Razonamiento Automatizado
Proyecto 01

Iker Lissarrague Berumen Javier Enriquez Mendoza
Mauricio Hernandez Olvera

April 2019

Part I
SAT

1 Introducción

Nuestro objetivo con este proyecto es explorar la resolución de una configuración (de ser posible) de buscaminas modelandolo en Forma Normal Clausular.

2 Reglas de Buscaminas

Se comienza con un tablero de N columnas, M filas y un número m que define la cantidad de marcas de mina que se pueden colocar (y, consequentemente el número de minas distribuidas uniformemente en el tablero).

Las reglas de Buscaminas son las siguientes:

- Inicialmente todas las casillas están cubiertas.
- En cada turno, el jugador puede realizar una de las siguientes tres acciones:
    1. **Marcar** (de haber marcas de minas disponibles, $m > 0$) una casilla cubierta como una mina. Esta acción decrementa el número de marcas de minas en uno.
    2. **Desmarcar** una casilla previamente marcada como mina. Esta acción incrementa el número de marcas de minas en uno.
    3. **Explorar** una casilla cubierta. Si esta casilla contiene una mina, se pierde la partida inmediatamente. De otra forma, se muestra el número de casillas adyacentes (horizontal, vertical y diagonalmente) que contienen una mina. Si ninguna de las casillas adyacentes contiene una mina, el recuadro se mostrará vacio (es consistente con un 0, pero facilita la interacción del usuario).

- Se gana la partida si se han utilizado las m marcas de minas, y además estas marcas de mina **fueron colocadas en casillas minadas**.

3 Definiciones y notación

Definición 1. Una *configuración* de buscaminas es un tablero de $N \times M$ de casillas posiblemente cubiertas, un número m de marcas de minas y una asignación minas en las casillas del tablero (la asignación de minas no es accesible al jugador). El tablero puede estar parcialmente cubierto de números y marcas de minas.

Definición 2. Una *configuración inicial* de buscaminas es una configuración con el tablero totalmente cubierto, $m < N * M$ marcas de mina y una asignación de m minas distribuidas en el tablero uniformemente

Definición 3. Una *configuración válida* de buscaminas es una configuración a la que se puede llegar jugando buscaminas en una configuración inicial.

Determinar si una configuración es válida o no es un problema **NP**-Completo per se y no se profundizará en este proyecto.

Veamos además que en las configuraciones válidas todas las casillas corresponden a una de las siguientes clases:

- Es una mina correctamente marcada
- Es una mina incorrectamente marcada
- Es una casilla cubierta
- Es una casilla descubierta y tiene un número asociado a ella.

Para facilitar su analisis, estaremos manejando únicamente configuraciones válidas que, de contener marcas de mina, estas hayan sido correctamente marcadas, simulando así a un "jugador ideal" (no se marca una casilla como mina a menos que se tenga absoluta certeza de que esta es una mina).

Utilizaremos coordenadas de pantalla empezando de 0 para identificar univocamente las casillas, de tal forma que la casilla $(0, 0)$ corresponde a la esquina superior izquierda y la casilla $(M-1, 0)$ corresponde a la esquina superior derecha.

Definición 4. Diremos que los *vecinos* de una casilla (i, j), denotado como $N(i, j)$, es el conjunto de casillas adyacentes vertical, horizantal y diagonalmente a la casilla (i, j).

Tomemos como ejemplo la figura 1:

- $N(1, 1) = \{(0, 0), (1, 0), (2, 0), (0, 1), (2, 1), (0, 2), (1, 2), (2, 2)\}$

- $N(0,1) = \{(0,1),(1,0),(1,1)\}$

Definición 5. Definamos la *etiqueta* de una casilla descubierta (i,j), denotado como $L(i,j)$, como el número de vecinos con minas.

Veamos que, como hablamos únicamente de configuraciones válidas y por las reglas de buscaminas, $L(i,j)$ es exactamente el número que tiene asociada cada casilla descubierta.

Definición 6. Definamos como *vecinos marcados* de una casilla descubierta (i,j), denotado como $MN(i,j)$, como los vecinos (i,j) que han sido marcados como minas.

Definición 7. Definamos como *vecinos efectivos* de una casilla descubierta (i,j), denotado como $EN(i,j)$, como los vecinos cubiertos y no marcados de (i,j).

Definición 8. Diremos que la *etiqueta efectiva* de una casilla descubierta (i,j), denotado por $EL(i,j)$, es la etiqueta de (i,j) menos la cantidad de vecinos marcados de (i,j):

$$EL(i,j) = L(i,j) - |MN(i,k)|$$

4 Buscaminas en CNF

Para cada casilla cubierta (i,j), tendremos una variable $M_{i,j}$ que sera verdadera si y solo si la casilla (i,j) contiene una mina.

Pensemos primero en los casos sencillos. Supongamos que las siguiente configuraciones son válidas y las marcas de mina han sido colocadas correctamente:

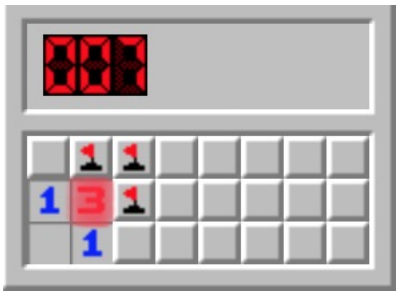

Figure 1: configuración de 8x3 y m=10

Observemos la casilla marcada en la figura 1. Veamos que $|MN(1,1)| = 3$. Por lo tanto tenemos que $EL(1,1) = L(1,1) - |MN(1,1)| = 3 - 3 = 0$. Es fácil ver que los demás vecinos efectivos de (1,1) son, con certeza, libres de mina.

Más formalmente

$$EL(x,y) = 0 \Rightarrow \forall (i,j) \in EN(x,y) \Rightarrow \neg M_{i,j}$$

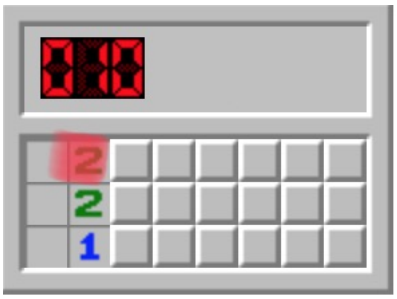

Figure 2: configuración de 8x3 y m=10

Similarmente, observemos la casilla marcada en la figura 2. Veamos que $|EN(1,0)| = 2 = EL(1,0)$. Es fácil ver que todos los vecinos efectivos de (1,0) son casillas minadas.

Más formalmente

$$|EN(x,y)| = EL(x,y) \Rightarrow \forall (i,j) \in EN(x,y) \Rightarrow M_{i,j}$$

Ya considerados los casos 'base', veamos un caso un poco más complejo:

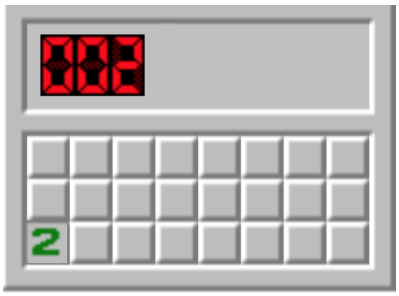

Figure 3: configuración de 8x3 y m=2

Sabemos que exactamente dos de las casillas (0,1), (1,1), (2,2) contienen una mina.

Esto es logicamente equivalente (y en general suele hacerse al 'traducir' a CNF) a:

- $LB(0,2)$: De las casillas (0,1), (1,1), (2,2) al menos dos contienen una mina

y

- $UB(0,2)$: De las casillas (0,1), (1,1), (2,2) a lo más dos contienen una mina

$LB(0,2)$ es equivalente a: $(M_{0,1} \vee M_{1,1}) \wedge (M_{0,1} \vee M_{2,2}) \wedge (M_{1,1} \vee M_{2,2})$

La intuición detrás de esto es la siguiente. Para que LB sea satisfacible todas las clausulas tienen que ser verdaderas. Si vemos las clausulas como implicaciones, por la equivalencia logica $p \Rightarrow q \equiv \neg p \vee q$, decimos que si alguna variable no es verdadera, digamos $M_{x,y}$, ($\neg M_{x,y}$ para denotar que es falsa) entonces la otra variable en las clausulas que contengan a $M_{x,y}$ ($M_{0,2}$ está en exactamente dos clausulas) deberán de serlo. Notemos que sí todas las variables son verdaderas, se sigue cumpliendo la especificación.

$UB(0,2)$ es equivalente a: $(\neg M_{0,0} \vee \neg M_{1,1} \vee \neg M_{2,2})$

La intuición detrás de esto es la siguiente. Para que UB sea satisfacible al menos una variable, digamos $\neg M_{x,y}$ debe de ser verdadera, forzando así la especificación.

Ahora, ¿cómo generalizamos estos dos conjuntos de clausulas para cada casilla descubierta?

Notemos que para capturar la intuición descrita anteriormente en el conjunto $UB(x,y)$, hicimos una clausula por combinación sin repetición de $\{\neg M_{i,j} \mid (i,j) \in EN(x,y)\}$ en $(EL(x,y)+1)$

Analogamente para el conjunto $LB(x,y)$ hicimos una clausula por combinación sin repetición de $\{M_{i,j} \mid (i,j) \in EN(x,y)\}$ en $(|EN(x,y)| - EL(x,y)+1)$

De tal forma que podemos aplicar la siguiente función a cada casilla descubierta para conseguir las clausulas de la configuración:

Clausulas *(x, y)*:
  if $EL(x,y) == 0$ **then**
    foreach $(i,j) \in EN(i,j)$ **do** Creamos la clausula $M_{i,j}$;
  else if $|EN(x,y)| == EL(x,y)$ **then**
    foreach $(i,j) \in EN(i,j)$ **do** Creamos la clausula $\neg M_{i,j}$;
  else
    $LB(x,y) \cup UB(x,y)$;

Si alguna casilla descubierta arrojó $M_{i,j}$, podemos marcar la casilla (i, j) como mina y volver a computar las clausulas de la configuración.

Analogamente para clausulas de la forma $\neg M_{i,j}$, podemos explorarlas con la certeza de que estas no son casillas minadas y volver a computar las clausulas de la configuración.

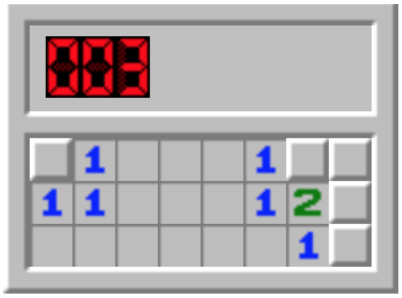

Figure 4: configuración de 8x3 y m=3

Utilizar la función mencionada previamente nos arroja las siguientes clausulas:

1. $M_{7,2} \vee M_{7,1} \vee M_{6,0}$
2. $M_{7,2} \vee M_{7,0} \vee M_{6,0}$
3. $M_{6,0}$
4. $M_{7,2} \vee M_{7,0} \vee M_{7,1}$
5. $M_{0,0}$
6. $M_{7,2} \vee M_{7,1}$
7. $\neg M_{7,1} \vee \neg M_{7,2} \vee \neg M_{6,0}$
8. $\neg M_{7,2} \vee \neg M_{6,0} \vee \neg M_{7,0}$
9. $M_{7,0} \vee M_{6,0} \vee M_{7,1}$
10. $\neg M_{7,2} \vee \neg M_{7,1} \vee \neg M_{7,0}$
11. $\neg M_{7,2} \vee \neg M_{7,1}$
12. $\neg M_{7,1} \vee \neg M_{6,0} \vee \neg M_{7,0}$

Marcando las casillas (0,0) y (6,0) como minas (por la clausula 5 y 3 respectivamente), y volviendo a computar las clausulas obtenemos las siguientes clausulas:

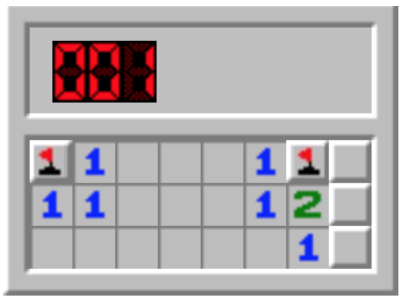

Figure 5: configuración de 8x3 y m=3

1. $\neg M_{7,1} \vee \neg M_{7,0}$
2. $\neg M_{7,2} \vee \neg M_{7,1}$
3. $M_{7,2} \vee M_{7,0} \vee M_{7,1}$
4. $M_{7,2} \vee M_{7,1}$
5. $\neg M_{7,2} \vee \neg M_{7,0}$

¿Qué podemos decir de estas clausulas?

Traduciendo las clausulas a Minisat obtenemos lo siguiente:

```
p cnf 3 5
-2 -1 0
-3 -2 0
3 1 2 0
3 2 0
-3 -1 0
```

Donde $M_{7,0} \to 1$, $M_{7,1} \to 2$, $M_{7,2} \to 3$

Y obtenemos el siguiente resultado:

```
SAT
-1 2 -3 0
```

Si forzamos a Minisat a no considerar el modelo anterior, obtenemos lo siguiente:

```
SAT
-1 -2 3 0
```

Lo que significa que hay más de un modelo posible. Sin embargo, cada configuración tiene una única asignación de minas. Por lo que podemos concluir que existe un elemento no-determinista en Buscaminas!

No encontraremos forma de librarnos completamente de adivinar, pero se puede hacer un analisis estadistico para así elegir casillas 'más seguras', minimizando así el número de inevitables partidas perdidas. Esto, sin embargo, va más allá del alcance de este proyecto.

Actualmente no tomamos en cuenta la variable global m, que nos dice que, de las casillas que quedan cubiertas, exactamente m son minas.

De forma parecida a lo que hicimos con las casillas, esto es logicamente equivalente a:

- GLB: De las casillas cubiertas restantes, al menos m son minas.

y

- GUB: De las casillas cubiertas restantes, a lo más m son minas.

Para GUB hacemos una clausula por combinación sin repetición de $\{\neg M_{i,j} \mid (i,j)$ es una casilla cubierta$\}$ en $(m+1)$

Para GLB hacemos una clausula por combinación sin repetición de $\{M_{i,j} \mid (i,j)$ es una casilla cubierta$\}$ en (# de casillas cubiertas-$m+1$)

Hacer esto descarta algunos posibles modelos pero aumenta considerablemente el número de clausulas.

Definamos entonces el algoritmo general para un movimiento de buscaminas (Supongamos la existencia de funciones CasillasCubiertas(c) y CasillasDescubiertas(c), que regresan el conjunto de casillas cubiertas y descubiertas de una configuración c):

MovimientoBuscaminas *(c : Configuración)*:
  $cnf \longleftarrow \{\}$;
  $cd \longleftarrow CasillasDescubiertas(c)$;
  $cc \longleftarrow CasillasCubiertas(c)$;
  foreach $(i, j) \in cd$ **do** $cnf \longleftarrow (cnf \cup Clausulas(i, j))$;
  $cnf \longleftarrow cnf \cup GUB(c) \cup GLB(c)$;
  if *cnf contiene una clausula unitaria positiva* $M_{x,y}$ **then**
    Marcamos $M_{x,y}$ como mina;
  else if *cnf contiene una clausula unitaria negativa* $\neg M_{x,y}$ **then**
    Exploramos $M_{x,y}$;
  else if $cnf \cup \{\neg M_{x,y}\}$ *es insatisfacible para algún* $(x, y) \in cc$ **then**
    Marcamos $M_{x,y}$ como mina;
  else if $cnf \cup \{M_{x,y}\}$ *es insatisfacible para algún* $(x, y) \in cc$ **then**
    Exploramos $M_{x,y}$;
  else
    Exploramos (x, y) para algún $(x, y) \in cc$ aleatorio;

Part II
Model Checking

5 Algoritmo de Euclides

```
int euclides(int u,int v)
{
    int t;
    while(v != 0)
    {
        t = 0;
        v = u % v;
        u = t;
    }
    return u;
}
```

Primero transformamos el algoritmo anterior para que el paso de parámetros sea por referencia. Rsultando el programa del código siguiente.

```
void euclides(int *u,int *v, int *t)
{
    while(*v != 0)
```

```
    {
        *t = 0;
        *v = *u % *v;
        *u = *t;
    }
}
```

Ahora lo transformaremos a un programam GO-TO

```
void euclides(int *u,int *v, int *t)
{
    eu:
        *t = 0;
        *v = *u % *v;
        *u = *t;
    if(*v != 0)
    {
        goto eu;
    }
}
```