Optimización Genética de Problemas SAT

Thesis · November 1999

CITATIONS
0

READS
242

1 author:

Eduardo Rodriguez-Tello
Center for Research and Advanced Studies of the National Polytechnic Institute
52 PUBLICATIONS **406** CITATIONS

SEE PROFILE

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY

CAMPUS MORELOS

DIVISIÓN DE INGENIERÍA Y CIENCIAS

OPTIMIZACIÓN GENÉTICA DE PROBLEMAS SAT

TESIS

QUE PARA OBTENER EL GRADO DE MAESTRO
EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

EDUARDO ARTURO RODRÍGUEZ TELLO

NOVIEMBRE DE 1999

ASESOR:
DR. JOSÉ TORRES JIMÉNEZ

Optimización Genética de Problemas SAT

Por:
Eduardo Arturo Rodríguez Tello

Aprobada por:

Dr. Jóse Torres Jiménez
Investigador ITESM Campus Morelos, Programa de Graduados en Informática
Asesor de Tesis

Dr. Eduardo Morales Manzanares
Investigador ITESM Campus Morelos, Programa de Graduados en Informática
Revisor de Tesis

Dr. Juan Frausto Solís
Investigador ITESM Campus Morelos, Programa de Graduados en Informática
Revisor de Tesis

Optimización Genética de Problemas SAT

por

Eduardo Arturo Rodríguez Tello

Sometida al Programa de Graduados en Informática
en Noviembre de 1999, en Cumplimiento Parcial de los
Requerimientos para el Grado de
Maestro en Ciencias de la Computación

Resumen

El problema de satisfactibilidad (SAT) tiene relevancia desde un contexto teórico y práctico. En el contexto teórico el problema SAT es fundamental para el análisis de la complejidad computacional de diversos problemas. En el contexto práctico, la solución de problemas SAT es requisito fundamental de diversos métodos de razonamiento. De esta manera el encontrar un procedimiento eficiente para resolver el problema SAT es muy importante. En esta tesis se plantea un enfoque novedoso para resolver problemas SAT usando algoritmos genéticos (AG), el algoritmo está basado en el reordenamiento de las variables que intervienen en las cláusulas del problema SAT, y la aplicación posterior de un AG tradicional. Los resultados presentados en este trabajo evidencian que el enfoque propuesto es consistentemente mejor que un AG tradicional para resolver problemas SAT duros.

Por otra parte las principales aportaciones de esta tesis son:

- La implementación de un algoritmo de Recocido Simulado que permite resolver el Problema de Minimización del Ancho de Banda de Grafos (PMABG) de manera competitiva.
- El uso de una métrica especial para el PMABG (γ) que es substancialmente mejor que la medida empleada comúnmente (β), y además permite bandear grafos pesados.
- Lograr identificar las características que debe cumplir un buen generador de casos de prueba para problemas SAT. Estas son: garantizar el control del grado de repetibilidad de las variables que intervienen en el problema; proveer un mecanismo que garantice el balance entre las apariciones positivas y negativas de cada variable del problema; evitar generar cláusulas triviales, así como cuidar que la razón entre cláusulas y variables sea aproximadamente entre 2 y 4.
- Un enfoque novedoso para generar casos de prueba duros del problema SAT a partir de su analogía geométrica con grafos.

Índice General

Índice de Tablas

Índice de Figuras

Dedicatorias

A Dios, por permitirme llegar a culminar esta meta.

A mis padres Santa y Eduardo, por todo su apoyo pero principalmente por sembrar en mi ese gusto por aprender y esforzarme cada día para lograr lo que me proponga.

A mi hermana Elda Claudia, por su apoyo incondicional y por todo su cariño.

A Liss mi novia y gran amiga, por toda esa paciencia y apoyo que me brindo cuando más lo necesite.

Agradecimientos

Quisiera agradecer de manera muy especial a mi asesor de tesis el *Dr. José Torres Jiménez* por su constante guía y apoyo a lo largo de mis estudios de maestría. José me enseñó la importancia de la intuición, la perseverancia y muchas otras habilidades indispensables en la investigación. Dado su gran entusiasmo y disponibilidad en todo momento para la discusión de nuevas ideas, difícilmente podría haber encontrado un mejor asesor. También quisiera agradecer a los revisores de mi trabajo de tesis *Dr. Eduardo Morales Manzanares* y *Dr. Juan Frausto Solís* por sus valiosos comentarios que enriquecieron este trabajo. A los directivos del ITESM Campus Morelos que económicamente apoyaron el desarrollo de mis estudios de maestría. A *Giovani Gómez Estrada* por permitirme consumir tanto tiempo de CPU.

Al *CD Juan Manuel Orozco y Sánchez* por todo el apoyo que me ha brindado para el desarrollo de mis estudios. Gracias por permitirme poner en práctica lo que alguna vez aprendí en las aulas de clase y por brindarme su confianza al emprender nuevos proyectos.

A *Lety, Glorise, Nancy* y *Pepe* por darme oportunidad de robarles la atención de José durante los meses que duro el desarrollo de mi tesis.

A mis padres *Eduardo* y *Santa*, así como a mi hermana *Elda Claudia* que sin reclamos permitieron que me dedicara completamente a cumplir con esta meta propuesta y me apoyaron siempre en todas las formas posibles.

A *Lisset* por su cariño, paciencia y comprensión; gracias por haberme apoyado de manera incondicional y aceptar el escaso tiempo que te brindé estos últimos meses. Sé perfectamente el esfuerzo que realizaste, y por esta razón este triunfo es también tuyo.

Capítulo 1

Introducción

El problema de *Satisfactibilidad* (*SAT*), consiste en encontrar alguna asignación de valores de verdad que logren satisfacer una colección de cláusulas en forma normal conjuntiva (FNC). Fue el primer problema computacional que se mostró que es NP-Completo [Coo71], aún cuando se restrinja el número de variables por cláusula a tres (3-SAT) [GJ79].

La satisfactibilidad es de gran importancia en diferentes campos de investigación ya que muchos problemas pueden representarse como un problema SAT. Algunas de las áreas de estudio en donde se aplica son: Investigación de operaciones [SKC95], problemas de planificación de tareas [KS92], síntesis y prueba de circuitos [Coe97], y el estudio de la complejidad computacional [Pap94], por mencionar sólo algunas.

Actualmente son muy variadas las técnicas desarrolladas para la solución del problema SAT, como ejemplos podemos mencionar el recocido simulado [Spe93], las redes neuronales [Spe96], y búsqueda tabú [MSG97]. Por otra parte se ha detectado que los algoritmos genéticos no han sido probados a profundidad y comparados contra otras técnicas de solución del problema SAT, por lo que representan un nuevo y amplio campo de investigación.

Los AG son uno de los métodos incompletos más empleados para la solución de problemas NP-Completos y uno de los que más interés ha despertado en la comunidad de investigadores. Esto se debe principalmente al éxito que han tenido en la solución de problemas duros donde el espacio de posibles soluciones es muy grande.

Para el caso particular del problema SAT, dado que consiste en una búsqueda sobre n variables booleanas que resulta en un espacio de soluciones de tamaño 2^n, la mejor opción para

resolverlo usando AG es una codificación con cadenas binarias de longitud n.

Existen fuertes razones para creer que si bajo una representación particular los bits relacionados no están cercanos, se espera que el AG no se desempeñe adecuadamente. Esta idea está fundamentada en [Hol75], donde se reporta que un AG procesa eficientemente esquemas cortos de bajo orden con aptitud arriba del promedio. Basándose en lo anterior, es posible crear una etapa de preprocesamiento que se encargue de colocar las variables relacionadas en posiciones cercanas dentro del cromosoma, para que posteriormente el AG resuelva el problema SAT con un desempeño mejor.

Considerando lo anterior la implementación computacional que se propone en esta tesis para resolver problemas SAT duros, consiste de dos etapas. En la primera etapa se emplea un algoritmo de preprocesamiento que permitirá cambiar la representación del problema a resolver, para disminuir su epístasis, con el fin de que en una segunda etapa sea posible utilizar eficientemente un AG para solucionar el problema SAT.

Esta tesis está organizada en 8 capítulos que describimos brevemente a continuación:

- **Capítulo 2.** Presenta un panorama general del estado del arte del problema SAT, y de los algoritmos que se han desarrollado para su solución.

- **Capítulo 3.** Contiene un estudio de los algoritmos genéticos (AG), que resalta las principales diferencias entre los AG y los métodos tradicionales de búsqueda y optimización, así como de los principales aspectos a considerar al momento de realizar la implementación de un AG para resolver un problema particular.

- **Capítulo 4.** Describe los requisitos que una representación debe cumplir para ser utilizada en AG y plantea la idea de realizar un preprocesamiento de problemas SAT para mejorar el desempeño al momento de ser resueltos con AG.

- **Capítulo 5.** Contiene un estudio del problema de minimización del ancho de banda de grafos, así como de los algoritmos más empleados para resolver este problema.

- **Capítulo 6.** Se definen los detalles específicos de la implementación del algoritmo propuesto llamado PAG (Preprocesamiento y Algoritmo Genético) para resolver problemas SAT.

- **Capítulo 7.** Muestra un análisis comparativo del desempeño del algoritmo PAG comparado con un AG al momento de resolver problemas SAT duros.
- **Capítulo 8.** Presenta las conclusiones obtenidas de este trabajo de tesis, así como las posibles líneas de investigación futuras.

Capítulo 2

El problema de Satisfactibilidad

El problema de *Satisfactibilidad* (SAT) fue el primer problema computacional que se mostró que es NP-Completo [Coo71], y aún cuando se restrinja el número de variables por cláusula a tres (3-SAT), éste sigue siendo NP-Completo [GJ79]. Esto significa que no podemos esperar encontrar un algoritmo determinístico eficiente que resuelva este problema en un tiempo polinomial. Sin embargo, sí es posible resolver algunos casos especiales del problema de satisfactiblidad en tiempo polinomial; un ejemplo son los problemas denominados 2-SAT, donde el número de variables por cláusula se restringe a dos.

El problema SAT se define formalmente de la siguiente manera:

> *Dada una colección C, de m cláusulas, $C_1, ..., Cm$, en Forma Normal Conjuntiva (FNC), las cuales se encuentren formadas por la disyunción $v_1 \vee ... \vee v_n$ de hasta n variables lógicas seleccionadas de un conjunto V, encontrar alguna asignación de valores de verdad, para dichas variables, tal que C sea verdadera.*

La satisfactibilidad es de gran importancia en diferentes campos de investigación ya que muchos problemas pueden representarse como un problema SAT. Algunas de las áreas de estudio en donde se aplica son: Investigación de operaciones [SKC95], problemas de planificación de tareas [KS92], síntesis y prueba de circuitos [Coe97], y el estudio de la complejidad computacional [Pap94], por mencionar sólo algunas.

Para la Inteligencia Artificial este problema es en especial importante por que tiene una conexión directa con algunos métodos de razonamiento. El razonamiento deductivo es simple-

mente una extensión del problema SAT: Dada una colección de hechos F, es posible deducir una sentencia a, si y sólo si, $F \cup \{\sim a\}$ no es satisfactible. Muchas otras formas de razonamiento también recurren directamente a la satisfactibilidad, incluyendo diagnósticos, planeación e interpretación de imagenes.

Una importante variante del problema SAT, es conocida como el problema de *Máxima Satisfactibilidad (MAX-SAT)*, el cual se describe de la siguiente forma:

> *Dada una colección C, de m cláusulas, $C_1, ..., Cm$, en Forma Normal Conjuntiva (FNC), las cuales se encuentren formadas por la disyunción $v_1 \vee ... \vee v_n$ de hasta n variables lógicas seleccionadas de un conjunto V, encontrar alguna asignación de valores de verdad, para dichas variables, tal que se maximice el número de cláusulas satisfechas de C.*

Para entender el problema de satisfactibilidad es necesario realizar primero un estudio de la demostración automática de teoremas, así como de las diferentes técnicas propuestas para la solución de este problema. Por esta razón a continuación se presenta un recuento histórico del problema SAT.

2.1 Antecedentes Históricos del Problema SAT

Desde hace mucho tiempo el hombre ha pretendido encontrar un procedimiento general de decisión que le permita comprobar sus teoremas de forma automática. Uno de los primeros en intentarlo fue Leibniz a finales del siglo XVII, pero no fue sino hasta 1920 en que Hilbert y sus seguidores retomaron la idea. En 1936 Turing y Church, probaron que era imposible esta comprobación. Cada uno de ellos, de manera independiente, mostraron mediante sus trabajos que es imposible tener un procedimiento general de decisión para verificar la validez o inconsistencia de fórmulas de la lógica de primer orden [Chu36][Tur36]. Sin embargo, existen procedimientos de prueba capaces de verificar que una fórmula es válida. En general para fórmulas inválidas este tipo de procedimientos no tienen un número de pasos finito.

La evolución en la investigación de esta área de la lógica ha tenido un gran número de avances en un corto tiempo, pero sin duda una gran aportación para la demostración automática de teoremas fue la que realizó Herbrand en 1930. Él desarrolló un algoritmo para encontrar una

interpretación que pudiera hacer que el resultado de la evaluación de una fórmula fuera falso (F). Sin embargo, si la fórmula dada es en efecto válida [1], no existe tal interpretación, y su algoritmo se detendrá después de un número finito de iteraciones. Este método propuesto es la base de la mayoría de los procedimientos modernos de prueba automática de teoremas [CLC73].

Una de las primeras personas en implementar computacionalmente el algoritmo propuesto por Herbrand, fue Gilmore, su programa fue diseñado para detectar la inconsistencia de la negación de una fórmula dada. El funcionamiento básico es el siguiente: durante la ejecución del programa son generadas periódicamente fórmulas proposicionales para posteriormente verificar su inconsistencia. Si la negación de la fórmula dada es inconsistente, el programa detectará eventualmente este hecho. El programa de Gilmore probaba un conjunto simple de fórmulas, pero encontraba importantes dificultades con la mayoría de las fórmulas de la lógica de primer orden [Gil60].

Con el tiempo, minuciosos estudios sobre esta implementación revelaron que este método de prueba de inconsistencia de una fórmula proposicional era ineficiente. Por lo que Davis y Putnam [MD60] lo mejoraron pocos meses después de que los resultados de Gilmore fueran publicados. Sin embargo, sus mejoras no fueron suficientes. Muchas fórmulas válidas de la lógica de primer orden aún no podían ser probadas mediante el uso de un programa computacional, en un lapso de tiempo razonable.

Más adelante Robinson realizó una gran aportación a la prueba mecánica de teoremas, la cual consistió en el desarrollo del conocido *Principio de Resolución* [Rob65]. El procedimiento de prueba mediante resolución es mucho más eficiente que cualquiera de los algoritmos que fueron propuestos anteriormente. Desde la introducción del principio de resolución, múltiples mejoras se han implementado en un intento de incrementar su eficiencia. Algunos ejemplos de estas mejoras son: resolución semántica, resolución bloqueada, resolución lineal, y la resolución unitaria [CLC73].

Como hemos podido apreciar los origenes del problema SAT están intimamente relacionados con la demostración automática de teoremas. Esto se debe principalmente a que la demostración de teoremas consiste en probar que un teorema es una consecuencia lógica de un conjunto de axiomas (es decir que su conjunción establece una fórmula valida), para lo cual un procedimiento

[1] Por definición, una fórmula válida es verdadera bajo cualquier interpretación.

comunmente empleado es el de resolver el problema de satisfactibilidad asociado.

Aun cuando el problema SAT es un problema NP-Completo[Coo71], muchos métodos han sido propuestos para satisfacer las necesidades prácticas. En las secciones siguientes se analizan algunos de los métodos más representativos.

2.2 Métodos para Resolver el Problema SAT

En general, los métodos propuestos para resolver el problema SAT, suelen clasificarse de manera general en dos grandes categorias: Métodos exactos[2] y completos[3] y métodos exactos e incompletos. Como ejemplos de la primera categoría podemos mencionar los basados en la lógica: como el de Davis-Putnam-Loveland [DLL62] [Lov70], o el principio de resolución de Robinson [Rob65]; los que utilizan redes de restricciones: como el de propagación local, y el de programación lineal 0/1[JMB91]. La segunda categoría de esta clasificación incluye métodos tales como los algoritmos evolutivos, el recocido simulado (*simulated annealing*), los algoritmos de búsqueda local y los que hacen uso de un enfoque glotón (*greedy*) como el Greedy-SAT [BS92]. Los métodos completos tienen la ventaja de que teóricamente son capaces de encontrar todas las soluciones de cualquier problema booleano, o en su defecto, probar que no existe una solución. Sin embargo, la naturaleza combinatoria del problema SAT convierte a los métodos completos en imprácticos cuando el tamaño del problema a resolver se incrementa. Una forma de superar este obstáculo es la empleada por los métodos incompletos. Estos usan heurísticas eficientes para tratar de limitar el problema del aumento de la complejidad lo más posible, además cabe resaltar que se ha demostrado recientemente que los métodos incompletos son capaces de resolver instancias más difíciles de SAT de una clase más amplia de problemas [JS89].

2.2.1 Métodos Completos

Los métodos completos, como mencionamos, tienen la ventaja de que teóricamente son capaces de encontrar todas las soluciones de cualquier problema booleano, o en su defecto, probar que no existe una solución al problema. En esta sección analizaremos algunos métodos como son:

[2] Exacto (soundness): Significa que cualquier solución encontrada sea correcta.
[3] Completo (completeness): Se refiere a que una solución debe ser encontrada al problema si es que una existe.

- El Teorema de Herbrand
- El Método de Davis y Putnam
- El Principio de Resolución
- La Resolución Semántica
- El Algoritmo de Dos Fases para SAT y MAX-SAT

Así como una propuesta para paralelizar el problema SAT.

El Teorema de Herbrand

El teorema de Herbrand es muy importante en lógica simbólica; es la base para los más modernos procedimientos de prueba, en la demostración mecánica de teoremas. Este teorema tiene la finalidad de probar si un conjunto de cláusulas es insatisfactible, para ello sólo considera las interpretaciones sobre el universo de Herbrand. Si el conjunto de cláusulas es falso, bajo todas las interpretaciones, entonces nosotros podemos concluir que las cláusulas son insatisfactibles. Debido a que hay usualmente un número infinito de interpretaciones, nosotros debemos organizarlas en alguna forma sistemática. Esto puede ser realizado mediante el uso de árboles semánticos.

El *Teorema de Herbrand* se enuncia como sigue:

> *Un conjunto S de cláusulas es insatisfactible si y sólo si existe un conjunto insatisfactible finito S' de instancias de las cláusulas aterrizadas de S* [CLC73].

Un ejemplo muy sencillo y conocido es el siguiente: Sea S, un conjunto instatisfactible tal que $S = \{P(x), \sim P(f(y))\}$. Entonces de acuerdo con el teorema de Herbrand, existe un conjunto insatisfactible finito S' de instancias de las cláusulas aterrizadas de S. Uno de estos conjuntos es $S' = \{P(f(a)), \sim P(f(a))\}$.

El teorema de Herbrand sugiere un procedimiento de demostración basado en la refutación. En otras palabras, dado un conjunto insatisfactible de cláusulas a probar, si existe un procedimiento mecánico que pueda generar sucesivamente conjuntos $S_1, ..., S_n$, de instancias de cláusulas en el Universo de Herbrand, y se puede verificar la insatisfactibilidad de cada uno de

estos conjuntos, entonces, como lo garantiza el teorema de Herbrand, este procedimiento puede detectar en un número finito de iteraciones la insatisfactibilidad de S.

Gilmore fue una de las primeras personas en implementar la idea expuesta en el párrafo anterior. En el año de 1960, escribió un programa de computadora que generaba sucesivamente conjuntos de instancias de las cláusulas de S, obtenidas de reemplazar las variables por constantes tomadas del Universo de Herbrand de S. Debido a que cada conjunto de interpretaciones es una conjunción de cláusulas, es posible utilizar cualquier método de la lógica proposicional para verificar su insatisfactibilidad. Gilmore utilizó un método basado en convertir cada conjunto de cláusulas producido, en su *Forma Normal Disyuntiva (FND)*; para remover cada conjunción en donde exista un par complementario de variables. Si algún conjunto termina estando vacío, entonces ese conjunto es insatisfactible y una prueba fue encontrada.

Este método usado por Gilmore es bastante ineficiente, Davis y Putnam en 1960 desarrollaron y publicaron un nuevo método más eficiente para verificar la insatisfactibilidad de un conjunto de cláusulas.

El Método de Davis y Putnam

Este método consiste en el uso iterativo de cuatro reglas que permiten realizar la comprobación de la insatisfactibilidad de un conjunto de cláusulas. Estas reglas permiten ir reduciendo la cantidad de cláusulas hasta llegar al punto en que se tiene un fórmula que es válida o que es insatisfactible y que contiene el mínimo número de cláusulas [MD60].

A continuación se describe cada una de estas reglas:

Sea S un conjunto de cláusulas instanciadas en el Universo de Herbrand.

1. *Regla de la Tautología:* Borre todas las cláusulas aterrizadas de S que son tautologías. El conjunto resultante S' es insatisfactible si y sólo si S lo es.

2. *Regla de la Literal Unitaria:* Si existe una cláusula aterrizada unitaria L en S, obtenga S' de S al borrar aquellas cláusulas en S que contengan L. Si S' está vacio, S es satisfactible. De lo contrario, obtenga un conjunto S'' de S' al borrar $\sim L$. S'' es insatisfactible si y sólo si S lo es. Note que si $\sim L$ es una cláusula aterrizada unitaria, entonces la cláusula se convierte a una cláusula vacía cuando $\sim L$ es borrada de la cláusula.

3. *Regla de la Literal Pura:* Una literal L en una cláusula aterrizada de S, se dice que es pura en S si y sólo si la literal $\sim L$ no aparece en ninguna cláusula aterrizada en S. Si una literal L es pura en S, borre todas las cláusulas aterrizadas que contengan L. El conjunto S' que resulta de esto, es insatisfactible si y sólo si S lo es.

4. *Regla de Expansión:* Si el conjunto S puede ser puesto en la forma: $(A_1 \vee L) \wedge ... \wedge (A_m \vee L) \wedge (B_1 \vee \sim L) \wedge ... \wedge (B_n \vee \sim L) \wedge R$, donde A_i, B_i, y R están libres de L y $\sim L$, entonces obtenemos los conjunto $S_1 = A_1 \wedge ... A_m \wedge R$ y $S_2 = B_1 \wedge ... B_n \wedge R$. S es insatisfactible si y sólo si $(S_1 \vee S_2)$ es insatisfactible, esto significa que S_1 y S_2 también lo son.

El método descrito anteriormente para verificar la insatisfactibilidad (inconsistencia), es más eficiente que el implementado por Gilbert. El método de Davis y Putnam, a diferencia del de Gilbert, puede ser aplicado a cualquier fórmula de la lógica proposicional. Por supuesto, primero se tiene que transformar esa fórmula en su *forma normal conjuntiva*, para posteriormente aplicar las cuatro reglas descritas anteriormente.

El Principio de Resolución

La idea esencial del principio de resolución es verificar cuando un conjunto de cláusulas S contiene la cláusula vacía. Si S la contiene, entonces S es insatisfactible. Pero si no, entonces la siguiente cuestión a verificar es si a partir de S se puede derivar la cláusula vacía.

Lo anterior visto en forma gráfica implica representar en un árbol semántico T, el conjunto S de cláusulas y verificar la inconsistencia en cada uno de los nodos. Se puede deducir fácilmente que S contiene la cláusula vacía, si y sólo si T consiste de un sólo nodo (el nodo raíz). Si S no contiene la cláusula vacía, entonces T, tiene más de un nodo. Sin embargo, si nosotros podemos reducir el número de nodos a uno, en algún momento la cláusula vacía puede ser forzada a aparecer. Esto es exactamente lo que el principio de resolución nos permite realizar.

En efecto, nosotros podemos ver el principio de resolución como una regla de inferencia que nos permite generar nuevas cláusulas a partir de S. Si colocamos estas nuevas cláusulas dentro de S, algunos nodos del conjunto original se estarán forzando a convertirse en nodos con falla. De esta forma el número de nodos en T puede ser reducido a uno y por lo tanto la cláusula vacía se presentará.

El principio de resolución, es en escencia una extensión de la regla de la literal unitaria, que propusieron Davis y Putnam, y de la cual hablamos en la sección anterior. Esta regla lo que nos indica es verificar si existen dos literales en las cláusulas que formen un par complementario, para posteriormente borrarlo.

Lo que realiza el principio de resolución es extender la regla anterior para cualquier par de cláusulas (no necesariamente cláusulas unitarias), con lo que surge la siguiente regla en la cual se expresa formalmente el *Principio de Resolución*:

> *Para cualquier par de cláusulas C_1 y C_2, si hay una literal L_1 en C_1 que es complementaria a una literal L_2 en C_2, entonces debe borrarse L_1 y L_2 de C_1 y C_2 respectivamente, y construir la disyunción de las cláusulas restantes. La cláusula resultante es un resolvente de C_1 y C_2.*

Si nosotros tenemos dos cláusulas unitarias, entonces el resolvente de ellas, si existe alguno, es la cláusula vacía. Más importante aún, si un conjunto S de cláusulas es insatisfactible, nosotros podemos a partir del principio de resolución generar la cláusula vacía a partir de S.

El principio de resolución es una poderosa regla de inferencia, y se aplica en muy diversas áreas de investigación. Un punto muy importante es que el principio de resolución es completo al probar la insatisfactibilidad de un conjunto de cláusulas por refutación. Esto significa, que siempre se podrá generar la cláusula vacía a partir de un conjunto insatisfactible de cláusulas.

Resolución Semántica

Muchas mejoras se han realizado al principio de resolución, cada una de ellas tiene su propio mérito. El método de *Resolución Semántica* fue originalmente propuesto por Slagle en 1967. Este método es una mezcla de los algoritmos de hiperresolución de Robinson y de resolución renombrable de Meltzer [CLC73].

El algoritmo de resolución semántica agrega una importante característica a los algoritmos existentes, tiene un mecanismo que reduce el número de cláusulas que no tienen un uso en la demostración de la inconsistencia del conjunto original de cláusulas [Sta67].

Para entender mejor esto comenzaremos por explicar que este método divide en dos conjuntos las cláusulas originales (S_1 y S_2). Con la característica de que no se permite que cláusulas

dentro del mismo grupo se resuelvan entre si. Esto naturalmente reducirá el número de cláusulas generadas, además de que este tipo de bloqueo no afecta la capacidad de deducir la cláusula vacía $\square$.

En este punto la pregunta importante es qué criterio seguir para agrupar las cláusulas. La respuesta es usando una interpretación, de aquí el nombre de esta técnica de resolución. Debido a que se esta trabajando con un conjunto de cláusulas insatisfactibles, ninguna interpretación puede satisfacer o falsificar todas las cláusulas. Sin embargo, cada interpretación divide S en dos conjuntos no vacíos de cláusulas, por lo que cualquier interpretación puede ser empleada.

Además de este método existen otros como la *Resolución Bloqueada*, propuesta en [Boy71], y la *Resolución Lineal* descrita en [Luc70]. Estos métodos al igual que el de resolución semántica tienen la propiedad de ser completos, es decir, cada uno puede ser usado por separado para derivar la cláusula vacía $\square$.

Algoritmo de Dos Fases para SAT y MAX-SAT

Este algoritmo como su nombre lo indica está dividido en dos etapas; en la primera, se busca encontrar una buena solución inicial al problema, para ello hace uso de una heurística conocida como Greedy-SAT [BS92], que consiste en una búsqueda local para satisfacer asignaciones de un conjunto de cláusulas proposicionales, trataremos con más detalle esta heurística más adelante en este capítulo. En la segunda fase, utiliza un procedimiento de enumeración basado en el algoritmo de Davis-Putnam-Loveland, para encontrar una probable solución óptima. Fue propuesto por Borchers y Furman [BF97].

El propósito de utilizar una primera etapa en donde se aplica una heurística, tiene el fin de aumentar el desempeño del algoritmo, al permitir obtener un límite superior del número de cláusulas insatisfechas (ub) en una solución óptima, que es un parámetro que puede ser utilizado para realizar una mejor poda del árbol de búsqueda.

La parte medular de este algoritmo es la segunda etapa, en ella es donde utiliza un algoritmo de búsqueda hacia atras para recorrer todas las posibles asignaciones de valores de verdad. Al final de este procedimiento se tiene un límite superior (ub) que se va actualizando conforme se van encontrando nuevas soluciones. A la vez también se va guardando un registro de la insatisfactibilidad ($unsat$), que puede considerarse como el número de cláusulas que se dejaron

sin satisfacer en la solución actual.

Dentro la segunda fase del algoritmo se emplean los valores obtenidos en la primer fase (ub y $unsat$), de forma que:

- Si $unsat \geq ub$, entonces la solución parcial no puede ser extendida para generar una mejor, así que se descarta la solución parcial encontrada y se realiza una búsqueda hacia atras (*backtrack*) para encontrar otra solución parcial. Para realizar el *backtrack*, se búsca la variable que se halla cambiado más recientemente hacia el valor de verdad falso en la fase uno; esta variable se niega y se continua el proceso.

- Si $unsat = ub - 1$, entonces se procesa usando el método de la cláusula unitaria. Una cláusula unitaria, para este algoritmo, es aquella donde todas, excepto una variable tienen valor de verdad falso. Cualquier literal dejada en una cláusula unitaria debe ser puesta como verdadera, para no incrementar el valor de la insatisfactibilidad de la solución a un número mayor que ub. Después de haber llevado a cabo el procedimiento de la literal simple, se debe de actualizar el valor de la insatisfactibilidad.

- Si se verifica la solución y se observa que $unsat < ub$, entonces se agrega otra variable lógica a la solución parcial actual. Como esta variable puede tomar dos valores entonces se generan dos subproblemas, por lo que se dice que el problema se divide en ramas. Para realizar la búsqueda a través de las ramas el algoritmo primero selecciona las cláusulas con el menor número de literales pendientes de asignarle un valor, entonces se selecciona la variable que aparece en el mayor número de cláusulas y se asigna a esta el valor de falso, así es como se sigue con la siguiente iteración del algoritmo. Este algoritmo finaliza cuando se han enumerado todas las posibles soluciones.

Este algoritmo se implementó con tres objetivos principales en mente: el primero fue determinar el tamaño de los problemas que podían ser resueltos, en un tiempo razonable. El segundo fue el determinar cuándo el uso de la heurística GSAT, mejoraba el desempeño global del algoritmo de dos fases. Y por último el determinar si el algoritmo de dos fases para la solución de MAX-SAT es competitivo con los resultados de la programación entera.

Con respecto al estudio de los tres puntos anteriores se reporta en [BF97] que:

1. El algoritmo de dos fases es un método que se comporta de manera estable en problemas que están formados por un número de variables entre cincuenta y ciento cincuenta, y con cien a setecientas cincuenta cláusulas.

2. Para verificar la utilidad del uso de GSAT en la primera fase del algoritmo, se ejecutó sin esta parte del algoritmo, de lo cual se observó que para problemas pequeños, el uso de la heurística no aumentaba considerablemente el desempeño del algoritmo. Sin embargo para problemas grandes el uso de GSAT, representó un dramático efecto en el tiempo de CPU utilizado para resolver los problemas. El algoritmo con GSAT es siete veces más rápido que el que no lo tiene.

3. En problemas 3-SAT, el algoritmo de dos fases es más competitivo que su equivalente en programación entera, pero a medida que el número de variables por cláusula se incrementa, el desempeño del algoritmo de dos fases también aumenta en un factor de cinco, según los resultados mostrados por Borchers y Furman.

Una Posibilidad de Paralelizar el Problema SAT

Un avance sobre la eficiencia del algoritmo de resolución es el procedimiento Davis-Putman [MD60], el cual consiste en asignar valores booleanos a una fórmula a la vez. Cada asignación permite satisfacer algunas cláusulas, y eliminar pares complementarios, tautologías y cláusulas con literales sencillas. Sobre esta misma idea en [WTC87] se propone una paralelización del algoritmo Davis-Putman y en [MYF92] se define una generalización de las reglas de inferencia:

- Considerar un conjunto de n variables booleanas $P_1, P_2, ..., P_n$
- Una variable P_i puede aparecer en forma positiva P_i o en su forma negativa $\sim P_i$
- Sea S un conjunto de cláusulas y P alguna variable en S. Entonces es posible particionar S en dos conjuntos de cláusulas $S(P)$ y $S(\sim P)$ con respecto a P de la siguiente manera:
  - Para cada cláusula C en S, si P aparece en C entonces se borra de C y el residuo se pone en $S(P)$
  - Si $\sim P$ aparece en C entonces se borra $\sim P$ de C y el residuo se coloca en $S(\sim P)$

– Si no existe una ocurrencia de P o $\sim P$ entonces colocar C en ambos conjuntos $S(P)$ y $S(\sim P)$

Como puede verse esta variante del algoritmo de Davis-Putman permite paralelizar fácilmente una búsqueda en la prueba automática de teoremas.

2.2.2 Métodos Incompletos para Resolver el Problema SAT

A continuación se analizan una serie de algoritmos diseñados para resolver el problema de satisfactibilidad. Cada uno de ellos es importante, ya que encontrar un método eficiente para la solución del problema SAT potencialmente representa la posibilidad de resolver cualquier problema NP-Completo. Esto debido a que cualquier problema NP-Completo por definición puede mapearse con un algoritmo de tiempo polinomial a un problema SAT. Los algoritmos y herramientas computacionales presentados cubren un gran panorama de los diferentes enfoques utilizados para resolver SAT, los algoritmos analizados son los siguientes:

- GSAT,
- RWS-GSAT
- Recocido Simulado SAT
- TSAT
- GSAT de Dos-Fases
- MASK
- NNSAT

Algoritmo GSAT

En 1992 el grupo de Bart Selman, Héctor Levesque y David Mitchell desarrollaron el algoritmo *Greedy-SAT* (*GSAT*) [BS92] y con esto una productiva línea de investigación. El procedimiento GSAT es una búsqueda local para satisfacer asignaciones de un conjunto de cláusulas proposicionales. Se utilizó inicialmente sobre conjuntos 3-SAT (tres variables por cláusula) generados

aletoriamente y con una probabilidad de cambiar el signo de la variable del 50%, esto genera conjuntos difíciles de satisfacer como se demostró de manera empírica en [DM92]. El algoritmo GSAT se define como sigue:

Procedimiento GSAT

Entrada: *Conjunto de cláusulas, Máximos cambios, Máximas iteraciones;*

Salida: *Asignación de verdad que satisfaga las cláusulas;*

Para $I = 1$ ***hasta*** *Máximas iteraciones*

$A =$ *una asignación de verdad generada de manera aleatoria;*

Para $J = 1$ ***hasta*** *Máximos cambios*

Si *A Satisface las cláusulas* ***entonces***

Regresa *"Solución encontrada";*

Sino

$P =$ *Variable que invirtiendo su valor de verdad incrementa el mayor número de cláusulas satisfechas;*

$A = A$ *sustituyendo* P con *el valor invertido;*

Fin Sino

Fin para J

Fin para I

Regresa *"Solución no encontrada";*

Fin GSAT

Extensiones al Algoritmo GSAT

El algoritmo GSAT es bastante general y como puede apreciarse puede utilizarse en más algoritmos de optimización. Algunas extensiones al algoritmo básico se resumen a continuación:

- ¿Es posible elegir una nueva asignación de verdad A de manera más inteligente? Esto puede aproximarse al *Recocido Simulado* [KGV83], en donde el factor T o temperatura indica la velocidad con la cual se "enfriará" el sistema (haciendo una analogía con el proceso físico de la industria metalúrgica). Es preciso comentar que según resultados experimentales publicados en [SK93] y [SKC95] se muestra que específicamente en SAT

el adicionar esquemas tipo recocido simulado no proporciona ninguna ventaja, excepto cuando este se reduce a Metrópolis con una temperatura constante [Jer92].

- ¿Siempre deberá de tomarse una variable P que maximice el valor de las cláusulas satisfechas? Sobre este punto se propuso una ligera variante en [JS89], conocida como *RWS-GSAT (Random Walk Strategy)*. En esta estrategia se realiza con probabilidad p, un cambio en alguna variable de la asignación A, y con una probabilidad $1-p$ se sigue el algoritmo normal GSAT. La pruebas que se presenta en [BSC94] son bastante interesantes debido a la propiedad que se tiene para escapar de óptimos locales e inclusive puede resolver varias instancias del "*DIMACS Challenge Benchmark Instances*".

- ¿Qué sucede si una variable aun siendo aleatoria se selecciona muchas veces y nos conduce a un mínimo local? Este problema se resolvió utilizando un contador por cláusula visitada y no satisfecha, y a partir de un umbral X ya no intentar satisfacer esa cláusula, sino "saltar" a otra diferente. En [BCM97] se muestran resultados experimentales de esta variante. El contador se reinicializa después de visitar *Máximos cambios*.

- Sobre al punto anterior también se ha propuesto guardar las variables que mejor se han comportado al momento de variar su valor, es decir que mejoren el número de cláusulas satisfechas. Teniendo estas variables podemos eliminar la parte aleatoria en la elección de la variable a cambiar, esto se podrá hacer en una estructura tipo FIFO (First In First Out). En [Fuk97] se presenta esta variante en la forma de elegir la variable P a partir de una lista, y se mostró que el utilizar una cola FIFO incrementa el poder del algoritmo.

- También al guardar los caminos que nos han conducido a mínimos locales podríamos prevenir que estos se presenten de manera consecutiva, es decir guardando una lista Tabú [Glo89]. El enfoque *Tabú-SAT (TSAT)* se ha utilizado en problemas reales que son del tipo k-SAT (a diferencia de los anteriores donde se prueba con 3-SAT ó 4-SAT). El TSAT es relativamente nuevo y se han conducido pocos experimentos utilizando esta Meta-Heurística, en [MSG97] se presenta una comparación entre el RWS-GSAT y TSAT. En esta TSAT mostró ser competitivo. Cabe señalar que la longitud de la lista Tabú es lineal con relación al número de variables.

- ¿Los algoritmos siempre necesitan comenzar desde cero? ¿Podrá ser útil tener almacenados los mejores resultados anteriores y a partir de ellos continuar las búsquedas? Esta extensión al GSAT, es conocida como *GSAT de Dos-Fases* es reportada en [SK93] y sugiere tomar información de la solución más cercana a la anterior. Esto reemplaza a la primera asignación aleatoria del GSAT original, y se comenzaría con una buena solución de la ejecución anterior.

El Algoritmo MASK

El método *MASK*, desarrollado por Hao y Dorne [HD93] al igual que los algoritmos genéticos (AG), está basado en tres elementos: una representación interna, que es un conjunto de cadenas binarias parcialmente instanciadas llamadas mascaras (masks), una función de evaluación y un mecanismo de evolución. El algoritmo de MASK está representado en el siguiente pseudocódigo:

Procedimiento MASK

Inicio

Inicializar la población de mascaras P;

Mientras *exista un bit libre en la mascara*

Evaluar P;

Ordenar P;

Seleccionar la mejor mitad de P (2^{k-1} mascaras);

Aplicar a cada mascara seleccionadas el operador División;

Fin Mientras

Fin MASK

Este algoritmo comienza con 2^k ($k \geq 1$) máscaras del mismo tamaño. El proceso de iteración continua con la evaluación de cada una de las 2^k máscaras usando un cierto mecanismo de evaluación, después elimina la peor mitad de ellas, y finalmente divide cada una de las máscaras restantes. Como ejemplo de la división empleada tenemos el siguiente: La división de la máscara $M = 101????$, produce dos máscaras nuevas $M' = 1010????$ y $M'' = 1011????$, las cuales reemplazan la máscara original M. Este proceso continúa hasta que todas las N posiciones en las máscaras tengan asignado un valor. De esta forma una posible solución es aquella en donde hay una máscara totalmente instanciada.

Tres factores distinguen al algoritmo MASK del resto de los algoritmos genéticos:

1. Una máscara es una cadena parcialmente instanciada de unos y ceros, es decir, algunos bits permanecen libres hasta que el operador de división es aplicado. La ventaja de esto es que el modelo del cromosoma empleado por los AG's sólo puede representar un determinado modelo, mientras que una máscara representa un conjunto de modelos potenciales. Más exactamente una máscara de longitud N con F bits libres cubre un subespacio de 2^F de un espacio total de búsqueda de tamaño 2^N.

2. El concepto de máscara difiere del concepto de esquemas utilizado por los algoritmos genéticos. En realidad el esquema es una notación conceptual para agrupar una colección de cromosomas que tienen genes en común, y no pueden ser directamente manipulados por los operadores genéticos, mientras que las máscaras, que son elementos básicos del método de MASK, son directamente manipulables por el operador de división.

3. La tercera diferencia tiene que ver con el mecanismo de evolución. En MASK, no existe el cruzamiento ni las mutaciones. Después de medir el grado de eficiencia de cada máscara y eliminar la peor mitad de la población, cada máscara restante es simplemente dividida en dos máscaras más especializadas.

En general se observa que de acuerdo a los resultados mostrados en [HD94], existen tres principales puntos sobre los cuales se pueden concluir.

- MASK escala hacia las soluciones mucho mejor que los AG's estándar, a medida que el número de variables se incrementa.

- MASK es más robusto que los algoritmos genéticos.

- Por último se observa que los operadores genéticos no son lo suficientemente apropiados para los problemas de satisfactibilidad. Por el concepto de esquemas necesitaríamos observar que ante un mínimo cambio en los genes l y $l+1$ (genotipo) responderá también con un cambio mínimo en el fenotipo. Esto no se ha conseguido realizar dado que la estructura intrínseca del problema muestra (con una representación binaria en donde cada gen corresponde a una variable) que modificar una parte mínima del gen produce cambios drásticos en el fenotipo.

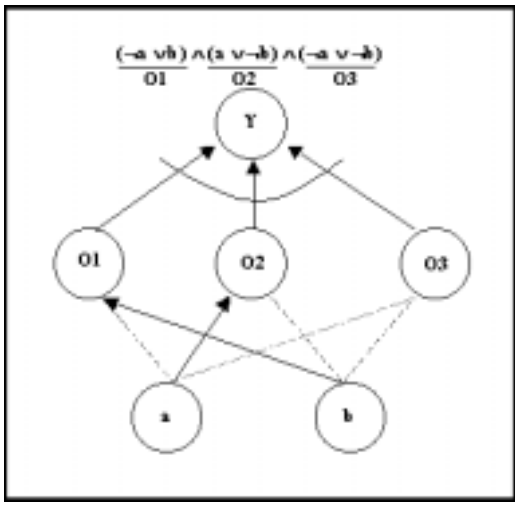

Figura 2-1: Red neuronal de un problema SAT

Algoritmo basado en Redes Neuronales (NNSAT)

Debido a que el problema SAT es un problema de satisfacer restricciones, este puede ser modelado como una red neuronal de Hopfield. En [Spe96] se describe un algoritmo llamado *NNSAT*, para resolver problemas de satisfactibilidad. Cabe mencionar que este algoritmo tiene la ventaja de no estar restringido a problemas en FNC, ya que puede manejar perfectamente expresiones booleanas arbitrarias. El grafo de la red neuronal está basado en un árbol Y/O, aun cuando no necesariamente sea un árbol. Esto se debe a que cada instancia de una variable booleana es representada por un sólo nodo hoja, el cual puede tener múltiples padres. La estructura resultante de esto es un grafo dirigido acíclico con raíz, donde todos los arcos del grafo están dirigidos hacia el nodo raíz. Un ejemplo es mostrado en la Figura 2-1, donde las negaciones de las variables se representan con líneas punteadas. Además tiene las siguientes características:

- Cada arco tiene un *peso* $w_{ij} = -1$, si existe una restricción de negación del nodo i al nodo j, de lo contrario tiene un peso de 1.
- Cada nodo i puede ser *verdadero* o *falso*, lo que es representado usando una *activación* a_i de 1 o 0, respectivamente.
- *Energy* es la energía globlal del estado de la red, y representa el grado en el cual las

restricciones booleanas son satisfechas.

- *Energy* es la combinación de las contribuciones locales de energía de cada nodo: $Energy = \sum_i Energy_i = \sum_i net_i a_i$. Si todas las restricciones son satisfechas, entonces cada contribución local de energía se maximiza y por lo tanto la energía total del sistema se maximiza también.

A continuación se describe el algoritmo propuesto por Spears:

Procedimiento NNSAT

Entrada: *Una expresión booleana, Max_temp y Min_temp;*

Salida: *Una asignación de verdad que satisfaga las cláusulas, si se encuentra;*

Inicio

$reinicios = 0, asignaciones = 0;$

Ciclo

$reinicios++;\ j = 0;$

aleatoriamente inicializar las a_i *para cada nodo* i;

Ciclo

Si *la expresión es satisfecha termina exitosamente;*

$T = Max\_temp * e^{-j * decay\_rate};$

Si $(T < Min\_temp)$ *sal del ciclo;*

Ciclo *a través de todos los nodos i aleatoriamente*

Calcula la entrada de la red net_i;

$probabilidad(a_i = 1) = \frac{1}{1+e^{\frac{-net_i}{T}}};$

Fin Ciclo

$j++;$

Fin Ciclo

Fin Ciclo

Fin NNSAT

2.3 Aplicaciones

Son múltiples las aplicaciones practicas del problema SAT, pero la primera y más obvia es a otros problemas NP-completos tales como la coloración de grafos, el problema del agente viajero, etc. El esfuerzo computacional necesario para reducir muchos de estos problemas a instancias SAT es muy pequeño; por ejemplo, Nadia Creignou mostró que la mayoría de esos problemas son *linealmente equivalentes* a SAT, es decir, son reducibles a un problema SAT y viceversa en tiempo lineal [Cre93]. Otras aplicaciones de SAT son: *job shop Scheduling* [CB94]; diseño de circuitos VLSI [Dev89]; así como muchos otros problemas de razonamiento en Inteligencia Artificial. Incluso existe un algoritmo para resolver SAT inmerso en la máquina de inferencias de Prolog III [Col90].

Cabe mencionar que el problema SAT, está tomando gran importancia en muchas áreas de la Inteligencia Artificial, por ejemplo, Kautz y Selman mostraron una forma de codificar problemas de planeacion (*planning*) como problemas SAT y demuestran que realizar esa transformación además de permitirles tener un marco de trabajo más flexible para incluir diferentes tipos de restricciones en los planes, también refleja más confiablemente la teoría detrás de los modernos sistemas de planeación basados en restricciones [KS92]. Y Papadimitriou menciona que alguna forma de satisfactibilidad puede ser un mejor modelo para el razonamiento basado en sentido común que incluso los demostradores de teoremas [Pap91].

2.4 Casos de Prueba

Existen dos clases principales de casos de prueba para evaluar el desempeño de los algoritmos para resolver SAT: Problemas específicos, los cuales pueden ser codificaciones de problemas del "mundo real"; y problemas generados aleatoriamente, los cuales pueden ser muy difíciles de resolver pero no tienen una aplicación práctica conocida. Las subsecciones siguientes tratan estas dos clases de casos de prueba respectivamente.

2.4.1 Problemas Específicos

Actualmente, existen solamente dos colecciones de problemas específicos SAT, ampliamente difundidos. La primera fue creada por Mitterreiter y Randermacher en 1991 [MR91], y la

segunda fue creada en colaboración con el Segundo Desafío DIMACS en 1993. Esta colección consiste de 41 instancias de problemas de los cuales 27 son satisfactibles. Los tamaños varian y el más grande es de $7,250$ variables y $454,622$ cláusulas.

Los problemas de Mitterreiter y Randermacher están disponibles via FTP anónimo[4]. Los problemas del DIMACS son enviados por los participantes del desafío y son seleccionados por los miembros del comité organizador[5].

2.4.2 Problemas Generados Aleatoriamente

Por la forma en que se generan este tipo de casos de prueba no es extraño que la distribución de probabilidad empleada sea crucial para respaldar la validez de cualquier investigación que emplee este tipo de fórmulas. Cabe mencionar que algunas distribuciónes proporcionan una mejor fuente de material de prueba que otras, y que es preferible emplear fórmulas que se encuentran en la región "dura", misma que se cree está asociada con la transición entre la satisfactibilidad y la insatisfactibilidad y se piensa ocurre según se aumenta el número de cláusulas [ML96].

En términos generales podemos clasificar a los generadores de casos de prueba en tres grupos. Cada uno de los cuales tiene tres parámetros: El número de variables n, el número de cláusulas m y una función de distribución de probabilidad p sobre el número de literales en las m cláusulas. Otro parámetro también empleado es la relación entre cláusulas y variables $c = m/n$. Los grupos son los siguientes:

- *k-SAT Aleatorio*: Este tipo es el más ampliamente utilizado. En él las cláusulas son seleccionadas usando una función de distribución uniforme con remplazo del conjunto de todas las $2^k\binom{n}{k}$cláusulas no triviales[6] de longitud k, definidas sobre las n variables.

- *SAT de Densidad Constante*: En el caso más simple, una cláusula se construye al incluir cada una de las $2n$ literales con una probabilidad p. La longitud de las cláusulas está distribuida binomialmente, con un valor esperado de $2np$. Dado que las cláusulas vacias, unitarias y triviales son fácilmente simplificables, la mayoría de los investigadores utilizan ciertas variantes, que se generan a partir de evitar estos tipos de cláusulas. Si estos tipos

[4] *dimacs.rutgers.edu (128.6.75.16)* bajo el directorio *pub/challenge/sat/benchmarks/faw.cnf.Z.*
[5] Están disponibles via FTP en *dimacs.rutgers.edu* bajo el directorio *pub/challenge/sat/benchmarks.*
[6] Una cláusula es trivial si contiene una variable y su negación.

no se permiten el tipo de problema se conoce como εk-*SAT Aleatorio* (ε denota que k es un valor esperado).

- $[k, l]$-*SAT*: Es un caso donde la longitud de las cláusulas está distribuída uniformemente sobre un rango determinado entre $[k, l]$.

2.4.3 Tamaños de los Casos de Prueba

Los casos de prueba que han sido usados para medir el desempeño de los algoritmos desarrollados para resolver el problema SAT, varian mucho dependiendo del algoritmo que se desea probar. Por ejemplo en [BS92] se reportan problemas con un número de variables entre 50 y 500 y de 215 a 2150 cláusulas, dichos problemas son empleados para probar el algoritmo 3-GSAT. En [BF97], se reportan pruebas realizadas con un algoritmo de dos fases para MAX-2-SAT y MAX-3-SAT, utilizando para ellas casos de prueba donde el número de variables oscila entre 50 y 150 y el de cláusulas entre 100 y 600. Otro ejemplo se puede observar en [JMB91], donde se analiza el desempeño de un algoritmo de Ramificación y Poda (*Branch and Cut*) para resolver problemas MAX–2-SAT y MAX–3-SAT, donde los casos de prueba tienen hasta $25,409$ variables y $26,041$ cláusulas.

2.5 Comentarios Sobre el Problema SAT

Actualmente son muy variadas las técnicas utilizadas para la solución del problema SAT, como ejemplos podemos mencionar el recocido simulado [Spe93], las redes neuronales [Spe96], y búsqueda tabú [MSG97].

Por otra parte se ha detectado que los algoritmos genéticos no han sido probados a profundidad y comparados contra otras técnicas de solución del problema SAT, por lo que representan un nuevo y amplio campo de investigación.

Capítulo 3

Algoritmos Genéticos

Retomando lo expuesto en el capítulo anterior, es necesario comentar que uno de los métodos incompletos más empleados para la solución de problemas NP-Completos y uno de los que más interés ha despertado en la comunidad de investigadores, es el de los algoritmos genéticos (AG). Esto se debe principalmente al éxito que han tenido en la solución de problemas duros donde el espacio de posibles soluciones es muy grande. En general, los algoritmos genéticos consisten de una población de individuos compitiendo por su supervivencia, misma que está basada en su aptitud con respecto a algún ambiente definido. El algoritmo procede en pasos llamados generaciones. Durante cada generación, una nueva población de individuos es creada a partir de la anterior vía la aplicación de operadores genéticos (cruza, mutación, etc.), y evaluada como solución al problema dado. Debido a la presión efectuada por la selección, la población se adapta al ambiente a través de generaciones donde se garantiza que la aptitud promedio de estas mejora; característica que provoca una evolución hacia mejores soluciones [Gol89].

Existen algunos conceptos importantes que deben ser considerados al momento de implementar un algoritmo genético, los más importantes son: la representación del problema, el tamaño de la población, la inicialización de la población, la función de aptitud a emplear, el criterio de selección de individuos, los operadores genéticos que se utilizarán, las probabilidades de aplicación de los operadores genéticos, así como los criterios de paro.

Un punto muy importante en la utilización de AG para la solución de problemas NP-Completos, es que aun cuando todos los problemas NP-Completos son de una dificultad equivalente, hablando en un sentido computacional general, algunos tienen una representación más

directa en AG, lo que permite que al utilizar esta técnica sea posiblemente más fácil obtener buenos resultados.

3.1 Computación Evolutiva

Durante los últimos años, el interés en algoritmos basados en un enfoque evolutivo se ha incrementado. Dentro de esta categoría tenemos a los AG, la búsqueda tabú y el recocido simulado. Los AG han mostrado ser muy robustos y funcionan adecuadamente en dominios amplios [Gol89]. En particular, se ha visto que estos algoritmos resultan adecuados en problemas de optimización combinatoria y como un mecanismo para descubrir reglas en aplicaciones de sistemas de aprendizaje. Su nombre se deriva del hecho de que están basados en modelos de cambio genético sobre una población de individuos. Sus principales características son las siguientes:

- El operar con una población de soluciones potenciales
- La utilización de un mecanismo de selección basado en la aptitud de cada una de las soluciones potenciales. Donde este mecanismo permite definir la supervivencia de una solución o en su defecto de características de la misma (noción Darwiniana de aptitud o *fitness*)
- El uso de operadores genéticos sobre los individuos seleccionados para determinar la configuración genética de sus descendientes y así construir una nueva generación.

Estos algoritmos han dado lugar a diferentes esquemas teóricos que han sido genéricamente agrupados para conformar una nueva área llamada computación evolutiva. Esta puede ser subdividida en las siguientes ramas:

- **Estrategias Evolutivas (EE).** Estas se caracterizan por el uso de una representación de las soluciones usando números reales. Además emplean como operadores modificaciones gausianas [Sch81].
- **Programación Evolutiva (PE).** En este enfoque, los miembros de la población son máquinas de estado finito [LFW66].

- **Algoritmos Genéticos (AG).** Una característica importante de este enfoque es el uso de operadores genéticos inspirados en la genética natural: mutación, cruce e inversión [Hol75].

- **Programación Genética (PG).** Los miembros de la población son programas completos cuya aptitud es medida de acuerdo a la exactitud de resolver un problema específico [Koz91].

3.2 Componentes de los Algoritmos Genéticos

Los algoritmos genéticos (AG) fueron propuestos originalmente por Holland [Hol75], pero la difusión de los algoritmos genéticos se debe principalmente al trabajo publicado por Goldberg [Gol89]. A continuación se muestra el pseudocódigo de un AG.

Procedimiento AG

t← *0*

inicializa P(t)

evalúa P(t)

Mientras *(no se cumpla condición de terminación)*

$t \leftarrow t + 1$

selecciona P(t) a partir de P(t − l)

aplica cruce sobre P(t) con una probabilidad de cruce

aplica mutación sobre P(t) con una probabilidad de mutación

evalúa P(t)

Fin Mientras

Fin AG

Este tipo de algoritmos son de naturaleza estocástica y mantienen una población de elementos *P(t).* Tienen una gran habilidad de explotar información acumulada acerca de un espacio

de búsqueda que inicialmente era desconocido. Esto se debe a que guían sus búsquedas subsecuentes a subespacios útiles. La población inicial generalmente es creada de forma aleatoria. Cada elemento de la población representa una solución potencial para el problema que está siendo resuelto. Los elementos de la población o individuos son codificados utilizando para ello alguna representación especial, la cual se conoce como *genotipo*; cuando este es expuesto a un ambiente obtenemos entonces el *fenotipo* del individuo, el cual indica su aptitud, cabe mencionar que se da preferencia a los más aptos. Los elementos seleccionados son *alterados* empleando un conjunto de operadores específicos que varia dependiendo del enfoque evolutivo seleccionado. Estas alteraciones por lo general se efectúan a nivel genotípico. De esta forma, la nueva generación es expuesta al ambiente y se itera hasta que se alcance la condición de terminación [Tor97].

Las principales diferencias de un AG comparado con los métodos tradicionales de búsqueda y optimización son las siguientes:

- Los AG trabajan con un conjunto de parámetros codificados y no con los parámetros mismos
- Inician la búsqueda desde un conjunto de puntos, en vez de comenzar con uno solo.
- Utilizan una función a optimizar en lugar de la derivada de la función u otro conocimiento adicional
- Estos algoritmos usan reglas de transición probabilísticas, en vez de usar reglas determinísticas

Algunos de los principales puntos a considerar al momento de realizar la implementación de un algoritmo genético son los siguientes:

- *Representación*
- *Tamaño de Población*
- *Inicialización de la Población*
- *Función de Aptitud*

- *Criterio de Selección*
- *Operadores Genéticos*
- *Probabilidades de Aplicación de los Operadores Genéticos*
- *Criterios de Paro*

En las secciones subsecuentes se explicarán más a detalle cada uno de estos importantes conceptos.

3.2.1 Representación

La representación de un problema es considerada crucial para el éxito de un AG. Tanto es así que al realizar una determinada implementación uno se debe cuestionar si la representación del problema es adecuada para un AG [Dav91]. La representación más usada es la representación binaria [Hol75]. Pero está reportado el uso de representaciones usando un formato de números enteros y reales [Mic92].

Para tener una idea clara de cómo se puede aplicar un AG a un problema en particular veamos algunos casos especiales de representación:

Un Parámetro Discreto

Si el problema a resolver sólo requiere de un parámetro discreto, la manera más directa de representarlo es a través del uso de una cadena de bits que mapee el espacio de los valores posibles. El tamaño del espacio a ser representado es: $z = X_u - X_l$ y el tamaño de la cadena de bits que se requiere es:

$$\lceil log_2(X_u - X_l + 1) \rceil$$

donde X_u y X_l indican los valores inferior y superior del parámetro. Para mapear un valor X a su correspondiente cadena de bits se usa la expresión:

$$Bin\,(X - X_I)$$

donde Bin indica la conversión de su argumento a una cadena de bits. El mapeo de una cadena de bits S a su valor equivalente es realizado usando la siguiente expresión:

$$X_l + Dec(S)$$

donde Dec indica la conversión de la cadena de bits a un número decimal.

Un Parámetro Real

Si el problema requiere un parámetro real la representación es similar a la anterior. El tamaño del espacio a ser representado es:

$$z = X_u - X_l$$

y el número de bits necesarios es dado por la fórmula:

$$\left\lceil log_2 \left(\frac{X_u - X_l}{P}\right) \right\rceil$$

donde X_u y X_l representan los valores inferior y superior del parámetro y P indica la precisión deseada. Por ejemplo $P = 0.01$ indica que el espacio a ser representado se dividirá en 100 partes.

Para realizar el mapeo de un valor X a una cadena de bits, se necesita definir que hacer cuando un mapeo particular cae entre dos subdivisiones. Típicamente un redondeo o truncamiento es usado; las siguientes expresiones indican estas dos opciones:

$$Bin\left(Round\left(\frac{X - X_l}{P}\right)\right) \text{ o } Bin\left(Trunc\left(\frac{X - X_l}{P}\right)\right)$$

donde *Round* y *Trunc* indican las operaciones de redondeo y truncamiento. Para obtener el valor real de una cadena de bits S se usa la siguiente expresión:

$$X_l + P * Dec(S)$$

Una variante de esta representación consiste en una codificación en dos partes, una indicando los dígitos significativos y otra indicando el exponente [Tor97].

Varios Parámetros

Si un problema particular requiere varios parámetros $X_n, ..., X_l$ y los tamaños del espacio de cada parámetro son denotados por $z_n, ..., z_l$ es posible usar alguno de los siguientes enfoques:

Parámetro	$X_{\mathbf{u}}$	$X_{\mathbf{l}}$	z	*Expresión de Valor Posicional*	*Valor Posicional*
1	7	4	3	1	1
2	10	6	4	$z_1 + 1$	4
3	5	3	2	$(z_1 + 1) * z_2 + 1$	17
4	6	1	5	$((z_1 + 1) * z_2 + 1) * z_3 + 1$	35

Tabla 3.1: Representación Posicional para Cuatro Parámetros Discretos

1. Tratar los parámetros como un vector y obtener la representación como el resultado de concatenar n representaciones de un solo parámetro [Mic92].

2. Tratar los parámetros como un sistema posicional en el cual los *dígitos* son los valores de cada parámetro, y el valor posicional es obtenido usando una función que involucre los tamaños de los espacios de cada parámetro $(z_n, ...z_1)$, de este modo podemos convertir los n parámetros a un solo valor [Tor97].

La manera en la que se implementa el primer enfoque es obvia, pero no es este el caso para el segundo. Supongamos que un problema particular requiere cuatro parámetros discretos de acuerdo a la Tabla 7.1. Entonces, una posible representación de los parámetros $X_1 = 6$, $X_2 = 8$, $X_3 = 4$, $X_4 = 5$ sería:

$$(6 - 4) * 1 + (8 - 6) * 4 + (4 - 3) * 17 + (5 - 1) * 35 = 167$$

finalmente tenemos un problema de representación de un sólo parámetro.

Representación Usando Codificación Gray

Las representaciones binarias tienen la desventaja de que algunos números contiguos son mapeados a cadenas de bits que son completamente diferentes. Por ejemplo, el número 7 y el número 8 al codificarse generan cadenas de cuatro bits: 0111 y 1000, que son totalmente diferentes. Una estrategia para eliminar este problema consiste en usar codificación Gray [Mic92], cuya principal ventaja consiste en que todos los número contiguos difieren a lo más en un bit.

3.2.2 Tamaño de la Población

A la capacidad de un algoritmo de examinar gran parte del espacio de búsqueda, se le denomina *exploración*, y a la capacidad de buscar en la vecindad de una región del espacio de búsqueda se

le denomina *explotación*. De la misma manera que otros algoritmos de búsqueda, un AG trata de crear un balance entre la exploración y la explotación. Este balance se establece a través del uso de operadores genéticos adecuados, de fijar en un tamaño correcto la población y del criterio de selección empleado.

El tamaño de la población debe establecer un balance. Una población pequeña provoca mayor explotación, y menor exploración por lo que se aumenta el riesgo de converger a un máximo local. Por otra parte un tamaño de población muy grande indica mayor exploración y menor explotación, pero además emplea muchos recursos computacionales. Un criterio que se utiliza para establecer un tamaño adecuado de la población es el número de bits que requiere una representación particular, de esta forma a un mayor número de bits mayor será el tamaño de la población. Está reportado en [Gol85] que una aproximación para el tamaño adecuado está dada por el doble del número de bits de la representación. En la práctica la mayoría de las implementaciones usan un tamaño de población menor al propuesto por Goldberg, debido principalmente a las limitaciones del tiempo de cómputo necesario para obtener una solución aceptable, aun cuando esto puede limitar la cantidad de soluciones obtenidas.

Existen dos variantes empleadas en el manejo del tamaño de la población al momento de implementar un AG, la primera, y más usada, mantiene el tamaño de la población fijo; la segunda variante usa un tamaño de población que cambia dinámicamente [JSD89] [CM91].

3.2.3 Inicialización de la Población

En general la población inicial para un AG es creada de manera aleatoria, y aun cuando se disponga de conocimiento relativo al problema que permita crear individuos más aptos, es recomendable que una fracción de la población sea creada aleatoriamente para tener una mayor diversidad genética.

3.2.4 Función de Aptitud

La función de aptitud es la base para determinar cuáles soluciones tienen mayor o menor probabilidad de sobrevivir, ya que determina el ambiente al cual es expuesta cada una de las soluciones. Junto con la representación, son las características más importantes y que deben ser bien estudiadas al momento de realizar la implementación de un AG. El punto crítico de

una función de aptitud radica en encontrar un equilibrio entre una función que haga diferencias muy grandes entre los individuos, lo que provoca *convergencia prematura*, y una función que haga diferencias muy pequeñas, provocando estancamiento en la búsqueda de soluciones.

Una de las causas de la convergencia prematura es una función de aptitud mal diseñada. Si un elemento de la población tiene un desempeño muy superior a los demás, este podría en unas cuantas generaciones dominar, en otras palabras, todos los individuos de la población serían similares a él. La convergencia prematura detiene la evolución del AG, por lo que debe tratar de evitarse, para ello se siguen varias estrategias, una es inyectar diversidad genética cuando sea necesario, creando una nueva población con una parte aleatoria, otra es utilizar un mecanismo que mantenga la diversidad genética a través de la creación de nichos como lo propone Goldberg en [Gol89].

3.2.5 Criterios de Selección

La manera en la cual los elementos de una población son seleccionados para ser alterados haciendo uso de operadores genéticos, y así construir la siguiente generación es conocida como el criterio de selección de un AG. Un criterio de selección, para su buen funcionamiento debería preferir a los elementos con desempeño arriba del promedio. Los criterios de selección más empleados se describen a continuación:

- **Selección de Rueda de Ruleta:** Esta clase de selección usa la aptitud de cada elemento y calcula su probabilidad acumulada de ser seleccionado. Usando para ello un generador de números aleatorios uniformemente distribuidos, que determina cuáles elementos constituirán la base de la siguiente generación [Gol89].

- **Selección de Torneo**: Opera formando grupos de t elementos, (t es el tamaño del torneo), y entonces se selecciona al elemento con mejor aptitud de cada grupo, por esta razón el número de grupos debe ser igual al tamaño de la población [Tor97].

- **Selección por Rango:** Los elementos se ordenan de acuerdo a su aptitud. El número de veces que es seleccionado cada elemento, como base de la siguiente generación, está determinado por su posición relativa a los demás y no directamente por su aptitud [BT95].

3.2.6 Operadores Genéticos

Los operadores genéticos clásicos definidos por Holland [Hol75] son:

- Cruce de un solo punto
- Mutación de un solo bit
- Inversión de una subcadena de bits a partir de dos posiciones dadas

Estos operadores tienen un fundamento matemático muy sólido, pero no funcionan adecuadamente para algunas representaciones, ya que producen individuos repetidos. Por esta razón existen números investigadores dedicando sus esfuerzos a encontrar otro tipo de operadores. Algunos ejemplos de operadores especiales de cruce son: OX [Dav85] y ERX [DWF89].

3.2.7 Probabilidades de Aplicación de los Operadores Genéticos

En la práctica, en general, el operador de mutación tiene una pequeña probabilidad de ser aplicado mientras que el de cruce tiene una probabilidad alta de usarse. Normalmente las probabilidades permanecen fijas, aunque, según se describe en [Tor97], éstas pueden variar en forma dinámica.

3.2.8 Criterio de Paro

El criterio de paro de los AG puede ser: un número máximo de ciclos o generaciones, después de un cierto período de tiempo, después de encontrar una solución determinada de calidad, cuando el AG ha convergido, es decir, un gran porcentaje de la población tiene características similares, o cuando el algoritmo no genera nuevas soluciones mejoradas durante un determinado número de ciclos [Gol89][JS89].

3.3 Algoritmos Genéticos y el Problema SAT

Sin duda, es difícil imaginar un problema que tenga una mejor representación en AG que el problema SAT; ya que consiste en una búsqueda sobre n variables booleanas, lo que resulta en un espacio de soluciones de tamaño 2^n, de tal forma que para representar este espacio es muy

natural utilizar una codificación de una cadena binaria de longitud n, donde Verdadero es 1 y 0 es Falso y el i-ésimo bit de esta cadena es el valor de verdad de la i-ésima variable booleana de la expresión, además esta representación es libre de contexto, en el sentido de que el significado de un bit no se ve afectado al cambiar el valor de otros bits de la cadena.

Tomando en cuenta la ventaja de que los problemas NP-Completos pueden ser considerados equivalentes, en el sentido de que cualquier problema NP-Completo puede ser transformado a otro NP-Completo en tiempo polinomial [GJ79]; así como la buena representación que el problema SAT tiene en términos de AG. Se percibe que una posible estrategia a seguir consiste en la identificación de un problema NP-Completo canónico, en el cual los AG se comporten favorablemente en la búsqueda de soluciones, y resolver otros problemas NP-Completos indirectamente al mapearlos hacia el problema canónico identificado. Claramente el problema SAT, representa un buena opción de problema canónico.

Lo anterior deja de manifiesto la importancia computacional que tiene el encontrar un algoritmo eficiente que nos permita resolver el problema SAT, ya que un buen desarrollo de éste implicaría la posibilidad de resolver otro tipo de problemas también NP-Completos, los cuales no tienen una representación en AG, tan directa como el problema SAT.

Un buen ejemplo de la utilización de esta transformación de problemas, es mostrado en [JS89], donde se realiza una transformación del problema de circuitos hamiltonianos (CH) a un problema SAT, para posteriormente resolverlo con AG. El problema de CH consiste en encontrar un camino dentro de un grafo dirigido que pase por todos los nodos exactamente una vez. Naturalmente, si un grafo está conectado por completo, el problema de CH se vuelve trivial. Sin embargo, a medida que los arcos van siendo removidos el problema se vuelve mucho más complejo, en general este problema es considerado como NP-Completo. La definición del problema implica que para cualquier solución, cada nodo debe tener exactamente un arco de entrada y uno de salida. Si cualquier recorrido viola esta restricción, éste no puede ser solución. Por lo tanto un expresión booleana equivalente es simplemente la conjunción de los términos indicando la combinación válida para cada nodo. Las asignaciones para las variables de los arcos indican cuales forman parte del recorrido, donde un valor de 1 denota que un arco es incluido y de 0 si no. El computo de esta transformación es realizado en un tiempo polinomial, y la solución para un problema de CH existe, si y sólo si, la expresión booleana que lo representa es

satisfactible.

Otro ejemplo interesante, que muestra la robustez de esta técnica propuesta, es el de decisión de factorización (DF), debido principalmente a que es un problema identificado como NP-Completo y para el cual pocos algoritmos especializados han sido desarrollados. Muchos de los sistemas criptográficos hacen uso de números primos y factorización [Riv78]. Según se reporta en [JS89], Dan Hoey desarrolló un algoritmo que convierte un problema de DF en un problema equivalente de SAT. Por ejemplo, un problema de la forma "¿Tiene el número 689 un factor de cuatro bits?", puede transformarse en una expresión booleana de 105 cláusulas con 22 variables y 295 literales.

Este tipo de ejemplos nos dan idea de la importancia y versatilidad de aplicaciones que tiene el problema SAT y en consecuencia el potencial de uso que tiene implícito este problema, desde el punto de vista, de la transformación canónica explicada, para resolver diversos tipos de problemas en diferentes áreas de aplicación.

3.4 Comentarios Sobre Algoritmos Genéticos

En este capítulo se ha hablado de las principales diferencias entre los AG y los métodos tradicionales de búsqueda y optimización, así como de los principales aspectos a considerar al momento de realizar la implementación de un AG para resolver un problema particular.

Cabe resaltar que al efectuar la investigación del estado del arte de los AG, se encontró que existen muy pocas referencias a la utilización de este tipo de métodos en la solución de problemas de satisfactibilidad. Las fuentes de información encontradas, corresponden al trabajo de Kenneth De Jong y William Spears [JS89] [Spe92].

Capítulo 4

El Problema de la Representación en AG

Como hemos visto en el capítulo anterior, al momento de implementar un AG con el fin de resolver un problema específico, deben tomarse en cuenta un conjunto de características que influyen importantemente en el desempeño del algoritmo. Pero sin duda la representación de un problema es considerada crucial para el éxito de un AG [Dav91].

Para el caso particular del problema SAT, que consiste en una búsqueda sobre n variables booleanas, el tipo de representación más natural a utilizarse es una codificación de una cadena binaria de longitud n.

Una vez que está claro que la representación con cadenas binarias es la mejor opción para resolver el problema SAT con AG, es necesario tomar en cuenta algunos otros conceptos básicos relacionados con la representación que también influyen en el desempeño de un AG. En las siguientes secciones se analizan a detalle estos conceptos con el fin de resolver el problema de la representación del problema SAT usando AG.

4.1 Esquemas

Los AG, a diferencia de otros algoritmos de optimización, utilizan en el proceso de búsqueda de una solución no sólo los valores de las medidas de aptitud, sino también las similitudes que existen entre ciertos patrones que siguen las cadenas con aptitud alta, es decir, cambian el énfasis

de la búsqueda de cadenas completas por el descubrimiento de cadenas parcialmente adaptadas. Por lo anterior es útil estudiar las semejanzas de estas cadenas junto con sus correspondientes medidas de aptitud, y especialmente investigar las correlaciones estructurales de cadenas con medidas de aptitud excepcionales.

Formalmente, se definen las semejanzas entre cadenas mediante un *esquema*. Un esquema (sobre el alfabeto binario sin que esto signifique perdida de generalidad) es una cadena del siguiente tipo:

$$(a_1, a_2, ..., a_i, ..., a_l),\ a_i \in \{0, 1, *\}.$$

El símbolo "$*$" se usa para representar un comodín que puede tomar los valores de 0 y 1. Un esquema es una plantilla que describe un subespacio de cadenas [Gol89]. Por ejemplo $S_1 = (11 * 00*)$ es un esquema en una población de cadenas de longitud $k = 6$ que representa las cadenas: $111001, 111000, 110001, 110000$. Por lo tanto, entre más símbolos "$*$" contenga un esquema este se vuelve menos específico, es decir, describe más cadenas. Es necesario recalcar que el símbolo "$*$", no es explícitamente procesado por el algoritmo genético.

Algunos otros conceptos importantes en el estudio de la teoría de esquemas son: El *orden de un esquema* $o(S)$, que se define como el número total de símbolos de la cadena menos el número de comodines $(*)$, por lo tanto $o(S_1) = 4$; por otra parte la *longitud de un esquema* $\delta(S)$ es la distancia entre los dos símbolos más distantes del esquema que no sean comodines, para el ejemplo $\delta(S_1) = 4$.

En la siguiente sección hablaremos del teorema fundamental de los algoritmos genéticos.

4.2 Teorema de Esquemas

Este teorema es considerado la base teórica de los algoritmos genéticos y permite explicar porque son capaces de resolver una gran diversidad de problemas. Sus fundamentos matemáticos giran en torno a la observación de que al evaluar la aptitud de un individuo de la población, se está derivando información implícita del esquema que describe esa cadena. La confiabilidad de esta extrapolación depende del grado de especificidad del esquema dado. Al observar un AG a nivel básico, puede observarse que el proceso de búsqueda de una solución es visto a través del espacio de cadenas. Sin embargo, la esencia misma del teorema de esquemas, muestra que uno puede

también observar el proceso de cambio de poblaciones como una búsqueda sobre el espacio de esquemas de los cuales los individuos fueron instanciados. Dado que cada cadena es una instancia de 2^l posibles esquemas, donde l es la longitud de la cadena, al probar la aptitud de esta, también se deriva una gran cantidad de información implícita concerniente a la "aptitud" de los esquemas a los que pertenece. Holland llamó a este hecho *paralelismo implícito*, el cual es una de las partes más importantes en la explicación del poder de los AG, y significa que el esfuerzo de búsqueda es dirigido simultáneamente en muchos hiperplanos del espacio.

En general el *teorema de esquemas* puede enunciarse como sigue:

> *Los esquemas pequeños de bajo orden con aptitud arriba del promedio reciben un incremento exponencial de representantes en las siguientes generaciones de un algoritmo genético[Hol75].*

Esos esquemas reciben el nombre de *bloques de construcción (building blocks).*

4.3 Epístasis

Comúnmente los investigadores se preguntan qué dominios de problemas son apropiados para resolverse usando algoritmos genéticos. Pero dado que los AG no trabajan directamente con el dominio del problema, debido a que este último es substituido por la representación, la pregunta debería reemplazarse por la siguiente: ¿La representación (del problema) promueve la eficiencia del AG?. Esta pregunta además de ser más consistente con el teorema de esquemas, también es más clara en cuanto a su significado.

El teorema de esquemas de Holland, como hemos visto, enuncia los prerrequisitos que un esquema debe poseer para utilizar una búsqueda con AG. Estos son, una estructura de cromosomas compuestos por bloques de construcción cortos y de bajo orden. Sin embargo, no provee las herramientas necesarias para evaluar la conveniencia de una representación, ya que no toma en cuenta el efecto que tiene la interdependencia entre los genes de un cromosoma sobre el desempeño de los AG.

La interacción de genes en los cromosomas de un AG, de igual forma que en la genética natural, es un aspecto de gran importancia que se conoce como *epístasis.* Este término se emplea para describir la situación donde la ausencia o presencia de ciertos genes activa o desactiva la

expresión de características fenotípicas de otro gen [Dav91]. Cuando la epístasis de la representación de un problema es alta, la generación de bloques de construcción no se llevará a cabo [Gol89], y el AG tendrá gran dificultad para encontrar buenas soluciones.

El efecto de la epístasis de una representación sobre el desempeño de un AG, puede clasificarse en tres casos:

- Cuando la epístasis es alta, el operador genético de cruza produce hijos que son fenotípicamente diferentes a los padres, por lo que un algoritmo de búsqueda aleatoria posiblemente se desempeñe mejor que un AG.

- Si la representación tiene una epístasis muy baja, es posible que un AG se comporte aceptablemente. Sin embargo, un algoritmo de ascensión de colinas (*hill climbing*) tiene generalmente mejor desempeño.

- Bajo una representación con epístasis media un AG se desempeña muy bien y es posible que supere tanto al algoritmo de búsqueda aleatoria como a uno de ascenso de colinas.

Ahora que estamos en posibilidades de apreciar que existen factores adicionales que afectan el desempeño esperado de un AG además de la naturaleza de los bloques de construcción, podemos concluir que si la epístasis para una determinada representación puede ser medida, entonces ésta ofrecerá una importante medida de la conveniencia de utilizar un AG.

En la siguiente sección se plantea la posibilidad de utilizar un algoritmo que permita realizar el reacomodo de las variables de un problema SAT, con el fin de reducir su epístasis y de esta forma lograr un mejor desempeño al resolver el problema usando AG.

4.4 Preprocesamiento de Problemas SAT

La posibilidad de un preprocesamiento de los problema SAT para reducir su epístasis, y resolverlos exitosamente mediante AG, es una idea fundamentada en [Hol75], donde se reporta que un AG procesa eficientemente esquemas cortos de bajo orden con aptitud arriba del promedio. Por esta razón, si bajo una representación particular los bits relacionados no están cercanos, se espera que el AG no se desempeñe adecuadamente bajo esa representación.

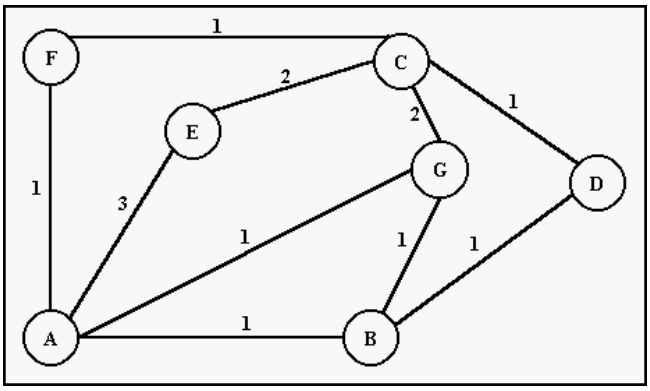

Figura 4-1: Grafo pesado que representa un problema SAT

Basandose en lo anterior la étapa de procesamiento previo, deberá tener la finalidad de realizar una transformación la cual consiste en reordenar las variables para cumplir con la restricción de que las variables relacionadas estén en posiciones cercanas dentro del cromosoma; de esta forma el desempeño esperado de un AG que resuelva el problema aumentará.

Una forma de lograr este objetivo es representando el problema SAT mediante una matriz de relaciones entre las variables del problema, que también puede verse como un grafo pesado. Un ejemplo se muestra a continuación: Dado un problema SAT en formato CNF:

$$(C \vee D \vee E) \wedge (C \vee \sim G \vee E) \wedge (A \vee E \vee G) \wedge (C \vee F \vee G) \wedge (\sim A \vee B \vee \sim C) \wedge$$
$$(A \vee B \vee \sim E) \wedge (A \vee \sim E \vee \sim F) \wedge (\sim B \vee D \vee \sim G)$$

El grafo pesado resultante se muestra en la Figura 4-1. En este caso el peso sobre cada arco corresponde al número de veces que está relacionada la variable i con la variable j. Bajo este enfoque el reordenar las variables es equivalente a resolver el problema de minimización del ancho de banda del grafo (PMABG). Por lo anterior, se realizó una investigación con el fin de evaluar los algoritmos más comunes para resolver este problema, y así estar en posibilidades de implementar el que mejor se ajuste a las necesidades específicas de este proyecto.

Es importante mencionar que el algoritmo de preprocesamiento debe cumplir un compromiso entre calidad de la solución entregada y el tiempo de computo empleado para lograrla. La meta es llegar al orden bandeado óptimo o sub-óptimo haciendo un esfuerzo computacional que sea menor que el resolver el problema original, ya que de lo contrario esta etapa no proporcionaría

ventajas en cuanto a tiempos totales de cómputo para resolver el problema de satisfactibilidad.

4.5 Comentarios sobre el Problema de la Representación en AG

En este capítulo se presentaron los prerequisitos que una representación debe poseer para utilizar una búsqueda con AG según el teorema de esquemas de Holland y el efecto que tiene la interdependencia entre los genes de un cromosoma sobre el desempeño de los AG (epístasis).

Además se planteó la idea de realizar un preprocesamiento de problemas SAT (utilizando un algoritmo para resolver el PMABG), que consiste en reordenar las variables para cumplir con la restricción de que las que están más relacionadas se encuentren en posiciones cercanas dentro del cromosoma, es decir, disminuir su epístasis, de este modo se espera que un AG pueda resolver de una mejor manera el problema SAT dado.

En el siguiente capítulo presentamos un estudio del PMABG.

Capítulo 5

El PMABG

En el capítulo anterior se planteó la idea de implementar un algoritmo de preprocesamiento de problemas SAT, que consiste en reordenar las variables con el fin de disminuir su epístasis. Esto significa resolver el PMABG asociado y de este modo solucionar de una mejor forma, haciendo uso de AG, el problema SAT dado.

En este capítulo analizaremos a fondo el problema de minimización del ancho de banda de grafos, así como una seríe de algoritmos para resolverlo.

5.1 Definición del PMABG

Formalmente, el problema el PMABG se define de la siguiente forma: Sea $G = (V, E)$ un grafo finito no dirigido, donde V define el conjunto de los vértices (etiquetados de 1 a N) y E el conjunto de arcos. Además $\tau = \{\tau_1, \tau_2, ..., \tau_N\}$ es un acomodo lineal de G, tal que es una permutación sobre $\{1, 2, ...N\}$, donde τ_i denota el etiquetado de los vértices que originalmente estaban identificados con la etiqueta i. El ancho de banda β de G para un etiquetado τ es:

$$\beta_\tau(G) = Max_{\{u,v\}\in E}|\tau(u) - \tau(v)|. \tag{5.1}$$

Entonces el PMABG puede ser definido como encontrar un etiquetado τ para el cual $\beta_\tau(G)$ es mínima [Tor97].

El primer trabajo que presentó un algoritmo para resolver el PMABM fue publicado en 1965 [AM65], en el se mostró que bandeando una matriz se mejoraba sensiblemente el desempeño de los algoritmos de solución. Posteriormente se atacó el PMABM modelándolo como un grafo [Har64], [Har67], para posteriormente identificarlo como el PMABG. Este problema comúnmente se resuelve representando el grafo como una matriz de $N \times N$, donde el grafo equivalente tendrá N vértices y un número de arcos correspondiente al número de elementos diferentes de cero de la matriz. La reducción del ancho de banda se realiza permutando dos renglones y dos columnas de la matriz, con lo cual no se modifica la estructura topológica del grafo.

5.2 Complejidad Computacional del PMABG

Los problemas de la clase polinomial (P) son aquellos para los que existen algoritmos para solucionarlos, cuyo orden está acotado por un polinomio, pero existen una clase de problemas no polinomiales (NP) que se caracterizan porque no es conocido un algoritmo con orden polinomial capaz de resolverlos. Por su parte los problemas no polinomiales completos (NP-Completos) [PS82][GJ79] tienen las siguientes características:

- No existe hasta el momento un algoritmo de tipo polinomial capaz de resolver ningún problema de esta clase.
- Si uno solo de los problemas de esta clase puede ser resuelto por un algoritmo de tipo polinomial, entonces todos los problemas podrán ser resueltos por algoritmos de tipo polinomial.

Cuando fue definido el PMABG, se pensaba que sería posible construir un algoritmo polinomial para resolverlo de manera óptima. Sin embargo, los algoritmos que se propusieron sólo funcionaban aceptablemente para ciertos tipos de grafos [Tur82]. El primer trabajo que ayudó a entender la complejidad computacional del PMABG fue desarrollado por Papadimitriou, quien demostró que el problema de decisión asociado al PMABG es un problema NP-Completo [Pap76]. Posteriormente fue demostrado en [MGK78] que el PMABG es NP-Completo incluso para árboles con un grado máximo de tres.

5.3 Algoritmos Para Resolver el PMABG

Existe un gran número de algoritmos reportados para atacar el PMABG, comúnmente para su estudio se dividen en exactos[1] y no exactos[2]. Algunos de los más utilizados se presentan a continuación:

5.3.1 Algoritmos Exactos

En [MGK78] se demuestra que el PMABG, para el cual el ancho de banda es dos o menos, puede ser resuelto en tiempo lineal . En [GS84] se reporta un algoritmo con orden $O(N^K)$ para probar si un grafo tiene un ancho de banda k.

5.3.2 Algoritmos no Exactos

En esta categoría se encuentran los algoritmos que presentan la característica de lograr encontrar soluciones aproximadas al PMABG sin requerir para ello de tiempos de computo excesivos. A continuación mencionaremos las características particulares de algunos de los más comunes.

Algoritmo Para Orugas

Un grafo de tipo oruga (*caterpillar* en inglés), es aquel que es creado por una trayectoria, llamada eje central, y muchas trayectorias, conocidas como cabellos, agregadas al eje central. En [JHM91] se encuentra reportado un algoritmo aproximado que garantiza encontrar una solución, la cual es a lo mucho $log(N)$ veces la solución óptima, para el caso especial de grafos de tipo oruga.

Algorimo Para Árboles Balanceados de Altura Generalizada k

En un trabajo reciente [HM97], se presenta un algoritmo aproximado que encuentra una solución que es a lo más $log(k)$ veces la solución óptima para un árbol balanceado de altura generalizada k. Un árbol de este tipo, es un árbol en el cual se cumple que en cualquier vértice la diferencia de altura de sus subárboles es cuando mucho k.

[1] Algoritmos que obtienen la solución óptima al problema.
[2] Algoritmos que no garantizan encontrar la solución óptima al problema.

Algoritmo de Cuthill-McKee

Los dos algoritmos más utilizado para el PMABG son por un lado el creado por Cuthill y McKee [CM69] y la modificación propuesta por George [GL81], la cual es identificada como el algoritmo RCM (*Reverse Cuthill McKee*). Este algoritmo se muestra a continuación:

1. Determinar dos vértices que están a una distancia máxima en el grafo.
2. Seleccione uno de los vértices anteriores como INICIO y asignarle el número 1.
3. Repetir desde $i = 1$ hasta N.

   (a) Determinar los vértices no-numerados que son vecinos del vértice numerado con i.

   (b) Numérense los vértices anteriores en forma secuencial siguiendo el orden de menos a mayor grado.
4. Renumere los vértices usando la expresión $N - j + 1$. Donde j indica el número que tiene asignado actualmente un vértice.

Algoritmo de Gibbs-Poole-Stockmeyer

Otro de los algoritmo que más se han usado es el conocido como *GPS* (Gibbs-Poole-Stockmeyer) [NGS76]. La función básica de este algoritmo consiste en la agrupación de los vértices en grupos denominados niveles y en la numeración de los vértices de acuerdo al grado dentro de cada nivel.

Este algoritmo ha mostrado ser el más rápido y entregar menores anchos de banda que el *RCM* bajo un amplio espectro de grafos según se reporta en [NGS76].

Algoritmo de Dueck y Jeffs

También existen algoritmos que utilizan recocido simulado para resolver el PMABG, según [Tor97], Dueck y Jeffs reportan en [DJ95] el uso de este tipo de algoritmos para resolver el PMABG para casos con menos de 200 vértices.

Algoritmo Genético Para el PMABG

En [Tor97], se reporta la implementación de un AG para el PMABG, el cual tiene un comportamiento superior a los algoritmos RCM y GPS bajo una extensa variedad de grafos.

5.4 Cotas Para el PMABG

Las cotas superior e inferior para el PMABG están basadas en invariantes de un grafo como son: el grado del grafo (Δ), el diámetro del grafo (D), el número de vértices (N), el ancho de banda (β) y el número de arcos (e) [Tor97]. A continuación se presentan algunas cotas reportadas para este problema:

En [Chv70] la siguiente expresión se encuentra reportada para definir las cotas superior e inferior:

$$\left[\frac{N-1}{D}\right] \leq \beta \leq N - D$$

Así como una expresión para definir la relación entre el ancho de banda, el número de vértices y el número de arcos de un grafo:

$$N - \frac{1+\sqrt{(2N-1)^2-8e}}{2} \leq \beta$$

Por otra parte en [Chv81], se demuestra que:

$$\frac{\Delta}{2} \leq \beta$$

5.5 Comentarios Sobre el PMABG

En este capítulo se mostró el PMABG y las técnicas más comunmente usadas para resolver este problema. Esto nos ha permitido tener bases suficientes para realizar la implementación del algortimo que mejor se ajusta a las necesidades específicas de este proyecto y así tener una etapa de preprocesamiento que se comporte de manera eficiente.

En el siguiente capítulo explicaremos a detalle el enfoque de solución propuesto para el problema SAT.

Capítulo 6

Enfoque de Solución

Este capítulo presenta una descripción detallada de la implementación computacional propuesta en esta tesis para resolver problemas SAT difíciles, la cual consiste de dos etapas, como puede observarse en la Figura 6-1. En la primera etapa se emplea un algoritmo de preprocesamiento que permitirá cambiar la representación del problema a resolver, con el fin de que en una segunda etapa sea posible utilizar eficientemente un algoritmo genético para solucionar el problema SAT.

El capítulo está organizado en dos secciones principales, dedicadas a cada una de las etapas mencionadas.

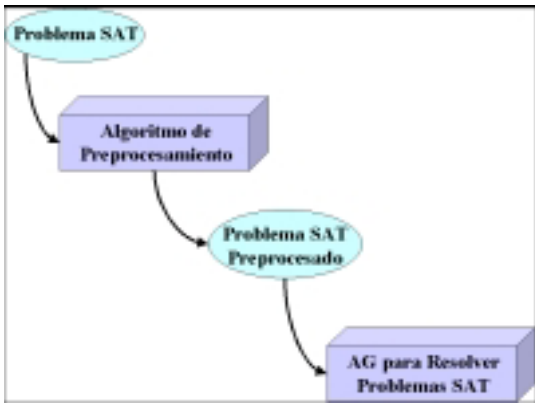

Figura 6-1: Esquema General del Proyecto

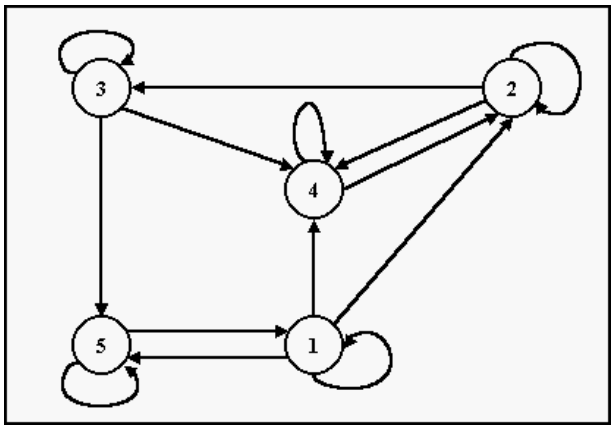

Figura 6-2: Grafo con $\beta = 4$

6.1 Algoritmo de Preprocesamiento

En esta sección se realiza el análisis del algoritmo de preprocesamiento propuesto, el cual se encuentra basado en un algoritmo para la minimización del ancho de banda de grafos pesados.

Se decidió emplear un algoritmo heurístico de *Recocido Simulado* (RS), principalmente por que los resultados presentados en [DJ95] demuestran que este enfoque permite reducir el ancho de banda de una amplia variedad de grafos en tiempos razonables. Analizando el trabajo de Duek y Jeffs se pudo observar que la medida empleada para medir el ancho de banda de los grafos tenía ciertas deficiencias. La más fuerte es que no es posible emplearse en grafos pesados. Por esta razón se desarrolló una medida alternativa (γ) que refleja más exactamente el cambio del ancho de banda de los grafos al momento de resolver el problema, a la vez de que permite bandear grafos pesados. En las siguientes subsecciones analizaremos tanto esta medida como el algoritmo de RS.

6.1.1 La Medida β

La medida β, explicada en el capítulo anterior, es la más utilizada por los algoritmos para resolver el PMAB [CM69] [NGS76] [AET] [JHM91] (una excepción es el trabajo de Dueck donde utiliza una medida en la cual no sólo β es usada, sino también son tomadas en cuenta las diferencias entre los vértices adyacentes cercanos a β [DJ95]). Por ejemplo, si deseamos

calcular β para el grafo de la Figura 6-2, es suficiente con identificar que vértices tienen la mayor diferencia absoluta entre sus etiquetas. Para este grafo en caso particular son los vértices 1 y 5, por lo que $\beta = 4$.

Dado que el número de vértices es N, entonces β puede tomar sólo N valores diferentes (de 0 a $N-1$), $\beta = 0$ implica un grafo que no tiene arcos o que sólo tiene arcos reflexivos. Así, el espacio de todas las posibles soluciones ($N!$) es particionado en N diferentes clases de equivalencia.

A continuación, se analizan algunas de las características de la medida β. Sea ω_i la cardinalidad de la clase de equivalencia con $\beta = i$. Entonces, $\omega_0 = 2^N - 1$ (se resta el caso de un grafo sin arcos) y $\omega_1 = 2^N(2^{2(N-1)} - 1)$ entonces:

$$\omega_i = \prod_{j=1}^{i-1} \left(2^{2(N-j)} \left(2^{2(N-i)} - 1\right)\right) \tag{6.1}$$

Ahora, para verificar que todos los posibles grafos han sido considerados, definiremos una sumatoria parcial sobre las cardinalidades de las clases de equivalencia $S_i = \sum_{j=0}^{i} \omega_i$; entonces $S_0 = \omega_0$, $S_1 = S_0 + \omega_1$, y

$$S_i = S_{i-1} + \omega_i = 2^N \prod_{j=1}^{i} \left(2^{2(N-j)} - 1\right) \tag{6.2}$$

Sin embargo, es relativamente sencillo mostrar que la sumatoria de las cardinalidades de las clases de equivalencia es igual al número total de grafos menos 1 (el caso del grafo sin arcos):

$$S_{N-1} = 2^{N^2} - 1 \tag{6.3}$$

donde 2^{N^2} es de hecho el número de todos los posibles grafos.

La medida β es muy gruesa con muy pocas clases de equivalencia, en consecuencia, cada clase de equivalencia tiene alta cardinalidad. En este sentido no hay forma de distinguir entre elementos que pertenecen a la misma clase de equivalencia β, lo que causa que el proceso de búsqueda no sea eficiente. Adicionalmente la medida β no toma en cuenta todas las diferencias

absolutas entre etiquetas de vértices adyacentes, en lugar de esto, sólo toma en cuenta la máxima diferencia absoluta.

6.1.2 La Medida γ

Dadas las características de β, desarrollamos una nueva medida, llamada γ, la cual toma en cuenta todas las diferencias absolutas del grafo. La medida propuesta, representada por un sistema posicional numérico de base variable es la siguiente:

$$\gamma = \sum_{i,j|(i,j)\in E} P(N, |i-j|) \tag{6.4}$$

Donde la suma es realizada para todos los arcos del grafo y P es definido como:

$$P(N,k) = \begin{cases} 1 & k = 0 \\ (N+1)\prod_{j=2}^{k}(2N-2j+3) & 1 \leq k \leq N \end{cases} \tag{6.5}$$

Puede ser verificado, que si i y j difieren significativamente, $P(N,k)$ resulta en un valor muy grande, y que el número total de clases de equivalencia es dado por la expresión $P(N,N)$.

Para un grafo particular hay: W_0 diferencias absolutas entre vértices adyacentes con valor cero, correspondientes a los arcos reflexivos; W_1 diferencias con valor uno; ...; y W_{N-1} diferencias con valor $N-1$. Entonces el número total de grafos equivalentes para ese grafo particular es:

$$\binom{N}{W_0}\binom{2(N-1)}{W_1}...\binom{2(N-i)}{W_i}...\binom{2}{W_{N-1}} \tag{6.6}$$

esto puede ser expresado en una forma más corta con la fórmula:

$$\binom{N}{W_0}\prod_{i=1}^{N-1}\binom{2(N-1)}{W_i} \tag{6.7}$$

Para demostrar que la sumatoria de las cardinalidades de todas las clases de equivalencia es igual al número de todos los posibles grafos, es necesario usar la Fórmula 6.7 instanciada con todos los posibles valores de $W_0, W_1, ..., W_{N-1}$, calcular la suma, y verificar si es igual a 2^{N^2}. Esto puede ser expresado como:

$$\sum_{i=0}^{N} \left(\begin{array}{c} N \\ i \end{array} \right) \prod_{j=1}^{N-1} \sum_{r=0}^{2(N-j)} \left(\begin{array}{c} 2(N-j) \\ r \end{array} \right) \tag{6.8}$$

pero como $\sum_{i=0}^{N} \binom{N}{i} = 2^N$ y $\sum_{r=0}^{2(N-j)} \binom{2(N-j)}{r} = 2^{2(N-j)}$ entonces la Fórmula 6.8 se simplifica como:

$$2^N \prod_{j=1}^{N-1} 2^{2(N-j)} \tag{6.9}$$

reescribiendo la Ecuación 6.8:

$$2^N 2^{2\sum_{j=1}^{N-1}(N-j)} = 2^N 2^{2\frac{N(N-1)}{2}} = 2^{N^2} \tag{6.10}$$

y finalmente:

$$2^N 2^{2\frac{N(N-1)}{2}} = 2^{N^2} \tag{6.11}$$

6.1.3 Comparando γ y β

En esta subsección se comparan las medidas β y γ. Primero se contrastan las diferencias entre el número de clases de equivalencia de β y γ representadas como ω_β (puede ser verificado que su valor es N) y ω_γ ($\omega_\gamma = P(N, N)$), y posteriormente se ilustran el promedio de las cardinalidades de las clases de equivalencia para cada una de estas medidas.

En la Tabla 6.1, $\boldsymbol{\omega}_\beta$ y $\boldsymbol{\omega}_\gamma$ se muestran para diferentes valores de N, es importante enfatizar que $\boldsymbol{\omega}_\beta$ tiene un incremento lineal mientras que el de $\boldsymbol{\omega}_\gamma$ es exponencial.

N	ω_β	ω_γ
5	5	5670.0
10	10	7.202 x 10^{9}
20	20	6.7165 x 10^{24}
40	40	3.2709 x 10^{60}
70	70	6.7587 x 10^{121}
150	150	5.6674 x 10^{308}
200	200	1.0156 x 10^{436}
300	300	6.1101 x 10^{705}

Tabla 6.1: Valores de las Clases de Equivalencia de Beta y Gamma

N	$2^{N^2}/\omega_\beta$	$2^{N^2}/\omega_\gamma$
5	6.7109 x 10^{6}	5917.9
10	1.2677 x 10^{29}	1.7601 x 10^{20}
20	1.2911 x 10^{119}	3.8447 x 10^{95}
40	1.1116 x 10^{480}	1.3593 x 10^{421}
70	1.5918 x 10^{1473}	1.6486 x 10^{1353}
150	9.9727 x 10^{6770}	2.6395 x 10^{6464}
200	7.9213 x 10^{12038}	1.5599 x 10^{11605}
300	1.6691 x 10^{27090}	8.1952 x 10^{26386}

Tabla 6.2: Cardinalidad Promedio de Gamma y Beta

La Tabla 6.2 muestra algunos promedios de las cardinalidades de las clases de equivalencia de β y γ ($2^{N^2}/\omega_\beta$ y $2^{N^2}/\omega_\gamma$) de acuerdo con el número de vértices (N) de un grafo.

Como se observa en las Tablas 6.1 y 6.2, γ es una medida más fina que β dado que tiene la habilidad de crear más clases de equivalencia con una menor cardinalidad.

6.1.4 Calculando γ para Valores Grandes de N

Un problema en el cálculo de γ es que los valores resultantes pueden exceder fácilmente la precisión de la computadora empleada cuando N toma valores grandes. Una posible solución puede ser utilizar el logaritmo de γ, pero a simple vista parece imposible obtener el logaritmo de la expresión: $\gamma = \sum_{i,j|(i,j)\in E} P(N, |i-j|)$, debido a que es una sumatoria. Sin embargo, se obtuvo una expresión que permite calcular γ en términos de logaritmos y además no emplea números demasiado grandes. Para ilustrar esto, veamos el ejemplo del grafo representado en la Figura 6-3, del cual presentamos su matriz de adyacencias en la Tabla 6.3.

Para este grafo en particular: $W_0 = 5$, $W_1 = 3$, $W_2 = 4$, $W_3 = 1$, $W_4 = 2$ (donde W_i repre-

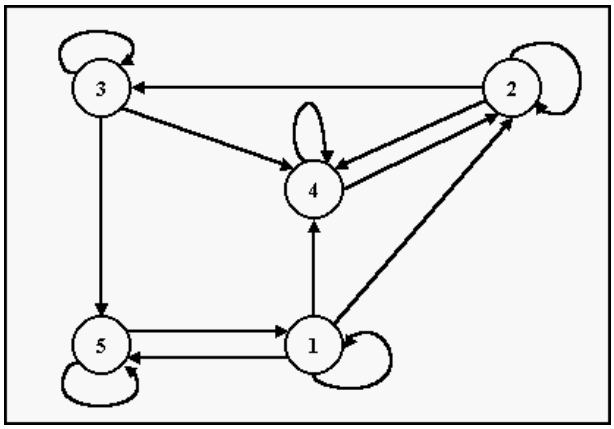

Figura 6-3: Ejemplo de un Grafo con 5 Vertices.

1	1	0	1	1
0	1	1	1	0
1	0	1	1	1
0	1	0	1	0
1	0	0	0	1

Tabla 6.3: Matriz de Adyacencias para el Grafo de 5 Vertices

senta el número de diferencias absolutas con valor i entre vértices adyacentes); adicionalmente tenemos que $P(5,0) = 1$, $P(5,1) = 6$, $P(5,2) = 54$, $P(5,3) = 378$, $P(5,4) = 1890$.

Entonces, $\gamma = \sum_{i=0}^{4} W_i P(5,i) = 5 + 18 + 216 + 378 + 3780 = 4397$, pero γ puede ser expresado en la siguiente forma:

$$\gamma = \left(\begin{array}{l} \dfrac{\dfrac{W_0}{\left(\frac{P(5,1)}{P(5,0)}\right)} + W_1}{\left(\frac{P(5,2)}{P(5,1)}\right)} + W_2 \\ \dfrac{\left(\frac{P(5,3)}{P(5,2)}\right)}{\left(\frac{P(5,4)}{P(5,3)}\right)} + W_3 \\ \qquad\qquad + W_4 \end{array} \right) P(5,4),$$ aplicando logaritmos a ambos lados de la ecuación:

$$\log(\gamma) = \log\left(\frac{\frac{\frac{\frac{W_0}{\left(\frac{P(5,1)}{P(5,0)}\right)}+W_1}{\left(\frac{P(5,2)}{P(5,1)}\right)}+W_2}{\left(\frac{P(5,3)}{P(5,2)}\right)}+W_3}{\left(\frac{P(5,4)}{P(5,3)}\right)}+W_4\right)+\log(P(5,4)) \tag{6.12}$$

La gran ventaja de la expresión anterior es que nunca se calculan números muy grandes, todos los denominadores en el primer término del lado derecho de la ecuación anterior corresponden a la serie: $(N+1),(2N-1),(2N-3),(2N-5),...,7,5$ y el segundo término puede ser expresado en la siguiente forma:

$$\log(P(N,K)) = \begin{cases} \log(N+1)+\sum_{j=2}^{k}\log(2N-2j+3) & 1 \leq k \leq N \\ 0 & k=0 \end{cases} \tag{6.13}$$

entonces el logaritmo de γ es:

$$\log(\gamma) = \log\left(\frac{\frac{\frac{W_0}{N+1}+W_1}{2N-1}+W_2}{2N-3}+W_3}{2N-5}+W_4\right)+\log(N+1)+\sum_{j=2}^{N-1}\log(2N-2j+3) \tag{6.14}$$

Resumiendo, se ha demostrado que es posible obtener el logaritmo de γ sin calcular números de gran tamaño, y en esta forma, evitar el problema de la precisión numérica de las computadoras.

Adicionalmente, el $\log(\gamma)$ puede ser normalizado a valores entre 0 y 1 usando la fórmula:

$$\gamma_{Norm} = \frac{\log(\gamma)}{\log(P(N,N))} \tag{6.15}$$

La importancia de esta normalización de γ, es que permite ver la similitud con β, esto puede

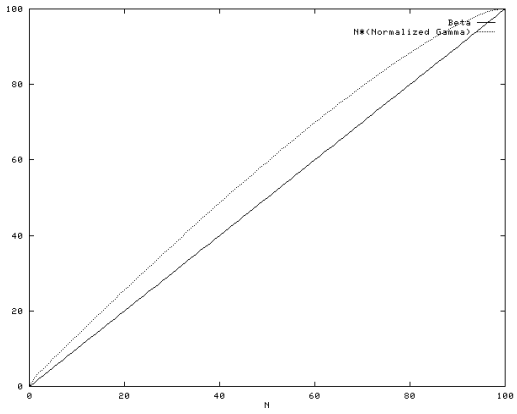

Figura 6-4: Gráfica Comparativa entre β y $N * \gamma_{Norm}$.

ser observado en la Figura 6-4 (es importante recalcar que γ_{Norm} mantiene la capacidad de representar más clases de equivalencia con menor cardinalidad, lo que permite guiar mejor el proceso de búsqueda de soluciones).

Ahora en la siguiente subsección analizaremos con detalle el algoritmo de Recocido Simulado propuesto.

6.1.5 Recocido Simulado

El recocido simulado es un buen algoritmo para buscar soluciones aceptables a problemas de optimización combinatoria. La operación básica en esta técnica es un *movimiento*. Un movimiento es una transición de una solución en el espacio de soluciones a otra. Dado que cada solución tiene un costo asociado, intuitivamente, se buscan transiciones o movimientos que decrementen el costo de la función objetivo. El criterio que la mayoría de los métodos tradicionales consideran (aquellos utilizados principalmente en la investigación de operaciones) es tomar una nueva solución si y sólo si su costo es menor que el de la solución actual. A estos algoritmos se les denomina de manera genérica glotones o avaros (dependiendo de si se maximiza o minimiza, respectivamente). Por otro lado, es común que después de ciertos movimientos se llegue a una solución que corresponde a un mínimo o máximo local, según corresponda. Tradicionalmente se han empleado dos estrategias para evitar que el método quede atrapado en mínimos locales: la primera se conoce como *reinicio*, la cual consiste en ejecutar el método varias veces partiendo

de diferentes puntos iniciales. La segunda estrategia consiste en que el método de optimización provea un mecanismo que le permita escapar de un mínimo local. Este último es el caso del recocido simulado (*Simulated Annealing*), análogo computacional de un proceso de la física del estado sólido.

En el recocido simulado los movimientos son elegidos aleatoriamente. Si un movimiento decrementa el costo, es aceptado (en ese sentido funciona como cualquier algoritmo avaro/glotón), pero a diferencia de ellos, en el caso de obtener una solución con costo mayor al actual, ésta se puede aceptar con probabilidad dada por la función de Boltzmann $P(\Delta E) = e^{-\Delta E/T}$ donde T es la temperatura y ΔE es el incremento en costo que resultará del movimiento. Inicialmente T es grande, y virtualmente todos los movimientos son aceptados. El valor de dicho parámetro se va disminuyendo conforme transcurre la ejecución del algoritmo, hasta que eventualmente el sistema alcanza un estado en el cual muy pocos movimientos son aceptados, se dice entonces que el sistema está *congelado*. La secuencia de decremento de temperatura es conocida como esquema de enfriamiento (*cooling schedule*). La siguiente temperatura es obtenida por $T_{n-1} = \alpha Tn$ donde α es la tasa de enfriamiento.

El algoritmo heurístico usado aproximará el ancho de banda de G examinando etiquetas de G generadas aleatoriamente, sin asumir la estructura de un etiquetado óptimo. Las etiquetas de un par de vértices se intercambian, y esto corresponde a un movimiento.

El proceso de Recocido Simulado comienza incializando la temperatura, *temp*; el número máximo de movimientos aceptados de cada temperatura max_moves; el número máximo de movimientos que serán intentados para cada temperatura, $max\_attempted\_moves$; max_frozen que es el número de iteraciones consecutivas soportadas para el cual el número de movimientos aceptados es menor que max_moves; así como la tasa de enfriamiento, $cool\_rate$. La elección de estos parámetros se discutirá en secciones posteriores. El algoritmo procede generando al azar un movimiento y después calculando el cambio en el costo del nuevo etiquetado. Si el costo decrementa entonces el movimiento es aceptado. Un movimiento que eleva el costo es aceptado con probabilidad $P(\Delta E) = e^{-\Delta E/T}$ donde ΔE es el incremento en el costo que resulta de este movimiento propuesto. El ancho de banda del Recocido Simulado, sa_band, es el ancho de banda mínimo del etiquetado generado por el algoritmo hasta el momento. Por lo tanto, si un movimiento crea un etiquetado con un ancho de banda más grande que sa_band, no es

cambiado. Se cuenta el número de movimientos aceptados y si este cae por debajo del valor dado, se dice que el sistema está congelado. El pseudocódigo para el algoritmo general se da a continuación:

Procedimiento Anneal*(G,best_map)*

temp = 0.00004;

cool_rate = 0.85;

map = *etiquetado aleatorio*;

best_map = map;

sa_band = AnchodeBanda(G,best_map);

max_moves = $4 * |E|$;

max_attempted_moves = 2∗ *max_moves*;

max_frozen = 10;

frozen = 0;

Mientras *(frozen* ≤ *max_frozen)*

moves = *attempted_moves* = 0*;*

Mientras *((moves* ≤ *max_moves)* **Y** *(attempted_moves* ≤ *max_attempted_moves)*

attempted_moves++;

genera un movimiento aleatorio map_ran;

Si *(El movimiento es aceptado)*

map = map_ran;

moves++;

Si *(sa_band < AnchodeBanda(G,map))*

best_map = map;

sa_band = AnchodeBanda(G,map);

Fin Si

Fin Si

Fin Mientras

temp = temp ∗ cool_rate;

Si *(attempted_moves > max_attempted_moves)*

frozen++;

Sino

frozen = 0;

Fin Si

Fin Mientras

Fin Anneal

La elección de parámetros se determinó en base a los resultados mostrados en [KGV83] y [DJ95], y a una serie de experimentos que permitieron afinar el algoritmo aquí reseñado para que se comportara de una manera más aceptable. Los parámetros considerados se mencionan a continuación:

- El número máximo de movimientos debe estar relacionado con el número de posibles movimientos. El número de posibles movimientos está directamente relacionado con el número de vertices ($|V|$). Existen $n(n-1)$ posibles movimientos, donde $|V| = n$. Sin embargo, se han obtenido mejores resultados correlacionando el máximo número de movimientos con el número de arcos $|E|$, ya que se requieren más movimientos para grafos más densos.

- La razón de enfriamiento ($cool\_rate$) empleada es de 0.85, de esta manera $T_{n-1} = 0.85 * Tn$.

- Sólo después de 10 iteraciones sin éxito el algoritmo se parará ($max\_frozen = 10$). Modificando este parámetro se puede obtener el resultado más rápido, pero no estarán cercanos a la banda óptima. Se encontró que el valor anterior para el parámetro da un buen balance entre la calidad del resultado y el esfuerzo invertido.

- La medición del costo de una solución está basada en la función γ definida previamente. Y dado que se uso el valor de γ normalizado, el algoritmo de recocido simulado no tiene problemas de escalamiento al variar el tamaño del problema.

6.2 Algoritmo Genético para Resolver Problemas SAT

En esta sección se hace énfasis en la manera en que se implementó el algoritmo genético para resolver problemas SAT. Los principales puntos que trataremos son: La representación del

problema, el tamaño de la población, la inicialización de la misma, la función de aptitud a emplear, el criterio de selección de individuos, los operadores genéticos que se utilizarán, las probabilidades de aplicación de los operadores genéticos, así como los criterios de paro.

Pero primeramente presentamos el pseudocódigo del AG propuesto, el cual describe de manera general las partes que lo componen. Este procedimiento es de naturaleza estocástica y mantiene una población de elementos $P(t)$. Cada elemento de la población representa una solución potencial para el problema SAT que está siendo resuelto. Los elementos de la población o individuos están codificados en cadenas binarias como se mencionó en el capítulo anterior; cuando estos individuos son expuestos a un ambiente obtenemos entonces el *fenotipo* del individuo, el cual indica su aptitud. Los elementos más aptos son seleccionados, para formar parte de la siguiente generación, y posteriormente son *alterados* empleando los operadores genéticos de cruza y mutación. Este proceso se realiza de manera iterativa y continua hasta que se alcance la condición de terminación.

Procedimiento AG

$t \leftarrow 0$

inicializa $P(t)$

evalúa $P(t)$

Mientras *(no se cumpla condición de terminación)*

$t \leftarrow t + 1$

selecciona $P(t)$ *a partir de* $P(t - 1)$

aplica cruce sobre $P(t)$ *con una probabilidad de cruce*

aplica mutación sobre $P(t)$ *con una probabilidad de mutación*

evalúa $P(t)$

Fin Mientras

Fin AG

En las siguientes subsecciones se explicarán más a detalle cada uno de los detalles de la implementación de este algoritmo.

6.2.1 Representación

Hao [Hao95], propuso una representación donde cada cláusula del problema SAT es codificada utilizando un grupo de tres bits (genes). Por ejemplo la cláusula $(A \vee B \vee \sim E)$ se representaría como la cadena 110. Este tipo de representación tiene una gran desventaja, veámoslo con un ejemplo, si una instancia SAT tiene 100 variables, da como resultado individuos con una longitud de 1290 bits, complicando con esto el proceso de búsqueda.

Una vez observada esta desventaja y dado que el algoritmo genético propuesto tiene la finalidad de resolver el problema SAT, el cual consiste en encontrar una asignación de valores de verdad capaz de satisfacer una expresión booleana formada por un conjunto de cláusulas en FNC, entonces el tipo de representación que más naturalmente se adapta al problema es una codificación binaria. Una ventaja de emplear este tipo de codificación es que permite hacer uso de los operadores genéticos clásicos definidos por Holland [Hol75], y obtener excelentes resultados gracias al fuerte fundamento teórico que tienen.

6.2.2 Tamaño de la Población

Durante la implementación del AG se examinó minuciosamente la influencia que tiene el tamaño de la población en el desempeño global del algoritmo, con el fin de lograr un equilibrio adecuado entre las capacidades de *exploración* y *explotación* del mismo.

Tomando en cuenta el trabajo de Goldberg [Gol85], donde se reporta que una aproximación para el tamaño adecuado de una población está dado por el doble del número de bits de la representación, comenzamos una etapa de experimentación, de donde se desprendió la observación de que una población pequeña provoca mayor explotación, y menor exploración por lo que se aumenta el riesgo de converger a un máximo (mínimo) local. Por otra parte un tamaño de población muy grande indica mayor exploración y menor explotación, pero además emplea muchos recursos computacionales.

Como producto de las pruebas realizadas se decidió trabajar con un enfoque donde se mantiene el tamaño de la población fijo. El tamaño empleado se obtiene al multiplicar 1.4 por el número de variables involucradas en el problema SAT.

6.2.3 Inicialización de la Población

La población inicial con que arranca el AG propuesto es creada de manera aleatoria, con la restricción de que no se permiten individuos repetidos. Esto es recomendable ya que permite tener una mayor diversidad genética al momento de iniciar el proceso de búsqueda y favorece el buen desempeño del AG.

6.2.4 Función de Aptitud

La función de aptitud es la base para determinar cuáles soluciones tienen mayor o menor probabilidad de sobrevivir, y determina el ambiente al cual es expuesta cada una de las soluciones. El punto crítico de una función de aptitud radica en encontrar un equilibrio entre una función que haga grandes diferencias entre los individuos, provocando *convergencia prematura*, y una función que haga diferencias muy pequeñas, causando estancamiento en la búsqueda de soluciones.

Dado que hemos elegido una representación basada en cromosomas codificados como cadenas de bits una manera obvia de evaluar la aptitud sería considerar el valor de verdad de la fórmula completa que se desea satisfacer. Sin embargo, en este caso no poseemos información sobre el gradiente de la función de aptitud, y el proceso de búsqueda genética se degrada al punto de llegar a convertirse en una búsqueda aleatoria. Otro enfoque utilizado es el propuesto por De Jong y Spears [JS89], ellos utilizan una función de aptitud basada en promediar los valores de verdad de las cláusulas, pero para ello emplean una codificación de punto flotante y ha demostrado no dar resultados tan buenos como la representación binaria.

En este trabajo hemos empleado como función de aptitud el conteo del número de cláusulas satisfechas que da como resultado el evaluar la función booleana a resolver con los valores de verdad codificados por ese individuo. Esto nos proporciona la ventaja de equilibrio que se mencionó al principio de esta subsección.

6.2.5 Criterios de Selección

La manera en la cual los elementos de una población son seleccionados para ser alterados haciendo uso de operadores genéticos y de esta manera construir la siguiente generación, es conocida

como el criterio de selección de un AG. Un criterio de selección, para su buen funcionamiento, debería preferir a los elementos con desempeño arriba del promedio.

Para el desarrollo de este trabajo se experimentó con el criterio de selección conocido como selección por torneo. Esta técnica opera formando grupos de t elementos, (t es el tamaño del torneo), y entonces se selecciona al elemento con mejor aptitud de cada grupo.

Cabe mencionar que se ha observado que el criterio de selección por torneo que mejor desempeño proporciona al AG es cuando este es de tamaño 3.

6.2.6 Operadores Genéticos

En esta implementación se emplearon dos de los operadores genéticos clásicos definidos por Holland [Hol75]. Estos son:

- Cruce de un solo punto
- Mutación de un solo bit

Hemos elegido estos operadores ya que tienen un fundamento matemático muy sólido y funcionan de manera muy aceptable bajo la representación que estamos empleando.

6.2.7 Probabilidades de Aplicación de los Operadores Genéticos

En la práctica en general el operador de mutación tiene una pequeña probabilidad de ser aplicado mientras que el de cruce tiene una probabilidad alta de usarse. Normalmente las probabilidades permanecen fijas, aunque, según se describe en [Tor97], éstas pueden variar en forma dinámica.

Para el caso de esta implementación se emplea una probabilidad de cruza del 80% así como una probabilidad de 1% de aplicar el operador de mutación, para cada gen dentro del cromosoma.

6.2.8 Criterio de Paro

Basados en [Gol89] y [JS89], los criterios de paro que se utilizaron en esta implementación son: un número fijo de generaciones, así como la condición de que el AG llegue a satisfacer todas las cláusulas del problema.

6.3 Comentarios Sobre el Enfoque de Solución

En este capítulo se han definido los detalles específicos de la implementación propuesta para resolver problemas SAT. Es particularmente importante resaltar algunos puntos con respecto a cada una de las dos fases del método propuesto. En la primera etapa o fase de preprocesamiento:

1. El algoritmo propuesto para preprocesar problemas SAT permite disminuir la epístasis de la representación de un problema.

2. La métrica desarrollada para este tipo de problemas (γ) es substancialmente mejor que la medida empleada comúnmente (β) por el resto de los investigadores. Y en particular como se uso el valor de γ normalizado en el rango de 0 a 1, el algoritmo de recocido simulado no tiene problemas de escalamiento.

3. γ a diferencia de β, permite minimizar el ancho de banda de grafos pesados.

4. Se han logrado con nuestro algoritmo de MABG un desempeño sobresaliente, tanto en tiempo de computo, como en calidad de solución, esto puede observarse en las tablas correspondientes en el siguiente capítulo.

En la fase de solución del problema:

1. La representación empleada permite emplear los operadores genéticos clásicos, los cuales tienen un fuerte fundamento teórico.

2. El criterio de selección por torneo con tamaño 3 permitió evitar el problema de la convergencia prematura.

3. La probabilidad de 0.80 y 0.01 para los operadores de cruza y mutación respectivamente permiten un buen desempeño del AG.

En el siguiente capítulo se presentarán los resultados obtenidos haciendo uso del enfoque de solución que se detalló en las secciones anteriores, el cual llamaremos a partir de este momento PAG (Preprocesamiento y Algoritmo Genético).

Capítulo 7

Resultados Experimentales

En este capítulo se realiza un análisis del desempeño del algoritmo propuesto para solucionar problemas SAT (PAG). Comenzaremos por discutir algunas formas de comparar el desempeño de los algoritmos de esta categoría, así como pruebas de desempeño (*benchmarks*) empleadas para medir la rapidez de la computadora utilizada para realizar los experimentos. Posteriormente presentaremos un estudio realizado a algunos de los generadores de casos de prueba más citados en la literatura, discutiremos sus ventajas y desventajas y presentamos un enfoque novedoso para generar casos de prueba difíciles para el problema SAT. Finalmente, presentamos y analizamos los resultados obtenidos con el algoritmo PAG.

7.1 Medidas de Desempeño

Comúnmente se emplean tres medidas para evaluar el desempeño de los algoritmos para resolver el problema SAT: El número de evaluaciones realizadas para resolver el problema, el número total de cláusulas satisfechas, y el tiempo de CPU empleado para ello. El número de evaluaciones por sí solo no es suficiente porque no indica la cantidad de tiempo que le toma al algoritmo resolver el problema. Pero tampoco el tiempo de CPU es suficiente porque es dependiente de la plataforma, del compilador empleado, y difícil de medir con precisión. Por lo tanto es importante complementar un análisis con ambas medidas.

Problema	Tiempo de CPU en Segundos
r100.5.b	0.01
r200.5.b	0.12
r300.5.b	1.13
r400.5.b	6.79
r500.5.b	25.71

Tabla 7.1: Pruebas de Desempeño (DIMACS) de la Computadora Empleada

7.2 Pruebas de Desempeño de la Computadora

Cuando se analizan los tiempos de CPU, es necesario tener una medida apropiada de la velocidad de la computadora empleada. Esto en general es difícil de realizar porque diferentes programas tienen también diferentes conjuntos de instrucciones que ejecutan más frecuentemente. Los organizadores del Desafío DIMACS trataron este problema al proveer una prueba de desempeño basada en un programa para encontrar cliques y cinco instancias de este tipo de problemas de diferente grado de dificultad. Ellos asumen que este programa tiene una mezcla de instrucciones que es similar a otros programas de búsqueda de cliques, coloreo de grafos y programas para probar satisfactibilidad[1].

La Tabla 7.1 presenta los tiempos de CPU de estos problemas en la computadora que utilicé para la realización de todos los experimentos de este proyecto, una IBM RS/6000 modelo H50 con 512MBytes de memoria RAM, corriendo AIX 4.3.2. El programa para realizar las pruebas de desempeño así como nuestro algoritmo fueron compilados con "cc -O2" (es decir se compila un programa en C usando un nivel de optimización 2). El tiempo de CPU del problema r500.5.b es el número que debe ser tomado en cuenta para realizar comparaciones de los resultados de esta tesis con otros algoritmos y/o computadoras que resuelvan los mismos problemas. Por ejemplo si el problema r500.5.b tarda en otra computadora 51.42 seg, se debería de usar un factor de escalamiento de 2.

[1]Está disponible vía FTP anónimo en *dimacs.rutgers.edu* bajo el directorio *pub/challenge/sat/benchmarks/Machine*.

7.3 Generadores de Casos de Prueba

En esta sección presentaremos algunos de los métodos para generar casos de prueba citados en la literatura y explicaremos algunas de sus características más importantes.

7.3.1 Generador de Dubois para Problemas Insatisfactibles

Este programa para generar casos de prueba fue creado por Oliver Dubois. Para su funcionamiento requiere como parámetros los siguientes valores:

- n = Número de variables.
- p = Número de cláusulas.
- r = Número de variables por cláusula.
- g = Semilla empleada por el generador aleatorio de instancias SAT.

Básicamente genera casos de prueba aleatorios k-SAT con cláusulas de longitud constante igual a r con el fin de obtener instancias duras [DM92]. El algoritmo en general puede describirse así: Primero se seleccionan los valores de n, p y r; posteriormente el programa elige uniforme e independientemente p cláusulas con r distintas literales tomadas del conjunto de n variables. Experimentalmente se ha observado que la dificultad de las instancias SAT generadas de acuerdo con este modelo, depende fuertemente de la razón p/n. Actualmente, las instancias 3-SAT más duras que pueden ser resueltas, tienen una razón que varia entre 4.1 y 4.5.

7.3.2 Generador de Pretolani para Problemas Duros

Este generador fue creado por Daniele Pretolani [Pre93]. Produce esencialmente problemas de coloreo de grafos con dos colores junto con una restricción de paridad para forzar la insatisfactibilidad. Tiene la característica de poder generar tanto fórmulas k-SAT, como fórmulas donde el número de literales por cláusula varia en el intervalo $[l..r]$ con una distribución discreta uniforme (DDU). Toma como entradas los siguientes datos:

- n Número de variables

- m Número de cláusulas
- *tipo* 0 = k-SAT y 1 = DDU
- l Límite izquierdo (k si el tipo es 0)
- r Límite derecho (k si el tipo es 0)
- *seed* Semilla si es 0, la semilla actual se obtiene utilizando la función "time(0)" del sistema operativo

La salida de este generador es un archivo con extensión "cnf" y con el formato sugerido por los organizadores del Desafío DIMACS.

7.3.3 Generador de Selman para Problemas Duros de Coloreo de Grafos

Este generador fue propuesto por Bart Selman [SML96] y produce problemas duros de coloreo de grafos, cabe mencionar que incluso los algoritmos especializados en este tipo de problemas, tienen serias dificultades para resolverlos. Su algoritmo produce fórmulas aleatorias de acuerdo con el modelo de cláusulas de longitud fija. En la documentación de este generador el propio Selman reconoce que su código está muy desordenado debido principalmente a que es una mezcla de algunos otros generadores.

El método de generación posee un mecanismo para decidir si se desea crear un problema satisfactible o insatisfactible. El parámetro que controla esto, recibe el nombre de "forced", si este se fija en cero, la fórmula será insatisfactible de otra forma la fórmula se forzará a ser satisfactible, esto se logra gracias a que el algoritmo toma una asignación aleatoria y elimina toda cláusula que viole esa asignación, de esta manera se garantiza la satisfactibilidad del problema. Este tipo de problemas son extremadamente fáciles de resolver, y por lo tanto no representan buenos casos de prueba para los procedimientos de prueba de satisfactibilidad. Si se desean crear instancias extremadamente difíciles es necesario elegir el número de cláusulas igual a 4.3 veces el número de variables.

Problema	Variables	Cláusulas	SAT	Solución	Tiempo	Evaluaciones
aim-50-1_6-yes1-1	50	80	Si	80	2.39	1500
aim-50-2_0-yes1-2	50	100	Si	100	1.98	1000
dubois21	63	168	No	167	16.44	4914
aim-100-2_0-no-2	100	200	No	199	50.35	12800
aim-100-2_0-yes1-3	100	200	Si	200	48.83	12400
pret150_25	150	400	No	399	280.49	34500
par16-4-c	324	1292	Si	1292	1242.92	127008

Tabla 7.2: Resultados para problemas del Desafío DIMACS resueltos con AG

7.3.4 Generador de Iwama para Problemas 3-SAT Duros

Este generador de casos de prueba fue desarrollado por Kazuo Iwama, Eiji Miyano y Yuichi Asahiro [CM97]. Utiliza parámetros similares a los de los otros generadores presentados. El mérito real de este algoritmo es que es capaz de producir instancias satisfactibles e insatisfactibles independientemente del rango elegido de variables y cláusulas, de esta forma provee instancias que los generadores aleatorios convencionales difícilmente podrían generar. Al mismo tiempo, este generador tiene un estilo aleatorizado, que lo distingue de otros que utilizan un enfoque determinístico o de aquellos que se basan en la traducción de otro tipo de problemas.

Algunas otras ventajas de este método para generar casos de prueba son: que no crea instancias de baja relación entre el número de cláusulas y variables. Las instancias satisfactibles tienen sólo una solución (una asignación que vuelve el predicado verdadero); este tipo de instancias permiten realizar valiosas pruebas en algoritmo incompletos de prueba de satisfactibilidad.

7.4 Desventajas de los Generadores de Instancias Estudiados

Después de realizar experimentos con los casos de prueba del Desafío DIMACS, los cuales fueron creados con los generadores de instancias mencionados anteriormente, se observó que el tipo de problemas que producían no eran lo suficientemente duros de resolver, al menos para un AG, ya que estos podían ser resueltos óptimamente en unas cuantas generaciones, vease la Tabla 7.2.

A raíz de esta observación, procedimos a realizar un estudio completo de las características de cada uno de los métodos de generación de instancias, así como del código fuente de sus implementaciones. De lo anterior obtuvimos información muy valiosa, en especial en cuanto

a los factores que afectan directamente la dureza de las instancias de prueba generadas. De esto podemos afirmar que las características que debe cumplir un buen generador de casos de prueba para problemas SAT son las siguientes:

- Garantizar el control del grado de repetibilidad de las variables que intervienen en el problema.
- Proveer un mecanismo que garantice el balance entre las apariciones positivas y negativas de cada variable del problema.
- Eliminar la generación de cláusulas triviales[2].
- Procurar que la razón entre cláusulas y variables sea la adecuada (aproximadamente entre 2 y 4 dependiendo del tamaño del problema)[CM97][DM92][ML96][SML96].

Cabe mencionar que ninguno de los generadores mencionados anteriormente cumplen con las primeras dos características, esta es la razón principal por la que resolver instancias de prueba generadas con ellos se convierte en algo trivial para un AG.

En la siguiente sección explicaremos un enfoque novedoso que cumple con las cuatro características anteriores por lo que permite crear casos de prueba SAT más difíciles.

7.5 Una Propuesta para Generar Problemas 3-SAT Duros

Una vez que se observó que los generadores de instancias de prueba existentes no cumplían con las cuatro características mencionadas, nos dimos a la tarea de diseñar un algoritmo que si cumpliera con ellas. Para ello en un principio se planteó la posibilidad de generar aleatoriamente una matriz de relaciones entre variables, donde se forzara a respetar una cierta cota de repetibilidad de variables en las cláusulas, para posteriormente a partir de esa matriz generar el caso de prueba SAT. Esta idea fue desechada debido a que de un problema SAT es posible obtener una matriz de relaciones pero a partir de ésta es posible obtener diversos problemas SAT. Además para generar un problema de tamaño específico es necesario seguir ciertas reglas al momento de generar la matriz para no exceder el número de relaciones entre variables.

[2] Una cláusula es trivial si contiene una variable y su negación.

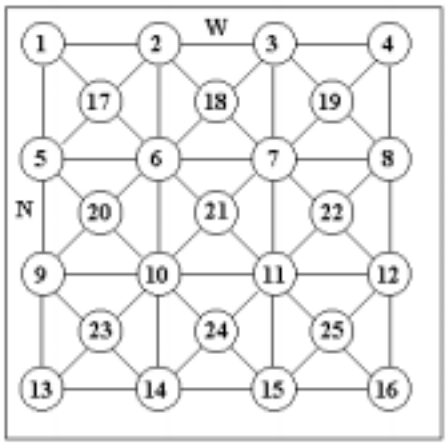

Figura 7-1: Malla con Centros para 3-SAT

Posteriormente se analizó la posibilidad de generar instancias de prueba del problema SAT a partir de grafos. En una primera aproximación se verificó la posibilidad de generar casos 3-SAT, esto debido a que si recordamos los problemas 2-SAT son triviales de resolver y además porque partimos de la premisa de que si era posible generar casos 3-SAT duros también es totalmente factible crear casos más grandes (4-SAT o 5-SAT), adoptando otro tipo de grafos.

Con esta idea en mente se estudiaron diferentes grafos y se llegó a la conclusión de que un modelo adecuado sería un grafo de tipo malla con la modificación de colocar un vértice al centro de cada cuadrado y unir éste con cada vértice del cuadrado, de este modo se forman cuatro triángulos, llamaremos a este tipo de grafos, *mallas con centros*, con el fin de identificarlos. Un ejemplo de este tipo de grafo es presentado en la Figura 7-1, este grafo de 4 renglones y 4 columnas, permite generar un problema 3-SAT de 25 variables y 36 cláusulas.

Si definimos a N como el número de renglones y a W como el número de columnas de la malla con centros, tenemos que el caso de prueba generado a partir de este grafo contiene $N * W + (N-1) * (W-1)$ variables y $4 * (N-1) * (W-1)$ cláusulas. Puede observarse en la figura que cada triángulo formado por tres vértices representa un cláusula (es decir cada arco de un triángulo representa la relación entre dos variables).

Las ventajas de generar los casos de prueba a partir de esta transformación de grafos a instancias SAT son principalmente que las cotas máxima y mínima del grado de repetibilidad de las variables que intervienen en el problema están perfectamente definidas y controladas,

observemos en la Figura 7-1 que la repetibilidad mínima es 2 (para el caso de los extremos), la máxima es 8 (en el centro del grafo), pero en promedio todas las variables se repiten 4 veces. Por otra parte al momento de transformar del grafo al problema SAT, es posible controlar el balance de las ocurrencias positivas y negativas de cada variable del problema, gracias a que desde un principio se conoce el número de veces que se repite, además de que la topología misma del grafo garantiza la satisfacción del requerimiento de la relación entre cláusulas y variables mencionada anteriormente. De esta manera nuestro generador de casos de prueba es superior a los existentes (esto se evidenció por la dificultad que tuvo el AG para resolver estas instancias y la facilidad con la que resolvio instancias de los otros generadores).

Después de hacer pruebas para resolver estos casos con un AG se observó que los problemas generados son más difíciles que los generados para el Desafío DIMACS. Un vez comprobado esto procedimos a diseñar otro tipo de grafos a partir de los cuales se generarán casos de prueba más difíciles. Intuitivamente un grafo que tenga un conjunto de nodos con más alto grado debería producir un problema más difícil, por esta razón modificamos el modelo del grafo de la Figura 7-1, conectando los vértices de los centros de los cuadrados entre sí, con lo cual se aumenta la repetibilidad máxima a 12, y en promedio todas las variables se repiten 8 veces, el resultado se muestra en la Figura 7-2, a este tipo de grafos los llamaremos *mallas con centros conectados*. Con el grafo de este ejemplo es posible producir una instancia SAT de 25 variables y 60 cláusulas con una razón entre cláusulas y variables de 2.4. Los problemas SAT generados a partir de este tipo de grafos en general tienen $N * W + (N-1) * (W-1)$ variables y $(4 * (N-1) * (W-1)) + (4 * (N-2) * (W-2)) + (2 * (N-2)) + (2 * (W-2))$ cláusulas, donde N y W son el número de renglones y de columnas de la malla con centros conectados.

7.6 Casos de Prueba Generados

Con el propósito de evaluar el desempeño del algoritmo PAG propuesto para resolver el problema SAT se eligieron una serie de problemas generados a partir de los dos grafos explicados en la sección anterior. Se tomaron 14 mallas con centros y 15 mallas con centros conectados.

Los nombres de los problemas están formados con la letra "p", dos números separados por un guión bajo que significan el número de renglones y columnas del grafo respectivamente y

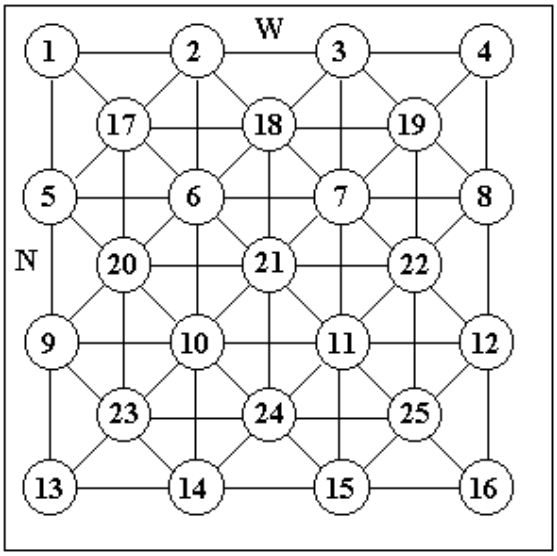

Figura 7-2: Malla con Centros Conectados para 3-SAT

por la letra “A” cuando se trata de una malla con centros o una letra “B” cuando es una malla con centros conectados.

Para cada caso de prueba se ejecutaron 10 corridas, 5 utilizando preprocesamiento y un AG (PAG), y las 5 restantes resolviendo el problema únicamente con el uso del mismo AG. Al momento de reportar los resultados para cada enfoque se realiza un promedio de las 5 corridas correspondientes. Cabe mencionar que en el algoritmo PAG al momento de reportar resultados, se contabiliza también el tiempo que toma la etapa de preprocesamiento.

En la siguiente sección se muestran los resultados obtenidos con nuestra implementación del algoritmo de preprocesamiento, el cual, es básicamente un algoritmo de RS para la MABG.

7.7 Resultados del Algoritmo de Preprocesamiento (MABG)

Los resultados obtenidos con el algoritmo de RS demuestran que es posible usar este enfoque para resolver el PMABG de una manera competitiva. A continuación se presentan dos tablas, donde se muestran los resultados para mallas con centros (Tabla 7.3) y mallas con centros conectados (Tabla 7.4), respectivamente. Cada una contiene columas donde aparece el nombre del problema; el número de nodos del grafo, n; la banda inicial, $\boldsymbol{\beta}_i$; la banda final, $\boldsymbol{\beta}_f$ y el

Prob.	**n**	β_i	β_f	**T**
p8_8A	113	64	16	7.84
p9_6A	94	54	12	4.58
p9_7A	111	63	14	7.10
p9_8A	128	72	16	9.67
p9_9A	145	81	19	11.53
p10_6A	105	60	12	5.28
p10_7A	124	70	14	8.21
p10_8A	143	80	16	12.40
p10_9A	162	90	19	15.38
p10_10A	181	100	20	21.89
p11_8A	158	88	17	17.85
p11_9A	179	99	19	22.77
p11_10A	200	110	23	23.30
p11_11A	221	121	44	18.95

Tabla 7.3: Resultados del algoritmo de MABG en Mallas con Centros

tiempo de computo (T), en segundos, que se consumió para resolver el problema.

7.8 Resultados del PAG en Mallas con Centros

En esta sección así como en la siguiente se presentan tablas donde se muestran los resultados comparativos entre los algoritmos PAG y AG. Los datos que se incluyen en esta comparación son: nombre del problema; número de variables n; número de cláusulas m; información referente al preprocesamiento como es el tiempo invertido T; y banda obtenida β. Por otra parte para cada enfoque se presenta: el número de cláusulas satisfechas, S; el tiempo de CPU en segundos, T; así como el número de evaluaciones E.

Para los problemas de Mallas con Centros al utilizar el algoritmo PAG se logró satisfacer el total de cláusulas de las 14 instancias propuestas, mientras que el AG sólo fue capaz de satisfacer el total de cláusulas en 10 de las 14 instancias.

Por otra parte el PAG proporcionó mejores resultados, haciendo un número menor de evaluaciones y por consecuencia consumió menor tiempo de CPU, para 12 de los 14 problemas. En la Tabla 7.5, se muestran en negritas los casos donde nuestro algoritmo es superado por el AG.

Prob.	**n**	β_i	β_f	T
p8_7B	98	56	14	8.44
p8_8B	113	64	16	8.59
p9_6B	94	54	12	8.32
p9_7B	111	63	15	8.06
p9_8B	128	72	17	12.39
p9_9B	145	81	18	14.43
p10_6B	105	60	12	5.60
p10_7B	124	70	15	7.29
p10_8B	143	80	17	13.63
p10_9B	162	90	19	16.23
p10_10B	181	100	20	35.68
p11_8B	158	88	17	22.97
p11_9B	179	99	21	23.36
p11_10B	200	110	24	18.10
p11_11B	221	121	44	21.58

Tabla 7.4: Resultados del algoritmo de MABG en Mallas con Centros Conectados

			P		**P+AG**			**AG**		
Prob.	**n**	**m**	β	**T**	**S**	**T**	**E**	**S**	**T**	**E**
p8_8A	**113**	**196**	**16**	**7.46**	**196**	**9.70**	**22594**	**196**	**5.64**	**46294**
p9_6A	94	160	12	4.28	160	4.46	2096	160	4.61	47291
p9_7A	111	192	14	6.41	192	8.20	2635	192	12.14	68665
p9_8A	128	224	16	9.56	224	10.24	5549	224	25.87	173809
p9_9A	145	256	19	11.38	256	12.82	10150	255	106.07	609000
p10_6A	105	180	12	3.21	180	3.38	2499	180	3.40	18228
p10_7A	**124**	**216**	**14**	**6.33**	**216**	**6.62**	**3979**	**216**	**6.52**	**43077**
p10_8A	143	252	16	12.40	252	13.60	9000	252	19.71	114000
p10_9A	162	288	19	14.25	288	15.68	10396	288	76.36	327700
p10_10A	181	324	20	19.93	324	22.10	11638	324	53.86	219857
p11_8A	158	280	17	17.74	280	19.53	11271	279	123.09	663000
p11_9A	179	320	19	22.77	320	29.16	32500	319	169.66	750000
p11_10A	200	360	23	21.00	360	22.57	8960	360	71.11	181440
p11_11A	221	400	44	17.14	400	35.09	63654	398	307.62	892392

Tabla 7.5: Resultados para Problemas de Mallas con Centros

			P		P+AG			AG		
Prob.	**n**	**m**	β	**T**	**S**	**T**	**E**	**S**	**T**	**E**
p8_7B	98	310	14	8.44	308	10.94	6713	306	61.07	411000
p8_8B	113	364	16	8.59	362	9.24	6162	360	61.32	329746
p9_6B	94	294	12	8.32	293	41.98	265930	291	54.25	380162
p9_7B	111	356	15	8.06	353	8.81	5580	353	34.53	182590
p9_8B	128	418	17	12.39	416	47.51	147138	413	127.04	537000
p9_9B	145	480	18	14.43	480	18.45	15428	480	121.38	431172
p10_6B	105	332	12	5.60	332	7.87	22050	330	74.1	441000
p10_7B	124	402	15	7.29	401	76.57	295830	399	126.95	519000
p10_8B	143	472	17	13.63	470	75.61	164600	469	200.24	600000
p10_9B	162	542	19	16.23	542	22.97	27572	540	117.53	336288
p10_10B	181	612	20	35.68	609	104.50	208978	606	147.67	371657
p11_8B	158	526	17	22.97	525	148.37	428298	523	224.39	663000
p11_9B	179	604	21	23.36	601	43.34	62750	600	242.53	641500
p11_10B	200	682	24	18.10	681	166.61	365400	679	356.14	815920
p11_11B	221	760	44	21.58	758	66.32	102588	755	194.83	406953

Tabla 7.6: Resultados para Problemas de Mallas con Centros Conectados

7.9 Resultados del PAG en Mallas con Centros Conectados

Al efectuar pruebas con este tipo de problemas el PAG logró satisfacer el total de cláusulas en 3 de las 15 instancias propuestas, mientras que el AG sólo fue capaz de satisfacer el total de cláusulas en uno de los problemas. Además un punto importante es el hecho de que el PAG superó en calidad de solución al AG en 13 de las 15 pruebas. Ambos algoritmos llegaron a la misma solución en dos ocasiones. En cuanto a tiempos de cómputo y número de evaluaciones el PAG venció al AG en las 15 pruebas efectuadas. Los resultados de las pruebas efectuadas se muestran en la Tabla 7.6.

7.10 Gráficas Comparativas entre PAG y AG

En esta sección presentamos 6 gráficas donde se puede apreciar claramente la ventaja de utilizar el algoritmo de preprocesamiento propuesto para resolver problemas SAT. En ellas se muestra el proceso de convergencia de los dos algoritmos estudiados. Tres corresponden a problemas generados a partir de mallas con centros y las restantes a problemas de mallas con centros conectados.

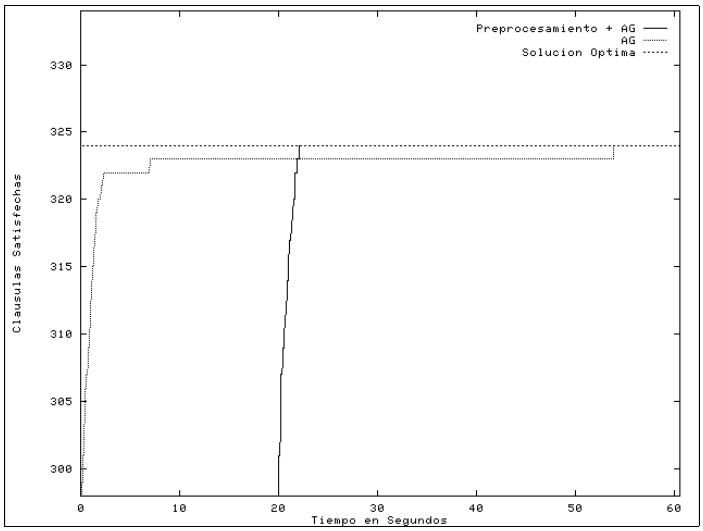

Figura 7-3: Gráfica del Problema *p10_10A*

Las gráficas fueron creadas utilizando como base los archivos de salida que producen ambos algoritmos. Las gráficas muestran en el eje de las abscisas el tiempo en segundos que ha consumido el algoritmo, y en el eje de las ordenadas el número de cláusulas satisfechas. También se graficó una línea que marca la solución óptima. Es importante resaltar que para el caso del algoritmo PAG su gráfica inicia a partir de un tiempo t_n, donde t_n es el tiempo de CPU que consumió la etapa de preprocesamiento, es por esto que se nota defasada con respecto a la gráfica del AG.

7.11 Comentarios de los Resultados Experimentales

En este capítulo se realizó un análisis comparativo del desempeño del algoritmo PAG y un AG para resolver problemas SAT, de él podemos concluir que en general el algoritmo PAG mostró un comportamiento más eficiente bajo las instancias de prueba utilizadas.

Es particularmente importante resaltar algunos puntos con respecto a cada una de las dos fases del PAG. En la primera etapa o fase de preprocesamiento:

- El algoritmo de preprocesamiento de problemas SAT propuesto, permitió reducir significativamente la banda de la matriz de adyacencias del problema SAT, en tiempos que

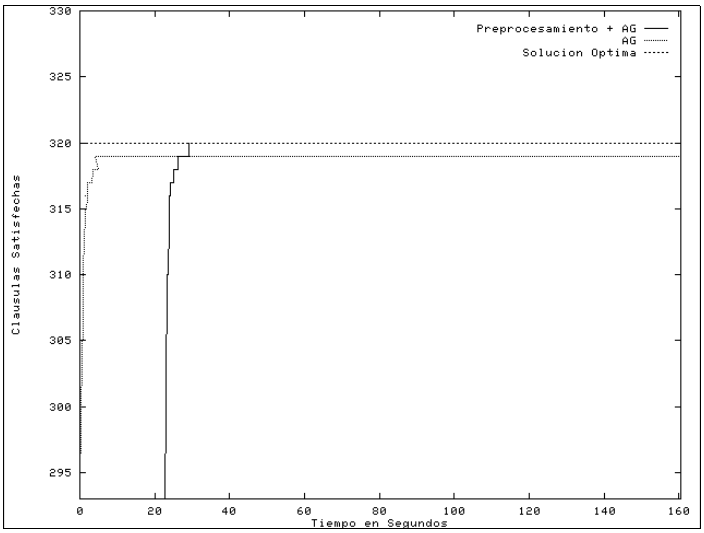

Figura 7-4: Gráfica del Problema *p11_ 9A*

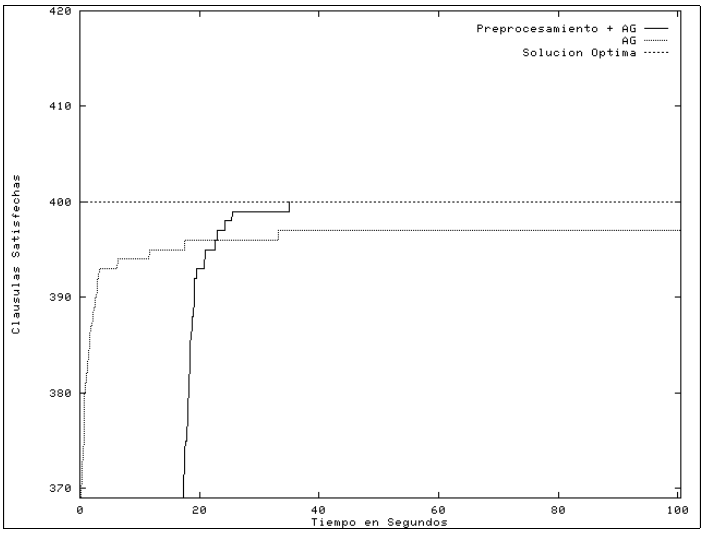

Figura 7-5: Gráfica del Problema *p11_ 11A*

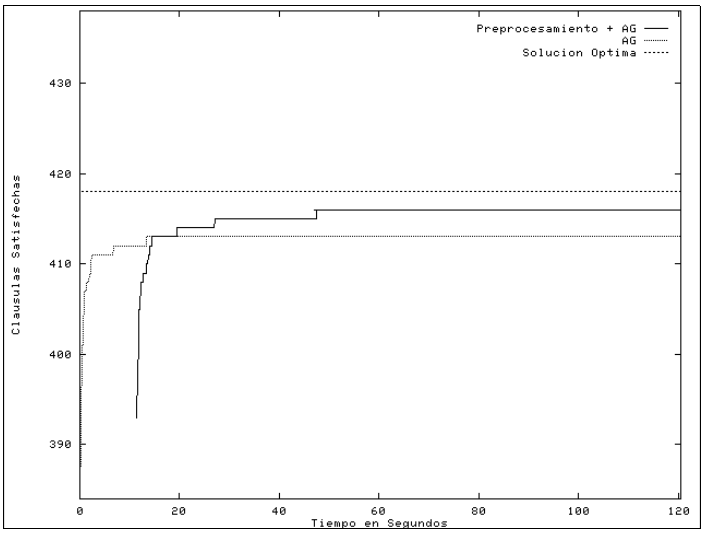

Figura 7-6: Gráfica del Problema *p9_ 8B*

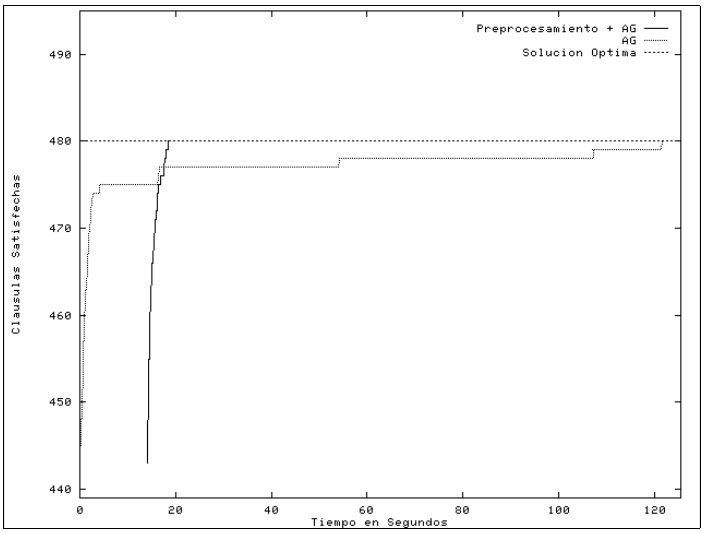

Figura 7-7: Gráfica del Problema *p9_ 9B*

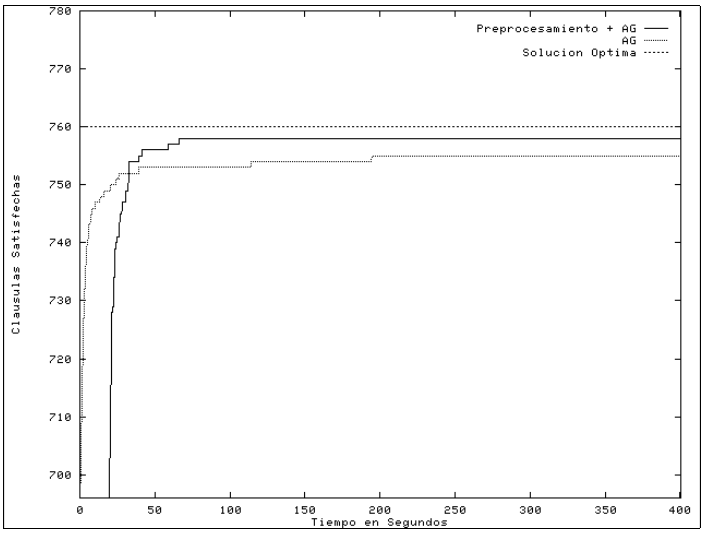

Figura 7-8: Gráfica del Problema *p11_ 11B*

no superan los 35 segundos para los casos de prueba presentados en las Tablas 7.5 y 7.6, donde el número de vértices es como máximo 221.

- Dado que el algoritmo de preprocesamiento cumple el compromiso entre calidad de la solución entregada y el tiempo de computo empleado para lograrla, permite la comparación con un AG y se observa que esta etapa realmente proporciona una ventaja en cuanto a tiempos totales de cómputo para resolver problemas duros de satisfactibilidad.

- Se comprobó que la métrica desarrollada especialmente para este tipo algoritmo (γ) es substancialmente mejor que la medida empleada comúnmente por el resto de los investigadores (β), en particular la normalización de γ no tiene ningún problema de escalamiento, i.e. funciona adecuadamente para tamaños variables del problema.

En la segunda fase, donde se aplica un AG para la solución del problema SAT:

- Se comprobó a traves de los experimentos que el utilizar el algoritmo de preprocesamiento propuesto para problemas SAT realmente ayuda a mejorar el desempeño de un AG, tanto en calidad de solución, como en tiempos de computo.

- Se ha demostrado que es posible resolver de manera competitiva problemas duros SAT

mediante el algoritmo PAG.

En el siguiente capítulo presentamos las conclusiones de este trabajo de tesis, así como las líneas de investigación futuras.

Capítulo 8

Conclusiones e Investigaciones Futuras

8.1 Conclusiones

Los resultados de este trabajo de investigación muestran que en general el algoritmo PAG, comparado con un AG, mostró un comportamiento más eficiente bajo las instancias de prueba utilizadas. Esto se debe en gran medida a que el algoritmo de preprocesamiento propuesto es capaz de encontrar buenas soluciones al PMABG en tiempos de cómputo razonables; que en general son menores a los tiempos que tomaría resolver ese mismo problema con un AG sin utilizar preprocesamiento.

Por otra parte se observó en las pruebas realizadas, que a medida que se incrementa el tamaño del problema SAT a resolver, las ventajas de usar el algoritmo de preprocesamiento aumentan. Esto se ve reflejado tanto en los tiempos totales de cómputo como en la calidad de la solución entregada.

Lo anterior nos permite concluir que la idea de realizar un preprocesamiento de problemas SAT para reordenar las variables y disminuir su epístasis proporciona grandes ventajas a un AG para resolver el problema dado.

Las principales aportaciones de esta tesis son:

1. Un algoritmo al que llamamos PAG que mostró ser consistentemente mejor que un AG

para resolver problemas SAT duros.

2. La implementación de un algoritmo de RS que permite resolver el PMABG de una manera competitiva.

3. El uso de una métrica especial para el PMABG (γ) que es substancialmente mejor que la medida empleada comúnmente (β). Es importante resaltar que la normalización de γ proporciona la ventaja de que no tiene ningún problema de escalamiento, por lo que funciona adecuadamente para diversos tamaños de problemas. Otra ventaja de γ con respecto a β es que permite bandear grafos pesados.

4. Lograr identificar las características que debe cumplir un buen generador de casos de prueba para problemas SAT. Estas son: garantizar el control del grado de repetibilidad de las variables que intervienen en el problema; proveer un mecanismo que garantice el balance entre las apariciones positivas y negativas de cada variable del problema; evitar generar cláusulas triviales, así como cuidar que la razón entre cláusulas y variables sea aproximadamente entre 2 y 4.

5. Un enfoque novedoso para generar casos de prueba duros del problema SAT a partir de su analogía geométrica con grafos. En especial se probaron dos tipos de grafos (mallas con centros y mallas con centros conectados) y se demostró que los grafos que tiene un conjunto de nodos con más alto grado producen problemas más difíciles.

8.2 Líneas de Investigación Futuras

A raíz de las trabajos realizados a lo largo de este proyecto de tesis se generaron algunas líneas de investigación que sería interesante explorar, a continuación presentamos algunas de ellas:

1. Al comprobar que el uso del algoritmo PAG mostró ser mejor que un AG para resolver problemas SAT duros, se abre la posibilidad de estudiar la manera de aplicar el mismo algoritmo de preprocesamiento a otro tipo de problemas que se deseen resolver mediante el uso de los AG.

2. Como se mencionó, los problemas NP-Completos pueden ser considerados equivalentes en el sentido de que cualquier problema de este tipo puede ser transformado en cualquier otro NP-Completo en tiempo polinomial [GJ79]. Una ventaja de esto es que se podría transformar un problema NP-Completo que no tenga una representación directa en AG a un problema SAT equivalente, resolver el problema SAT eficientemente usando un AG y mapear esta solución al dominio original del problema. Una posibilidad de investigaciones futuras sería el desarrollo de algoritmos que permitieran realizar la transformación explicada anteriormente de manera biyectiva.

3. Considerando el enfoque planteado para generar casos de prueba duros del problema SAT a partir de su analogía geométrica con grafos, existe la posibilidad de estudiar otros tipos de grafos más complejos que permitan crear problemas aún más duros para k-SAT en general, e incluso problemas SAT de densidad constante, es decir, problemas donde la longitud de las cláusulas es variable.

4. Un aspecto importante de las analogías geométricas entre grafos y problemas SAT, sería desarrollar algoritmos que permitan determinar que configuraciones de grafos generan casos satisfactibles y cuales no.

5. Para el caso del AG empleado se pretende realizar investigación de funciones de aptitud alternativas que orienten mejor el proceso de búsqueda.

Bibliografía

[AET] F. Malucelli A. Esposito, S. Fiorenzo and L. Tarricone. A wonderful bandwidth matrix reduction algorithm. In *Submitted to Operations Research Letters*.

[AM65] G.G. Alway and D.W. Martin. An algorithm for reducing the bandwidth of a matrix of symetrical configuration. *Computer Journal*, (8):264–272, 1965.

[BCM97] Y. Kambayashi B. Cha, K. Iwama and S. Miyasaki. Local search algorithms for partial maxsat. In *Proceedings of the AAAI 97*, pages 263–268, July 27-31 1997.

[BF97] Brian Borchers and Judith Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Technical report, Mathematics Department, New Mexico Tech, Socorro, NM, March 31 1997.

[Boy71] R. S. Boyer. *Locking: A restriction of Resolution*. PhD thesis, University of Texas at Austin, Texas, 1971.

[BS92] D. Mitchell B. Selman, H. Levesque. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Inteligence*, pages 440–446, San Jose CA, July 1992.

[BSC94] H. Kautz B. Selman and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI 94*, pages 337–343, July 1994.

[BT95] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth ICGA*, pages 9–16. Morgan Kaufmann Publishers, San Francisco, Ca., 1995.

[CB94] James M Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. *AAAAI*, 1994.

[Chu36] A. Church. An unsolvable problem of number theory. *Amer. J. Math.*, 58:345–363, 1936.

[Chv70] V. Chvátal. A remark on a problem of harary. *Czech. Math. Journal*, (20), 1970.

[Chv81] J. Chvátolová. *On the Bandwidth Problem for Graphs.* PhD thesis, University of Waterloo, Ontario, Canada, 1981.

[CLC73] Richard Char-Tung Lee Chin-Liang Chang. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[CM69] E. Cutchill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. *Proceedings 24th National of the ACM*, pages 157–172, 1969.

[CM91] H.M. Cartwright and G.F. Mott. Looking around: Using clues from the data space to guide genetic algorithms searches. In *Proceedings of the Fourth ICGA*, pages 108–114. Morgan Kaufmann Publishers, Los Altos Ca., 1991.

[CM97] Stephen A. Cook and David G. Mitchell. Finding hard instances of the satisfiability problem: A survey. *DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences*, 1997.

[Coe97] Carlos Coello Coello. Diseño óptimo de circuitos lógicos usando algoritmos genéticos. *Memorias de la sección de aprendizaje, primer encuentro nacional de computación, México*, 1997.

[Col90] Alain Colmerauer. An introduction to prolog III. *Comunications of the ACM*, 33(7):69–90, July 1990.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proc. Of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[Cre93] Nadia Creignou. The class of problems that are linearly equivalent to satisfiability or a uniform method for proving NP-completeness. In H. Kleine S. Martini E. Börger, G. Jäger and M.M. Richter, editors, *6th Workshop on Computer Science Logic (Lecture Notes in Computer Science, Volume 702)*, pages 115–133. Springer-Verlang, 1993.

[Dav85] L. Davis. Applying adaptative algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[Dav91] Y. Davidor. *Genetic Algorithms and Robotics, A Heuristic Strategy for Optimization.* World Scientific Publishing Co., 1991.

[Dev89] S. Devadas. Optimal layout via boolean satisfiability. In *ICCAD 89*, pages 294–297, November 1989.

[DJ95] G. Dueck and J. Jeffs. A heuristic bandwidth reduction algorithm. *Journal of combinatorial mathematics and computers*, (18), 1995.

[DLL62] M. Davis, G. Logenmann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM 5*, pages 394–397, 1962.

[DM92] H. Levesque D. Mitchell, B. Selman. Hard and easy distributions for sat problems. In *Proceedings of the Tenth National Conference on Artificial Inteligence*, pages 459–465, San Jose CA, July 1992.

[DWF89] T. Starkweather D. Whitley and D. Fuquay. Scheduling problems and traveling salesman: The genetic age recombination. In *Proceedings of the Third ICGA*, pages 133–140. Morgan Kaufmann Publishers, Los Altos, Ca., 1989.

[Fuk97] Alex S. Fukunaga. Variable-selection heuristics in local search for sat. In *Proceedings of the AAAI 97*, pages 275–280, Providence, Rhode Island, July 1997.

[Gil60] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM J. Res. Develop*, pages 28–35, 1960.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness.* Freeman and Company, 1979.

[GL81] A. George and W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Glo89] Fred Glover. Tabu search part 1. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[Gol85] D.E. Goldberg. Optimal initial population size for binary-coded genetic algorithms. Technical Report TCGA Report No. 85001, Tuscalosa, University of Alabama, 1985.

[Gol89] David E. Goldberg. *Genetic algoritms in search, optimization and machine learning.* Addison-Wesley Publishing Company, Inc., 1989.

[GS84] E.M. Gurari and I.H. Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem. *Journal of Algorithms, 5*, pages 531–546, 1984.

[Hao95] J. K. Hao. A clausal genetic representation and its evolutionary procedures for satisfiability problems. In *Proceedings of the International Conference on Artifitial Neural Networks and Genetic Algorithms*, France, April 1995.

[Har64] L.H. Harper. Optimal assignment of numbers to vertices. *Journal of SIAM*, 12:131–135, 1964.

[Har67] F. Harary. Theory of graphs and its aplications. page 161, 1967.

[HD93] J.K. Hao and R. Dorne. A population-based method for satisfiability problems. Technical Report 93-11-01, EERIE-LERI Parc Scientifique Georges Besse, Nimes, France, November 1993.

[HD94] J. K. Hao and R. Dorne. An empirical comparision of two evolutionary methods for satisfiability problems. Technical Report 94-11-01, EERIE-LERI Parc Scientifique George Besse, Nimes France, November 1994.

[HM97] J. Haralambides and F. Makedon. Approximation algorithms for the bandwidth minimization problem for a large class of trees. *Theory of Computer Systems*, (30):67–90, 1997.

[Hol75] J. Holland. *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press, 1975.

[Jer92] M. Jerrum. Large cliques elude the metropolis process. *Random Structures and Algorithms*, 3(4):347–359, 1992.

[JHM91] F. Makedon J. Haralambides and B. Monien. An aproximation algorithm for caterpillars. *Journal of Mathematical Systems Theory*, (24):169–177, 1991.

[JMB91] Steve Joy, John Mitchell, and Brian Borchers. A branch and cut algorithm for MAX-SAT and weighted MAX-SAT. In *DIMACS Series in Discrete Mathematics and Theoretical Science*, 1991.

[JS89] Kenneth A. De Jong and William M. Spears. Using genetic algorithms to solve NP-complete problems. In *Proceedings of the Third ICGA*, pages 124–132, Fairfax, Virginia, 1989.

[JSD89] L. Eshelman J.D. Schaffer, R. Caruana and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proc. Of the Third ICGA*, pages 51–60. Morgan Kaufmann Publishers, Los Altos Ca., 1989.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Koz91] J.R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1991.

[KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. Of the 10th European Conference on Artificial Intelligence (ECAI 92)*, pages 359–363, 1992.

[LFW66] A.J. Owens L.J. Fogel and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley, Chichester, UK, 1966.

[Lov70] D. W. Loveland. A linear format for resolution. *Proc. IRIA Symp. Automatic Demonstration, Versailles, France 1968*, pages 147–162, 1970.

[Luc70] D. Luckham. Refinements in resolution theory. *Symp. Automatic Demonstration, Versailles, France*, pages 163–190, 1970.

[MD60] H. Putnam M. Davis. A computing procedure for quantification theory: Its justification and realization. *J. ACM*, pages 201–215, 1960.

[MGK78] D.S. Johnson M.R. Garey, R.L. Graham and D.E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics 34*, pages 477–495, 1978.

[Mic92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlang Berlin Heidelberg New York, 1992.

[ML96] David G. Mitchell and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence 81*, 1996.

[MR91] I. Mitterreiter and F.J. Randermacher. Experiments on running time behavior of some algorithms solving propositional logic problems. Working paper, Technical Report, Forschungsinstitutfür anwendungsorientierteWissensverarbeitung, Ulm, Germany, 1991.

[MSG97] Bertrand Mazure, Lakhdar Sais, and Eric Gregoire. Tabu search for SAT. In *Proc. National Conference on Artificial Intelligence (AAAI-97)*, pages 281–285, 1997.

[MYF92] Wen-Tsuen Chen Ming-Yi Fang. Vectorization of a generalized procedure for theorem proving in propositional logic on vector computers. *IEEE Trans. on Knowledge and Data Engeneering*, (4):475–486, 1992.

[NGS76] W.G. Poole N.E. Gibbs and P.K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis 13*, pages 235–251, 1976.

[Pap76] C.H. Papadimitriou. The NP-Completeness of the bandwidth minimization problem. *Journal of Computing*, (16):263–270, 1976.

[Pap91] Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Conference on the Foundations of Computer Science*, pages 163–169, 1991.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Pre93] D. Pretonali. Solving satisfiability problems: An algorithm implementation challenge? In *Workshop Notes 2nd DIMACS Challenge*, 1993.

[PS82] C.H. Papadimitriou and K. Steighlitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice Hall, 1982.

[Riv78] R. L. Riverst. A method for obtaining digital signatures and public-key cryptosystems. *CACM*, pages 120–126, 1978.

[Rob65] J. A. Robinson. A machine oriented logic base on the resolution principle. *J. ACM*, 18:630–646, 1965.

[Sch81] H.P. Schewefel. *Numerical Optimization for Computer Models*. John Wiley, Chichester, UK, 1981.

[SK93] Bart Selman and Henry A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proc. Of the Eleventh National Conference on Artificial Intelligence (AAAI-93).*, 1993.

[SKC95] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *AAAI-92: Proceedings 10th National Conference on AI*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, January 1995.

[SML96] Bart Selman, David Mitchell, and Hector Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, pages 111–125, 1996.

[Spe92] William M. Spears. Using neural networks and genetic algorithms as heuristic for NP-complete problems. Technical report, AI Center, Naval Research Laboratory, Washington, DC 20375/ George Mason University, Fairfax Virginia, 1992.

[Spe93] William M. Spears. A NN algorithm for hard satisfiability problems. In *Proceedings of the International Conference on Neural Networks*, pages 1121–1126, Julio 1993.

[Spe96] William M. Spears. A NN algorithm for boolean sastisfiability problems. Technical Report AIC-96-009, AI Center, Naval Research Laboratory, Washington, DC 20375, 1996.

[Sta67] J. R. Stagle. Automatic theorem proving with renamable and semantic resolution. *J. ACM*, 14:687–697, 1967.

[Tor97] José Torres. *Minimización del Ancho de Banda de un Grafo Usando un Algoritmo Genético.* PhD thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey, Campus Morelos, 1997.

[Tur36] A. M. Turing. On computable numbers, with aplication to the entschiedungs problem. *Proc. London Math. Soc.*, 42:230–265, 1936.

[Tur82] J. Turner. *Bandwidth and probabilistic complexity.* PhD thesis, NorthWestern University, 1982.

[WTC87] L. L. Liu W. T. Chen. Parallel aproach for theorem proving in propositional logic. *Inform. Sci.*, 41:61–76, 1987.