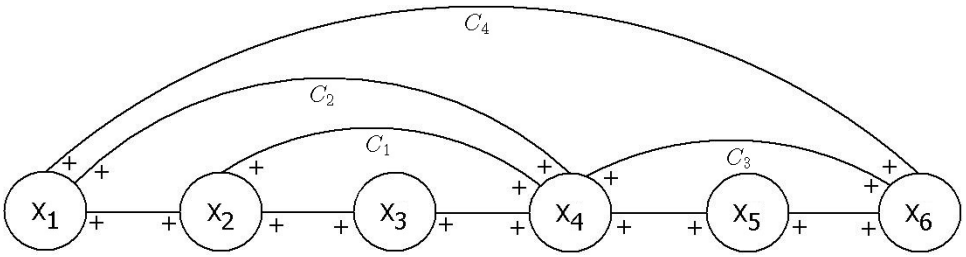

Figura 6.8: Grafo con ciclos anidados.

nuevo grafo compuesto por dos macros unidos por una arista de retroceso que puede ser procesado fácilmente como se muestra en la figura 6.9.

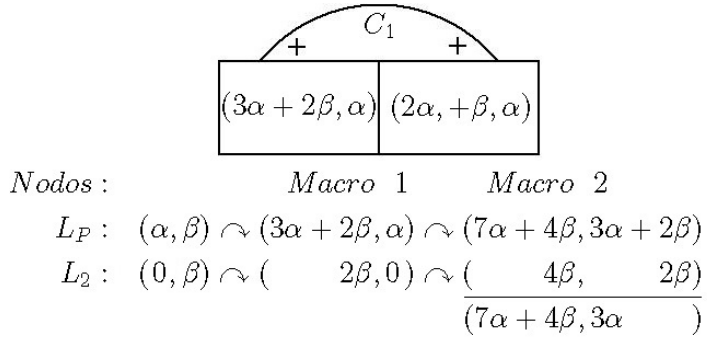

Figura 6.9: Reducción de grafo con ciclos anidados.

Al aplicar el algoritmo propuesto varias veces para reducir el grafo, generaliza el proceso para calcular el número de modelos sobre todo tipo de grafo con ciclos anidados manteniendo una complejidad lineal, esto es dado que el número de hilos computacionales simultáneos utilizados en ningún momento es mayor a tres, manteniendo así una complejidad lineal de $O(3m + n)$, donde m es el número de vértices y n el número de aristas del grafo asociado a la fórmula en $2 - FC$.

Capítulo 7

Conclusión

El conteo de modelos de 2-FC's (problema $\#2SAT$) es un tema relevante en el área de la Inteligencia Artificial. Las propuestas de su solución han sido abordadas de distintas formas, usando por ejemplo; algoritmos de aproximación, y algoritmos exactos que tienen diversos ordenes de complejidad en tiempo.

En este trabajo de tesis, para realizar el conteo de modelos, se diseñaron métodos basados en el recorrido en profundidad del grafo de restricciones definido por una $2-FC$, y al mismo tiempo que se hace el recorrido se van aplicando reglas de recurrencia que permiten llevar la cuenta de modelos, el resultado de este conteo se asocia a cada nodo del grafo. Se demostró también que hay diversas instancias de 2-FC's en las que $\#2SAT$ puede ser resuelto eficientemente.

En la primera parte del documento, se revisan las topologías más simples sobre el grafo de restricciones de la 2-FC que permiten el conteo eficiente de los modelos de la fórmula. Entre estas topologías, se presenta el caso de caminos y ciclos simples, y se extiende el caso de caminos simples para considerar árboles (grafos acíclicos). En este documento, se presentan algoritmos de complejidad lineal en tiempo para procesar todos los casos anteriores.

Entre el tipo de grafos que pueden pre-procesarse en tiempo polinomial para el cálculo de #2SAT, se encuentran grafos con aristas múltiples. Se termina la exposi-

ción de algoritmos presentando el caso del conteo de modelos sobre grafos con ciclos anidados, mostrando que también en este caso, hay un algoritmo de complejidad polinomial en tiempo que calcula #2SAT.

Además de definir un conjunto de algoritmos eficientes para el procesamiento de diversas topologías de los grafos, se muestran dos aplicaciones en las que el conteo de modelos juega un papel indispensable, estas aplicaciones tienen importancia en el análisis químico de compuestos, lo que demuestra que se puede desarrollar una gran variedad de aplicaciones dentro del área de inteligencia artificial, y que esta área se encuentra en constante innovación y desarrollo.

Al considerar un grafo general, que no cumpla ninguna de las topologías anteriormente mencionadas, se propone un procedimiento para procesar grafos compuestos por ciclos no simples. Al inicio del capítulo 5, se presenta un algoritmo de complejidad exponencial en tiempo con respecto al núumero de ciclos anidados para procesar este tipo de grafos.

7.1. Trabajo a futuro

Dentro del trabajo a futuro que puede extender este trabajo de tesis, se encuentra el de diseñar algoritmos para contar modelos sobre grafos con ciclos intersectados, así como determinar en cuales casos de este tipo de topologías, existen algoritmo eficientes para el conteo de modelos.

Otro aspecto de interés a considerar como trabajo a futuro, es el relacionado con el uso de las técnicas de análisis utilizadas en este trabajo, para aplicaciones específicas, o bien como parte de un sistema más complejo que involucre el conteo de modelos.

Es importante destacar que las aplicaciones que puede tener la teoría de conteo de modelos se puede diversificar, dado que el problema es un problema #P-completo,

y por tanto, hay una gran cantidad de problemas de conteo relacionados con el problema #2SAT, como puede ser el conteo de conjuntos independientes, conteo de cubiertas de vértices, conteo de coloreo de vértices en un grafo, etc. Aparte de que cada uno de los anteriores problemas relacionados, tiene a su vez, una buena cantidad de aplicaciones potenciales.

Bibliografía

[1] R. Bayardo y J. Pehoushek. Counting models using connected components. *Proceeding of the seventeenth National Conf. on Artificial Intelligence*, págs. 157–162, 2000.

[2] B. Ben-Moshe y B. Bhattacharya. Efficient algorithms for the weighted 2-center problem in cactus graph. *Lecture Notes in Computer Science*, págs. 693–703, 2005.

[3] Elaza Birnbaum y Eliezer Lozinskii. The good old davis putnam procedure helps counting models. *Revista de Investigacin en Inteligencia Artificial*, págs. 10:457–477, 1999.

[4] G. Chang, C. Chen, y Y. Chen. Vertex and tree arvoricities of graphs. *Journal of Combinatorial Optimization*, págs. 295–306, 2004.

[5] SA Cook. The complexity of theorem-proving procedures. *Conference Record of Third Annual ACM Symposium on Theory of Computing*, págs. 151–158, 1971.

[6] Vilhelm DahllÄof y Peter Jonsson. An algorithm for counting maximum weighted independent sets and its applications. *Annual ACM-SIAM Symposium on Discrete Algorithms*, págs. 292–298, 2002.

[7] Alvaro del Val. On 2-sat and renamable horn. *National Conference on Artificial Intelligence*, págs. 279–284, 2000.

[8] Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, págs. 81(1):49–64, 1991.

[9] S. Dalmao F. Bacchus y T. Pitassi. Dpll with caching a new algorithm for #sat and bayesian inference. *ECCC, Dep. of CS, U. of Toronto.*

[10] De Ita G. y Guillén C. Efficient computation of the degree of belief for a subclass of two conjunctive forms. *Inteligencia Artificial*, págs. 14(48):15–27, 2010.

[11] De Ita G. y Morales Luna G. *Propuestas algoritmicas en el tratamiento de los problemas de satisfactibiliad.* Departamento de Ingeniera Elctrica CINVESTAV-IPN, 1996.

[12] De Ita G, Bello P, y Contreras M. New polynomial classes for #2sat established via graph topological structure. *Engineering Letters*, págs. 15:4250–258, 2007.

[13] Michael Garey y David Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, 1984.

[14] Leslie Ann Goldberg y Mark Jerrum. Counting unlabelled subtrees of a tree is #p-complete. *LMS Journal of Computation and Mathematics*, págs. 3:117–124, 2000.

[15] Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Computational Complexity*, págs. 9(1):52–72, 2000.

[16] F. Harary y R. Z. Norman. The dissimilarity characteristic of husimi trees. *Annals of Mathematics6*, págs. 134–141, 1953.

[17] F. Harary y E. M. Palmer. Graphical enumeration. *Academic Press*, 1973.

[18] F. Harary y G. E. Uhlenbeck. On the number of husimi trees. *IEEE International Symposium on Circuits and Systems*, pág. 315322, 1953.

[19] H. Hosoya y K. Balasubramanian. Exact dimer statistics and characteristic polynomials of cacti lattices. *Theorical Chemistry Accounts.*, págs. 315–329, 1989.

[20] K. Husimi. Note on mayers'theory of cluster integrals. *The Journal of Chemical Physics*, págs. 682–684, 1950.

[21] Harry Hunt III, Madhav Marathe, Venkatesh Radhakrishnan, y Richard Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, págs. 27(4):1142–1167, 1998.

[22] Dexter Kozen. *The design and analysis of algorithms*. Springer-Verlag, 1992.

[23] J. L. Monroe. The bilayer ising model and a generalized husimi tree approximation. *Physica A: Statical Mechanics and its Applications*, págs. 563–576, 2004.

[24] R. J. Riddell. *Contributions to the theory of condensation*. Tesis Doctoral, University of Michigan, 1951.

[25] Herbert John Ryser. *Combinatorial Mathematics*. The Mathematical Association of America, 1963.

[26] P. Serra, J. F. Stilck, W. L. Cavalcanti, y K. D. Machado. Polymers with attractive interaction on the husimi lattice. *Proceeding of the seventeenth National Conf. on Artificial Intelligence*, págs. 157–162, 2004.

[27] Edith Spaan, Leen Torenvliet, y Peter van Emde Boas. Nondeterminism, fairness and a fundamental analogy. *Bulletin of the EATCS*, pág. 37:186, 1989.

[28] Szeider Stefan. On fixed-parameter tractable parameterizations of sat. *LNCS 2919*, págs. 188–202, 2004.

[29] D. Tomislav y S. L. Marie. Matching an independent sets in polyphenylene chains. *Communications in Mathematical and in Computer Chemistry*, págs. 313–330, 2011.

[30] Alan Turing. On computable numbers, with an application to the entscheidungs problem. *Proceedings of London Mathematical Society*, págs. 230–265, 1963.

[31] G. E. Uhlenbeck. Lectures in statical mechanics. *Journal of Applied Mathematics and Mechanics*, 1956.

[32] Salil Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, págs. 31(2):398–427, 2001.

[33] Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, pág. 8:189, 1979.

[34] Leslie Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, págs. 279–284, 2000.

[35] Wenhui Zhang. Number of models and satisfability of sets of clauses. *Theoretical Computer Science*, págs. 155(1):277–288, 1996.

[36] B. Zmazek. The obnoxious center problem on weighted cactus graphs. *Discrete Application Mathematics*, págs. 377–386, 2004.

[37] B. Zmazek. Estimating the traffic on weighted cactus networks in linear time. *Ninth International Conference on Information Visualization.*, págs. 536–541, 2005.

[38] B. Zmazek y J. Zerovnik. Computing the weighted wiener and szeged number on weighted cactus graphs in linear time. *Croatica Chemica Acta 76*, págs. 137–153, 2003.

[39] V. E. Zverovich. The ratio of the irredundance number and the domination number for block-cactus graphs. *Journal of Graph Theory 29*, págs. 139–149, 1998.