Chaff

Un resolvedor SAT Eficiente

1. Introducción

El problema de satisfacibilidad booleana (SAT) consiste en determinar una asignación *V* de variables, para que una función booleana *f* sea verdadera, o determinar que tal *V* no existe.
SAT es uno de los problemas NP-Completos centrales.

1. Introducción

Muchos de los resolvedores SAT disponibles públicamente (GRASP, POSIT, SATO, WalkSAT…) han sido desarrollados, usando alguna combinación de dos estrategias principales:
La búsqueda con backtrack de Davis-Putnam (DP)
Búsqueda heurística local. Estas técnicas no garantizan completidad

1.1 Especificación del Problema

La mayoría de los resolvedores operan en problemas donde *f* esta dado en forma normal conjuntiva (*CNF*). La cual consiste en *AND's* lógicos para una o mas cláusulas, las cuales consisten de *OR's* lógicos de una o mas literales. Todas las funciones booleanas pueden ser descritas en *CNF*. En esta forma, para que *f* sea satisfecha (*sat*), cada cláusula debe ser *sat*

1.2 Búsqueda *DP* básica

```
while(true) {
    if(!decide())
        return(satisfiable);
    while(!bcp()){
        if(!resolveConflict())
            return(nonSatisfiable);
    }
}
bool resolveConflict() {
    d= most recent decision not tried both ways;
    if(d==NULL)
        return false;
    flip the value of d;
    mark d as tried both ways;
    undo any invalidated implications;
    return true; }
```

2. BCP Optimizado

En la práctica, para la mayoría de los resolvedores SAT, la gran mayoría de su tiempo de ejecución (mayor a 90% en el mayor de los casos) es gastado en el proceso BCP
Por lo que un motor eficiente BCP es la clave para cualquier resolvedor SAT

2. BCP Optimizado

Una cláusula es “*implied*” si y solo si todas excepto una de sus literales son cero (falsas). Entonces, necesitamos encontrar una manera para visitar rápidamente todas las cláusulas que justo se convierten *implied* por medio de una decisión.

¿Cómo hacerlo?

Revisar cada cláusula, que contenga una literal que ha sido puesta en cero por la asignación actual

Llevar contador para cada cláusula de cuántas literales con valor cero están en la cláusula

Pero……

Si la cláusula tiene n literales, no hay necesidad de visitarlas cuando 1,2,3, …,n-2 literales son puestas a cero. Solo sería necesario cuando el numero de ceros vaya de n-2 a n-1.

2. BCP Optimizado

Escojamos entonces dos literales cualesquiera que no estén asignadas en cero para cada cláusula. Hasta que una de esas dos sea asignada a cero, podemos asegurar que la cláusula no será *implied*. Entonces, solo es necesario visitar una cláusula dada, cuando una de sus dos literales sea asignada a cero.

2. BCP Optimizado

Cuando visitamos una cláusula, pasara una de dos situaciones:
La cláusula no esta *implied*, por lo que al menos 2 literales no están en cero, incluida la otra literal observada. Por lo que hay otra variable no observada que no esta en cero. Escogemos esta como reemplazo de la variable que acabamos de asignar a cero.
La cláusula esta *implied*. Seguimos el procedimiento para visitar una cláusula *implied* (lo que usualmente genera una implicación, a menos que la cláusula ya sea sat). Nótese que la variable *implied* siempre es la otra literal observada

3. Heurísticas para la decisión

La decisión consiste en determinar que variable y estado para esta debe ser seleccionado cada vez que decide() es llamado. La falta de evidencia estadística clara que diga que estrategia es mejor sobre otra, es difícil determinar que hace una buena estrategia de decisión y que hace una mala

3. Heurísticas para la decisión

La estrategia mas simple posible, es escoger la nueva decisión aleatoriamente, de entre las variables sin asignar (RAND)
El otro extremo, se podría usar una heurística que envuelva la maximización de una función moderadamente compleja del estado de las variables y la base de datos de las cláusulas (BOHM, MOM)

3. Heurísticas para la decisión

Una de las mas populares, que esta en un punto medio de las anteriores, es la “Dinamyc Largest Individual Sum” (DLIS) en la que se selecciona la literal que aparece mas frecuentemente en las cláusulas sin resolver

¿Cuál es mas rápida?...

¡¡No hay respuestas claras!!

3.1 VSIDS

Variable State Independent Decaying Sum

Cada variable en cada polaridad tiene un contador, inicializado en 0
Cuando se añade una cláusula , el contador asociado a cada literal se incrementa
La variable (sin asignación) y polaridad con el mayor contador es escogida en cada decisión
Los empates son resueltos aleatoriamente
Periódicamente, todos los contadores son divididos por una constante

3.1 VSIDS

Para escoger la variable con el contador mas alto aun mas rápido a la hora de la decisión, se debe tener una lista de las variables sin asignar, ordenada por el contador

Esta estrategia, puede verse como un intento de resolver conflictos entre cláusulas, pero mas en particular, conflictos recientes entre cláusulas

Principales Caracteristicas

- BCP Eficiente
  - Solo necesitamos examinar clausulas donde dos literales observados son puestos a **False**
  - Podemos remover cualquier asignación de literales puestos a **True.**
  - Podemos ignorar cualquier asignación a variables no observadas.
- Procedimiento de Decisión Eficiente
  - Reducción del espacio de búsqueda
- Procedimiento de Reinicio
  - Incremento de Robustes

BCP

- **Ejemplo**

·v2 + v3 + v1 + v4 + v5
·v1 + v2 + v3’
·v1 + v2’
·v1’+ v4
·v1’

Ejemplo

- Literales Observados

- v2 + v3 + v1 + v4 + v5
- v1 + v2 + v3’
- v1 + v2’
- v1’+ v4
- v1’

BCP

Ejemplo

- Hacemos v1 = F

- v2 + v3 + v1 + v4 + v5
- v1 + v2 + v3’
- v1 + v2’
- v1’+ v4