Esbozo de un método rápido para probar Covering Arrays

Ing. Vicente Oscar Mier Vela

10 de julio del 2013

Curso propedéutico del 2013, a cargo del Dr Jose Torres-Jimenez, CINVESTAV, UNIDAD TAMAULIPAS, LABORATORIO DE TECNOLOGÍAS DE INFORMACIÓN, Parque Científico y Tecnológico TECNOTAM – Km. 5.5 carretera Cd. Victoria-Soto La Marina C.P. 87130 Cd. Victoria, Tamps. Teléfono: (834) 107 02 20 – Fax: (834) 107 02 24 y (834) 314 73 92, vinculacion@tamps.cinvestav.mx

Resumen

Un método sugerido de pruebas por Covering Arrays para probar la si una matriz binaria es, ó no, una Covering Array de t=2, v=2. NOTA: La efectividad de este método es hipotética. Hace falta ponerlo a prueba.

1. Introducción

Para probar si una matriz de valores binarios de un solo dígito, es, ó no es, una Covering Array de t=2, v=2, es necesario verificar que, en cada pareja posible de columnas, existan las combinaciones {00,01,10,11}.

Es posible probar si existe esta propiedad en cualquier matriz binaria, si aplicamos una verificación por fuerza bruta, que analice todas las parejas posibles de columnas de la matriz. Si la matriz es demasiado grande, quizás no sea conveniente aplicar este método, ya que el valor de pruebas necesarias sería proporcional a la cantidad de posibles combinaciones de columnas, dada por la fórmula

$$p=\binom{k}{2}$$

Donde p es la cantidad de posibles parejas de columnas, y k es la cantidad de columnas. La cantidad de columnas puede obtenerse por medio de

$$k \leq \binom{N-1}{\lceil \frac{N}{2} \rceil}$$

Donde N determina el tamaño de una Covering Array óptima en N de t=2, v=2.

Es posible construir una Covering Array de t=2, v=2 realizando un conteo binario ordenado en polinomios mayor-qué de las combinaciones posibles con $\lceil \frac{N}{2} \rceil$ 1s y $N - 1 - \lceil \frac{N}{2} \rceil$ 0s. Si este conteo se almacena en una matriz, en la que cada fila sea uno de los valores binarios ordenados correspondientes a cada una de las combinaciones posibles, y después, se rota 90 grados hacia la izquierda dicha matriz, y se le concatena una fila llena de 0s, se obtiene una Covering Array de t=2, v=2.

Las matrices generadas por este método requieren de una sola variable para ser generadas: N.

Si quisiéramos utilizar una Covering Aarray de fuerza 2 para probar combinaciones de parejas de columnas, esta tendría que contar con suficientes bits para generar los pares de valores decimales correspondientes a parejas de columnas de una matriz binaria a la que se le aplica la prueba. Llamaremos CA_ta la Covering Array de fuerza 2 con W_t columnas que utilizaríamos para probar si la matriz binaria M con W_M es, ó no es, una CA de de t=2, v=2.

2. Algoritmo

Si generamos CA_t utilizando el método anterior, a partir de un valor de N, deberían cumplirse las siguientes condiciones:

$$W_t \geq 2 \log_2 W_M$$

Esto, para que cada fila de CA_t pueda representar un par determinado de columnas. También:

$$W_t = \binom{N-1}{\lceil \frac{N}{2} \rceil}$$

Ya que, al rotar la matriz 90 grados hacia la izquierda, la cantidad de combinaciones posibles de los valores binarios mencionados anteriormente, se convierte en la cantidad de columnas de la CA resultante.

Es necesario entonces encontrar un valor de N tal que $W_t \geq 2 \log_2 W_M$

Para lograrlo, podríamos utilizar la siguiente aproximación:

$$\binom{n}{k} \approx \frac{n^k}{k!}$$

Si

$$n\text{=N-1}$$

$$k = \left\lceil \frac{N}{2} \right\rceil$$

Entonces

$$N \approx \sqrt[\lceil \frac{N}{2} \rceil]{\left\lceil \frac{N}{2} \right\rceil ! \times W_t} + 1$$

Dado que $N \geq 3$ (para que $W_t \geq 2$), debe cumplirse que

$$\left\lceil \frac{N}{2} \right\rceil \leq \left\lceil \frac{N}{2} \right\rceil ! \times W_t$$

$$\frac{N}{2 \lceil \frac{N}{2} \rceil !} \leq W_t$$

Para que el resultado de la raíz arroje, por lo menos, un valor mayor ó igual a 1. Sin embargo, puede resultar un poco difícil despejar N en la desigualdad anterior, por lo que es preferible utilizar una aproximación menos detallada:

$$\binom{n}{k} \approx n^k$$

$$N \approx \sqrt[\lceil \frac{N}{2} \rceil]{W_t} + 1$$

Utilizando el razonamiento anterior aquí, es fácil ver que debe cumplirse

$$N \leq 2W_t$$

Es seguro establecer que $N \leq 2W_t$ ya que $\frac{N}{2 \lceil \frac{N}{2} \rceil !} \leq \frac{N}{2} \leq W_t$ para $N \geq 3$

Esto establece un límite superior para N que nos permite acelerar la aproximación a una solución de N en

$$N \approx \sqrt[\lceil \frac{N}{2} \rceil]{\left\lceil \frac{N}{2} \right\rceil ! \times W_t} + 1$$

utilizando aproximaciones sucesivas ó tabulación.

Habiendo obtenido un valor apropiado de N, es posible generar CA_t. Construir CA_t con la N obtenida podría resultar en una CA_t con $W_t > 2 \log_2 W_M$, en cuyo caso, deberán eliminarse las columnas excedentes de CA_t. Debido a que, al eliminar una columna cualquiera de una CA de fuerza dos, las demás parejas de columnas siguen preservando la propiedad de contener {00, 01, 10, 11}, la eliminación de columnas de una CA no debe ser un problema. Sin embargo: quizás la eliminación de ciertas columnas pueda eficientizar el proceso de prueba de alguna manera.

Habiendo obtenido una CA_t con la cantidad deseada de columnas, es posible recorrer sus filas para seleccionar pares de números binarios de $\log_2 W_M$ bits que representen pares de columnas en M. Llamemos A al número binario con bits desde el bit 0 hasta el bit $\log_2 W_M - 1$ y B al número binario con bits desde el

bit $\log_2 W_M$ hasta el bit $2\log_2 W_M$-1 en cualquier fila de CA_t. Para realizar un barrido que compruebe la propiedad {00, 01, 10, 11} en un par de columnas C_A y $C_B$de M, deberá cumplirse que

$$A < B$$

De lo contrario, es preferible ignorar el barrido que sugiera la fila de CA_t, ya que podrían repetirse parejas de columnas de M, ó comparar una columna de M consigo misma.

3. Generador de CAs en C

El siguiente código fuente utiliza el método mencionado en la introducción para contruir una CA de t=2, v=2.

```
#include <stdio.h>
#include <stdlib.h>

int coef_bin2(int, int);

int main(int argc, char *argv[]) {
        int i, j, m, ceroextremo, cuentaunos, yaconteunos, \
unosconsecutivos, donde, cuantos, llenaceros, coef, r, c;
        int N = atoi(argv[1]);
        int k = N-1;
        int t = (N/2)+(N%2);
        int B[k];
        coef=coef_bin2(k, t);
        int CA[k+1][coef];
        for(i=0;i<t;i++){
                B[i]=1;
        }
        for(i=t;i<k;i++){
                B[i]=0;
        }
        int k1 = k;
        int k2 = 2*k;
        int k3 = 3*k;
        i=0;
        j=0;
        ceroextremo=0;
        cuentaunos=0;
        yaconteunos=0;
        unosconsecutivos=0;
        donde=0;
        cuantos=0;
```

```
llenaceros=0;
c = 0;
r = k;
while(1){
        if(i<k1){
                CA[r--][c]=B[i];
        }
        if(i>=k && i<k2){
                j=i-k;
                if(B[k-j-1]==0 && j==0){
                        ceroextremo=1;
                        i=k2;
                        continue;
                }
                if(B[k-j-1]==1 && j==0){
                        cuentaunos=1;
                        cuantos++;
                        i++;
                        continue;
                }
                if(cuentaunos == 1 && B[k-j-1] == 1){
                        cuantos++;
                        i++;
                        continue;
                }
                if(cuentaunos == 1 && B[k-j-1] == 0){
                        if(j==k-1){
                                break;
                        }
                        yaconteunos = 1;
                        cuentaunos = 0;
                        i++;
                        continue;
                }
                if(yaconteunos == 1 && B[k-j-1] == 0 && j < k-1){
                        i++;
                        continue;
                }
                if(yaconteunos == 1 && B[k-j-1] == 1 ){
                        unosconsecutivos=1;
                        donde=k-j;
                        cuantos++;
                        i=k2;
                        continue;
                }
                if(yaconteunos == 1 && B[k-j-1] == 0 && j >= k-1){
```

```
                break;
        }
}
if(i>=k2 && i<k3){
        j=i-k2;
        if(unosconsecutivos == 1 && j == donde-1){
                B[j]=0;
                i++;
                continue;
        }
        if(unosconsecutivos == 1 && j>=donde && \
        cuantos-- > 0 && ceroextremo == 0){
                B[j]=1;
                i++;
                continue;
        }
        if(unosconsecutivos == 1 && cuantos < 0 && \
        ceroextremo == 0){
                unosconsecutivos = 0;
                llenaceros = 1;
                B[j]=0;
                i++;
                continue;
        }
        if(llenaceros == 1 && ceroextremo == 0){
                B[j]=0;
                i++;
                continue;
        }
        if(ceroextremo == 1 && B[k-j-1] == 1){
                ceroextremo = 0;
                B[k-j-1]=0;
                B[k-j]=1;
                i++;
                continue;
        }
}
if(i>=k3){
        i=0;
        j=0;
        ceroextremo=0;
        cuentaunos=0;
        yaconteunos=0;
        unosconsecutivos=0;
        donde=0;
        cuantos=0;
```

```
                        llenaceros=0;
                        r=k;
                        c++;
                        continue;
                }
                i++;
        }
        for(i=0;i<coef;i++){
                CA[0][i]=0;
        }
        for(i=0;i<k+1;i++){
                for(j=0;j<coef;j++){
                        printf("%d",CA[i][j]);
                }
                printf("\n");
        }
        return 0;
}

int coef_bin2(int n, int k) {
        int c = 1;
        int limit = k-1;
        int stop = n-limit;
        int result = n;
        while(n>stop){
                result *= --n;
                result /= c++;
        }
        return result/c;
}
```

4. Notas finales

Todavía hace falta probar y codificar el algoritmo sugerido en 2. Su efectividad es todavía hipotética.