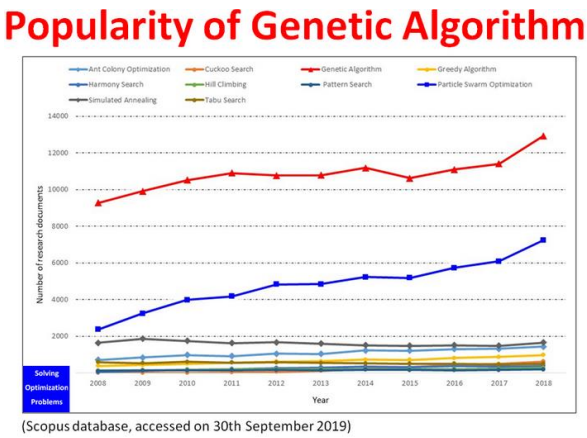
3. Algoritmos Genéticos

José J. Martínez P.
Jmcursosingenieria111@gmail.com
Febrero 2024

Entre los diferentes enfoques para atacar el problema de la IA, se presentó el enfoque de vida, los sistemas bioinspirados, en este caso del proceso evolutivo. El Algoritmo Genético es una de una serie de herramientas bioinspiradas entre las que se encuentran: Optimización por Colonia de Hormigas, Recocido Simulado, Inteligencia de Enjambre, Búsqueda Tabú y otras, que se han desarrollado para la solución de *problemas de optimización y* de *complejidad computacional*. En la figura 3.1 se presenta una comparación de popularidad de estos algoritmos.



(Scopus database, accessed on 30th September 2019)

Figura 3.1. Popularidad de los Algoritmos Genéticos frente a otros algoritmos de optimización

De la figura 3.1, se puede observar que el algoritmo genético es la herramienta más trabajada para la solución de problemas especialmente con alta complejidad computacional y de optimización. Esta es una de las razones para trabajar esta herramienta en el curso.

3.1 **Introducción a los Algoritmos Genéticos**

En los 1950s y 1960s varios investigadores estudiaron independientemente la evolución natural con la idea de que sus mecanismos se pudieran usar como una herramienta de optimización para solucionar problemas de ingeniería. La idea en todos estos sistemas era evolucionar una población de soluciones candidatas usando operadores inspirados en la variación genética y la selección

natural, para solucionar un determinado problema. En los 1960s y 1970s, John Holland con sus estudiantes y algunos colegas de la Universidad de Michigan, inventaron el Algoritmo Genético.

Holland estudio la adaptación tal como se presenta en la naturaleza y desarrolló la forma para llevar esos mecanismos al computador. En su libro "Natural and Artificial Systems, an Introductory Analysis with Applications to Biology, Control and Artificial Intelligence", publicado en 1975, presenta el Algoritmo Genético como una abstracción de la evolución natural y presenta una teoría sobre la adaptación usando Algoritmos Genéticos.

El Algoritmo Genético, AG, es un algoritmo de búsqueda estocástica, que se basa en los mecanismos de la selección natural. Dentro de un conjunto de individuos siempre va a encontrarse algunos mejores que otros en la solución de un problema específico. Se busca que estos mejores individuos, los más aptos, se seleccionen como padres para la siguiente generación y que a través de un intercambio de información estructurado y aleatorio, que imita los procesos de la evolución biológica, generen la siguiente generación que en principio va a solucionar mejor el problema. El problema que se pretende solucionar con el AG se presenta como una función de aptitud que debe satisfacer una o varias condiciones.

El AG trabaja muchas posibles soluciones en forma *paralela*. Estas se llaman individuos o cromosomas. La estructura de datos del cromosoma contempla todas las posibles soluciones del problema y se compone de genes. El número de cromosomas o individuos se conoce como el tamaño de la población. Inicialmente estos cromosomas se generan aleatoriamente formando una población, entre los cuales necesariamente algunos cromosomas serán mejores que otros al enfrentar el problema planteado.

En cada generación se crea un nuevo conjunto de individuos, estructuras de datos, que usan partes de las estructuras de individuos más aptos de las generaciones anteriores. La estructura de datos del cromosoma es una cadena finita de símbolos, una lista.

A través de un proceso de selección, se busca llevar los mejores individuos a la siguiente generación. Una vez conformada la nueva generación se seleccionan de a dos estructuras, para su apareamiento, con el fin de compartir información para la producción de dos descendientes. A estos descendientes, con base en una probabilidad generalmente baja, se les hace una mutación. Este proceso, evaluación, selección, cruce y mutación, se repite hasta que se satisface un criterio de evaluación o se llega a un número determinado de iteraciones que se conoce como el número de generaciones del algoritmo. En este momento se tiene una solución buena o muy buena que satisface las condiciones del problema. Como en todo Sistema Complejo Adaptativo, no se conoce la solución óptima.

En el caso del AGS, Algoritmo Genético Simple, que se estudia a continuación, los símbolos que almacena la estructura de datos son 0 y 1. La estructura del cromosoma debe representar todas las posibles soluciones del problema, es decir, define el espacio de búsqueda del AG. Por lo tanto, un cromosoma es un punto en el espacio de búsqueda de soluciones, es una solución candidata a ser tomada como la solución del problema. Cada posición de la estructura de datos que conforma el cromosoma se conoce como gen. El problema que se pretende solucionar, con todas sus posibles restricciones, se define como la función de aptitud.

El AG procesa la población actual de cromosomas, luego reemplaza esa población creando una nueva generación con base en el comportamiento de los individuos de la generación anterior, con respecto a la función de aptitud. Así avanzan las generaciones. La aptitud de cada cromosoma depende de qué tan bien la solución que representa resuelve el problema.

3.2 El Algoritmo Genético Simple, AGS

Los mecanismos de un AG son sorprendentemente sencillos, dado que su estructura de datos es una lista, las operaciones más complejas incluyen la copia de listas y el intercambio parcial del contenido de las listas. La simplicidad de operación y el poder de efectividad son dos de las principales atracciones de las soluciones por Algoritmos Genéticos.

Con el fin de tener mayor claridad sobre los términos empleados, veamos algunos nombres tomados de la biología, que a veces se utilizan en el contexto de los algoritmos genéticos:

Biología	**Alg. Genéticos**
Cromosoma	Lista
Gen	Elemento de la lista
Genotipo	Solución codificada, cromosoma
Fenotipo	Solución decodificada

Definamos algunos parámetros de los AG.
$\boldsymbol{l}$: número de genes del cromosoma; en AGS, número de bits o longitud del cromosoma
$\boldsymbol{k}$: un gen cualquiera del cromosoma
$\boldsymbol{K}$: número de cromosomas en la población
$\boldsymbol{M}$: número de generaciones del algoritmo
$\boldsymbol{x}$: el valor que representa un cromosoma; el cromosoma decodificado
$\boldsymbol{f(x)}$: función de aptitud aplicada en el punto $\boldsymbol{x}$.

Para trabajar con el Algoritmo Genético se han desarrollado varios operadores, sin embargo, los más importantes y comunes, que producen buenos resultados son los siguientes tres operadores:

1. Selección.
2. Cruce.
3. Mutación.

Selección. O reproducción, es un proceso en que ciertas listas individuales se llevan a la siguiente generación, teniendo en cuenta los valores de su función objetivo f con respecto al resto de la población, para comportarse como padres de la siguiente generación. La selección de estas listas se hace con base en el valor *promedio de aptitud*, dándole a las listas con un valor mayor de aptitud, mayor probabilidad de selección, para que contribuyan con uno o más descendientes en la siguiente generación. Este operador, claramente es una versión artificial de selección natural, un superviviente darwiniano de los más aptos entre las criaturas lista.

Esta selección se basa tanto en la *aptitud individual* de cada cromosoma al enfrentar el problema, como en la *aptitud global* de la población. Entre más apto sea un cromosoma con respecto a la

población, tiene mayor probabilidad de que se seleccione como padre de la siguiente generación. Este operador se aplica a todos los individuos, la función objetivo es el árbitro final de la vida o la muerte de las criaturas lista.

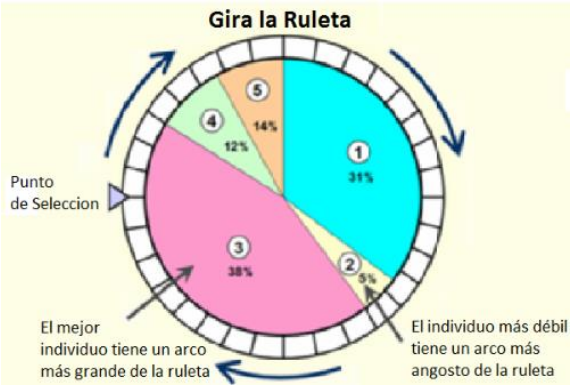


Figura 3.2 Selección por ruleta

Una forma básica de selección de los cromosomas más aptos como padres de la siguiente generación es la selección por ruleta. Como se puede observar de la figura 3.2, a los cromosomas o individuos más aptos se les asigna un arco mas amplio que a los individuos débiles.

Cruce simple o cruzamiento procede en dos pasos. **Primero,** la población de listas seleccionadas como padres entran en el juego del apareamiento, mezclándose al azar. **Segundo**, se efectúa el cruce al azar de cada par de listas como sigue: se selecciona con probabilidad uniforme, una posición $\boldsymbol{k}$, entre $\mathbf{1}$ y la longitud $\boldsymbol{l}$ de la cadena menos **1,** $[\mathbf{1}, \boldsymbol{l} - \mathbf{1}]$. Se crean dos nuevas cadenas entre las posiciones $\boldsymbol{k} + \mathbf{1}$ y $\boldsymbol{l}$ incluidas. Por ejemplo, considere las listas con $\boldsymbol{l} = 5$, y los cromosomas A1 y A2, que se han seleccionado como padres:

A1 = 0 1 0 1 1 1 0 1 0
A2 = 0 1 1 1 1 0 0 1 0

Suponga que se hizo la escogencia de un número aleatorio entre 1 y 8, y se obtuvo $\boldsymbol{k}$ = 5. El cruce resultante da dos nuevas cadenas A1' y A2' que entran a formar parte de la nueva generación:

A1' = 0 1 0 1 1 0 0 1 0
A2' = 0 1 1 1 1 1 0 1 0

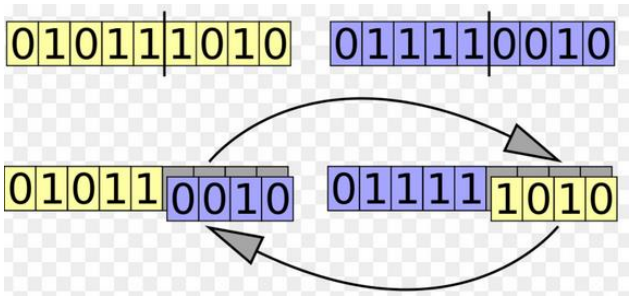


Figura 3.3. Cruce simple con $\boldsymbol{l} = 9$ y $k = 5$

Este operador permite combinar la información de cromosomas exitosos, para la producción de descendencia con información que se supone más exitosa que la de sus padres, lo que permite la explotación de información genética validada en determinadas partes del espacio de búsqueda.

Los mecanismos de reproducción y cruce son sorprendentemente simples: generación de números aleatorios, copia de listas, e intercambio parcial de contenido de listas. Es el énfasis combinado de reproducción y el intercambio estructurado de información, aunque al azar, lo que le da a los AG mucha de su potencia. Es sorprendente que, con dos operadores tan simples y computacionalmente tan sencillos, se pueda producir un mecanismo de búsqueda tan robusto y exitoso.

Mutación, es un operador genético de segundo orden en el AG, pues es mucho más provechoso el intercambio de información de estructuras exitosas en el cruce. Sin embargo, la mutación permite la exploración de áreas del espacio de soluciones, que puedan presentar picos de optimización más altos. En el caso del AGS simplemente, con base en una probabilidad, se cambia un 1 por un 0, o viceversa. Este operador se aplica a los individuos con una probabilidad $\boldsymbol{p_m}$ generalmente baja.

Evita que la solución se quede en un óptimo local, otra forma de verlo es que aumenta la diversidad genética de la población, posibilitando mejores soluciones. En muchos casos se usa la mutación como función del tiempo (generación) de evolución, dando una mayor probabilidad de mutación al comienzo del proceso. Comúnmente se aplica luego del cruce.

La repetición permanentemente de estos tres operadores a través de las generaciones, continúa hasta llegar o a un número determinado de generaciones o cuando se satisfaga algún otro criterio de finalización; tomando la mejor solución hasta ese momento como la respuesta al problema.

3.3 Algoritmo Genético Simple

A continuación, se presenta una descripción del AGS.

1. Se genera aleatoriamente la primera población. Se crean $\boldsymbol{K}$ cromosomas cada uno con $\boldsymbol{l}$ bits de longitud.
2. Se repiten $\boldsymbol{M}$ veces los siguientes pasos, o el número de veces que sea necesario hasta satisfacer alguna otra condición de terminación del algoritmo:
    i. Se obtiene el valor $\boldsymbol{x}$, por decodificación del cromosoma..
    ii. Se calcula la aptitud $\boldsymbol{f(x)}$, de cada cromosoma; se encuentra la aptitud global de la población, que es la sumatoria de las $\boldsymbol{f(x)}$; se calcula la

aptitud con que contribuye cada cromosoma a la aptitud global de la población, $f(x)$ dividida por la aptitud global de la población, esta se toma como la probabilidad de selección de ese cromosoma.

iii. Para cada cromosoma de la población actual se efectúa un proceso de selección con base en su probabilidad de selección, que define si tendrá o no descendientes en la siguiente generación. La selección se hace sin reemplazo, es decir, que un cromosoma puede seleccionarse más de una vez para ser padre. Se mantiene el mismo tamaño de la población.

iv. Se escoge un par de cromosomas de la población de padres. Se cruzan el par de cromosomas en un punto escogido aleatoriamente (con una probabilidad uniforme) para formar dos hijos.

v. De acuerdo con una probabilidad de mutación p_m, se elige sustituir un gen con valor 1 por un gen con valor 0 por 1. Una vez aplicados estos operadores se tiene la nueva población.

vi. Reemplaza la población actual por la nueva población. Se disminuye en 1, el conteo de generaciones.

vii. Regresa al paso 2.

Debido a que la aleatoriedad juega un papel importante en la ejecución del algoritmo genético, dos ejecuciones con diferentes semillas en la generación de los números aleatorios, pueden producir comportamientos diferentes en las primeras generaciones, para hacia al final converger a las mismas respuesta o a respuestas muy cercanas. Los reportes estadísticos, como aptitud promedio de la población, generación en la cual se encontró el individuo con la mejor aptitud, son en promedio iguales para diferentes ejecuciones del AG sobre el mismo problema. Además, es posible que en diferentes corridas no se obtengan los mismos resultados, para un número fijo de generaciones.

El procedimiento descrito es la base de las aplicaciones del AG. Existen otros detalles que se deben tener en cuenta en dichas aplicaciones, tales como el tamaño de la población y las probabilidades de cruce y mutación, ya que el éxito del algoritmo depende en gran parte de estos detalles.

3.4 Ejemplo de un Algoritmo Genético Simple

Supongamos que se quiere encontrar la raíz de un polinomio por AGS. Dado el polinomio $P(x) = x^2 - 1.23x - 6.26$, encontrar su raíz positiva que se encuentra entre *1* y *4.*

Teniendo en cuenta que necesitamos encontrar un número entre 1 y 4, con tres decimales, el cromosoma, la lista debe almacenar ceros y unos que representen ese intervalo, estos son los datos para escoger el número ***l*** de bits de la lista. Supongamos ***l*** *= 6,* los valores que se pueden representar son:

0	0	0	0	0	0

= 0

1	1	1	1	1	1

= 63

Dado que el límite inferior es 1, la evaluación binaria de la primera lista es 0, de manera que el 1 del problema se representa por 0, el límite superior del intervalo es 4, pero dado que el 0 representa 1, entonces el 4 del problema, es 1 + 3, así el 3 se representa por 63. Lo primero que calculamos es que dec es el valor decimal del cromosoma, entonces $x = 1 + 3 * \frac{dec}{63}$. Esta es la forma en que se decodifica la solución $\boldsymbol{x}$ del problema de la estructura lista, para este ejemplo especifico.

Ahora vamos a generar aleatoriamente 8 listas. Para decodificar los valores de cada lista, primero encontramos su valor decimal equivalente ***D*** al binario almacenado y luego lo transformamos a un valor entre 0 y 4, son los valores ***x***, con que vamos a evaluar la función de aptitud $\boldsymbol{P(x) = x^2 - 1.23x - 6.26}$. Entonces, evaluamos el polinomio con esos valores y miramos su resultado. En este caso definimos la aptitud como: $Aptitud = 10 - Valor\ absoluto(f(x))$.

A continuación, se desarrolla el ejemplo en Excel. Para crear la primera generación usamos la función **ALEATORIO.ENTRE(0,1).** Para encontrar la aptitud usamos la función **ABS( ).**

Pob. Nueva						***Dec***	***x***	***x2-1.123x-6.26***	***Aptitud***	***Probab.***	***Selección***						***Cruce***						***Mutación***					
0	0	1	0	0	1	9	1,429	-5,823469388	4,17653	0,0845	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1956	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0
1	1	1	1	1	0	62	3,952	4,922791383	5,07721	0,1028	1	0	0	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1
0	0	1	1	1	1	15	1,714	-5,246367347	4,75363	0,0962	1	0	1	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0
1	0	0	1	1	0	38	2,81	-1,521671202	8,47833	0,1716	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551	0,1944	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0
0	0	0	0	1	1	3	1,143	-6,237306122	3,76269	0,0762	1	0	0	1	1	0	1	0	0	1	1	1	1	0	0	1	0	1
0	0	0	1	0	1	5	1,238	-6,117501134	3,8825	0,0786	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0
								Aptitud total =	49,4002																			

En este caso definimos la aptitud como: $Aptitud = 10 - Valor\ absoluto(f(x))$

Se obtiene la aptitud de cada lista y la aptitud total de la población. La probabilidad de seleccionar una lista como padre de la siguiente generación, viene dada por la relación entre la aptitud propia de cada lista y la aptitud total de la población. A través de varios mecanismos, como puede ser la ruleta, se seleccionan las listas padre para la nueva población. Es común que la mejor lista se repita varias veces en la población de listas seleccionadas, en este caso la lista 101110. Se selecciona dos veces. Luego, el cruce, se toman dos listas seguidas y a partir de una posición aleatoria se intercambian los contenidos. Posteriormente, la mutación, se selecciona una posición para cambiar un 1 por 0, o viceversa. Se obtiene la nueva población.

Pob. Nueva						Dec	x	x2-1.123x-6.26	Aptitud	Probab.	Selección						Cruce						Mutación					
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1372	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	62	3,952	4,922791383	5,07721	0,0721	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551	0,1364	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1372	1	0	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	1	1	1	47	3,238	0,588879819	9,41112	0,1336	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	0	1	0	42	3	-0,629	9,371	0,1331	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	0	1	0	1	37	2,762	-1,733501134	8,2665	0,1174	1	0	1	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	0	1	0	42	3	-0,629	9,371	0,1331	1	0	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1
								Aptitud total =	70,4299																			

Ahora se tiene la nueva población y se repiten los mismos pasos. Aquí la aptitud global aumenta más de un 57%, con respecto a la generación anterior. Visto de otra manera, es más difícil ser competitivo en esta población. Es importante considerar que solo estamos utilizando 6 bits, por lo que tenemos muy poca resolución en la respuesta. También es importante observar que las mejores listas tienen

1	0	1

en sus tres primeras posiciones, este es un esquema exitoso. Así tenemos que cualquiera sea la solución del problema la lista más apta debe llevar esos tres bits con esos valores. El teorema del Algoritmo Genético dice que las mejores soluciones se componen de esquemas exitosos y lo que hace la evolución es conformar la mejor solución a partir de esquemas exitosos.

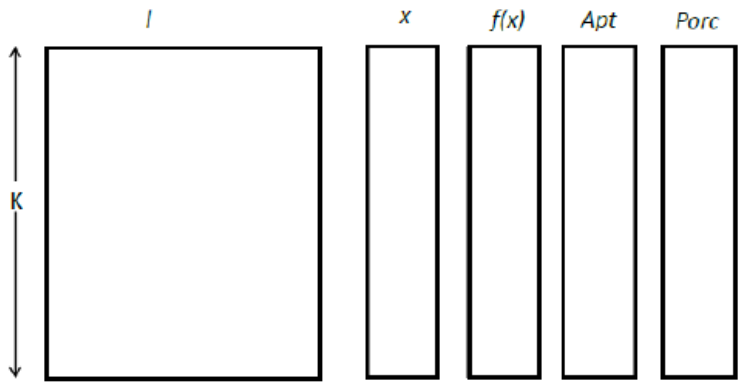
Pob. Nueva						Dec	x	x2-1.123x-6.26	Aptitud
1	0	1	1	1	1	47	3,238	0,588879819	9,41112
1	0	1	0	1	0	42	3	-0,629	9,371
1	0	1	1	0	1	45	3,143	0,088122449	9,91188
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	0	1	0	42	3	-0,629	9,371
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551
								Aptitud total =	76,6618

Ahora con la nueva población repetimos el proceso, pero lo detenemos, al encontrar que la tercera lista de la población nos da la solución del problema. Este es uno de los criterios de detención del A.G., el otro criterio, el más común es cuando el A.G. se ha corrido un número de generaciones definido previamente.

3.5 Codificación de las Rutinas Básicas de un Algoritmo Genético

Rutina principal

Corresponde a la descripción que acabamos de estudiar. La estructura de datos que se va a utilizar es la de una matriz de bits $\boldsymbol{l} * \boldsymbol{K}$, y cuatro vectores asociados; $\boldsymbol{x}$, que es el resultado de decodificar cada cromosoma, $\boldsymbol{f}(\boldsymbol{x})$ que es la evaluación de la función de aptitud, $\boldsymbol{Apt}$, que es la aptitud de cada cromosoma, y , $\boldsymbol{Porc}$ que es el porcentaje o probabilidad con que ese cromosoma contribuye a la aptitud total de la población.



El ciclo principal de la rutina se ejecuta $\boldsymbol{M-1}$ veces. La variable $\boldsymbol{i}$ se emplea como índice de las iteraciones. La estructura pob guarda los $\boldsymbol{K}$ cromosomas de longitud $\boldsymbol{l}$ que se generan inicialmente.

```
def Alg_Genetico(ecuacion,l, M, K, p_mut):
    pob = genera(l,K)  #Se genera aleatoriamente la población
    probab = eval_apt(pob,l) #Se evalúa esa primera población
    i =0;                #Es la primera generación
    while i < M:     #Mientras no se ejecuten todas las generaciones
        n_pob = seleccion(pob, probab) #Selecciona, los nuevo padres
        hijos = cruce(n_pob)         #Cruce de los padres generan hijos
        pob = mutacion(hijos, p_mut, l)   #Se mutan los hijos para generar otra población
        probab = eval_apt(pob,l) #Se evalua la población i, se calcula la prob. de selección
        i = i + 1
```

Rutina Genera

Esta rutina simula el lanzamiento de una moneda, para generar aleatoriamente un bit.

```
def genera(l,K):
    pob = [randint(0, 2, l).tolist() for _ in range(K)]
    return pob
```

Rutina evalúa

Una vez se tiene la primera generación, se debe evaluar el comportamiento de cada cromosoma con respecto a la función de aptitud. Luego encontrar cuál es su aporte a la aptitud global de la población para saber qué tan bueno es con respecto a los demás.

La aptitud total de la población se guarda en $\boldsymbol{apt\_pob}$, la aptitud del cromosoma $\boldsymbol{i}$ que guarda en el vector asociado apt_crom. La aptitud con que contribuye el cromosoma $\boldsymbol{i}$ a la aptitud global, en porcentaje se guarda en el vector asociado $\boldsymbol{probab}$.

```
def eval_apt(pob,l):
    apt_crom = []
```

Con formato: Inglés (Estados Unidos)

```
  apt_pob = 0
  for i in range(0, len(pop)):
    crom = pop[i]
    apt = evalua(crom,l)
    apt_pob += apt
    apt_crom.append(apt)
  probab = []
  for i in range(0, len(pop)):
    prob_crom = apt_crom[i]/apt_pob
    probab.append(prob_crom)
  return probab
```

Función F_aptitud

Esta función, que devuelve un valor real, es la encargada de evaluar la aptitud del cromosoma $\boldsymbol{i}$ de la rutina anterior. La decodificación del cromosoma $\boldsymbol{i}$ produce el valor $\boldsymbol{x}$ con que se evalúa esta función. Para ilustrar mejor el algoritmo, tomamos como ejemplo maximizar una función real arbitraria: $\boldsymbol{f}(\boldsymbol{x}) == 5 * x ** 5 - 3 * x ** 4 - x ** 3 - 5 * x ** 2 - x - 3$. Se llama a la función ***decodifica,*** que convierte la cadena binaria ***cromosoma*** en un valor del espacio de soluciones.

```
def evalua(crom,l):
  x = decodifica2(crom,l)
  y = ecuacion(x)
  apt = 15 - abs(y)
  return apt

def ecuacion(x):
  valor = 5*x**5-3*x**4-x**3-5*x**2-x-3
  return valor
```

Función Decodifica

Esta función convierte el contenido binario del cromosoma, el genotipo, en un valor real, el fenotipo, dentro del espacio de soluciones. Cómo existen diversas formas de representar el espacio de soluciones en la estructura del cromosoma, para cada una de esas formas hay un esquema propio de codificación.

La función ***decodifica*** es la que realmente hace que el AG sea universal, ya que con solo modificarla se pueden representar una amplia variedad de problemas. La variable ***rango*** define el intervalo en el cual puede estar el valor real del cromosoma, para el ejemplo rango es $[0.5, 1.5]$. En ***valorDecimal*** se calcula el valor del cromosoma en binario.

```
def decodifica(crom,l):
  xi = 0.5  # Limite inferior en el dominio real
  xf = 1.5  # Limite superior en el dominio real
  Max = 2 **(l)
  cromPot = [crom[i]*2**(l-i-1) for i in range(l)]
  #print(cromPot)
  valorDecimal = sum(cromPot)
  #print(valorDecimal)
```

```
    valDeco = ((xf-xi)/Max)*(valorDecimal)+xi
    return valDeco
```

Procedimiento de selección

Este procedimiento es la versión artificial de la selección natural, que busca un superviviente darwiniano entre los cromosomas más aptos; en las poblaciones naturales la aptitud está definida por la habilidad del individuo para sobrevivir a predadores, pestes y otras situaciones que reducen la posibilidad de que el individuo llegue a adulto y pueda reproducirse. El ciclo externo se debe a que se trabaja con población constante y se deben seleccionar $\boldsymbol{K}$ individuos. El ciclo interno es la búsqueda de los mejores individuos dentro de la población, que se debe repetir hasta completar los $\boldsymbol{K}$ individuos, es por esto los mejores individuos se seleccionan varias veces.

```
def seleccion(pob, probab):
    j = 0
    K = len(pob)
    #print("long pob", K)
    #k = len(probab)
    #print("long probab", k)
    limite = 2 * max(probab)
    pob_nueva = []
    while j < K:
        i = 0
        while i < K:
            aleat = random.uniform(0, limite)
            if probab[i] > aleat:
                pob_nueva.append(pob[j])
                j += 1
                if j >= K:
                    break
            i +=1
            if i == K:
                i = 0
    return pob_nueva
```

Procedimiento de Cruce

Este procedimiento no utiliza la probabilidad de cruce, se implementa fácilmente considerando que la población se ha ido generando aleatoriamente. Simplemente se seleccionan dos parejas consecutivas de la población seleccionada y se cruzan en un punto escogido aleatoriamente. ***pt*** es el punto para el cruce y es una posición aleatoria entre ***2*** y $\boldsymbol{l} - \mathbf{1}$. Este es el cruce más elemental, se pueden hacer cruces multipunto, lo que mejora el comportamiento del algoritmo pues elimina sesgos en la población.

```
def cruce(pob_nueva):
    i = 0
    K = len(pob_nueva)
    hijos =[]
```

```
  while i < K:
    crom1 = pob_nueva[i]
    crom2 = pob_nueva[i+1]
    pt = randint(1, len(crom1)-1)
    hijo1 = crom1[:pt] + crom2[pt:]
    hijo2 = crom2[:pt] + crom1[pt:]
    hijos.append(hijo1)
    hijos.append(hijo2)
    i = i +2;
  return(hijos)
```

Procedimiento de Mutación

Considerando que la probabilidad de mutación en general es muy baja y con el fin de ahorrar tiempo de proceso, se toma su inverso como un bloque o segmento en el cual debiera haber una mutación. Se encuentra el número de mutaciones y dentro del bloque se lanza ese número de aleatorios para cambiar en las posiciones dadas.

```
def mutacion(hijos, p_mut, l):
  K = len(hijos)
  totalbits = K * l
  segmento = 1/p_mut
  n_mutaciones = totalbits/segmento
  i = 0
  while i < n_mutaciones:
    muta = randint(0, totalbits-1)
    x = math.floor(muta/l)
    y = muta%l
    if hijos[x][y] == 0:
      hijos[x][y] = 1
    else:
      hijos[x][y] = 0
    i += 1
  return hijos
```

Otras funciones

Dependiendo del nivel de sofisticación del programa, se pueden incluir rutinas para la impresión de los cromosomas, sus aptitudes, graficas del mejor cromosoma por generación, y otros; según las necesidades de análisis del algoritmo. Por ejemplo, llevar el mejor cromosoma y su aptitud en cada generación, permite observar cómo es la convergencia del algoritmo. En caso de no llegar a una convergencia, se puede replantear la codificación de las variables, o la evaluación de la función, o el método de selección o la estructura del cromosoma.

De lo anterior se puede ver la potencialidad del AG, que a partir de un proceso aleatorio se van generando soluciones, en este caso, para funciones algebraicas. Sin embargo, el AG es una herramienta que no se limita a usar genes binarios, sino que se puede utilizar otros tipos de datos para la solución combinatoria de cualquier problema, como se mencionó.

En las aplicaciones que actualmente se trabajan, dependiendo del problema, se tienen diferentes tipos de datos para los genes, lo que lleva a variar y aumentar los operadores de cruce y mutación; aunque también es común la creación de nuevos operadores genéticos.

3.6 Problemas Tipo NP y NP-Completos

3.6.1 Problemas Tipo P

Un algoritmo cuyo tiempo de ejecución está dado por un polinomio que es función del tamaño de los datos de entrada, se conoce como un algoritmo con tiempo de ejecución polinomial. Un problema pertenece a la clase $\boldsymbol{P}$ si el número de pasos requeridos para hallar su solución, es decir, su tiempo de ejecución, está limitado o puede definirse por un polinomio, el problema se conoce como de Tipo P. Este tipo de problemas evita preocupaciones en cuanto al modelo de máquina, dado que todos los modelos de computación, incluyendo la máquina de Turing, manejan complejidad de tiempo relacionada polinomialmente. En su gran mayoría son los problemas que se trabajan permanentemente.

3.6.2 Problemas Tipo NP

Los problemas $\boldsymbol{NP}$ son intratables en el sentido de que no se ha encontrado una solución real y eficiente para el problema, cuando el tamaño de los datos de entrada está dado por un $\boldsymbol{n}$ grande. En este caso, el tiempo necesario para hallar una solución crece exponencialmente con $\boldsymbol{n}$. Un problema pertenece a la clase de problemas de tiempo polinomial no determinístico, $\boldsymbol{NP}$, si se puede resolver en tiempo polinomial en una máquina de Turing no determinística. Una máquina de Turing no determinística $\boldsymbol{MTN}$, es una máquina de Turing paralela que puede tomar muchos caminos computacionales simultáneos, con la restricción de que las máquinas de Turing paralelas no se pueden comunicar.

Algunos algoritmos han podido encontrar buenas soluciones a problemas **NP**, sin embargo, no se puede afirmar que se haya encontrado la solución óptima. Por ejemplo, el problema de asignar los horarios de los exámenes finales, en una universidad con $\boldsymbol{n}$ cursos y con solo 5 días para efectuar los exámenes. Otro es el bien conocido problema de encontrar el recorrido completo de un agente viajero, que debe visitar $\boldsymbol{n}$ ciudades tan solo una vez y al final llegar a la ciudad de partida, en un tiempo mínimo.

3.6.3 Problemas Tipo NP Completos

Los problemas NP conocidos, tienen una característica importante: todos se pueden reducir a uno. Es decir que, dados dos problemas NP, X y Y, existe un algoritmo de tiempo polinomial que redefine un problema de tipo X como un problema de tipo Y, y existe un algoritmo de tiempo polinomial que traslada una solución de un problema de tipo Y a una solución para un problema de tipo X. O sea, que, si se encuentra un algoritmo de tiempo polinomial para cualquiera de estos problemas, entonces, hay un algoritmo derivado, de tiempo polinomial, para todos los demás problemas del conjunto. Un problema que satisface estos requerimientos se denomina un problema NP - Completo.

Por ejemplo, el problema del agente viajero es un problema NP-Completo, así que cualquier otro problema que se pueda transformar en este problema también es un problema NP-Completo, que es lo que en general sucede con los problemas de asignación. Otro ejemplo es el problema del transporte, en el que se debe determinar un conjunto de rutas que debe seguir un conjunto de vehículos que parten de uno o más depósitos para satisfacer la demanda de varios clientes dispersos geográficamente.

Ha habido mucho trabajo entre los matemáticos alrededor de si un problema tipo P es igual a un problema tipo NP, pues esto llevaría a la solución en tiempo polinomial de muchos problemas tipo NP, sin embargo en el artículo de D. Cardona, se demuestra que P ≠ NP, utilizando A.C.s.

3.7 El Problema del Furgón

Para comprender como trabaja el AG en estos problemas, consideremos un ejemplo concreto, el problema del furgón que es un problema NP- Completo clásico. Estableciéndolo simplemente es: dada una colección de artículos que varían en peso y valor, que se quieren transportar en un furgón que tiene una determinada capacidad, se debe encontrar una combinación de artículos para transportar, que tenga el mayor valor total, pero que no exceda un peso límite.

En otras palabras, la meta está en llenar completamente un furgón hipotético con el lote más caro que puede llevar. Aunque es fácil de describir, esta meta puede ser difícil de realizar. Por ejemplo, una colección de solo 50 artículos presenta 2^{50} posibles combinaciones diferentes. Suponiendo que un computador pueda probar un millón de combinaciones diferentes por segundo, tomaría alrededor de 35 años para tratarlas todas.

Aquí se muestra cómo un AG resuelve tal problema, pero en aras de la ilustración, se usa un número mínimo de artículos. La colección contiene diez artículos, así que hay un poco más de 1.000 combinaciones posibles, para tratar en el problema. Hay cinco tipos de artículos diferentes, de 4 a 8 unidades de peso, de 6 a 12 unidades de valor y el número de artículos va de 1 a 3, ver Tabla 3.1. El furgón puede cargar un peso máximo de 24 unidades, así que puede llevar un A, o dos B's, etc. La Tabla 1 enumera todos los artículos, las unidades con sus pesos y valores; y el número disponible de cada tipo de artículo para transportar.

Tabla 3.1

ARTICULO	PESO	VALOR	No. DE ARTIC.
A	4	6	3
B	5	7	3
C	6	8	2
D	7	10	1
E	8	12	1

Aprovechando dar solución al problema anterior, a continuación, se presentan los pasos que se siguen para solucionar un problema por AG.

3.8 Diseño de un AG para la solución de problemas

Como ya se ha dicho, la estructura general de AG es común para la solución de problemas diferentes independiente de su dominio. Para diseñar la solución de un problema con AGs, se deben seguir los siguientes pasos:

1. Definir el problema. Debe haber claridad con respecto al problema que se pretende solucionar por algoritmos genéticos, en términos de complejidad computacional.
2. Definir la representación de los individuos, o diseñar los genes que conforman el cromosoma. Define el espacio de búsqueda de las todas las soluciones.
3. Definir los operadores genéticos. Buscar los operadores apropiados de selección, cruce y mutación, o crearlos.
4. Definir la función de aptitud. Nos permite medir el desempeño de cada individuo. En algunos casos es posible que haya que incluir penalidades.
5. Definir el criterio de parada y la población.
6. Definir otros parámetros. Como la población y la probabilidad de mutación.

Es importante aclarar que estos pasos son una guía, dado que en cualquier momento, dependiendo del comportamiento del algoritmo genético, se debe volver a algunos pasos anteriores para afinar ese comportamiento.

1. Definir el problema. El algoritmo genético es una técnica de optimización, en ese sentido se debe ver el problema, tener claridad en que se lo que se quiere optimizar.

2. Definir la representación de los individuos. El diseño del individuo tiene que ver con la forma en que se van a expresar las soluciones del problema. Si se busca la solución de una ecuación a través de una variable, la solución debe ser un valor real. Si la solución es un recorrido, la solución debe ser una serie de puntos (x, y).

El diseño de cada uno de los genes que conforman el cromosoma, define el tipo de datos de cada gen y su forma de codificación, que naturalmente dependen del dominio del problema que se va a solucionar.

El diseño del cromosoma, o la estructura de datos, que determina cómo representar una solución potencial del problema planteado, define el espacio de soluciones del problema, en términos de la codificación de los genes que la componen.
En algunos casos también se debe tener en cuenta que la longitud del cromosoma puede ser más larga o menos, al considerar la precisión que se le quiera dar a su solución, como en el caso de encontrar la raíz de una ecuación.

3. Definir los operadores genéticos. Los tipos de datos de los genes pueden ser binarios, enteros, reales, lógicos, o por extensión cualquier tipo de objeto. Nos dan la pauta para la selección o creación de los operadores apropiados. Dependiendo del dominio del problema, se deben definir y adecuar los operadores genéticos que se van a utilizar en el AG.

Generalmente, más que definir los operadores genéticos apropiados es escogerlos dentro de una gama de operadores bien conocidos. Los más comunes son los operadores binarios, aunque también en ellos se encuentra una gama de posibilidades. En el caso de los operadores de tipo real, aparecen diferentes opciones, dependiendo del dominio del problema a solucionar. Pero también

puede uno pensar en objetos y definir sus propios operadores genéticos. Supongamos que estoy trabajando con voz; podría definir una clase con objetos voz y sus operadores genéticos.

4. Definir la función de aptitud. Permite decidir y medir la bondad de la respuesta del AG, es una de las consideraciones más importantes a tener en cuenta en el diseño del AG, pues de ella depende todo el comportamiento del AG.

En muchos casos en que el problema tiene restricciones, como en el Problema del Furgón, se presenta que algunos cromosomas violan las restricciones. Los investigadores de AG han explorado muchos enfoques a la violación de restricciones, pero ninguno es perfecto. Aquí hay tres posibilidades:

> **Eliminación**. La eliminación intenta determinar si un cromosoma infringe la restricción, aún antes de su creación. Este enfoque tiene varios problemas. En primer lugar, puede ser demasiado costoso en su ejecución, o simplemente imposible. Segundo, al prevenir la creación de cromosomas que infrinjan las restricciones, se puede ocasionar que el AG pase por alto soluciones perfectamente válidas. Esto porque los infractores podrían producir descendientes legales (no infractores) que llevarían a una solución satisfactoria más rápidamente.
>
> **Alta Penalidad.**
> Este enfoque impone una pena alta a los infractores. Reduce la aptitud de los infractores, permitiéndoles, ocasionalmente, propagar su descendencia. Una debilidad de este enfoque se hace evidente, cuando una población contiene gran porcentaje de infractores. En este caso, los cromosomas legales dominarán las generaciones siguientes y los infractores se dejarán sin explotar. Este efecto podría conducir al estancamiento de la población.
>
> **Penalidad Moderada.** Un tercer enfoque impone una penalidad moderada a los infractores. Aumenta la probabilidad de que los infractores procreen, reduciendo así la oportunidad de que la población se estanque. También presenta sus propios problemas, especialmente cuando los infractores se clasifican mejor que los cromosomas legales. En este caso, si los infractores no crean listas legales, entonces los infractores dominarán las generaciones siguientes. Además, si los infractores clasifican más alto que las listas legales, entonces los criterios para terminar la búsqueda deben incorporar un mecanismo para detectar infractores.

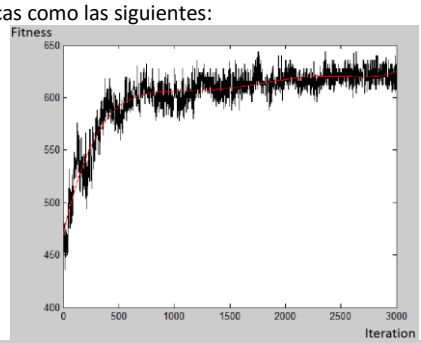
Con respecto a la población, el número de cromosomas es proporcional a la complejidad del problema. Esto puede conllevar gastos fuertes en poder de cómputo, pues el proceso de cada cromosoma requiere más cálculos. Este es uno de los defectos del AG, ya que después de procesar, en cada generación $\boldsymbol{K}$, cromosomas, al final solo se requiere un cromosoma como respuesta. Así la población debe ser lo suficientemente grande para crear un conjunto diverso de individuos, para que el AG pueda explotar mejor el espacio de búsqueda. El número de generaciones tiene que ver con la población, a mayor población, mayor número de generaciones.

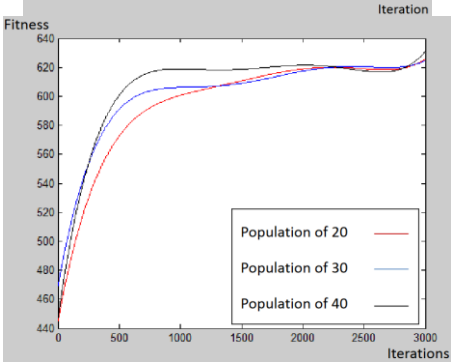
5. Definir el criterio de parada. El criterio de parada más sencillo hace referencia al número de generaciones del AG, pues en muchas ocasiones no es fácil establecerlo precisamente en términos de la función de aptitud. La idea es que el AG se va afinando para incluir un criterio de parada en términos de la solución.

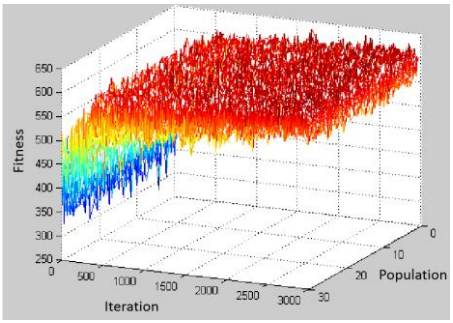
Estos pasos, como los de cualquier otro diseño en ingeniería, si bien se plantean secuencialmente, son de un ir y volver, hasta llegar a un mejor diseño. La combinación apropiada de estas tareas, le da al algoritmo genético la potencialidad para resolver el problema.

En el caso de problema NP-Completos, el criterio de parada más común es el número de generaciones, dado que la explosión combinatoria es tan alta que no se puede llegar a conocer la respuesta o configuración óptima.

7. **Definir otros parámetros.** Son varios los parámetros, además del número de generaciones, la población y la probabilidad de mutación, que se pueden considerar como: el mejor cromosoma de cada generación, la mejor aptitud en cada generación y la aptitud global de la población. Otra ayuda puede ser una representación gráfica del espacio de búsqueda, para poder observar cómo están distribuidas las soluciones y si falta explorar alguna parte del espacio de búsqueda, especialmente en las primeas generaciones. Con estos datos se pueden obtener graficas como las siguientes:







Con respecto a la probabilidad de mutación, como decíamos es muy baja, pero en algunos casos se puede hacer dependiente de la generación, disminuyendo en la medida en que aumenta el número de generaciones.

3.8.1 Diseño de un AG para el problema del Furgón

1. Definir el problema. Dada una colección de artículos que varían en peso y valor, según la tabla 3.1, se quiere transportar una combinación de artículos en un furgón con una capacidad de carga de máximo 24 unidades de peso, que tenga el mayor valor total.

2. Definir la representación de los individuos.. Lo que se va a evaluar en cada generación del AG es una población de cromosomas o listas con el número de elementos de cada artículo que se van a transportar. Se debe representar el número de elementos de cada artículo, que conforma la lista. De manera que el tipo de datos de cada gen puede ser: binario o entero.

Para el ejemplo vamos a usar binarios, por la aplicación de operadores genéticos conocidos. Así, para expresar 3, el número máximo de artículos A que se pueden transportar, se requieren 2 bits, el primer gen necesita dos posiciones binarias; el segundo gen para representar los artículos B necesita 2 bits; el tercero necesita para representar los artículos C necesita 2 bits; el cuarto para representar los artículos D necesita 1 bit; y el quinto para representar artículos E necesita 1 bit.

El cromosoma o estructura de datos permite incluir todos los detalles de la lista de artículos que se van a transportar en el furgón, los genes definidos anteriormente. En este caso concreto es un cromosoma con el número de artículos A, B, C, D y E, que va llevar el furgón.

Tabla 3.2 Esquema de codificación de los artículos para la lista

Posición	1- 2	3 - 4	5 - 6	7	8
Artículos	A	B	C	D	E

3. Operadores. Vamos a usar los operadores binarios ya conocidos de cruce y mutación, dado que se va a trabajar con números binarios.

4. Aptitud. En este caso una función de aptitud muy simple sería cuantificar el valor total de cada una de las listas de la población, así las listas de artículos con mayor aptitud serían las que den mayor valor transportado, es decir cargar todos los artículos en el furgón.

Sin embargo, hay que imponer la restricción del peso, así las listas de artículos con mayor peso menor que la restricción son las más aptas. ¿Pero qué pasa con las listas de artículos que pesan más? ¿Las eliminamos? ¿o las penalizamos? Su función de aptitud suma el valor de cada artículo y resta una penalidad moderada para los infractores. La penalidad es tres veces la cantidad del exceso de peso. Para esto primero se define V, el valor total de la lista y P, el peso total de la lista. Donde x_i es el número de elementos del artículo i; v_i el valor de un artículo i; y p_i el peso de un artículo i. Entonces,

$$V = \sum_{i=1}^{5} x_i \; x_i \quad \text{y} \quad P = \sum_{i=1}^{5} p_i \; x_i$$

De manera que $f_{apt}(X) = V$ si $P \leq 24$; o $f_{apt}(X) = V - 3(P - 24)$

Una primera generación aleatoria de 5 listas, podría ser como la que se muestra en la Tabla 3.3. La primera lista coloca seis artículos en el furgón: dos A, tres B y un E; para un peso total de 31 unidades y un valor total de 45 unidades. La segunda lista coloca seis artículos en el furgón: un A, un B, dos C's, un D y un E, para un peso de 33 y un valor de 51. La tercera cadena coloca solo dos artículos: un A y un D para un peso de 11 y un valor de 16. La cuarta lista coloca cuatro artículos: un A, un B y dos C's para un peso total de 21 unidades y un valor de 29. Por último, la lista cinco, carga 4 artículos: un A, un B, un D y un E para un peso de 24 unidades y un valor de 35.

Tabla 3.3. Listas generadas aleatoriamente, que representan los artículos cargados en el furgón

Artículo	A	B	C	D	E	PESO	VALOR
Posición	1,2	3,4	5,6	7	8		
Lista 1	1 0	1 1	0 0	0	1	31	45
Lista 2	0 1	0 1	1 0	1	1	36	51
Lista 3	0 1	0 0	0 0	1	0	11	16
Lista 4	0 1	0 1	1 0	0	0	21	29
Lista 5	0 1	0 1	0 0	1	1	24	35

La Tabla 3.4 muestra la aptitud resultante de aplicar la función de aptitud a las cinco listas tomadas de ejemplo anterior.

Tabla 3.4. Listas de artículos cargados en el furgón con pesos, valores y aptitud

Listas	1	2	3	4	5
Pesos	31	36	11	21	24
Valores	45	51	16	29	35
Aptitud	24	15	16	29	35

5. Criterio de parada. Dado que es un problema de combinatoria, el algoritmo va a parar cuando se cumpla el número de generaciones definido.

6. Definir otros parámetros. En este caso vamos a parar cuando se hayan corrido 100 generaciones, M = 100 con una población K = 30 individuos; Se escogió esta población más bien pequeña para ilustrar mejor como trabaja el AG.

3.8.2 Iniciación.

Después de adelantar los pasos anteriores, el AG está listo para correr. La primera tarea del AG es crear una población inicial de listas. Hay muchas maneras para seleccionar una población inicial; los enfoques oscilan desde definir aleatoriamente cada carácter de cada cadena, hasta modificar interactivamente los resultados de una búsqueda hecha anteriormente.

La composición de la población inicial puede afectar dramáticamente el desempeño del algoritmo genético. La mayor diversidad en la población inicial le da más oportunidades al AG, para explotar el espacio de búsqueda. El esquema del lanzamiento de la moneda tiene la ventaja de la simplicidad y la diversidad. Es simple pues no requiere ninguna información sobre el problema. El esquema es diverso porque la función crea listas que van desde casi cero hasta uno y todos los valores entre ellos.

Aquí es de destacar el algoritmo de generación de números aleatorios, porque muchos de los generadores de números aleatorios, realmente son pseudoaleatorios, se repiten después de cierto tiempo.

3.8.3 Tipos de selección

Selección de padres y procreación

Después que se construye la población, el programa principal entra en un ciclo while, y permanece allí hasta que se encuentre una solución. Es poco probable que alguna lista de la generación inicial contenga una solución. El primer paso para crear la siguiente generación es de nuevo la selección de dos padres. El método de seleccionar los padres en un AG es muy importante ya que puede incidir significativamente en la eficiencia de la búsqueda. Hay muchos tipos de formas de seleccionar los padres para la siguiente generación y veamos algunos.

Ruleta

Es el tipo de selección más conocido y que hasta ahora se hemos utilizado. En este caso la selección de una lista determinada es proporcional a su aptitud. En primer lugar, se calcula la aptitud de cada cromosoma; con base en estos valores de calcula la aptitud global de la población. Ahora, se divide la aptitud de cada cromosoma por la aptitud global de la población y así se tiene una probabilidad de selección de esa estructura, así las mejores estructuras, tienen mayor probabilidad de que se seleccionen. Se decide si el cromosoma se selecciona como padre, generando un número aleatorio, si es menor que la probabilidad del cromosoma, ese cromosoma se selecciona, de lo contrario no.

Selección por Elitismo

Hay varias versiones de este tipo de selección, la que se presenta a continuación permite mantener la diversidad genética incluyendo siempre individuos menos aptos. Aquí se toma la población con sus cromosomas ya calificados por la función de aptitud, se ordena con base en los valores de aptitud de cada estructura. Automáticamente se selecciona un porcentaje de las mejores estructuras de población. Por ejemplo, en una población de 100 cromosomas, se selecciona directamente el 20% es decir los 20 mejores. Luego simulando el lanzamiento de una moneda, se

van seleccionando uno de cada dos cromosomas de la población ordenada que queda. Así muchos cromosomas con una aptitud baja tienen una probabilidad de selección más alta que en la ruleta, lo que permite una mayor distribución inicial de las estructuras en el espacio de búsqueda.

Selección por clasificación
El propósito es impedir la convergencia demasiado rápida, ya que los mejores cromosomas no difieren mucho en su aptitud. Como en el caso anterior, los cromosomas de la población se ordenan de acuerdo con la aptitud, y se obtiene un nuevo valor de aptitud de cada cromosoma, que depende más de su clasificación que de su aptitud absoluta.

El mejor cromosoma tendrá una aptitud K, el tamaño de la población; y el último tendrá una aptitud de 1. La formación de clases evita darle mucha aptitud a los más mal clasificados, aunque tienen oportunidad de ser seleccionados, suponemos R clases. Se puede compartir mucho la descendencia, con un grupo pequeño altamente apto, y así reducir la presión de selección, cuando la varianza de la aptitud es alta. Para calcular la probabilidad de selección se calcula $SumR_i$, que es la suma de todas las aptitudes de la población por clases. $R_{i,j}$ es la clase del cromosoma j en la generación i.

$$SumR_i = \sum_{i=1}^{K} R_{i,j} \text{ y}$$

La probabilidad de selección del cromosoma j en la generacion i es:

$$p(R_{i,j}) = \frac{R_{i,j}}{SumR_i}$$

Selección ayudada por el diseñador
Sin embargo, como ya se ha dicho, estas son las formas convencionales de hacer la selección y puede haber muchas otras formas de hacerlo, combinando diferentes métodos, que en muchos casos dependen del dominio del problema; o con la participación del diseñador. En este caso se presentan dos formas.

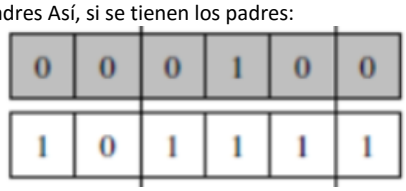
1. Ayudada o Interacción. Aquí el diseñador va observando el desarrollo del proceso y encuentra que por alguna razón ciertos espacios del espacio de búsqueda no se han explorado, entonces directamente inserta algunos cromosomas como padres, que permitan realizar la exploración en ciertas partes del espacio de búsqueda.

2. Directa. En este caso es el diseñador quien directamente califica cada cromosoma. Para esto primero define los criterios de calificación del cromosoma dependiendo del tipo de solución que se esté trabajando. Puede ser por ejemplo en un estudio sobre imágenes, donde el ojo del diseñador define la aptitud de cada imagen de la población, en estos casos la población no debe ser muy grande, para que el ojo no se fatigue y pueda valorar los cromosomas sin parcialidad. También se puede dar en aplicaciones de audio.

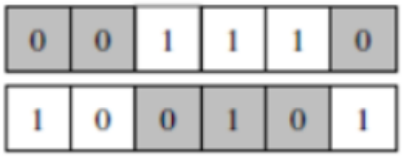
3.8.4 Cruce
Lo mismo que con el operador de Selección hay varios tipos de cruce, el común, es el cruce uniforme que ya hemos estudiado. Que tiene problemas de parcialidad ya que muchas veces los primeros y los últimos bits no se cruzan tan frecuentemente como los otros bits de la lista.

Cruce en un punto
Es el que hemos estudiado en los ejemplos anteriores. Sin embargo, tiene un par de limitaciones. Una que es muy probable que genes vecinos sigan juntos y la otra que los genes de los extremos nunca estén juntos. Este sesgo posicional puede llevar, dependiendo del problema, a una exploración pobre en el espacio de soluciones.

Cruce multipunto.
En este caso se definen aleatoriamente varios puntos donde efectuar el corte, dependiendo de la longitud del cromosoma. Por ejemplo, si se definen tres puntos, aleatoriamente se distribuyen los puntos para cada par de padres Así, si se tienen los padres:



Se seleccionan dos puntos de cruce en los padres, los hijos quedan:



3.8.6 Mutación.
Mutación por Similitud
A veces los hijos experimentan mutación. En el ejemplo clásico del furgón usa un operador de mutación interesante. En lugar de una probabilidad fija de mutación, se usa una probabilidad que cambia con base en la conformación de la población. Esta rutina compara los dos padres del hijo; la mayor similitud entre padres aumenta la probabilidad que ocurra una mutación en el hijo.

Mutación variable
Otra forma de operar la mutación es haciéndola variable en el tiempo. En este caso la probabilidad inicial de mutación es muy alta, en la medida en que pasan las generaciones la probabilidad es más baja. La razón para usar una probabilidad de mutación variable está en reducir la oportunidad de que haya convergencia prematura. Esta condición ocurre cuando la población se precipita hacia una solución mediocre y entonces simplemente se desperdician recursos. Hay poca diversidad y la búsqueda se convierte en una caminata aleatoria entre el promedio.

Encontrando la Respuesta
El AG continúa durante muchas generaciones aplicando los operadores ya definidos, pero se requiere un criterio de parada. Uno sería cuando encuentre una lista que resuelve el problema, pero en general este no es el caso. Pero si se desconoce la mejor solución, ¿cómo puede el programa determinar si ha encontrado la mejor respuesta, o por lo menos, una de las mejores respuestas? Un primer enfoque sería resolver para una solución fija, que encuentre algún valor predeterminado como mínimo aceptable. Un segundo enfoque es el de correr el programa hasta que caiga su tasa

para encontrar nuevas respuestas mejores, o la tasa de mejoramiento de esas respuestas se disminuya drásticamente.

Veamos el ejemplo del furgón durante tres generaciones.

	A	B	C	D	E	Valor	Peso	Aptit.	Prob.	Selección	Nueva Población
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,131	1 0 1 1 0 0 1 1	1 0 1 1 0 0 1 1
1 0 1 1 0 0 1 1	2	3	0	1	1	91	38	49	0,189	1 1 0 1 0 0 0 1	1 1 0 1 0 0 0 1
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,12	1 0 1 1 0 0 1 1	1 0 0 1 0 0 0 1
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,131	1 1 0 1 0 0 0 1	1 1 1 1 0 0 1 1
1 0 1 1 0 0 1 1	2	3	0	1	1	55	16	55	0,212	1 0 1 1 0 0 1 1	1 0 1 1 0 0 0 1
0 0 0 0 1 1 0 0	0	0	3	0	0	24	18	24	0,093	1 1 0 1 0 0 0 1	1 1 0 1 0 0 1 1
1 1 1 1 0 0 1 1	3	3	0	1	1	61	42	7	0,027	1 1 0 1 0 0 0 1	1 1 0 1 0 0 0 1
1 0 1 1 0 1 1 0	2	3	0	1	0	43	30	25	0,097	1 0 0 1 0 0 0 1	1 0 0 1 0 0 0 1
	Aptitud Total							**259**	**1**		

	A	B	C	D	E	Valor	Peso	Aptit.	Prob.	Selección	Nueva Población
1 0 1 1 0 0 1 1	2	3	0	1	1	55	38	13	0,066	1 1 0 1 0 0 0 1	1 1 0 1 0 0 0 1
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,173	1 0 0 1 0 0 0 1	1 0 0 1 0 0 0 1
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,157	1 1 0 1 0 0 0 1	1 1 0 1 0 0 0 1
1 1 1 1 0 0 1 1	3	3	0	1	1	61	42	7	0,036	1 0 0 1 0 0 0 1	1 0 0 1 0 0 0 1
1 0 1 1 0 0 0 1	2	3	0	0	1	45	31	24	0,122	1 0 0 1 0 0 1 0	1 0 0 1 0 0 1 0
1 1 0 1 0 0 1 1	3	1	0	1	1	47	32	23	0,117	1 0 1 1 0 0 0 1	1 0 1 1 0 0 0 1
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,173	1 1 0 1 0 0 0 1	1 1 0 1 0 0 0 1
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,157	1 0 0 1 0 0 0 1	1 0 0 1 0 0 0 1
	Aptitud Total							**197**			

	A	B	C	D	E	Valor	Peso	Aptit.	Prob.	Selección	Nueva Población
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,137	1 1 0 1 0 0 0 1	1 0 0 1 0 0 0 1
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,125	1 0 0 1 0 0 0 1	1 1 0 1 1 0 0 1
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,137	1 0 0 1 0 0 0 1	1 0 0 1 0 0 0 0
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,125	1 0 0 1 0 0 1 0	1 0 0 1 0 0 1 1
1 0 0 1 0 0 1 0	2	1	0	1	0	29	20	29	0,117	1 1 0 1 0 0 0 1	0 1 0 1 0 0 1 1
1 0 1 1 0 0 0 1	2	3	0	0	1	45	31	24	0,097	1 0 0 1 0 0 1 0	1 1 0 1 0 0 1 0
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	0,137	1 1 0 1 0 0 0 1	1 0 0 1 0 0 1 0
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	0,125	1 0 0 1 0 0 1 0	1 1 0 1 0 0 0 1
	Aptitud Total							**248**			

	A	B	C	D	E	Valor	Peso	Aptit.	Prob.
1 0 0 1 0 0 0 1	2	1	0	0	1	31	21	31	
1 1 0 1 1 0 0 1	3	1	2	0	1	53	37	14	
1 0 0 1 0 0 0 0	2	1	0	0	0	19	13	19	
1 0 0 1 0 1 1 1	2	1	1	1	1	49	34	19	
0 1 0 1 1 0 1 1	1	1	2	1	1	51	36	15	
1 1 0 1 0 0 1 0	3	1	0	1	0	35	24	35	
1 0 0 1 0 1 1 0	2	0	1	1	0	30	21	30	
1 1 0 1 0 0 0 1	3	1	0	0	1	37	25	34	
	Aptitud Total							**197**	

El ejemplo del furgón se corre hasta que se encuentre una respuesta conocida, en este caso el cromosoma óptimo es el número 6, con una aptitud de 35. Desde luego, este problema es lo suficientemente pequeño para resolver con métodos tradicionales, pero sirve para observar cómo opera el código en forma detallada y darle una buena idea de cómo trabaja el AG.

3.6 Operadores de Punto Flotante

En particular, para problemas de optimización con variables en dominios continuos, se puede experimentar con genes codificados en punto flotante, para los cuales se utilizan operadores especialmente diseñados para ellos. Vamos a ver aquí los operadores utilizados en la solución de problemas de Programación Lineal por AG. Tomando como la función a optimizar $f(x_1, x_2, \ldots\ldots, x_q)$, se denomina D el conjunto convexo formado por las restricciones y por los rangos de las variables. Estos operadores son propuestos por Michalewicz

Con base en lo anterior, se define un cromosoma como una solución factible a un problema de Programación Lineal como la cadena.

$$(gen_1, gen_2, \ldots., gen_q)$$

donde el gen_i representa la variable x_j x_i, y el cromosoma representa el punto factible:

$$<x_1, \ldots, x_q>$$

Para estas nuevas cadenas se define un conjunto de operadores genéticos susceptibles de modificarse o ampliarse son base en los resultados de las experimentaciones.

De la convexidad del conjunto D se sigue que para cada punto del espacio de búsqueda $< x_1, x_2, \ldots, x_q >$, hay un rango $[Izquierda(k), derecha(k)]$ de la componente o variable x_k (para cada $t \leq k \leq q$), donde las otras variables $x_i (i = 1, \ldots, k-1, k+1, \ldots., q)$ permanecen fijas.

En otras palabras, para un punto $< x_1, \ldots, x_k, \ldots, x_q > \in D$, la componente x_k, está en el rango $[Izquierda(k), derecha(k)]$, si y solo si $< x_1, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_q > \in D$, donde todos los demás x_i $(i = 1, \ldots, k-1, k+1, \ldots, q)$ permanecen constantes. Por lo tanto, si el conjunto de restricciones está vacío para el espacio de búsqueda convexo, formado por los dominios de las variables x_k ($l_k \leq x_k \leq u_k$ para $k = 1,2,\ldots,q$), donde los limites inferior y superior, $l_k = izquierda(k)$ y $u_k = derecha(k)$ (para $k = 1,2,\ldots,q$); esto significa que los operadores propuestos constituyen un conjunto válido sin la presencia de restricciones.

A continuación, se presentan siete nuevos operadores genéticos como cruces, mutaciones y selección de la nueva población de cada generación.

3.6.1 Mutación Uniforme

Este operador requiere un único padre *x* y produce un único descendiente *x´*. Suponiendo que cada padre cuenta con *n* elementos o genes, el operador selecciona un componente aleatorio x_k con $k \in \{1,2,\ldots,n\}$, del vector $x=<x_1,\ldots,x_q>$, y produce $x' = <x_1,\ldots, x_k',\ldots,x_n>$, donde x' es un valor aleatorio con distribución de probabilidad uniforme dentro del rango [*Izquierda(k), derecha(k)*].

El operador juega un papel importante en las primeras fases del proceso de evolución, pues permite que las soluciones se desplacen libremente dentro del espacio de búsqueda. En particular es esencial cuando la población consta de múltiples copias del mismo punto factible. También es útil en las últimas etapas del proceso evolutivo puesto que permite movimientos lejanos de un óptimo local, búsqueda de una mejor solución.

3.6.2 Mutación No Uniforme

Este operador requiere un único padre x y es el responsable de la capacidad de ajuste de las soluciones encontradas, por estar los cromosomas en punto flotante. Se define de la siguiente manera:

Para un padre *x,* se selecciona para mutar un elemento x_k, resultando como nuevo descendiente $x'=<x_1,\ldots, x_k',\ldots,x_n>$, donde:

$$x_k' = x_k + \Delta(t, derecha(k), x_k),$$ con probabilidad *0.5*

$$x_k' = x_k - \Delta(t, x_k, izquierda(k)),$$ con probabilidad *0.5*

La función $\Delta(t, y)$ retorna un valor en el rango $[0, y]$, de tal manera que la probabilidad de que $\Delta(t, y)$ esté muy cercana a cero y va aumentando a medida que *t*, la generación actual, crece, donde t , representa la generación. Esta propiedad hace que el operador al comienzo haga una búsqueda uniforme en el espacio de las soluciones, cuando *t* es pequeño, y en etapas posteriores lo haga muy localmente.

La función se define como: $\Delta(t, y) = y * (1 - r^{(1-t/T)^b})$

Donde r es un número aleatorio entre 0 y 1, T es el número de generaciones y b es un parámetro del sistema que entre mayor sea, hace que la variación sea menor. Generalmente se toma como 2.

3.6.3 Mutación en la Frontera

Este operador requiere también un único padre x y produce un único descendiente x'. Es una variación de la *Mutación no uniforme*, es en donde x_k' es o bien *izquierda(k),* es o bien *derecha(k)*, cada uno con igual probabilidad.

El operador se construye con el fin de encontrar soluciones que estén cerca o sobre el espacio de búsqueda factible. Si el conjunto de restricciones está vacío y el límite de los dominios de las variables es muy grande el operador es ineficaz, sin embargo, es muy útil en presencia de restricciones.

3.6.4 Cruce Aritmético Simple

Este es operador binario, cuya definición es la siguiente:

Si $x^1 = < x_1, \ldots, x_n >$ y $x^2 = < y_1, .., y_n >$, se han seleccionado para cruce en la posición $k - ésima$ seleccionada también aleatoriamente, los descendientes son:

$$x'^1 = < x_1, \ldots, x_k, y_{k+1}, \ldots, y_n >$$ y $$x'^2 = < y_1, \ldots yx_k, x_{k+1}, \ldots, x_n >$$

Como esta operación puede producir descendientes que no se encuentren dentro de D, se utiliza la propiedad del espacio convexo que garantiza la existencia de un *a[0,1]* tal que:

$$x'^1 = < x_1, \ldots, x_k, y_{k+1} * a + x_{k+1} * (1 - a), \ldots\ldots\ldots, y_n * a + x_n * (1 - a) >$$ y

$$x'^2 = < y_1, \ldots, y_k, x_{k+1} * a + y_{k+1} * (1 - a), \ldots\ldots\ldots, x_n * a + y_n * (1 - a) >$$

pertenecen al dominio D. Para encontrar el mayor valor de a, de modo que se obtenga un intercambio considerable de información, se comienza con $a = 1$ y si por lo menos uno de los descendientes no pertenece a D, se decrementa la constante a en $(1/q)$, de manera que después de q intentos a tiende 0, y ambos descendientes siguen perteneciendo a D, puesto que son idénticos a sus padres.

3.6.5 Cruce Aritmético Completo

Este es otro operador binario que se define como la combinación lineal de dos vectores, de tal manera que si se van a cruzar X^1 y X^2 el resultado es:

$$X^1 = a * X^1 + (1 - a)X^2 \text{ y } X^2 = a * X^2 + (1 - a)X^1$$

los cuales pertenecen a *D*.

3.6.6 Cruce Heurístico

Este operador utiliza los valores de la función objetivo para determinar la dirección de la búsqueda. Produce un solo descendiente x^3 de dos padres x^1 y x^2, de acuerdo con la siguiente regla: $x^3 = r(x^2 - x^1) + x^2$

donde r es un número aleatorio generado entre $[0,1]$ y el padre x^2 es más apto que x^1. Es posible que este operador genere un descendiente que se sitúe en una región no factible. En tal caso se genera otro número aleatorio r y se crea otro descendiente; después de w intentos no se encuentre un descendiente factible, el operador deja como descendiente el mejor de sus padres.

El **Cruce Heurístico** además de contribuir con la precisión de la solución, es el responsable de la búsqueda en una dirección promisoria.

3.7 Aplicaciones

Optimización de funciones numéricas.

Los AGs se han utilizado mucho en esta tarea, para la solución de ecuaciones especialmente complicadas, como puede ser la búsqueda de raíces en polinomios de 3er. grado en adelante. O en la solución de ecuaciones diferenciales donde se hace necesaria la solución numérica. También en funciones no continuas o funciones con mucho ruido.

Procesamiento de imágenes

Hay muchas aplicaciones que van desde el arte, generando nuevas expresiones artísticas hasta sistemas de ayuda para la obtención de las caras de sospechosos a partir de los detalles que puede suministrar un testigo. El AG puede generar caras que se pueden ir acercándose a los perfiles descritos, en este caso es el humano el que define la aptitud de los seres cara.

Optimización Combinatoria

Es la tarea de encontrar soluciones a problemas que involucran configuraciones de objetos discretos. A mi modo de ver es uno de los nichos donde mejor se desempeñan los AGs. Son muchos los ejemplos, tales como el problema de la mochila, el problema del agente viajero, programación de tareas en máquinas y muchos otros.

Diseño

En diseño también se mezclan muchos problemas de optimización combinatoria. Por ejemplo, diseño de puentes, diseño de sistemas de distribución de fluidos, y otros.

3.8 Ejercicios y Problemas.

1. Suponga que usted es el jefe de gobierno y está interesado en que pasen los proyectos de su programa político. Sin embargo, en el congreso conformado por 5 partidos, no es fácil su tránsito, por lo que debe repartir el poder, conformado por ministerios y otras agencias del gobierno, con base en la representación de cada partido. Cada entidad estatal tiene un peso de poder, que es el que se debe distribuir. Suponga que hay 50 curules, distribuya aleatoriamente, con una distribución no informe entre los 5 partidos esas curules. Defina una lista de 50 entidades y asígneles aleatoriamente un peso político de 1 a 100 puntos. Cree una matriz de poder para repartir ese poder, usando AGs.

2. Una empresa proveedora de energía eléctrica dispone de cuatro plantas de generación para satisfacer la demanda diaria de energía eléctrica en Cali, Bogotá, Medellín y Barranquilla. Cada una puede generar 3, 6, 5 y 4 GW al día respectivamente. Las necesidades de Cali, Bogotá, Medellín y Barranquilla son de 4, 3, 5 y 3 GW al día respectivamente. Los costos por el transporte de energía por cada GW entre plantas y ciudades se dan en la siguiente tabla:

	Cali	Bogotá	Medellín	Barranq.
Planta C	1	4	3	6
Planta B	4	1	4	5
Planta M	3	4	1	4
Planta B	6	5	4	1

Los costos del KW-H por generador se dan en la siguiente tabla:

Generador	$KW-H
Planta C	680
Planta B	720
Planta M	660
Planta B	750

Encontrar usando AGs el mejor despacho de energía minimizando los costos de transporte y generación.

2. En el siguiente enlace se encuentra un programa Python para la solución del TSP (Traveling Salesman Problem):
https://github.com/rocreguant/personal_blog/blob/main/Genetic_Algorithm_Python_Example/Traveling_Salesman_Problem.ipynb

3. Estúdielo y úselo como laboratorio, usando diferentes tipos de selección y diferente número de generaciones. Imprima las gráficas correspondientes.

4. Genere aleatoriamente una población de 50 palabras, que se escuche por el parlante del computador. Tomando como función de aptitud una palabra suya, usando AGs, con base en las palabras generadas aleatoriamente llegue a la palabra que usó como función de aptitud.

5. Tome un problema de los anteriores, u otro cualquiera, y utilice la librería Python de AGs Pygad y para solucionarlo.

3.9 Bibliografía

Cardona, Daniel, P NOT EQUAL TO NP, Universidad Nacional, 2024, https://arxiv.org/pdf/2401.15079.pdf
Goldberg David, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.
Haupt Randy et al, PRACTICAL GENETICALGORITHMS, JOHN WILEY & SONS, INC., 2004
Holland John, Natural and Artificial Systems, an Introductory Analysis with Applications to Biology, 1995. 2nd. Edition, 1995
Mitchell Melanie, An Introduction to Genetic Algorithms, MIT Press, 1999.
Martínez José J., Rojas Sergio, Introducción a la Informática Evolutiva, Nuevo Enfoque para Solucionar Problemas de Ingeniería, Facultad de Ingeniería, Universidad Nacional de Colombia, 2000.