Centro de Investigación en Matemáticas A.C.
Programación y Algoritmos 1 (2019)

Tarea 12 - UMDA

Marco Antonio Esquivel Basaldua

Un algoritmo genético está inspirado en la evolución biológica y en su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

En el algoritmo UMDA (*Uivariate Marginal Distribution Algorithm*), se hace evolucionar una población de individuos bajo la selección de los individuos más aptos. Este es el modelo probabilístico más simple posible. En cada generación la distribución de probabilidad conjunta, $p_l(x)$, que sirve para estimar el comportamiento de los individuos seleccionados, se factoriza como un producto de distribuciones marginales univariantes e independientes. Es decir:

$$p_l(x) = p(x|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_l(x_i)$$

El pseudocódigo de algoritmo UMDA es el siguiente:

UMDA

$D_0 \leftarrow$ Generar M individuos (la poblacion inicial) al azar

Repeat for $l = 1, 2, \ldots$ hasta que se verifique el criterio de parada

$D_{l-1}^{Se} \leftarrow$ Seleccionar $N \leq M$ individuos de D_{l-1} de acorde al metodo de seleccion

$p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_l(x_i) = \prod_{i=1}^{n} \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i|D_{l-1}^{Se})}{N} \leftarrow$ Estimar la distribucion de probabilidad conjunta

$D_l \leftarrow$ Muestrar M individuos (la nueva poblacion) de $p_l(\boldsymbol{x})$

Cada distribucion marginal se estima a partir de las frecuencias marginales :

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta(X_i = x_i|D_{l-1}^{Se})}{N}$$

Donde:

$$\delta(X_i = x_i|D_{l-1}^{Se}) = \begin{cases} 1 & \text{si en el j-ésimo caso de } D_{l-1}^{Se}, X_i = x_i \\ 0 & \text{en otro caso} \end{cases}$$

Resultados

El programa entregado encuentra la localización del mínimo de las funciones $f_1(x) = x^2$ y $f_2(x) = (1-x)^2 + 1$. La selección de cada una de las funciones se hace a través de la variable de tipo entero `func`, si esta variable es igual a 1 se trabaja con la función $f_1(x)$, si es igual a 0 se trabaja con la función $f_2(x)$.

Para evitar trabajar con una matriz de unos y ceros para indicar los genes de cada individuo, se trabaja con la representación tipo entera de esta cadena de bits, `vbits`, que está determinada por la cantidad de bits que conforma a un individuo, `nbits`. Al trabajar con una representación tipo entera se tiene un máximo de 15 bits para representar a un individuo (quitando el bit de signo).

Dado el valor entero de la representación de un individuo y el rango de búsqueda del óptimo, se mapea la representación en la recta de los reales para conocer el valor de x en la función `mapeo` de la siguiente forma:

$$x = limInferior + \frac{limSup - limInf}{2^{nbits} - 1}(vbits)$$

Dentro del código, un `Individuo` está representado por un `pair` cuyo primer elemento es `f(x)`, el segundo elemento es otro `pair` del que el primer elemento es `vbits` y el segundo es `x`. La razón de colocar `f(x)` en la primer posición del individuo es por que se hace uso de un multimapa para representación de la `Poblacion` y se aprovecha para tener a los individuos ordenados dentro de `Poblacion`. Los Individuos al principio del multimapa son los más aptos para formar la siguiente generación. La forma de cada individuo es la siguiente:

```
pair<double f(x),pair<int vbits,double x> > Individuo
```

A continuación se presentan los resultados al trabajar la función $f_1(x) = x^2$, con una cantidad de bits por individuo de `nbits = 6`, un tamaño de población de 10 individuos y seleccionando como aptos a la mitad de la población para una total de 10 generaciones en un rango de búsqueda de $[-2, 8]$.

```
➜ Marco Esquivel Tarea12 make
g++ umda_sinPOO.cpp
./a.out
0.145125 15 0.380952
1.032 19 1.01587
3.27438 24 1.80952
3.87402 25 1.96825
17.5601 39 4.19048
17.5601 39 4.19048
39.1121 52 6.25397
39.1121 52 6.25397
39.1121 52 6.25397
51.9315 58 7.20635

0.0493827 14 0.222222
0.145125 15 0.380952
0.145125 15 0.380952
0.533132 8 -0.730159
1.37969 20 1.1746
4.52406 26 2.12698
5.22449 27 2.28571
6.77652 29 2.60317
6.77652 29 2.60317
8.53011 31 2.92063

0.00403124 13 0.0634921
0.0493827 14 0.222222
0.145125 15 0.380952
0.326531 9 -0.571429
0.533132 8 -0.730159
1.09751 6 -1.04762
2.72512 23 1.65079
2.72512 23 1.65079
2.83094 2 -1.68254
5.97531 28 2.44444
```

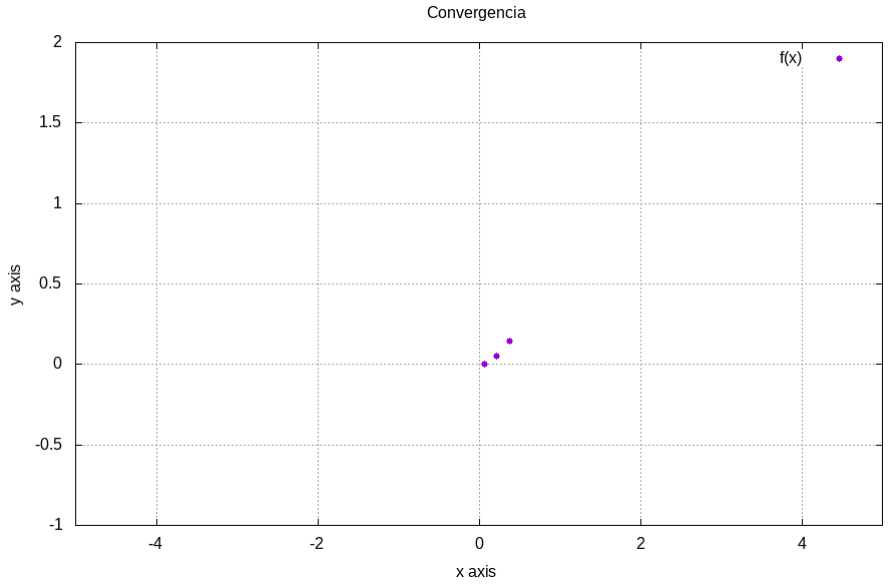
```
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00907029 12 -0.0952381
0.0493827 14 0.222222
0.145125 15 0.380952
0.326531 9 -0.571429
0.326531 9 -0.571429
0.326531 9 -0.571429
0.533132 8 -0.730159

0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.0493827 14 0.222222
0.0493827 14 0.222222
0.145125 15 0.380952
6.77652 29 2.60317

0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.00907029 12 -0.0952381
0.0493827 14 0.222222
```

```
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921

0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921

0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
```

```
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921
0.00403124 13 0.0634921

El minimo se encuentra en la posicion:
x = 0.0634921, y = 0.00403124
El valor entero de su representacion binaria es 13
```

Se puede observar que el algoritmo converge a un valor de $x = 0.0634921$ y $f(x) = 0.00403124$, que son buenos aproximadores para la función $f_1(x) = x^2$ cuyo mínimo se encuentra en $(0, 0)$.

De forma gráfica, los mejores individuos de cada generación se visualizan en la siguiente imagen.



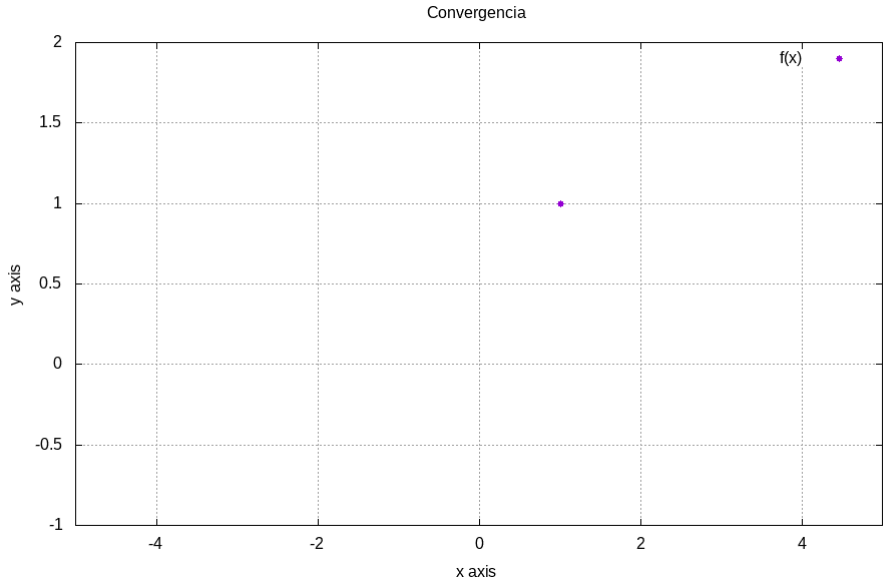
De igual forma de obtienen los resultados para la función $f_2(x) = (1-x)^2 + 1$.

```
➜ Marco Esquivel Tarea12 make
g++ umda_sinPOO.cpp
./a.out
1.00025 19 1.01587
1.87705 13 0.0634921
2.27009 26 2.12698
2.27009 26 2.12698
4.10431 30 2.7619
4.10431 30 2.7619
6.5936 4 -1.36508
15.6337 43 4.8254
22.3356 48 5.61905
39.5188 58 7.20635

1.00025 19 1.01587
1.93752 25 1.96825
2.27009 26 2.12698
2.27009 26 2.12698
2.65306 27 2.28571
3.08642 28 2.44444
3.08642 28 2.44444
4.10431 30 2.7619
4.68884 31 2.92063
4.68884 31 2.92063

1.00025 19 1.01587
1.00025 19 1.01587
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
```

```
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571
2.65306 27 2.28571

1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
26.9615 51 6.09524

1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
```

```
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587

1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.42353 23 1.65079

1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
```

```
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587
1.00025 19 1.01587

El minimo se encuentra en la posicion:
x = 1.01587, y = 1.00025
El valor entero de su representacion binaria es 19
```



En este caso se puede observar una rápida convergencia y estancamiento a los resultados $x = 1.01587$ y $f(x) = 1.00025$. Se sabe que el mínimo de la función $f_2(x) = (1 - x)^2$ es $(1, 1)$.

POO

Tuve problemas para realizar las abstracciones y aplicar el algoritmo usando clases y programación orientada a objetos. El programa descrito es el que lleva por nombre *umda_sinPOO.cpp*. Queda de mi aprender y aplicar la programación orientada a objetos para las tareas que vienen y el proyecto final.