PRÁCTICA 3. Algoritmo genético

Alumno: Isaac André Canalizo Mendoza 201728726

I Objetivos

General: Diseñar un algoritmo genético para los problemas de TSP, Set covering y Schwefel.

Específico: Para cada uno de los problemas encontrar cuál es la configuración de algoritmo genético que obtiene en promedio el mejor resultado.

II Desarrollo

Set covering

1) **Parámetros fijos:**
   a) **Tamaño de la población:**
      Se estableció con un valor de 1000, todos los elementos son generados aleatoriamente con la función genPop()

```
def genPop():
    pob = []
    binArray = []
    n = nPop = 0
    while(nPop < 1000):
        binArray = np.random.randint(0,2,63009)
        pob.append(binArray)
        nPop += 1
    return pob
```

   b) **Generaciones:**
      Se estableció en 100 repeticiones el ciclo que abarca desde la evaluación de la población hasta el reemplazo.

```
while(nGen < 100):
    listaFitness = genListaFit(population, lista)
    listOnlyFitVal = listaFitness[1]
    idPadres = genPadres(listOnlyFitVal, numHijos, tamMaxPool)
    idPadres = np.array(idPadres)
    listaFitnessHijos = getHijos(idPadres[:,0], listaFitness,
crossoverPoint)
    listaFitnessHijos = genListaFit(mutacion(listaFitnessHijos[0]), lista)
    population = replacement(listaFitness, listaFitnessHijos)
    nGen += 1
```

      b

c) **Selección de padres: Torneo binario**
Este se realiza con dos métodos, uno que genera los padres, las piscinas y sus respectivos tamaños y uno que encuentra al ganador junto con su posición

```
def genPadres(listaFitness, numHijos, tamMaxPool):
    padres = []
for i in range(numHijos):
        ban = ban1 = False
        while(ban == False or ban1 == False):
            if(ban == False):
                n1 = random.randint(2, tamMaxPool)
                if(n1 % 2 == 0):
                    ban = True
            if(ban1 == False):
                n2 = random.randint(2, tamMaxPool)
                if(n2 % 2 == 0):
                    ban1 = True

        pool1 = getElements(n1, listaFitness)
        pool2 = getElements(n2, listaFitness)

        winner = binTournament(pool1, pool2)
        padres.append(winner)
    return padres
```

```
def binTournament(pool1, pool2):
    min1, min2 = min(pool1[1]), min(pool2[1])
    pos1, pos2 = pool1[0][pool1[1].index(min1)],
pool2[0][pool2[1].index(min2)]
    if(min1 < min2):
        return [pos1, min1]
    else:
        return [pos2, min2]
```

d) **Elitismo:**
Al seleccionarse un porcentaje del 25% de la población con peor valor de fitness para ser reemplazada por los hijos, los mejores resultados siempre se mantienen

Parámetros seleccionados:

1. **Política de reemplazo:**
   Se seleccionó una política de estado estable, siendo 25% el porcentaje de cambio elegido.
2. **Probabilidad de mutación**
   30%

```
def mutacion(hijos):
    umbral = .30
    for hijo in hijos:
        if(random.random() < umbral):
            r = random.randint(10, 100)
            for i in range(r):
                ban = False
                while(ban == False):
                    pos = random.randint(0, 63009)
                    if(hijo[pos] == 1):
                        hijo[pos] == 0
                        ban = True
    return hijos
```

3. **Operaciones genéticas**
   a. **Cruza**
   b. **Mutación**

```
def crossover(elemento1, elemento2, crossoverPoint):
    child1 = child2 = []
    i = 0
    tam = len(elemento1)
    while(len(child1) < 63009 and len(child2) < 63009 ):
        if(i < crossoverPoint):
            child1.append(elemento2[i])
            child2.append(elemento1[i])
        else:
            if(i < tam - crossoverPoint):
                child1.append(elemento1[i])
                child2.append(elemento2[i])
            else:
                child1.append(elemento2[i])
                child2.append(elemento1[i])
        i += 1
    return [child1, child2]
```

Resultados

No se logró obtener satisfactoriamente realizar las ejecuciones necesarias para ser documentadas al contemplar una población con mil elementos y cien generaciones ya que se corrobora que todos sean soluciones válidas, al realizarlo con poblaciones más pequeñas el mejor resultado obtenido fue de 31350.