Índice

Aclaraciones Generales **2**

Introducción **3**

Situaciones de la vida real que se pueden modelar utilizando MAX-SAT **3**

Algoritmo exacto para MAX-SAT **3**

Algoritmo de fuerza bruta **3**

Algoritmo de backtracking **4**

Heurística constructiva para MAX-SAT **4**

Heurística de búsqueda local para MAX-SAT **4**

Metaheurística de búsqueda tabú para MAX-SAT **4**

Aclaraciones Generales

- La implementación de todos los algoritmos se realizó en lenguaje C++.
- Para calcular los tiempos de ejecución de los algoritmos se utilizó la función gettimeofday(), que se encuentra en la librería $< sys/time.h >$. Dado que dicha función funciona solamente en sistemas operativos de tipo linux, se debe compilar con el flag -DTIEMPOS en este tipo de sistemas para poder hacer uso de las mismas.
- Para la realización de los gráficos se utilizó Qtiplot

Introducción

En el presente trabajo se buscó realizar diferentes aproximaciones a la resolución del problema MAX-SAT. El problema MAX-SAT es un problema de optimización proveniente del problema de decisión SAT.

El problema SAT se basa es decidir si un conjunto de clausulas en forma normal conjuntiva, tiene alguna asignación de las variables que las componenen, tal que la evaluación de todas las clasulas sea verdadera con dicha asignación.

El problema SAT es un problema muy importante dentro del campo de la teoria de la complejidad, esto se debe a que SAT fue el primer problema que se identificó como NP-Completo. El Teorema de Cook demuestra que el algoritmo SAT pertenece a esta clase de algoritmos.

La importancia de este algoritmo no radica solamente en haber sido el primero en ser caracterizado como NP-Completo, se demostró que el problema SAT puede ser reducido al problema 3-SAT, que es basicamente el mismo problema pero en el cual todas las clausulas tienen un máximo de 3 literales. Además de probar la reducción, se demostro que este problema también pertenece a la clase NP-Completo (A diferencia del problema 2-SAT, para el cual se conoce un algoritmo polinomial para resolverlo). Esta reducción del problema a 3-SAT es un resultado importante ya que luego para probar que otros problemas se encuentran tambien en esta clase se utilizaron reducciones a 3-SAT mostrando la equivalencia en cuanto a la complejidad de resolución.

Situaciones de la vida real que se pueden modelar utilizando MAX-SAT

Algoritmo exacto para MAX-SAT

Como su nombre lo indica, el algoritmo exacto para Max-Sat se encarga de resolver el problema exactamente, arrojando la asignacion que valida la mayor cantidad de clausulas posibles. Dado que no se conoce ningún algoritmo polinomial para resolver este problema, se implementaron 2 algoritmos de complejidad exponencial. Por un lado se implementó un algoritmo de fuerza bruta de simple implementación pero de muy baja eficiencia, en cuanto a tiempo de ejecución. Por otro lado se implemento un algoritmo exacto mediante backtracking para poder evitar visitar todas las asignaciones de las variables posibles.

Algoritmo de fuerza bruta

En este algoritmo la idea es muy simple, se generan absolutamente todas las asignaciones posibles que existen, siendo estas 2^v donde v es la cantidad de variables. Luego, por cada una de las asignaciones se verifica cuantas clausulas valida, en el momento que una asignación supera el máximo de clausulas hasta el momento, se actualiza la cantidad de clausulas validadas, así como cual es la asignación que generó este máximo.

La idea de este algoritmo es tener una resolución muy simple del problema, es claro que el tiempo de ejecución va a ser muy malo ya que se revisan todas y cada una de las asignaciones posibles, y estas crecen en orden exponencial en

función de la cantidad de variables. Sin embargo, cabe destacar que el algoritmo provee una resolución exacta del problema y con baja probabilidad de errores dada la simpleza del mismo.

A continuación se presenta el pseudocódigo del mismo:

```
maxSatExacto(Vector clausulas, int variables)
vector asignacion
int max := 0
inicializar asignacion todos en falso
Para i = 1 hasta 2^variables
    int sat := 0
    Para j = 1 hasta tamanio(clausulas)
       Si haceTrue(asignacion, clausulas[j])
          sat:= sat + 1
       fin si
    fin para
    si sat > max
        actualizar max
        actualizar asignacionMax
    fin si
    asignacion := siguiente(asignacion,i+1)
fin para
devolver asignacionMax, max
```

Lo que muestra el pseudocódigo anterior es como, por cada asignación posible, se mira cada clausula y si la función *haceTrue* devuelve true, entonces se suma 1 a la cantidad de satisfechas por esa asignación. Por último, se mira cual asignación es la que tiene más clausulas satisfechas.

A continuación el muestra el pseudocodi

Algoritmo de backtracking

Heurística constructiva para MAX-SAT

Heurística de búsqueda local para MAX-SAT

Metaheurística de búsqueda tabú para MAX-SAT