ORGANIZACION Y ARQUITECTURA DE COMPUTADORAS
SEGUNDO TERMINO 2013 - 2014
PRIMER PROYECTO

GRUPO # 2
JAIME CASTELLS PEREZ
JAVIER RON ARTEAGA

PROJECT WRITE-UP

En esta primera entrega de proyecto se presentaran las funcionalidades requeridas en la tarea publicada en el sistema SIDWeb bajo el nombre de “First Project – Assembly Language”.

La tarea consiste en 5 partes descritas a continuación con sus respectivas respuestas y soluciones:

PARTE A:

Usando el generador de números aleatorios del simulador MARS, crear un arreglo de tamaño 600x2 y llenarlo con números mayores a 0, y menores a 1000; y guardar esta información en un archivo llamado “randomMIPS.txt”.

SOLUCION:

Se creó un archivo generarAleatorios.asm, que cumple con la funcionalidad pedida.

```
        li      $t0, 600        # $t0 = number of rows
        li      $t1, 2        # $t1 = number of columns
        move    $s0, $zero      # $s0 = row counter
        move    $s1, $zero      # $s1 = column counter
        move    $t2, $zero      # $t2 = the value to be stored

       li   $v0, 13       # system call for open file
       la   $a0, fout     # output file name
       li   $a1, 1        # Open for writing (flags are 0: read, 1: write)
       li   $a2, 0        # mode is ignored
       syscall            # open a file (file descriptor returned in $v0)
       move $s6, $v0      # save the file descriptor

#  Each loop iteration will store incremented $t1 value into next element of matrix.
#  Offset is calculated at each iteration. offset = 4 * (row*#cols+col)
#  Note: no attempt is made to optimize runtime performance!

loop:   mult    $s0, $t1       # $s2 = row * #cols  (two-instruction sequence)
        mflo    $s2            # move multiply result from lo register to $s2
        add     $s2, $s2, $s1  # $s2 += column counter
        sll     $s2, $s2, 2    # $s2 *= 4 (shift left 2 bits) for byte offset

       li  $v0, 42                # service 42 is to set up the upperbound of the random number
       addi $a1, $zero, 1000  # loading upperbound
       syscall



       ############################## FUN ###################################


       li      $t3,    10
       div     $a0,    $t3
```

Ilustración 1: generarAleatorios.asm

Del cual se obtuvieron los resultados. Se generaron archivos aleatorios, y cada dígito, uno a uno, se lo transforma en strings para ser guardados en un archivo de texto. Este archivo se llama randomMIPS.txt.

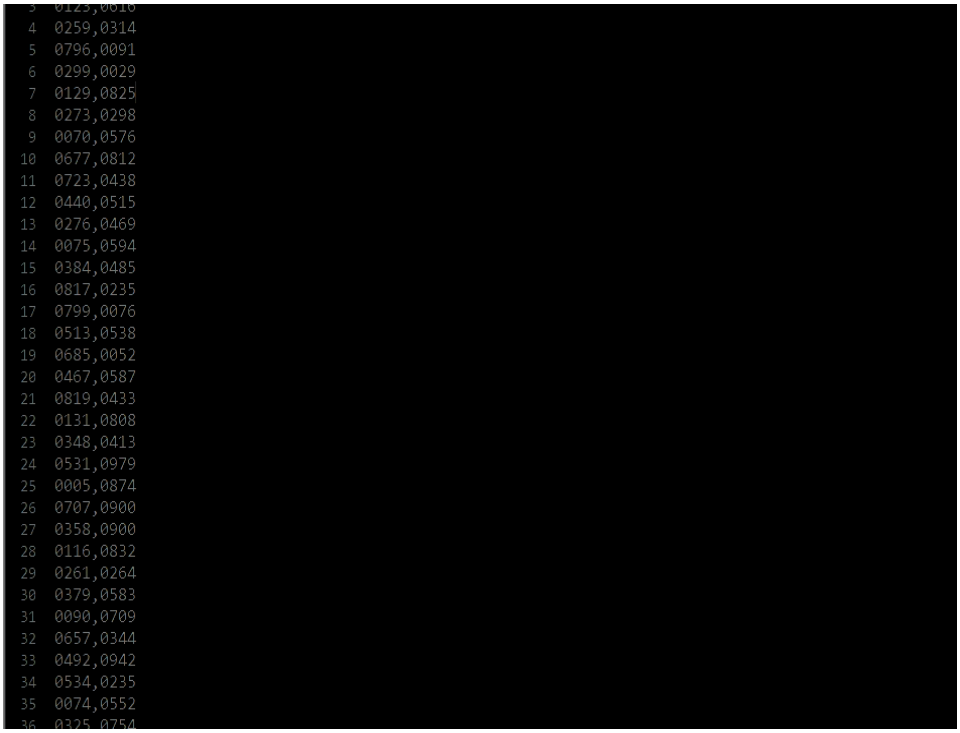

Ilustración 2: randomMIPS.txt

PARTE B:

Para esta parte se requirió algo similar a la la parte A, con la excepción de que en lugar de usar el generador proveído por MIPS, se implemente un generador de aleatorios de tipo “Multiply-With-Carry”, inventado por George Marsaglia.

SOLUCION:

Se modificó el archivo de la parte A, para reemplazar el generador de aleatorios. El archivo es marsaglia.asm.

```
44        #li  $v0, 42                 # service 42 is to set up the upperbound of the random number
45        #addi $a1, $zero, 1000  # loading upperbound
46        #syscall
47
48  newRand:li $t5, 36969
49        li $t6, 65535
50        li $t7, 18000
51
52        and $t8, $s4, $t6
53        srl $t9, $s4, 16
54        mul $s4, $t5, $t8
55        add $s4, $s4, $t9
56
57        and $t8, $s3, $t6
58        srl $t9, $s3, 16
59        mul $s3, $t7, $t8
60        add $s3, $s3, $t9
61
62        sll $t5, $s4, 16
63        addu $a0, $t5, $s3
64
65
66
67        li $t5, 1023
68        and $a0, $t5, $a0
69
70        li $t5, 1000
71
72        bgt $a0, $t5, newRand
73
74        ############################ FUN ################################
75
76
77        li      $t3,    10
```

Ilustración 3: marsaglia.asm

Con esto, se obtuvieron los resultados, codificando el algoritmo que se dio en el Sidweb. Se usó el código anterior para basarse en este. El resultado se ve en randomMarsaglia.txt.

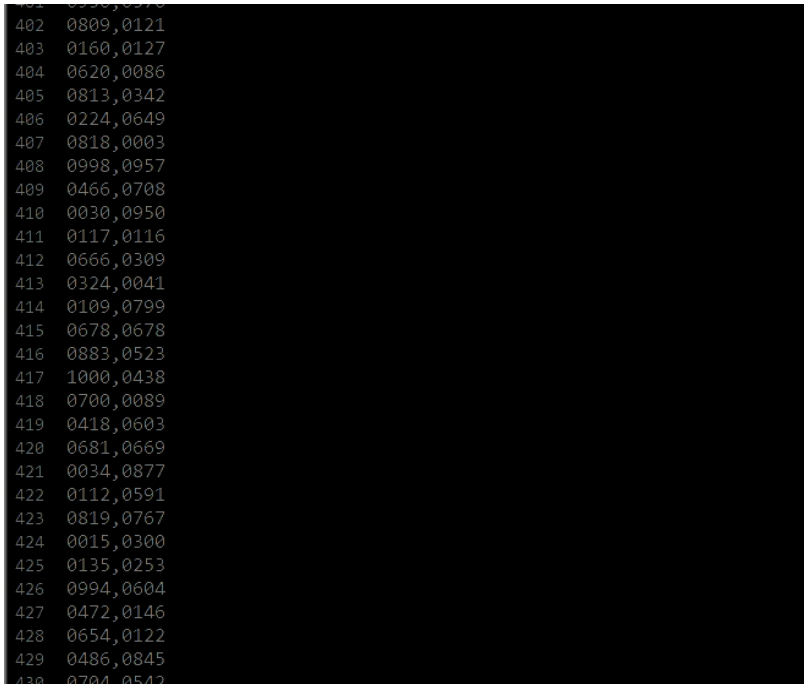

Ilustración 4: randomMarsaglia.txt

PARTE C:

En esta parte se pide graficar los resultados de las 2 partes anteriores, y analizar dichas gráficas comparándolas una con otra.

SOLUCION:

Podemos ver que con ambos generadores se logra una concentración algo uniforme dentro de todo el espacio usado; y es difícil notar alguna diferencia marcada entre los dos métodos.

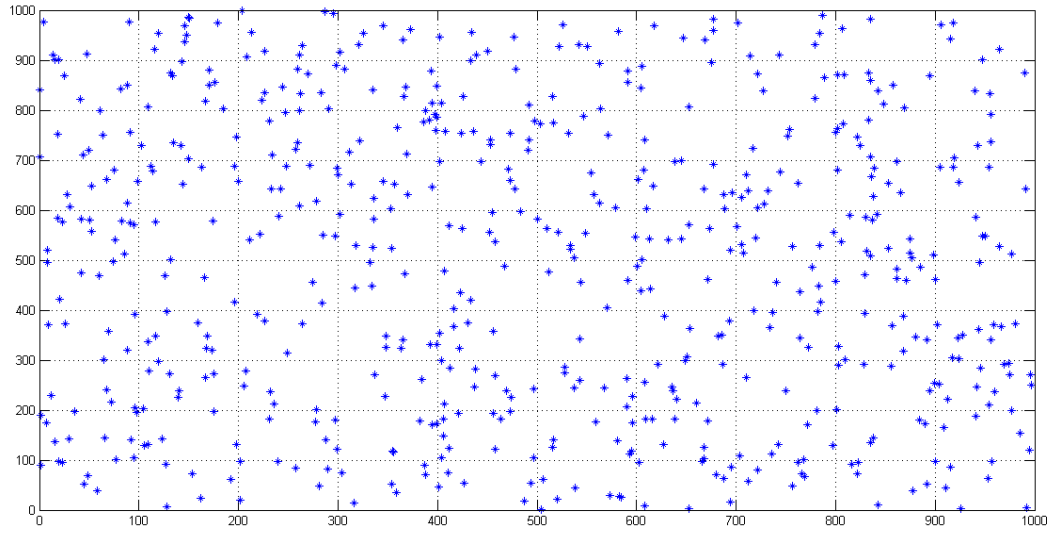

Ilustración 5: Puntos generados por generarAleatorios.asm

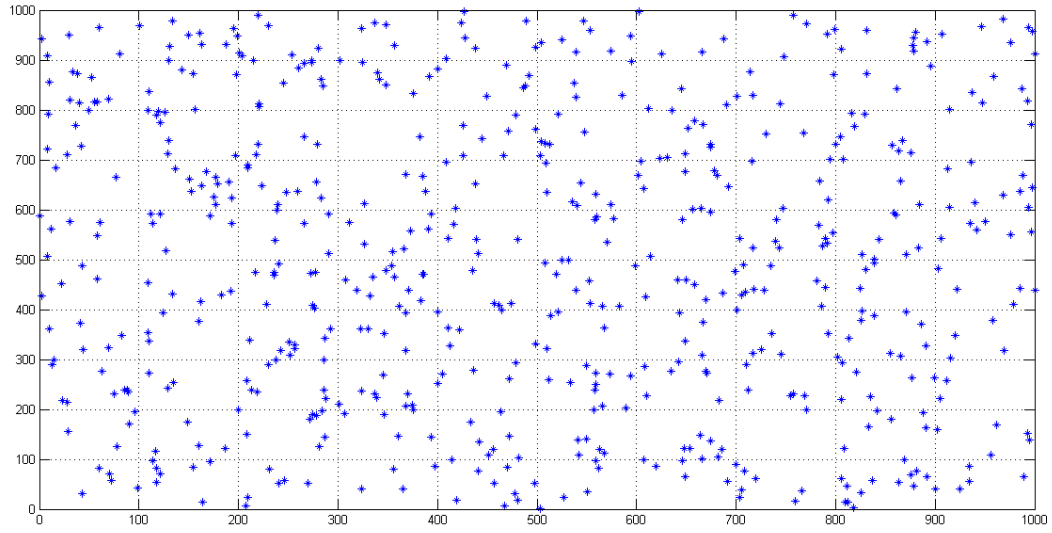

Ilustración 6: Puntos generados por marsaglia.asm

PARTE D:

Para esta sección se pide generar las siguientes configuraciones específicas de clusters sintéticos usando los generadores de aleatorios implementados previamente y, la funcionalidad implementada para imprimir datos en archivos:

1. Dos dimensiones – Dos clusters
2. Tres dimensiones – Dos clusters
3. Cuatro dimensiones – Dos clusters
4. Dos dimensiones – Tres clusters
5. Tres dimensiones – Tres clusters
6. Cuatro dimensiones – Tres clusters
7. Dos dimensiones – Cuatro clusters
8. Tres dimensiones – Cuatro clusters
9. Cuatro dimensiones – Cuatro clusters
10. Dos dimensiones – Cinco clusters
11. Tres dimensiones – Cinco clusters
12. Cuatro dimensiones – Cinco clusters

SOLUCION:

Primero se implementó el código generarAleatoriosClusters.asm, basado en generarAleatorios.asm, que genera clusters de puntos, es decir, se genera un punto aleatorio, y a partir de este se generan puntos cercanos a este con un rango mucho menor de aleatoriedad. Después de esto se implementó funcionalidad para crear tantos clusters con tantas dimensiones sean necesarias. Esto se define al principio del programa. Finalmente, se implementó que el usuario pueda especificar cuántos clusters y dimensiones quiere.

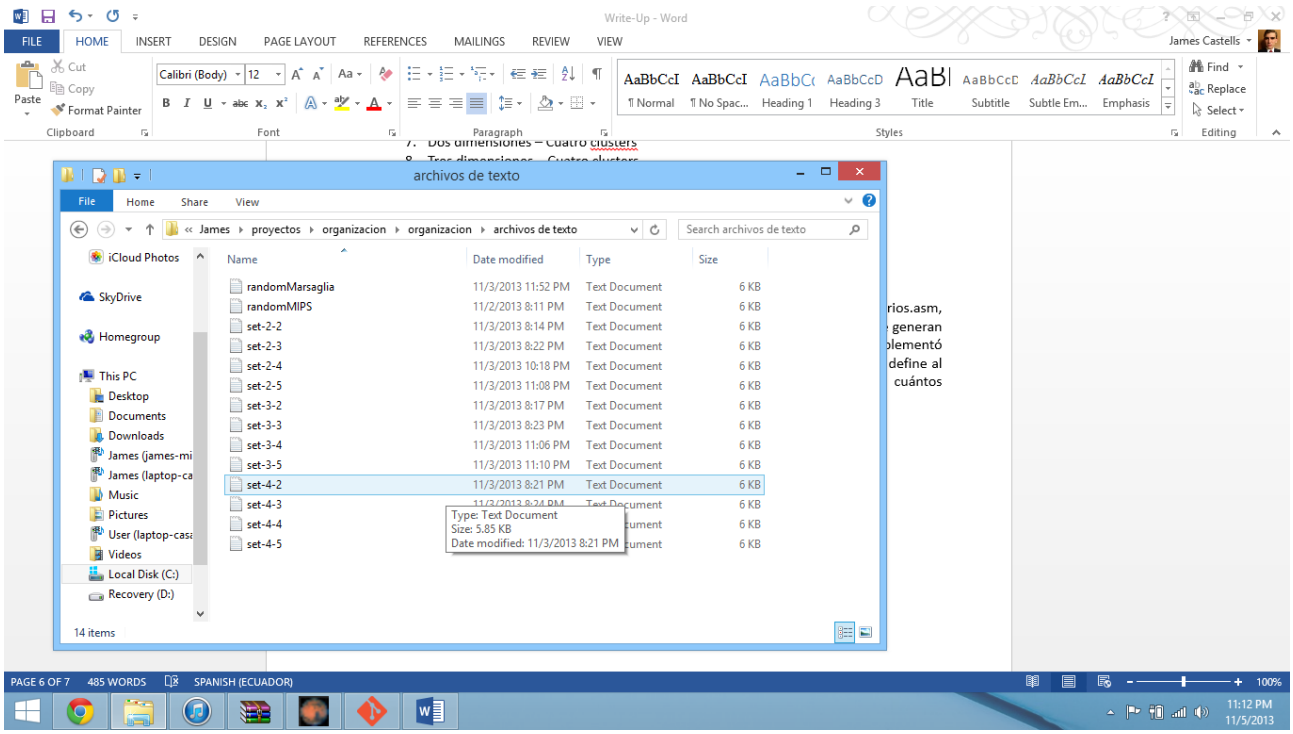

Ilustración 7: Los archivos con clusters generados.

PARTE E:

En esta parte se pide graficar algunos de los datos obtenidos para observar los clusters generados.

SOLUCION:

Se pueden observar claramente que los archivos en 2D y 3D ofrecen clusters definidos:

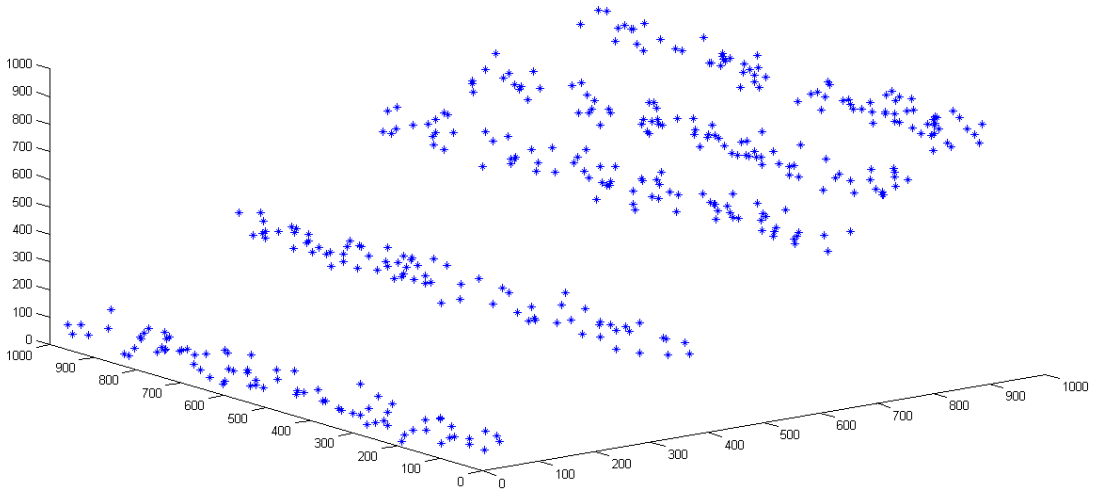

Ilustración 8: 3 dimensiones con 5 clusters.

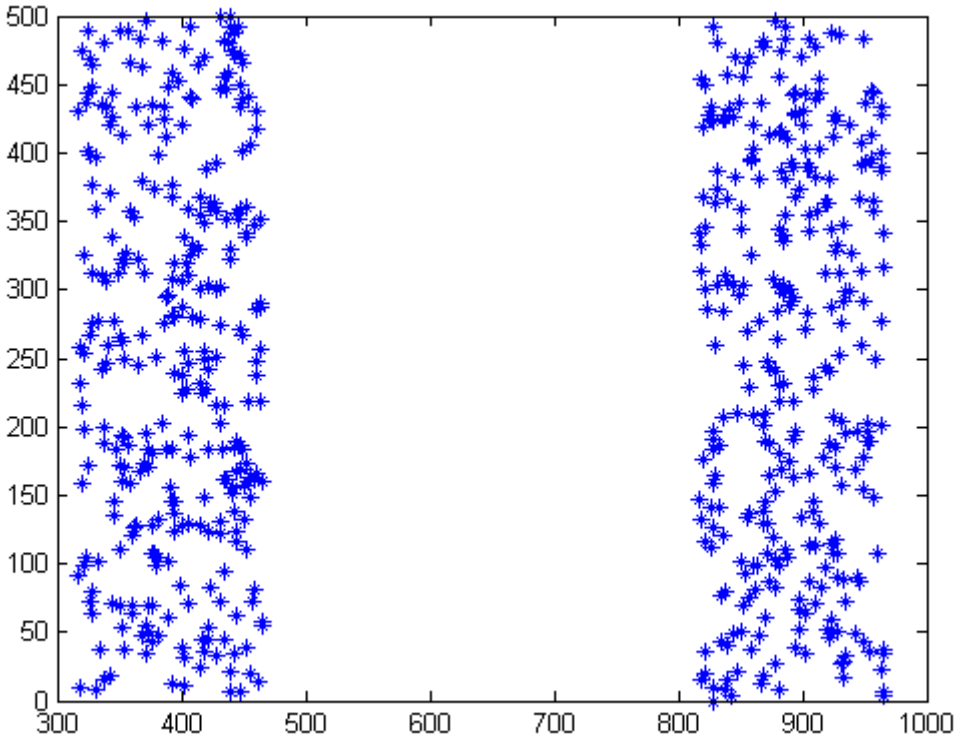

Ilustración 9: 2 dimensiones con 2 clústers.

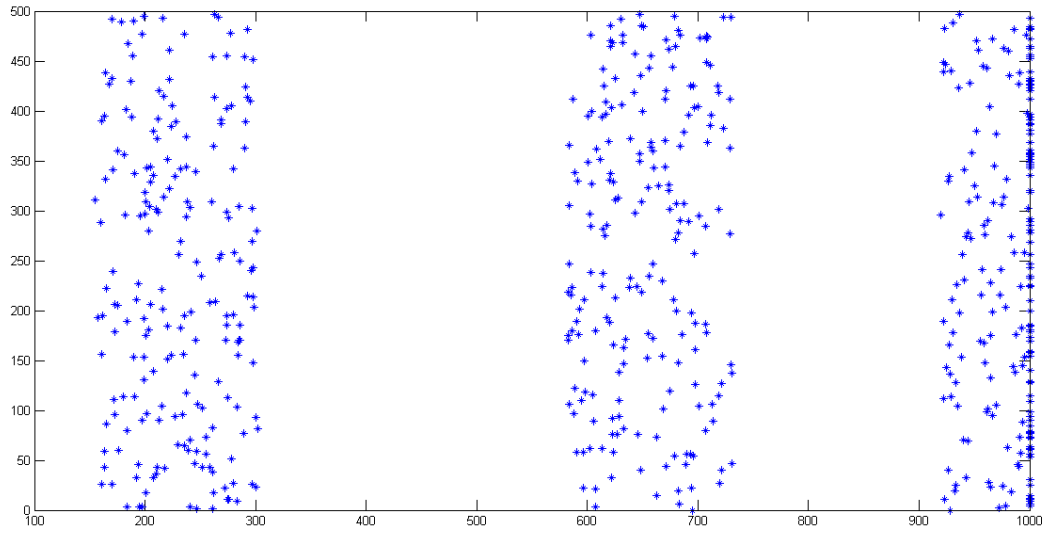

Ilustración 10: 2 dimensiones con 3 clústers.

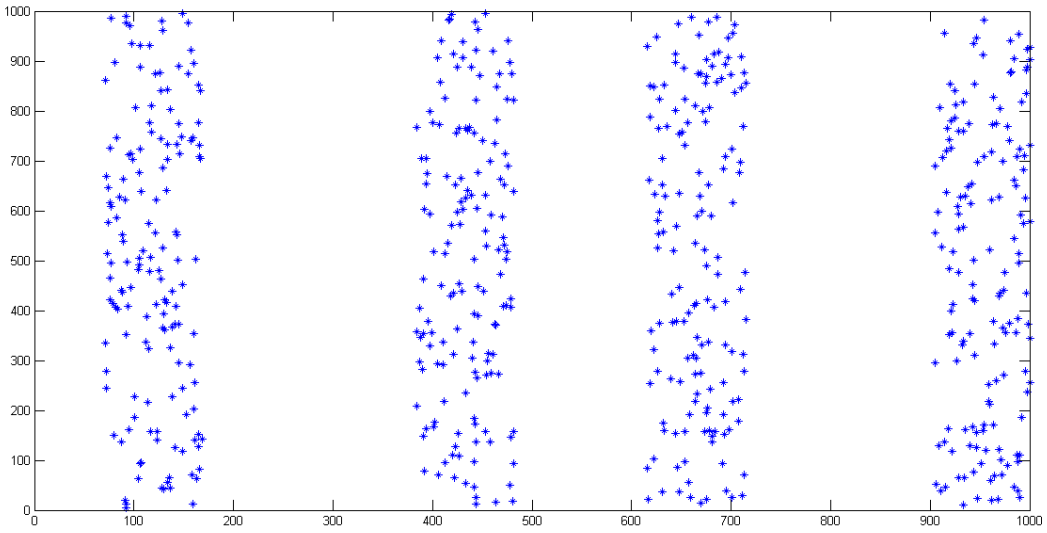

Ilustración 11: 2 dimensiones con 4 clústers.

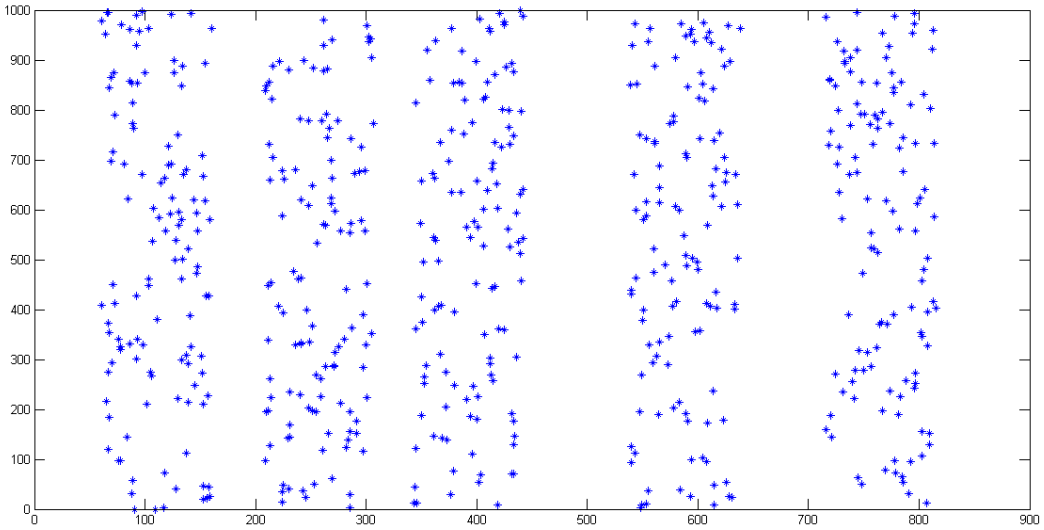

Ilustración 12: 2 dimensiones con 5 clústers.

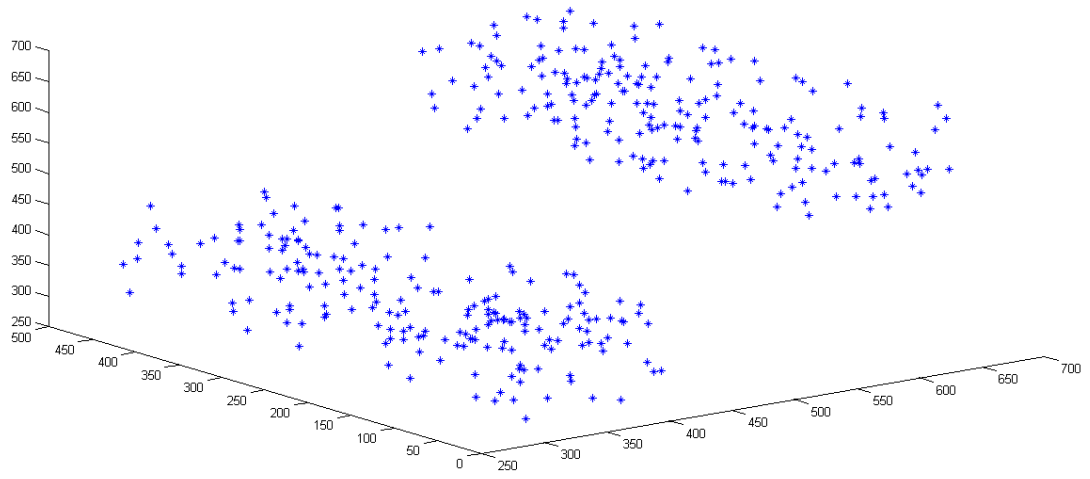

Ilustración 13: 3 dimensiones con 2 clústers.

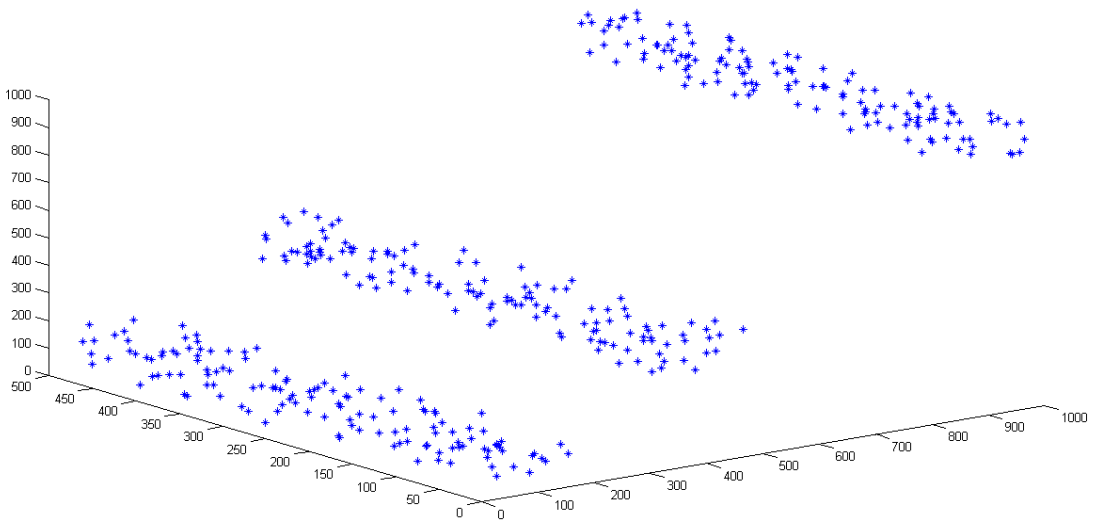

Ilustración 14: 3 dimensiones con 3 clústers.

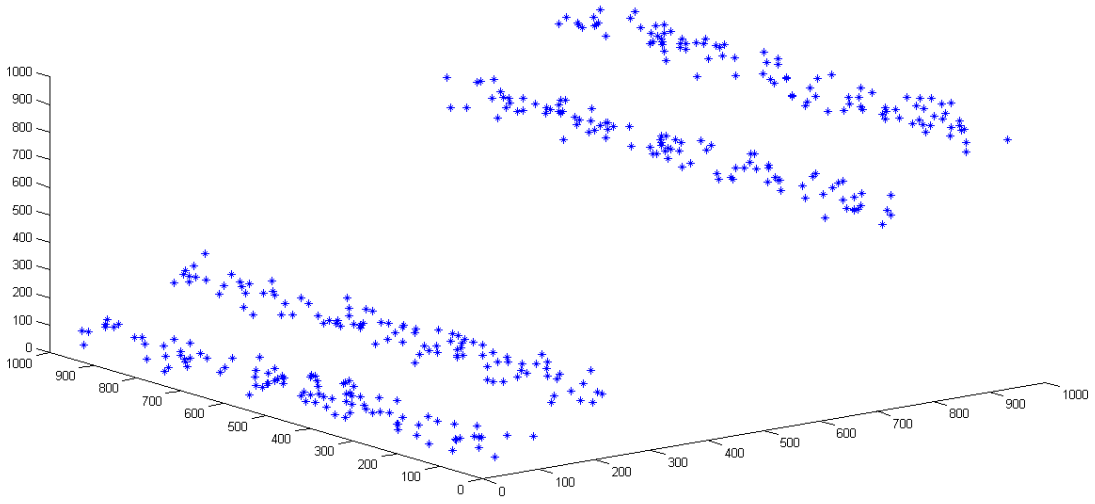

Ilustración 15: 3 dimensiones con 4 clústers.

PARTE F:

El programa debe tener buena interfaz para el usuario.

SOLUCION:

Al pedir números de clústers y dimensiones, se presenta información clara al usuario:

```
==========================

Ingrese el numero de dimensiones (max. 4): 5
Error! Vuelva a ingresar el numero.

==========================

Ingrese el numero de dimensiones (max. 4): 8
Error! Vuelva a ingresar el numero.

==========================

Ingrese el numero de dimensiones (max. 4): 3
==========================

Ingrese el numero de clusters que desea (max. 5): 8
Error! Vuelva a ingresar el numero.

==========================

Ingrese el numero de clusters que desea (max. 5): 2
==========================
El archivo se ha generado. Se llama randomMIPSclusters.txt.
```

Ilustración 16: Pedida de dimensiones y clusters al usuario.