

Centro de Investigación en Cómputo
Instituto Politécnico Nacional
Metaheurísticas
Actividad No. 17

Solución de problemas mediante Algoritmos Genéticos Estacionarios
Curso impartido por: Dra Yenny Villuendas Rey

Adrian González Pardo

14 de diciembre de 2020

1. Ventajas y Desventajas de GA

Ventajas	Desventajas
Permite realizar multiples busqueda de soluciones a los problemas	Puede que el metodo de mutación o cruza seleccionado puede que no ayude a encontrar una buena solución
Esta bioinspirado en la genética y en la selección natural (Darwinismo)	Puede que el que en la aplicación en alguna etapa del GA ya no avance
Es una heurística poblacional	Puede que genere bastante uso de recursos en memoria y procesamiento
Es posible el trabajar soluciones de formas paralelizables o distribuidas	Puede ser difícil de implementar

2. Genotipo vs Fenotipo

Genotipo: es una representación en cadenas de bits en la cual generalmente es trabajada para generar un nuevo individuo en el algoritmo.

Fenotipo: es la representación que tiene la cadena de bits en el ambito del problema, es decir, la cadena de bits puede representar números reales \mathbb{R} , números enteros \mathbb{Z} , valores binarios $\{0, 1\}$, índices de la solución a algún problema.

3. Operadores

3.1. Mutación

3.1.1. Aleatoria para fenotipo real

Para este tipo de mutación es necesario conocer la dimensión del arreglo, de modo en que para la mutación de este tipo podemos hacer dos metodos distintos hacer la modificación mediante una selección aleatoria en cualquier x_i con valores aleatorios que van de $[min, max]$ y la otra que itera sobre cada x_i bajo una función de probabilidad cuyo valor es pequeño y si se genera un valor aleatorio cuyo valor compite en la función de probabilidad y este es menor se generara el valor aleatorio en el intervalo del metodo anterior.

3.1.2. Aleatoria para fenotipo binario

Para esta mutación se hará uso de un valor de probabilidad pequeño de modo en que se debe obtener un valor pequeño de tal modo que este significara que hay una modificación sobre cada bit particular $b_i = 0, 1$ de tal forma en que se puede realizar esto con cada índice b_i o se puede iterar sobre todos y cada uno de ellos

3.1.3. Aleatorio para fenotipo de orden

Para este tipo de mutación al igual que los anteriores podemos apoyarnos del uso del valor de probabilidad para mutar sobre dos valores p_i que intercambia lugar con p_j de tal modo en que mediante esta permutación podremos generar una nueva solución, por otro lado se puede realizar esta permutación una sola ocasión o realizarla como si se iterara sobre cada elemento de la solución es decir N veces

3.2. Cruzamiento

3.2.1. Dos puntos de corte

Es una forma de intercambiar información entre dos padres. Esta forma de cruce se selecciona dos puntos aleatorios de corte de forma que se los cortes van de $[1, a]$, $[a, b]$, $[b, N - 1]$, quedando un ejemplo de la siguiente forma:

0	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0

Pasan a:

0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0

3.2.2. Uniforme

Este tipo de cruzamiento se realiza bajo un vector con valores de probabilidad, destacando que cada elemento del arreglo tiene una probabilidad de cruce del 0,5 por lo que generaran valores aleatorios en el vector de tal manera que si $v_i \geq 0,5$ este índice realiza la cruce de datos de ese índice.

Un ejemplo de ello seria el siguiente: **Se genera un vector con los siguientes datos**

0.55	0.03	0.67	0.77	0.30	0.25	0.99	0.78
------	------	------	------	------	------	------	------

Teniendo los siguientes padres:

0	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0

Entonces:

1	0	1	0	1	0	1	0
0	0	0	1	0	1	1	1

3.2.3. Aritmético

Para este tipo de algoritmo es necesario pensar en que nuestros dos vectores padres son valores reales los cuales van a ser cruzados a través de un valor aleatorio $\alpha \in [0, 1]$ de modo que actuaran como valores escalares para nuestros vectores, de tal forma que existe un segundo valor $\beta = 1 - \alpha$ formando la siguiente ecuación para realizar la cruce:

Sean los vectores

$$\begin{aligned} X^1 &= x_0^1, x_1^1, \dots, x_N^1 \\ X^2 &= x_0^2, x_1^2, \dots, x_N^2 \\ y &= \alpha x^1 + \beta x^2 \end{aligned}$$

3.3. Selección

3.3.1. Torneo

Este modelo se trata de ir realizando un k número de arreglos permutados de forma aleatoria de tamaño N que es el tamaño dimensional de la respuesta de tal modo en que se busca seleccionar el valor índice de aquellos datos que esten mejor evaluados en el torneo, gráficamente el torneo se ve de la siguiente manera:

$k = 2$, entonces $[0, 5, 1, 4, 0, 1]$, la generación aleatoria, generaria lo siguiente por ejemplo:

2	1	4	3
4	2	3	1

Se accedera a los valores de cada subíndice del arreglo en su valor y se evaluara el menor valor vía índices es decir de arriba hacia abajo para seleccionar al mejor.

4	1	4	1
---	---	---	---

3.3.2. Proporcional

Para este tipo de selección se usa la función

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j}$$

Donde:

p_i es la probabilidad de ser seleccionado p_i

f_i es la función de fitness para ese valor de i

Por tanto al obtener esto tendremos un pequeño sesgo probabilístico a la hora de seleccionar un área por tanto definiremos al error selectivo como:

$$n_i = p_i \times N = \left\lfloor N \times \frac{f_i}{\sum_{j=1}^N f_j} \right\rfloor$$

En esta ecuación nos podremos percatar de que redondeamos el error a un valor entero, lo que significa que cada individuo podría perderse cualquiera que sea su valor relativo de aptitud.

3.3.3. Por Emparejamiento Variado Inverso (NAM)

Un padre se escoge aleatoriamente, para el otro selecciona N_{nam} padres y escoge el más lejano al primer ($N_{nam} = 3, 5, \dots$). Está orientado a generar diversidad.

4. Impacto en convergencia a la solución con GA

Como bien sabemos la existencia de un algoritmo bioinspirado en la selección natural y el la biología en si misma nos hace pensar que en cuestion de generación de soluciones ya sean aleatorias en cada nueva iteración o con alguna conservación en sus estados es de suma importancia ya que de este modo podemos explorar ya sea de forma continua o paralela el espacio de soluciones de modo en que se puede converger en la solución al problema de una forma más rápida y sencilla, por lo que a mayor tamaño de población podría significar mayor procesamiento pero si se implementa alguna técnica de paralelización pueda que en tiempo sea casi aproximado a las soluciones de los otros algoritmos heurísticos como lo son Random-Mutation Hill-Climbing (RMHC) o Simulated Annelling (SA).

5. Estructuras de datos necesarias para la implementación en problemas

5.1. Knapsack

Modelación Matemática

Sean dos funciones

$$f(x) = \sum_{i=1}^N x'_i h(x_i)$$

$$g(x) = \sum_{i=1}^N x'_i p(x_i) \leq \text{peso_máximo}$$

Donde:

X es un vector de la forma $X = [x_1, x_2, \dots, x_N]$

X' es un vector el cual es parecido a X pero sus valores son binarios $x'_i \in \{0, 1\}$

$h(x_i)$ es una función la cual devuelve el beneficio total de los objetos x_i en la mochila

$p(x_i)$ es una función la cual devuelve el peso total de los objetos x_i en la mochila

De modo que para los operadores GA en este problema el genotipo puede ser la cadena de bits de modo que el fenotipo que representa de la cadena es la representación de un valor binario de si el objeto es incluido o no es incluido en la solución.

Finalmente la estructura de datos para esta solución es un arreglo binario el cual contendrá un valor 0 o 1 de tal forma en que se pueda evaluar cada arreglo a la selección y torneo de GA, por tanto así podría calcularle los valores de su función de maximización de beneficio y de tratar de minimizar o no sobrepasar el peso máximo de la mochila

5.2. Travel Salesman Problem (TSP)

Modelación Matemática

Sea la función

$$f(x) = \sum_{i=1}^N x_i \cdot g(y_{x_i, i})$$

Donde:

Y es una matriz de $N \times N$ la cual trabajara para obtener el costo con la función $g(x)$

X es un vector de dimensión N el cual contiene el índice del vértice al cual pasa de i a j

$g(y_{x_i, i})$ es una función la cual devuelve el costo de ir de i hacia el contenido de x_i

De modo en que nuestro GA puede ser implementado con un fenotipo entero el cual representa la ciudad n hacia que vértice se conecta posteriormente.

Ejemplo de arreglo X : $x = [2, 3, 1, 0]$ de modo que sabemos que nuestro grafo tiene 4 aristas y pensando en que todos están conectados partiremos de la siguiente manera $(0 \rightarrow 2)$, $(2 \rightarrow 1)$, $(1 \rightarrow 3)$, $(3 \rightarrow 0)$ de tal forma que el formato en como realiza el movimiento de un vértice a otro es $(i \rightarrow j)$ siendo el valor de j el que será evaluado en la matriz Y para obtener su costo.

Finalmente para este es tan sencillo como la realización de un arreglo de orden que contenga los valores de cada nodo o ciudad de la solución para así realizar el cálculo de su costo y así sobre cada solución.

5.3. Función de Minimización en D dimensiones

Modelación Matemática

Sea la función

$$f(x) = \sum_{i=1}^D x_i^2$$

Donde:

$x_i \in \mathbb{R}$

Descrito en los intervalos $x_i \in [-10, 10]$

Donde sabemos que los puntos mínimos de cada x_i los encontramos cuando el valor asignado a el es $x_i = 0$ por lo tanto el ir variando los valores para que el programa se acerque hacia 0

Por lo tanto para este GA podemos implementarlo con un fenotipo de valor real que vaya del intervalo indicado. Para este problema podemos pensar en un arreglo de valores reales los cuales nos permitan realizar o almacenar las soluciones para así pasar por cada etapa del GA.

6. Equipo de ejecución para la solución de los problemas

6.1. Características de Hardware

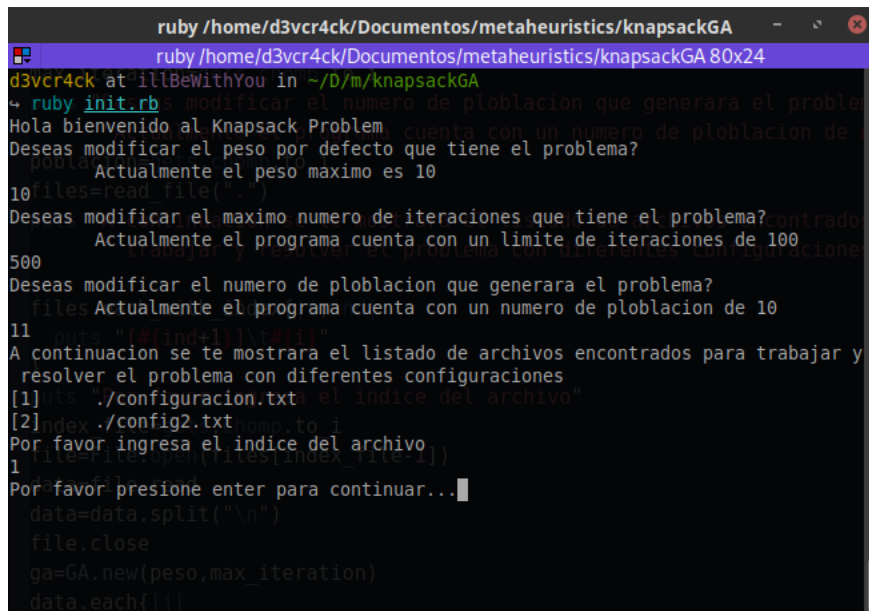
- Procesador Intel Core i5-3210M CPU 2.50GHz

- Memoria RAM DDR3 12 GB
- Sistema Operativo Fedora 32 x86_64
- Ruby 2.7.2p137 para la ejecución

7. Implementación e interfaces

Para el lenguaje seleccionado (Ruby) es difícil la realización de una interfaz gráfica por ello se implementa una interfaz por comandos o ventana la cual muestra el comportamiento de los programas

7.1. Knapsack

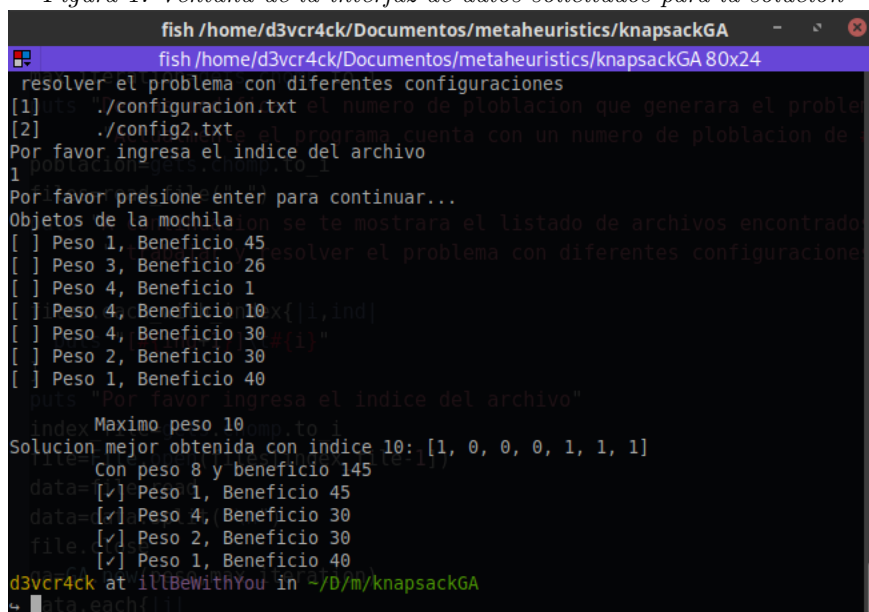


```

ruby /home/d3vcr4ck/Documents/metaheuristics/knapsackGA
d3vcr4ck at illBeWithYou in ~/D/m/knapsackGA
← ruby init.rb modificar el numero de poblacion que generara el problema
Hola bienvenido al Knapsack Problem cuenta con un numero de poblacion de
Deseas modificar el peso por defecto que tiene el problema?
poblacion Actualmente el peso maximo es 10
10 files=read file(" ")
Deseas modificar el maximo numero de iteraciones que tiene el problema?
Actualmente el programa cuenta con un limite de iteraciones de 100
500
Deseas modificar el numero de poblacion que generara el problema?
files Actualmente el programa cuenta con un numero de poblacion de 10
11 puts "[index+1]x{1}"
A continuacion se te mostrara el listado de archivos encontrados para trabajar y
resolver el problema con diferentes configuraciones
[1] puts "./configuracion.txt" el indice del archivo"
[2] index = ./config2.txt.chomp.to_i
Por favor ingresa el indice del archivo
1 file=file.open(files[index], "r")
Por favor presione enter para continuar...
data=data.split("\n")
file.close
ga=GA.new(peso,max_iteration)
data.each{|i|

```

Figura 1: Ventana de la interfaz de datos solicitados para la solución



```

fish /home/d3vcr4ck/Documents/metaheuristics/knapsackGA
fish /home/d3vcr4ck/Documents/metaheuristics/knapsackGA 80x24
resolver el problema con diferentes configuraciones
[1] puts "./configuracion.txt" el numero de poblacion que generara el proble
[2] ./config2.txt el programa cuenta con un numero de poblacion de
Por favor ingresa el indice del archivo
1 poblacion=gets.chomp.to_i
Por favor presione enter para continuar...
Objetos de la mochila on se te mostrara el listado de archivos encontrado
[ ] Peso 1, Beneficio 45 resolver el problema con diferentes configuracione
[ ] Peso 3, Beneficio 26
[ ] Peso 4, Beneficio 1
[ ] Peso 4, Beneficio 10 x{1, ind
[ ] Peso 4, Beneficio 30 #1)"
[ ] Peso 2, Beneficio 30
[ ] Peso 1, Beneficio 40
puts "Por favor ingresa el indice del archivo"
index Maximo peso 10
Solucion mejor obtenida con indice 10: [1, 0, 0, 0, 1, 1, 1]
Con peso 8 y beneficio 145
data=[✓] Peso 1, Beneficio 45
data=[✓] Peso 4, Beneficio 30
file. [✓] Peso 2, Beneficio 30
file. [✓] Peso 1, Beneficio 40
d3vcr4ck at illBeWithYou in ~/D/m/knapsackGA
← data.each{|i|

```

Figura 2: Ventana de la interfaz con la solución dada por el programa

7.2. Travel Salesman Problem (TSP)

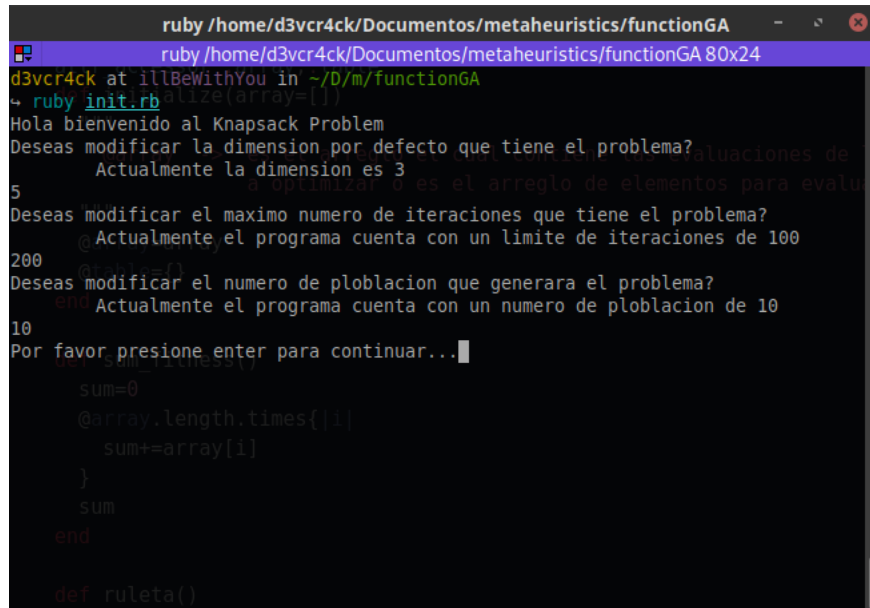
```
ruby /home/d3vcr4ck/Documentos/metaheuristics/tspGA
ruby /home/d3vcr4ck/Documentos/metaheuristics/tspGA 80x24
d3vcr4ck at illBeWithYou in ~/D/m/tspGA
➔ ruby init.rb
poblacion=10
Hola bienvenido al Travel Salesman Problem TSP
Deseas modificar el maximo numero de iteraciones que tiene el problema?
Actualmente el programa cuenta con un limite de iteraciones de 100
200
Deseas modificar el numero de poblacion que generara el problema?
Actualmente el programa cuenta con un numero de poblacion de 10
10
A continuacion se te mostrara el listado de archivos encontrados para trabajar y
resolver el problema con diferentes configuraciones
[1] ./configuracion.txt
Por favor ingresa el indice del archivo
1
Por favor presione enter para continuar...
18
19 files.each_with_index{|i,ind|
20 puts "#{ind+1} \t#{i}"
21 }
22 puts "Por favor ingresa el indice del archi
23 index_file=gets.chomp.to_i
```

Figura 3: Ventana de la interfaz de datos solicitados para la solución

```
fish /home/d3vcr4ck/Documentos/metaheuristics/tspGA
fish /home/d3vcr4ck/Documentos/metaheuristics/tspGA 80x24
Deseas modificar el maximo numero de iteraciones que tiene el problema?
Actualmente el programa cuenta con un limite de iteraciones de 100
200
Deseas modificar el numero de poblacion que generara el problema?
Actualmente el programa cuenta con un numero de poblacion de 10
10
A continuacion se te mostrara el listado de archivos encontrados para trabajar y
resolver el problema con diferentes configuraciones
[1] ./configuracion.txt
Por favor ingresa el indice del archivo
1
Por favor presione enter para continuar...
Solucion mejor obtenida con indice 4:
Camino:
Nodo 0 -> Nodo 1
Nodo 1 -> Nodo 2
Nodo 2 -> Nodo 3
Nodo 3 -> Nodo 4
Nodo 4 -> Nodo 5
Nodo 5 -> Nodo 0
Y costo: 17
d3vcr4ck at illBeWithYou in ~/D/m/tspGA
```

Figura 4: Ventana de la interfaz con la solución dada por el programa

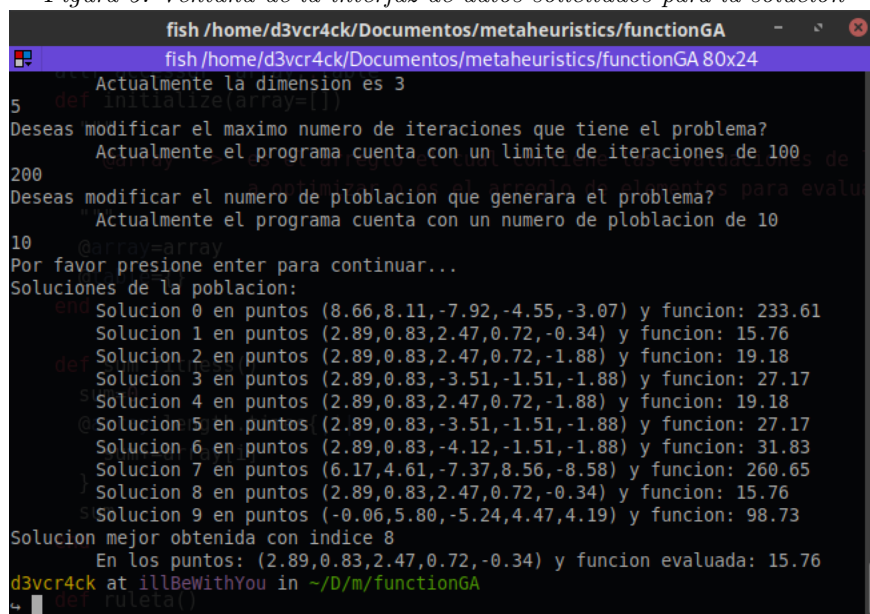
7.3. Función de Minimización en D dimensiones



```
ruby /home/d3vcr4ck/Documents/metaheuristics/functionGA
ruby /home/d3vcr4ck/Documents/metaheuristics/functionGA 80x24
d3vcr4ck at illBeWithYou in ~/D/m/functionGA
← ruby init.rb initialize(array=[])
Hola bienvenido al Knapsack Problem
Deseas modificar la dimension por defecto que tiene el problema?
  Actualmente la dimension es 3
5  a optimizar o es el arreglo de elementos para evaluar
Deseas modificar el maximo numero de iteraciones que tiene el problema?
  Actualmente el programa cuenta con un limite de iteraciones de 100
200 @ Actualmente el programa cuenta con un limite de iteraciones de 100
Deseas modificar el numero de poblacion que generara el problema?
  Actualmente el programa cuenta con un numero de poblacion de 10
10
Por favor presione enter para continuar...
  sum=0
  @array.length.times{|i|
    sum+=array[i]
  }
  sum
end

def ruleta()
```

Figura 5: Ventana de la interfaz de datos solicitados para la solución



```
fish /home/d3vcr4ck/Documents/metaheuristics/functionGA
fish /home/d3vcr4ck/Documents/metaheuristics/functionGA 80x24
  Actualmente la dimension es 3
5 def initialize(array=[])
Deseas modificar el maximo numero de iteraciones que tiene el problema?
  Actualmente el programa cuenta con un limite de iteraciones de 100
200
Deseas modificar el numero de poblacion que generara el problema?
  Actualmente el programa cuenta con un numero de poblacion de 10
10 @array=array
Por favor presione enter para continuar...
Soluciones de la poblacion:
end Solucion 0 en puntos (8.66,8.11,-7.92,-4.55,-3.07) y funcion: 233.61
  Solucion 1 en puntos (2.89,0.83,2.47,0.72,-0.34) y funcion: 15.76
  Solucion 2 en puntos (2.89,0.83,2.47,0.72,-1.88) y funcion: 19.18
  Solucion 3 en puntos (2.89,0.83,-3.51,-1.51,-1.88) y funcion: 27.17
  Solucion 4 en puntos (2.89,0.83,2.47,0.72,-1.88) y funcion: 19.18
  Solucion 5 en puntos (2.89,0.83,-3.51,-1.51,-1.88) y funcion: 27.17
  Solucion 6 en puntos (2.89,0.83,-4.12,-1.51,-1.88) y funcion: 31.83
  Solucion 7 en puntos (6.17,4.61,-7.37,8.56,-8.58) y funcion: 260.65
  Solucion 8 en puntos (2.89,0.83,2.47,0.72,-0.34) y funcion: 15.76
  Solucion 9 en puntos (-0.06,5.80,-5.24,4.47,4.19) y funcion: 98.73
Solucion mejor obtenida con indice 8
  En los puntos: (2.89,0.83,2.47,0.72,-0.34) y funcion evaluada: 15.76
d3vcr4ck at illBeWithYou in ~/D/m/functionGA
← def ruleta()
```

Figura 6: Ventana de la interfaz con la solución dada por el programa