TRABAJO # 2

Sobre algoritmos genéticos

John Fredy Zapata Zapata

Margarita Maria Zuluaga

Jeison David Atehortua Alvarez

Inteligencia Artificial

Pedro Atencio

Medellín

Marzo de 2019

INFORME

Para afrontar el trabajo, hicimos un listado de los pasos requeridos para llegar a la solución, a continuación los pasos:

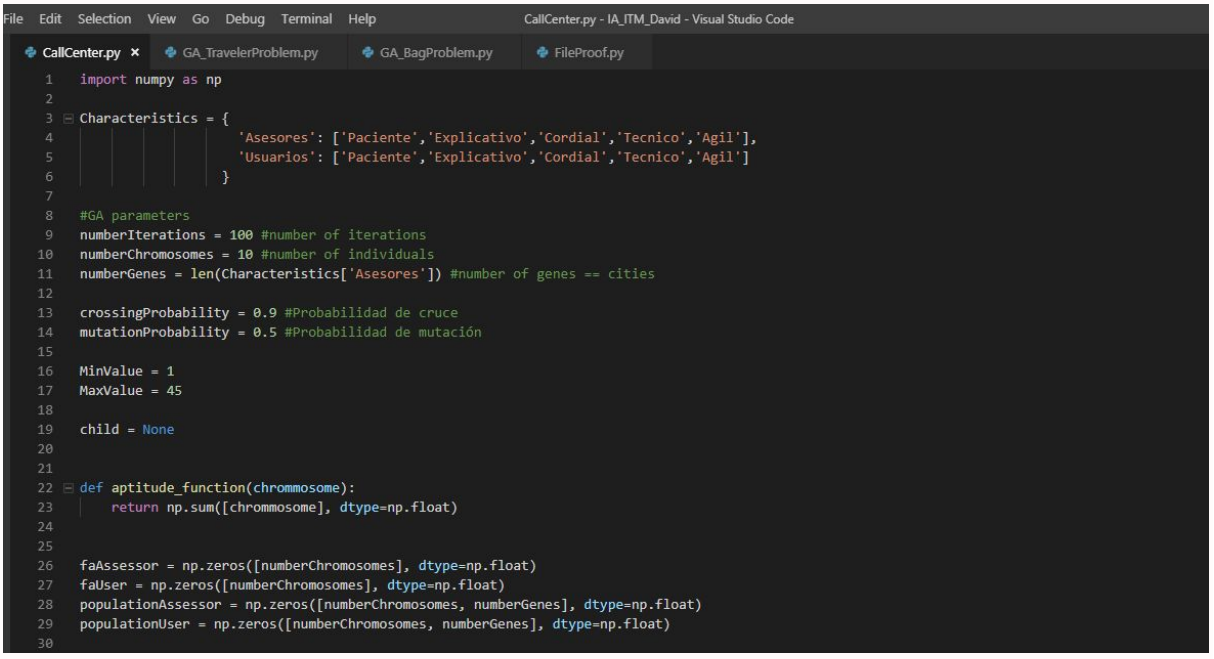
1. Listar características del asesor (Punto 1 del taller)
2. Listar características del usuario (Punto 1 del taller)
3. Ver si es posible emparejar las características usuario/asesor (Punto 1 del taller)
4. Definir tipo de vector (Binario, Entero, Flotante) **Flotante** Min 1 Max 45 (Punto 2 del taller)
5. Generar población de asesores (Punto 3 del taller)
6. Generar población de usuarios (Punto 3 del taller)
7. Función de aptitud para los asesores (Punto 3 del taller)
8. Función de aptitud para los usuarios (Punto 3 del taller)
9. Meta-Parámetros GA (Punto 4 del taller)
10. Método Selección Ruleta (Punto 5 y 6 del taller)
    a. Suma de todos los valores de “fa”
    b. Calcular porcentaje para cada “fa”
    c. Generar número aleatorio
    d. Sumas parciales hasta que supere r
11. Ciclo optimización algoritmo genético (Punto 5 del taller)
    a. Selección Ruleta
    b. Cruce
    c. Mutación
    d. Evaluación de los nuevos hijos
    e. Inserción nuevos hijos
12. Indicar que asesor atiende a determinado usuario
    a. Ordenar Descendentemente la población de asesores, según su “fa”
    b. Ordenar Ascendentemente la población de usuarios, según su “fa”
    c. Asignarle a cada usuario el asesor que corresponda según sus posiciones ordenadas Usuario[i] = Asesor[i]
13. Comparar desempeño selección ruleta contra selección aleatoria

Logramos identificar del método ruleta, que es una manera sencilla de selección, es una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores; pero a la vez se vuelve un poco ineficiente en medida que aumenta el tamaño de la población, además se corre el riesgo de elegir el peor individuo más de una vez.

Se puede concluir que con el método ruleta se tiene una mejor solución de forma repetida, diferente a la selección aleatoria, que en algunos casos puede o no ser más efectiva.

Otra conclusión a la que se llega, es que la efectividad del algoritmo genético, depende mucho del método de selección que se utilice, dado que se obtienen resultados más efectivos según el método utilizado.

Pantallazos de la solución codificada




```
#population initialization and aptitude function calculated
for i in range(numberChromosomes):
    for j in range(numberGenes):
        populationAssessor[i, j] = np.random.uniform(MinValue,MaxValue)
        populationUser[i, j] = np.random.uniform(MinValue,MaxValue)
    faAssessor[i] =  aptitude_function(populationAssessor[i])
    faUser[i] =  aptitude_function(populationUser[i])

print('\n-------------Población Inicial Asesor------------- \n', populationAssessor)

#Ciclo principal del algoritmo genético:
#Seleccion->Cruce->Mutación->Evaluación->Inserción
def ruletaSelection(fa):

    totalFa = np.sum([fa], dtype=np.float)
    percentFa = np.zeros([numberChromosomes], dtype=np.float)
    selectedCrhomosomes = np.zeros([2], dtype=np.int)

    for i in range(numberChromosomes):
        percentFa[i] = fa[i]/totalFa

    selectedCrhomosomes[0] = np.argmax(np.cumsum(percentFa) >= np.random.rand())
    selectedCrhomosomes[1] = np.argmax(np.cumsum(percentFa) >= np.random.rand())

    return selectedCrhomosomes
```

```python
for i in range(numberIterations):

    selectedCrhomosomes = ruletaSelection(faAssessor)
    chromoSelectP1 = selectedCrhomosomes[0]
    chromoSelectP2 = selectedCrhomosomes[1]

    #Cruce Aritmetico Completo
    if(np.random.rand() <= crossingProbability):
        a = 0.5
        fatherA = populationAssessor[chromoSelectP1]
        fatherB = populationAssessor[chromoSelectP2]
        child = np.zeros([0], dtype=np.float)
        for m in range(numberGenes):
            child = np.append(child,a*fatherA[m]+(1-a)*fatherB[m])

    if(child is None):
        continue

    #Mutación, suma número aleatorio a un par de genes seleccionados aleatoriamente (cod. real)
    if(np.random.rand() <= mutationProbability):
        randomPosition = np.random.randint(numberGenes)
        child[randomPosition] =  child[randomPosition] + np.random.uniform(MinValue,MaxValue)/100
        randomPosition = np.random.randint(numberGenes)
        child[randomPosition] = child[randomPosition] + np.random.uniform(MinValue,MaxValue)/100

    #Evaluamos el hijo
    evaluationChild = aptitude_function(child)

    #Inserción
    randomPositionFa = np.random.choice([chromoSelectP1, chromoSelectP2])
    if(evaluationChild > faAssessor[randomPositionFa]):
        populationAssessor[randomPositionFa] = child
        faAssessor[randomPositionFa] = evaluationChild

populationAssessorOrder = np.zeros([numberChromosomes, numberGenes], dtype=np.float)
count = 0

for i in np.argsort(faAssessor)[::-1]:
    populationAssessorOrder[count] = populationAssessor[i]
    count = count + 1

populationUserOrder = np.zeros([numberChromosomes, numberGenes], dtype=np.float)
count = 0

for i in np.argsort(faUser):
    populationUserOrder[count] = populationUser[i]
    count = count + 1

print('\n------------Población final Asesor------------\n', populationAssessor)
print('\n------------Población Usuario----------------\n', populationUser)

#Asignación de asesores a usuarios:
#Los asesores con mayor calificación son asignados a los usuarios con las caracteristicas mas bajas.
#En este orden de ideas, el asesor mas calificado atenderá al usuario con las preferencias de menor calificación.
for i in range(numberChromosomes):
    print('\nEl usuario con preferencias ',aptitude_function(populationUserOrder[i]),
        'Sera atendido por el asesor con caracteristicas ',aptitude_function(populationAssessorOrder[i]))
```

Anexamos archivo python CallCenter.py