

Seminario de solución de problemas de inteligencia artificial

Practica 01

Javier Emmanuel Astudillo Salamanca

213496188

[Seminario de solucion de problemas de inteligencia artificial](#)

[Practica 01](#)

[Javier Emmanuel Astudillo Salamanca](#)

[213496188](#)

[Descripcion](#)

[Marco teorico](#)

[Heuristica](#)

[Metaheuristica](#)

[Algoritmos genéticos evolutivos](#)

[Descripcion](#)

[Genotipo](#)

[Fenotipo](#)

[Fitness](#)

[Metodo de seleccion](#)

[Apareamiento](#)

[Condicion de parada](#)

[Desarrollo](#)

[Individuo](#)

[Poblacion](#)

[Metodo de eleccion](#)

[Apareamiento](#)

[Selección de los mejores](#)

[Ciclo principal](#)

[Resultados](#)

[Generacion 0](#)

[Generacion 6](#)

[Generacion 12](#)

[Generacion 18](#)

[Generacion 24](#)

[Generacion 30](#)

[Conclusiones](#)

Descripcion

Considere el problema de

$$\min f(x) = x^4 + 5x^3 + 4x^2 + 4x + 1 : \min f(x) \varepsilon - 4 \leq \min f(x) \leq -1$$

Marco teorico

Heurística

Algunos problemas de optimización son difíciles de solucionar por medio de métodos tradicionales debido a la complejidad, cantidad de respuestas o simplemente por que con el poder de cómputo actual tardaríamos una eternidad en resolverlos.¹

Es ahí cuando entran en acción las metaheurísticas que nos define heurística de la siguiente manera:

“Procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido”. (Zanakis y Evans, 1981) ”²

Así pues se utilizan cuando no existe un método exacto de resolución, cuando el que existe demora bastante tiempo para encontrar una respuesta óptima o cuando se tienen limitaciones de tiempo.

Metaheurística

Las metaheurísticas como las heurísticas, tampoco garantizan la obtención de un óptimo al problema, a diferencia de las heurísticas las metaheurísticas tratan de evitar los óptimos locales, enfocados en el óptimo global.

La lógica de las técnicas metaheurísticas es similar: el punto de partida es una solución (o conjunto de soluciones) que típicamente no es óptima. A partir de ella se obtienen otras parecidas, de entre las cuales se elige una que satisface algún criterio, a partir de la cual comienza de nuevo el proceso. Este proceso se detiene cuando se cumple alguna condición establecida previamente.

¹ (2014). Algoritmos Evolutivos y Algoritmos Genéticos. Retrieved September 6, 2016, from <http://www.it.uc3m.es/~jvillena/irc/practicas/estudios/aeag>.

² (2013). Técnicas metaheurísticas - Ingeniería de Organización y Logística. Retrieved September 6, 2016, from <http://www.iol.etsii.upm.es/arch/metaheuristicas.pdf>.

Algoritmos genéticos evolutivos

Su raíz procede de los algoritmos bioinspirados, basados en imitar el comportamiento o algún fenómeno existente en la naturaleza para resolver el problema, emplean métodos heurísticos no deterministas de “búsqueda”, “aprendizaje”, evolución.

Descripcion

En el caso del algoritmo en cuestión, se basa en la máxima evolutiva de “Los más aptos sobreviven y dejan descendencia”, por lo tanto este algoritmo utiliza una función para evaluar a los más aptos, lo que se llama “fitness” esta función sirve como un indicador de cuáles son los individuos más aptos y cuales son los que nos interesan para “dejar descendencia”, la idea de que se reproduzcan los más aptos es que con cada generación se tenga una solución más cercana al óptimo global y que cuando se alcance la condición de parada se tenga un resultado lo más cercano al óptimo global posible.

Genotipo

Cada individuo cuenta con un genotipo que consta de valores binarios “0s” y “1s” agrupándolos y representando estos grupos características o el “fenotipo” del individuo.

Fenotipo

Dado el genotipo del individuo se puede descifrar cuales son sus características.

Fitness

Es una magnitud que mide qué tan apto es el individuo (solución) con respecto a sus pares, indicándonos un mayor valor que es más cercano al óptimo.

Metodo de seleccion

Existen varios métodos de selección siendo los más populares el de “ruleta” o “torneo”, en el presente trabajo se utilizara el método de ruleta. Que consiste en evaluar a cada individuo para obtener su “fitness” y en base a ese valor otorgarle una probabilidad, a mayor “fitness” mayor probabilidad de ser elegido.

Una vez calculada esa probabilidad mediante una función “random” que elija dos individuos diferentes para aparearse.

Apareamiento

Una vez que se han elegido los dos individuos para aparearse el método que usaremos para hacer esto es el de “swap” que consiste en tomar el genotipo de cada individuo y mediante una función random elegir un número entre (1 y n -1), n siendo el número de genes que tiene la población.

Una vez teniendo este número, se procede a hacer un intercambio de los genes a partir de esa posición por ejemplo:

X = 2

Individuo 1 : 10 01001

Individuo 2 : 11 10110

Hijo 1 : 10 01001

Hijo 2 : 11 10110

Condicion de parada

Esta condición de parada es arbitraria y puede ser un número determinado de generaciones, determinado tiempo, cuando las generaciones no cambien mas, etc.

Para este trabajo se decidió establecer la condición de parada en 30 generaciones.

Desarrollo

Primero declaramos las clases y atributos que nos ayudaran a resolver el problema:

Individuo

```
class Individuo(object):  
    """docstring for Individuo"""  
    def __init__(self, genoma, fenotipo):  
        super(Individuo, self).__init__()  
  
        self.genoma = genoma  
        self.fenotipo = fenotipo  
        self.fitness = -1
```

```
def setFitness(self, fitness): self.fitness = fitness
```

Que cuenta con 3 atributos: genoma, fenotipo, fitness.

Un constructor que toma por parametros “genoma” y “fenotipo”.

Poblacion

```
class Poblacion(object):
```

```
    """docstring for Poblacion"""
```

```
    diccionarioGenomico = \
```

```
    {
```

```
        "0000" : -4.0, "0001" : -3.8, "0010" : -3.6, "0011" : -3.4,
```

```
        "0100" : -3.2, "0101" : -3.0, "0110" : -2.8, "0111" : -2.6,
```

```
        "1000" : -2.4, "1001" : -2.2, "1010" : -2.0, "1011" : -1.8,
```

```
        "1100" : -1.6, "1101" : -1.4, "1110" : -1.2, "1111" : -1.0
```

```
    }
```

```
    def __init__(self, funcionFitness):
```

```
        super(Poblacion, self).__init__()
```

```
        self.poblacion = []
```

```
        self.poblacionNueva = []
```

```
        self.funcionFitness = funcionFitness
```

```
        self.totalFitness = 0
```

Cuenta con 2 atributos “población” y “poblacionNueva” que son listas que almacenan la población actual de la que se eligen los individuos más aptos y “poblacionNueva” que almacena los hijos de los individuos que se aparean.

Asi mismo cuenta con una variable estática “diccionarioGenomico” que es un diccionario al que se le hacen consultas, se le pregunta por el genoma y devuelve el fenotipo.

Ej: diccionarioGenomico[“0000”] → -4.0

Metodo de eleccion

Para este trabajo se eligió el método de ruleta que se implementa de la siguiente manera:

```
def ruleta( self ):
    totalProbabilidad = 0
    elegido = round( random.random() , NUM_DECIMALES )

    for individuo in self.poblacion:
        aux = (individuo.fitness / self.totalFitness)
        totalProbabilidad += aux

        if elegido <= totalProbabilidad:
            return individuo
```

Se inicializa la variable auxiliar “totalProbabilidad” a 0 que sirve como acumulador del rango de probabilidad que tiene cada individuo, la variable “elegido” guarda un valor aleatorio entre 0 y 1 con 4 cifras decimales y por cada individuo se asegura que el valor elegido no haya caído en el rango asignado a ese individuo, si es así regresa al individuo.

Apareamiento

Para la función aparear se eligió el método “swap”.

Primero se elige el número de alelos que se van a conservar por cada individuo, con una función random que elige un número entre (1 y n - 1), siendo n el número total de alelos.

Enseguida se emplean 2 variables auxiliares que guardaran los genomas resultantes del swap e inicializará 2 objetos individuo inicializarlos con el genotipo resultante y el fenotipo que se obtiene haciendo una llamada a “diccionario Genómico” y se regresan los dos individuos resultantes.

```
def aparear(self, padre, madre):
    numAlelos = random.randint( 1, len( padre.genoma ) - 1 )

    genoma1 = padre.genoma[ : numAlelos ] + madre.genoma[ numAlelos : ]
    genoma2 = madre.genoma[ : numAlelos ] + padre.genoma[ numAlelos : ]

    return Individuo( genoma1, self.diccionarioGenomico[genoma1] ) , \
        Individuo( genoma2, self.diccionarioGenomico[genoma2] )
```

Selección de los mejores

Para la elección de los mejores para aparearse se utiliza la ruleta que regresa a los individuos más aptos es necesario que ambos individuos sean diferentes, por lo que es necesario asegurarse que esto no ocurra, una vez que tenemos a dos individuos diferentes, se aparean y se integran a la lista “poblacionNueva”.

```
def seleccionMejores( self ):
    aux1 = self.ruleta()

    flag = True

    while( flag ):
        aux2 = self.ruleta()

        if aux1 != aux2:
            flag = False

    hijos = self.aparear( aux1, aux2 )

    self.poblacionNueva.append( hijos[0] )
    self.poblacionNueva.append( hijos[1] )
```

Ciclo principal

```
def cicloPrincipal( maxGeneraciones ):
    numGeneraciones = 0

    p = Poblacion( lambda x: -( x**4 + 5*x**3 + 4*x**2 - 4*x + 1 ) + OFFSET )
    p.inicializaPoblacion()

    while( numGeneraciones != maxGeneraciones ):
        del( p.poblacionNueva[:] )
        p.resetTotalFitness()
        p.calcularFitness()

        while( len( p.poblacionNueva ) != NUM_POBLACION ):
            p.seleccionMejores()

        p.setPoblacion( sorted( p.poblacionNueva , key= lambda ind: ind.fenotipo ) )
        numGeneraciones += 1
```


Inicializamos el “numGeneraciones” a 0, así como la variable “poblacion” pasándole como parametro la función con la que evaluara el “fitness”, inmediatamente después se inicializa la población.

Se inicia un ciclo que tiene como condición de paro que el número de generaciones sea igual al máximo.

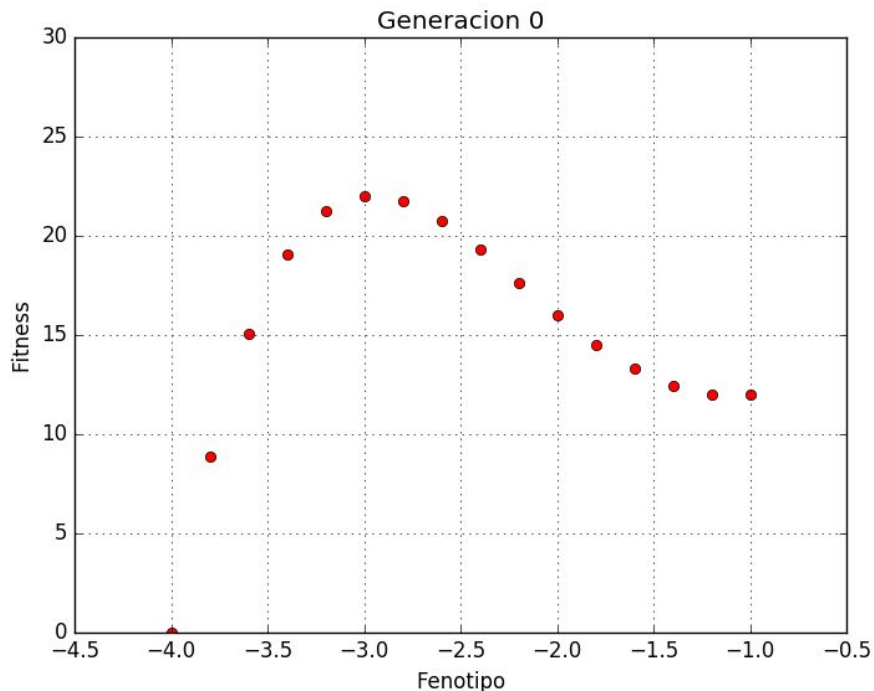
Al inicio de cada ciclo se eliminan todos los elementos en “poblacionNueva”, poner a 0 el “totalFitness” y asignar a cada individuo el fitness actual.

Así mismo se entra a un segundo ciclo que ejecuta la selección de los mejores individuos, los aparea e inserta los hijos a la nueva población, repitiendo esto hasta que la “nuevaPoblacion” tenga el tamaño de requerido.

Por último se asigna la “nuevaPoblacion” a la “poblacion” y se incrementa el contador de generaciones, se repite el ciclo hasta que se cumple con la condición de paro.

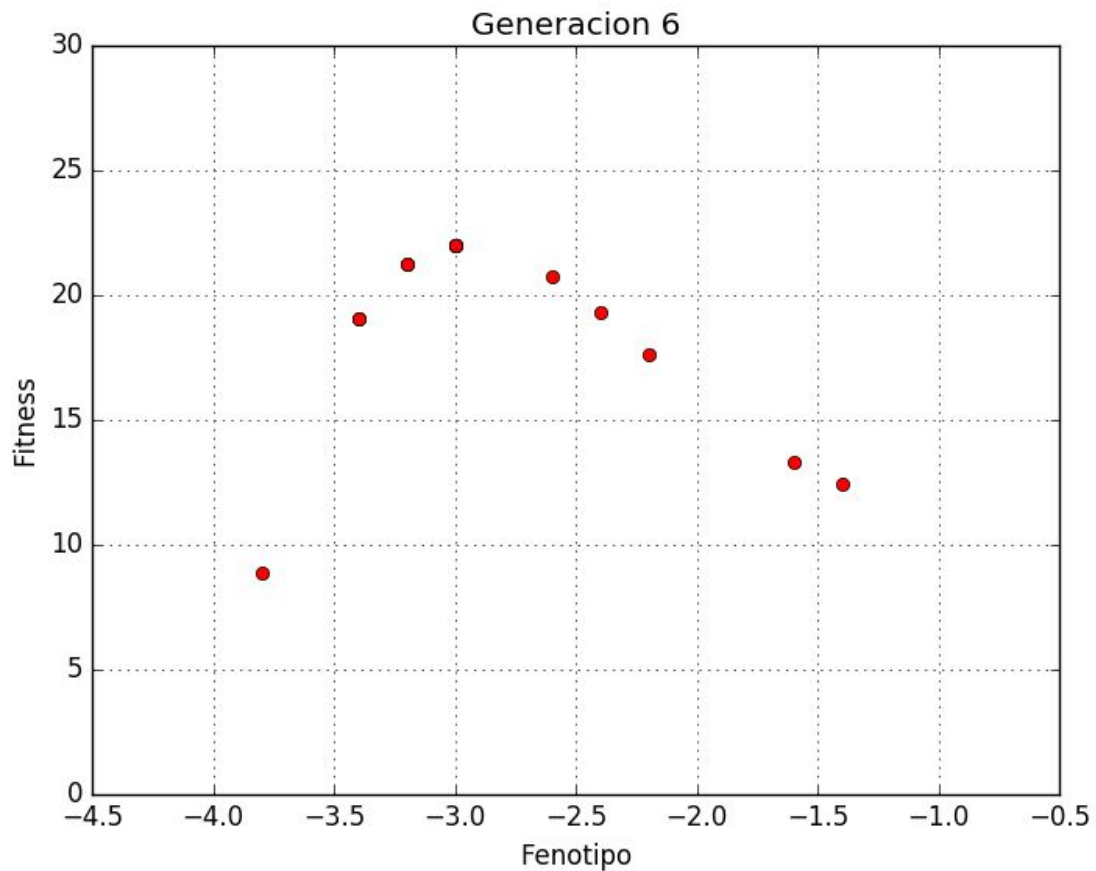
Resultados

Generacion 0



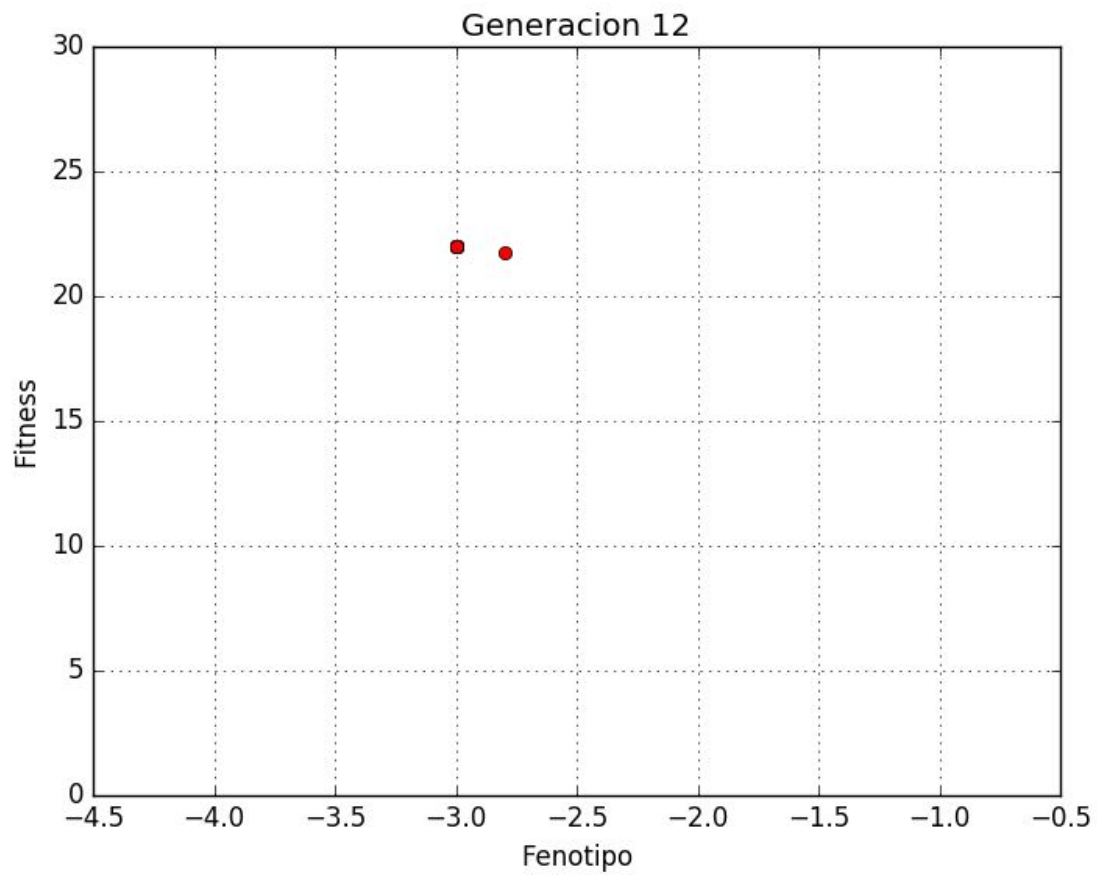
En la generación 0 se muestra a los 16 individuos iniciales distribuidos en el rango de -4 a -1 separados a intervalos de 0.2

Generacion 6



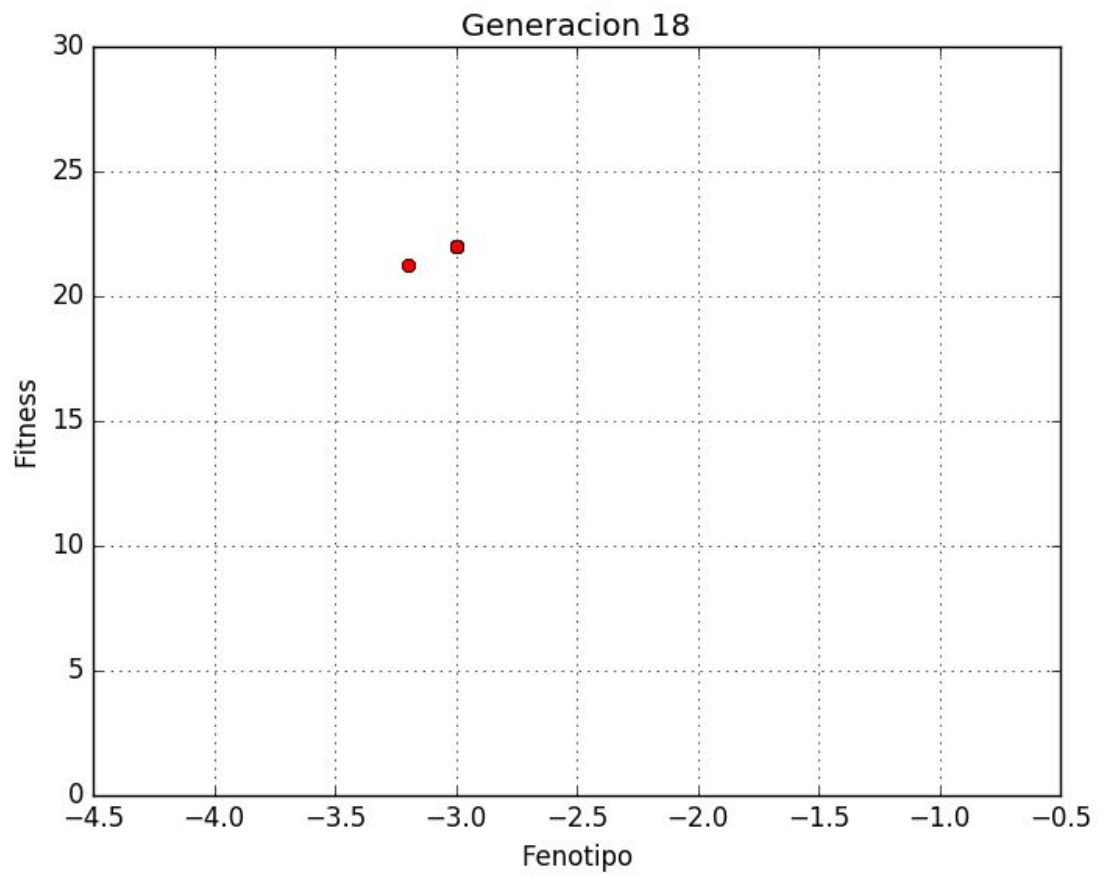
En la generación 6 vemos como la población va evolucionando y se agrupan en el rango (-3.4 - -2.4) con algunos individuos en otros rangos.

Generacion 12



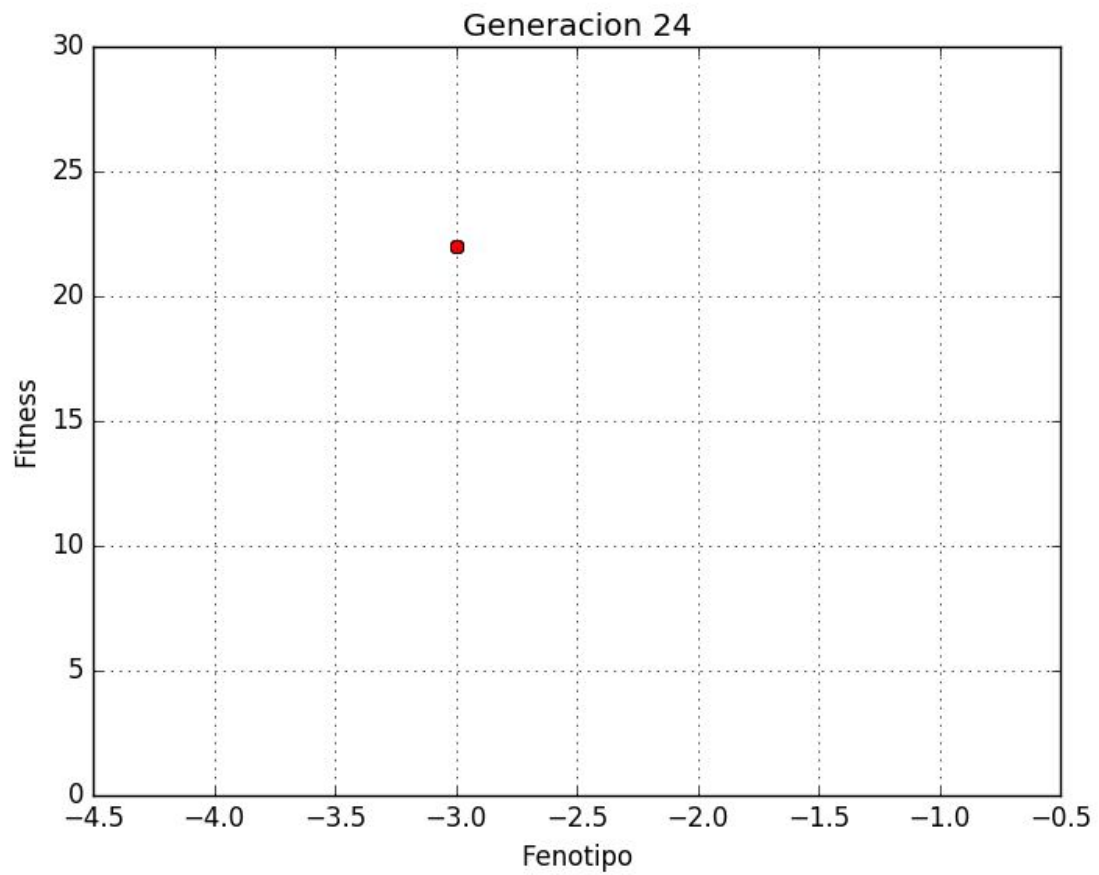
En la generacion 12, vemos como solo quedan 2 grupos de individuos en torno a (-3.0 y -2.8) que se acerca al máximo global de la función que evaluamos.

Generacion 18



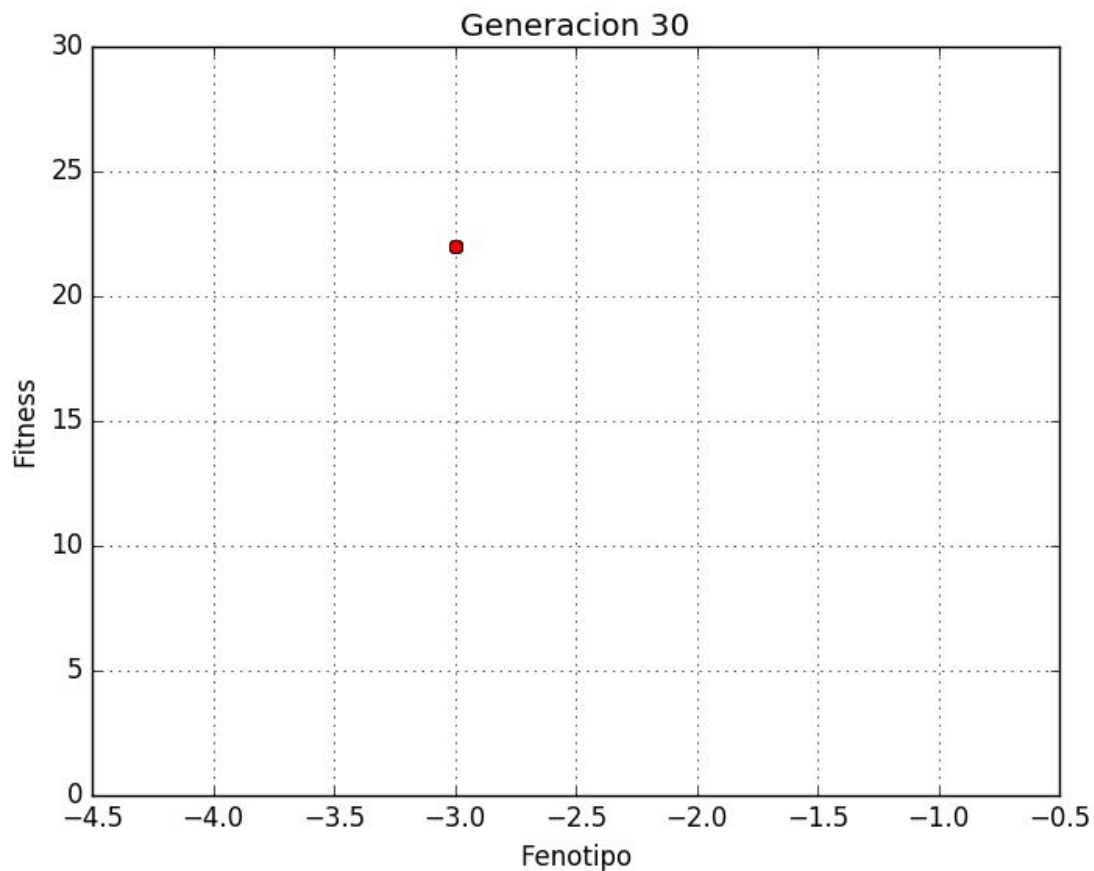
En la generación 18 vemos como hubo un cambio en la población que se sigue agrupando en torno al -3.0

Generacion 24



En la generación 24 todos los individuos tienen el “fenotipo” -3.0.

Generacion 30



En la generación 30 queda exactamente igual a la 24.

Conclusiones

Las metaheurísticas como los algoritmos genéticos evolutivos” generan buenas soluciones cuando no se dispone de tiempo o poder de cómputo para buscar exhaustivamente la solución con algún algoritmo voraz.

Acerca de las diferentes variantes que se pueden probar con este algoritmo, me parece importante señalar que en este caso usamos un offset de 20, entre mayor sea el offset menor serán las diferencias entre los individuos más aptos y los menos, siendo más probable que un individuo menos apto pueda aparearse y dejar descendencia.

En el método de elección, me parece que es mejor el método de torneo por lo que lo intentaré en próximas prácticas.