Algoritmo genético para la reducción de ecuaciones de funciones binarias expresadas en maxitérminos y minitérminos.

Carlos Delgado, Edgar Moncada, Luis F. Vargas

Junio de 2012

Resumen

En este proyecto se busca la simplicación de ecuaciones de funciones booleanas utilizando un algoritmo evolutivo, que busque automatizar el proceso que se realiza con los mapas de Karnaugh

1. Fundamentación del problema

Se busca que en la construcción de circuitos digitales para el cálculo de funciones booleanas sea del menor tamaño posible, existen dos posible formas de representar una función booleana en términos de compuertas *and* y *or*:

- **Minitérminos** Se trata de representar la función en cláusulas, cada clausula es determinada por cada *1* que tiene la función, cada claúsula se crea conectando las entradas con compuertas *and* y las cláusulas entre sí se conectan con compuertas *or*
- **Maxitérminos** Se trata de representar la función en cláusulas, cada clausula es determinada por cada *0* que tiene la función, cada cláusula se crea conectando las entradas con compuertas *or* y las cláusulas entre sí se conectan con compuertas *and*, cada entrada se representa como su negación

Por ejemplo, para la siguiente función:

x_1	x_0	$f(x)$
0	0	1
0	1	1
1	0	0
1	1	0

En representación de minitérminos es así.

$$(\sim x_1 \wedge \sim x_0) \vee (\sim x_1 \wedge x_0) \tag{1}$$

En respresentación de maxitérminos:

$$(\sim x_1 \vee x_0) \wedge (\sim x_1 \wedge \sim x_0) \tag{2}$$

Sin embargo, cuando las funciones tienen muchas variables de entrada, el costo (número de compuertas) presenta un gran incremento, por lo que se deben simplificar las funciones, existen dos métodos:

- **Algebra de Boole** Mediante el uso de relaciones lógicas se simplifican las expresiones, sin embargo, no es bueno cuando las funciones son extensas, ya que se torna engorroso.
- **Mapas de Karnaugh** Una buena solución, ya que utiliza una representación de matrices ordenadas en codificación Grey (cada símbolo cambia un bit en cada columna o fila) y permite asociar directamente para realizar la simplicación, éste metodo es muy bueno hasta cierto tamaño donde las matrices ya son muy grandes y requiere algún esfuerzo para simplificar el problema.

-	x_1	$\sim x_1$
x_0	1	1
$\sim x_0$	0	0

Para la ecuación, en minitérminos:

$$(\sim x_1 \wedge \sim x_0) \vee (\sim x_1 \wedge x_0) \tag{3}$$

En maxitérminos:

$$(\sim x_1 \vee \sim x_0) \wedge (\sim x_1 \vee x_0) \tag{4}$$

2. Algoritmo Genético

2.1. El cromosoma

El cromosoma se codifica como una matriz de tamaño número de clausulas (que es un número entre 1 y número de posibles entradas que es el peor caso donde existe una clausula por cada entradas donde $entradas = 2^{bits}$) ejemplo:

x_1	x_0	$f(x)$
0	0	0
0	1	1
1	0	1
1	1	0

Un cromosoma que representa esta función es:

x_1	$\sim x_1$	x_0	$\sim x_0$
0	1	1	0
1	1	0	0
0	0	1	0

Que en representación de minitérminos:

$$(\sim x_1 \wedge x_0) \vee (\sim x_1 \wedge x_1) \vee x_0 \tag{5}$$

Que en representación de maxitérminos:

$$(\sim x_1 \vee x_0) \wedge (\sim x_1 \vee x_1) \wedge x_0 \tag{6}$$

2.2. Población inicial

Para el problema se utilizan poblaciones iniciales de 200 individuos por defecto, aunque el usuario puede indicar un número, que debe ser mínimo 100 y máximo 1000

2.3. Función de aptitud

La función de aptitud considera los siguiente factores:

- **Número de cláusulas** Número de cláusulas en el cromosoma, se busca minimizar este valor.
- **Acercamiento a la función** Es el factor más importante se busca que este valor sea 0, es decir que la función encontrada sea correcta

La función de aptitud se calcula
$F_{aptitud} = 25 * N_{clausulas} + 75 * (N_{aciertosEntrada} - N_{posible_entrada})$
Se le da un mayor peso a la diferencia entre los aciertos con la entrada y el número de entradas ya que el objetivo principal es que la función que encuentre pueda representar la entrada.

Se busca que la función de aptitud sea las mas pequeña posible, es decir los mejores cromosomas son aquellos que tengan el menor valor de función de aptitud.

2.4. Función de selección

En este caso se utiliza la selección por ruleta.

2.5. Cruce

Se seleccionan dos cromosomas dentro del grupo de seleccionados, denominados padre y madre, y se igualan la cantidad de cláusulas, es decir que si la madre tiene 3 cláusulas y el padre tiene 2, se toma la primera cláusula de la madre y se inserta al final del padre para que tengan el mismo número.

Se genera un número aleatorio α entre 1 y el número de cláusulas menos 1. Con este valor se generaran dos hijos de la siguiente manera:

- **Hijo 1:** se construye con α cláusulas, y sus cláusulas contienen los primeros $\alpha/2$ elementos del padre y de la madre, si α es impar se toma al azar un elemento del padre o de la madre.
- **Hijo 2:** se construye con numero de clausulas de los padres menos α mas 1 cláusulas, y sus cláusulas contienen el resto $\alpha/2$ elementos del padre y de la madre, si α es impar se toma al azar un elemento del padre o de la madre.

Ejemplo:

Tabla 1: Cromosoma A

x_1	$\sim x_1$	x_0	$\sim x_0$
0	1	1	0
1	1	0	0
0	0	1	0

Tabla 2: Cromosoma B

x_1	$\sim x_1$	x_0	$\sim x_0$
0	0	1	0
1	1	1	0

Se igualan los cromosomas y dado que el Cromosoma B tiene menor número de claúsulas queda de la siguiente manera:

Tabla 3: Cromosoma B

x_1	$\sim x_1$	x_0	$\sim x_0$
0	0	1	0
1	1	1	0
0	0	1	0

En ese caso se ha insertado la primera posición en la última, queda como una claúsula redundante.

Ahora se tienen que el número de cláusulas es 3 por tanto α puede valer 1 ó 2. Se elige a α con valor 2 para este ejemplo y el primer hijo es:

Tabla 4: Hijo 1

x_1	$\sim x_1$	x_0	$\sim x_0$
0	1	1	0
0	0	1	0

En este caso se ha insertado un cromosoma del padre y uno de la madre.

Para el segundo hijo el número de clausulas será 3 (número de cláusula padre) menos 2 (α) mas 1 igual a 2 y se toman las cláusulas restantes:

Tabla 5: Hijo 2

x_1	$\sim x_1$	x_0	$\sim x_0$
0	0	1	0
1	1	1	0

En este caso se ha insertado un cromosoma del padre y uno de la madre.

2.6. Mutación

Para mutar se selecciona el 2 % de los individuos, en éstos se selecciona aleatoriamente una cláusula, con probabilidad del 50 % se realiza alguna de estas dos acciones

- **Borrar cláusula** Borra una clausula, si es la única de la función, la salida de la misma es siempre 0
- **Cambiar el valor de una variable** Se selecciona una posición de la cláusula y se cambia el valor que tiene asignado por su negación

Si dado el caso se llega seleccionar borrar clausula en un cromosoma con una sóla clausula se aplica la segunda que es cambiar el valor de una variable, para evitar inconsistencias.

2.7. Selección de los cromosomas que pasan a la siguiente generación

En este algoritmo se mantiene el número de la población inicial, por lo tanto para ingresar los hijos se eliminan los peores padres.

2.8. Criterio de parada

El criterio de parada se presenta en dos casos

- Al pasar 200 generaciones
- Si la aptitud del mejor cromosoma en cada generación, no tiene cambios en 10 generaciones

3. La aplicación

3.1. Parámetros de entrada

Tabla 6: Parámetros de la aplicación

Parámetro	**Tipo dato**	**Descripción**
-f	Cadena de caracteres	Nombre archivo de entrada. Por defecto *input.txt*
-p	Número natural	Población inicial (Por defecto 200) Min: 100 Máx: 1000
-i	Número natural	Número de interacciones o generaciones (Por defecto 500) Min: 1 Máx: 1000
-t	Booleano (0 o 1)	Indica si se va trabajar por minitérminos o maxitérminos 0 o 1 respectivamente (por defecto 0 o false)
-o	Cadena de caracteres	Nombre archivo de salida. Por defecto *output.txt*

3.2. Representación de la población

La población se representa como un vector de matrices, donde cada matriz representa un individuo.
Ejemplo:

Tabla 7: Representación de población

Individuo	x_1	$\sim x_1$	x_0	$\sim x_0$
Cromosoma 1	0	1	1	0
	1	1	0	0
	0	0	1	0
Cromosoma 2	0	1	1	0
	1	1	0	0
	0	0	1	0
Cromosoma N	0	1	1	0
	1	1	0	0
	0	0	1	0
	0	1	1	0

3.3. La entrada

La entrada se codifica en un archivo de entrada, en cuya primera linea tiene un número natural B que indica el número de bits de la entrada, en las siguiente 2^B líneas se especifica la función booleana, ésta función debe

ingresarse en orden de codificación binaria y en la última columna debe tener el valor que toma ante una entrada específica, en la siguiente línea tiene un número natural para indicar el tamaño de la siguiente entrada, sí este número es 0 significa que no hay más entradas.

```
2
0 0 1
0 1 1
1 0 0
1 1 0
3
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 0
1 1 0 0
1 1 1 1
0
```

3.4. Salida

La salida muestra las ecuaciones resultantes de cada solución, la salida tiene la forma solución # <numero de solución>y a continuación la ecuación en términos de x que la representan, ejemplo:

Si es en minitérminos

```
Solución # 1:
(∼x0 and x1) or (∼x1)
Solución # 2:
(∼x0 and x2) or (∼x4) or (x4 and x3)
```

Si es en maxitérminos

```
Solución #1:
(∼x0 or x1) and (∼x1)
Solución #2:
(∼x0 or x2) and (∼x4) and (x4 or x3)
```

4. Análisis y resultados

Se realizan varias pruebas, variando el número de bits de entrada y los tipos de funciones, los resultados son los siguientes

4.1. Entradas pequeñas

Las pruebas se realizan con dos entradas de 1,2 y 3 bits, se encuentra que el sistema encuentra la ecuación resultante de forma rápida (menos

de 20 iteracciones).

Las entradas de prueba son:

```
1
0 0
1 1
2
0 0 1
0 1 1
1 0 0
1 1 0
3
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 0
1 1 0 0
1 1 1 1
0
```

La salida correspondiente es:

```
-------------------------------------------------

SOLUCION ENTRADA #1
-------------------------------------------------

Numero de iteracciones 11

Datos Mejor cromosoma
1 0
Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
0                 0                         0
1                 1                         1

-------------------------------------------------

SOLUCION ENTRADA #2
-------------------------------------------------

Numero de iteracciones 11

Datos Mejor cromosoma
0 1 0 0
Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
```

```
00                1                          1
01                1                          1
10                0                          0
11                0                          0

-------------------------------------------------

SOLUCION ENTRADA #3
-------------------------------------------------

Numero de iteracciones 19

Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
000               1                          1
001               0                          0
010               1                          1
011               0                          0
100               1                          1
101               0                          0
110               1                          0
111               1                          1
```

Como se ve en el caso de los 3 bits, no corresponde la función encontrada ante la entrada 110

4.2. Entradas grandes

Se realiza una prueba con 8 bits. Se muestran sólo aquellos donde la función falla.

```
-------------------------------------------------

SOLUCION ENTRADA #8
-------------------------------------------------

Aptitud 9675
Numero de iteracciones 13

Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
00000001                  1                0
00000010                  1                0
00000011                  1                1
00000100                  1                0
00011000                  1                0
01001001                  1                0
01001010                  1                0
```

01001011	1	0
01010111	1	0
01011010	1	0
01011011	1	0
01011100	1	0
01011101	1	0
01101101	1	0
01101110	1	0
01110100	1	0
01110101	1	0
01111010	1	0
01111011	1	0
01111100	1	0
01111101	1	0
01111110	1	0
01111111	1	1
10000000	1	1
10000001	1	0
10000010	1	0
10000011	1	0
10000100	1	0
10000101	1	0
11000111	1	0
11001010	1	0
11001100	0	0
11001110	1	0
11010010	1	0
11010011	1	0
11100010	0	1
11100011	0	1
11100100	1	0
11100101	1	0
11100110	1	0
11100111	1	0
11101000	0	0
11101001	0	1
11101010	0	1
11101011	0	1
11101100	0	1
11101101	1	0
11101110	1	0
11101111	1	0
11110000	1	0
11110001	1	0
11110010	1	0
11111111	1	0

4.3. Entradas de funciones con muchos 1

Se ingresa una entrada y la solución es la siguiente:

```
-------------------------------------------------

SOLUCION ENTRADA #1
-------------------------------------------------

Aptitud 325
Numero de iteracciones 12

Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
0000              0                          1
0001              0                          0
0010              0                          0
0011              0                          1
0100              1                          1
0101              1                          1
0110              1                          1
0111              1                          1
1000              1                          1
1001              0                          1
1010              1                          1
1011              0                          0
1100              1                          1
1101              1                          1
1110              1                          1
1111              1                          1
```

4.4. Entradas de funciones con muchos 0

Se ingresa una entrada y la solución es la siguiente:

```
-------------------------------------------------

SOLUCION ENTRADA #1
-------------------------------------------------

Aptitud 100
Numero de iteracciones 15

Datos Mejor cromosoma
1 0 0 1 0 0 1 1
Comparaciones
Entrada | Tabla de verdad entrada | Tabla
    de Verdad cromosoma
0000              0                          0
```

0001	0	0
0010	0	0
0011	0	0
0100	0	0
0101	0	0
0110	0	0
0111	0	0
1000	1	1
1001	0	0
1010	0	0
1011	0	0
1100	1	1
1101	0	0
1110	0	0
1111	0	1

5. Conclusiones

- La codificación de cada variable y su negada muestra ser apropiada para ser aplicada a un algoritmo evolutivo, ya que permite realiza con gran facilidad las operaciones de mutación y cruce correctamente.
- Se encuentra que el algoritmo se acerca muy rápidamente a una solución cerca a la óptima pero tarda mucho en converger, lo que explica que los resultandos incorrectos fallan en la salida en pocas entradas. Cuando se elimina la condición de parada de 10 generaciones, la ganancia no es muy notoria. Lo que permite concluir que la función de aptitud debe ser más detallada para poder conseguir mejores resultados.
- El comportamiento ante entradas pequeñas, es muy bueno, ya que encuentra buenas soluciones, en el caso de entradas grandes encuentra una aproximación que no es tan buena, pero cubre en gran medida la función, el algoritmo no es bueno ante entradas con muchos 1 y es muy bueno ante entradas con muchos 0. Esto indica que se debe realizar un análisis del comportamiento de los operadores de cruce con respecto al paso de las características de los padres a los hijos.
- Se debe elaborar una mejor función de aptitud para que el sistema encuentre en todos los casos una solución total ante una entrada, ya que las pruebas realizadas indican que en las entradas grandes, se encuentra una buena aproximación pero no la función que concuerda totalmente con la entrada. Sin embargo, en las pruebas realizadas en el proyecto indica que la taza de error es baja.
- El método aplicado para diseñar el algoritmo evolutivo muestra ser una buena herramienta para crear una aplicación para la simplificación de ecuaciones de funciones booleanas, si se realizan algunos ajustes.

6. Referencias

Referencias

[1] Ronald J. Tocci,Neal S. Widmer, “Sistemas digitales: Principios y aplicaciones”, Prentice Hall, 2003

[2] Garcia. Angel, “Guias de Clase Computación Evolutiva”, Universidad del Valle, 2012