NUMBER MIND

Carlos Sevilla Barceló || Francisco Javier Fernández Montero

GRUPO 2 Inteligencia Artificial

Introducción

En este documento se describe los operadores de mutación y cruce diseñados para resolver el problema. Además del algoritmo genético concreto que se ha utilizado para buscar la solución.

Representación

Cada individuo de nuestro algoritmo genético tendrá un genotipo numérico. De tamaño igual al tamaño de las pistas. En nuestro caso tendrá una longitud de 16.
La función de valoración se basa en las pistas proporcionadas. La valoración es proporcional al número de pistas que cumple el individuo. La solución óptima es la solución que cumple todas las pistas.

Operadores de mutación

Para los operadores de mutación hemos elegido intercambio aleatorio y mutación uniforme.

Intercambio aleatorio

```
def intercambioAleatorio(indiv,cambioporparametro):
    cambios=0
    listaNuestra=indiv.getLista()

    if (cambioporparametro > 16) or (cambioporparametro == 0):
        cambios = randint(1,16)

    else:
        cambios = cambioporparametro

    for i in range(cambios):
        pos1=randint(0,15)
        pos2=randint(0,15)

        numero1=listaNuestra[pos1]
        numero2=listaNuestra[pos2]

        listaNuestra[pos1] = numero2
        listaNuestra[pos2] = numero1

    res = Individuo(listaNuestra)

    return res
```

Esta es la implementación que hemos realizado para pasar a python el método de intercambio aleatorio visto en clase.
Se le pasan por parámetros 1 individuo y el número de genes que se intercambian. El método nos devuelve otros 1 individuo, que es el individuo mutado resultantes de ese individuo pasado por parámetro.

Mutación Uniforme

```
def mutacionUniforme(indiv, probabilidadDeMutacion):
    mutacion=indiv.getLista()

    if(not(probabilidadDeMutacion>0 and probabilidadDeMutacion<10)):
        probabilidadDeMutacion=randint(0,10)

    for i in range(0,len(mutacion)-1):
        aleatorio2=randint(0,10)
        if(aleatorio2>probabilidadDeMutacion):
            mutacion[i]=randint(0,10)

    res = Individuo(mutacion)
    return res
```

Esta es la implementación que hemos realizado para pasar a python el método de mutación uniforme visto en clase. Se le pasan por parámetros 1 individuo y la probabilidad de que un gen mute. El método nos devuelve otros 1 individuo, que es el individuo mutado resultantes de ese individuo pasado por parámetro.

Operadores de cruce

Hemos elegido los siguientes operadores de cruces, porque son los mejor adaptados a nuestro problema, al tipo de genotipo:

Recombinación en un punto

```
def recombinacionEnUnPunto(indiv1, indiv2):
    genotipo1 = indiv1.getLista()
    genotipo2 = indiv2.getLista()

    hijo1 = genotipo1[:]
    hijo2 = genotipo2[:]

    aleatorio=random.randrange(1,len(genotipo1))


    for i in range(aleatorio,len(genotipo1)):
        hijo1[i]=genotipo2[i]
        hijo2[i]=genotipo1[i]


    hijo1=Individuo(hijo1)
    hijo2=Individuo(hijo2)

    return hijo1,hijo2
```

Esta es la implementación que hemos realizado para pasar a python el método de recombinación en un punto visto en clase.
Se le pasan por parámetros 2 individuos, y el método nos devuelve otros 2 individuos, que son los hijos resultantes de esos 2 padres.

Recombinación en dos puntos

```
def recombinacionEnDosPuntos(indiv1, indiv2):
    genotipo1 = indiv1.getLista()
    genotipo2 = indiv2.getLista()

    hijo1 = genotipo1[:]
    hijo2 = genotipo2[:]

    aleatorio1=random.randrange(1,len(genotipo1))
    aleatorio2=random.randrange(aleatorio1,len(genotipo1))

    #print('aleatorio1 = ', aleatorio1)
    #print('aleatorio2 = ', aleatorio2)

    for i in range(aleatorio1,len(genotipo1)):
        if (i>=aleatorio1 and i<=aleatorio2):
            hijo1[i]=genotipo2[i]
            hijo2[i]=genotipo1[i]


    hijo1=Individuo(hijo1)
    hijo2=Individuo(hijo2)

    return hijo1,hijo2
```

Esta es la implementación que hemos realizado para pasar a python el método de recombinación en dos puntos visto en clase.
Se le pasan por parámetros 2 individuos, y el método nos devuelve otros 2 individuos, que son los hijos resultantes de esos 2 padres.

Recombinación uniforme

```
def recombinacionUniforme(padre1, padre2,probabilidadDeMutacion):
    padre1=padre1.getLista()
    padre2=padre2.getLista()

    hijo1 = padre1[:]
    hijo2 = padre2[:]

    if(not(probabilidadDeMutacion>0 and probabilidadDeMutacion<10)):
        probabilidadDeMutacion=random.randrange(10)

    for i in range(0,len(padre1)-1):
        aleatorio2=random.randrange(10)
        if(aleatorio2>probabilidadDeMutacion):
            hijo1[i]=padre2[i]
            hijo2[i]=padre1[i]

    hijo1=Individuo(hijo1)
    hijo2=Individuo(hijo2)

    return hijo1,hijo2
```

Esta es la implementación que hemos realizado para pasar a python el método de recombinación uniforme visto en clase.
Se le pasan por parámetros 2 individuos, y el método nos devuelve otros 2 individuos, que son los hijos resultantes de esos 2 padres.

Estrategia de Selección

Para que haya una mayor intensificación hemos elegidos estas estrategias de selección:

Selección por torneo

```
def seleccionPorTorneo(listaDeIndividuos, numeroIndividuosASeleccionar, k):
    listaAuxiliar=listaDeIndividuos[:]
    listaDeSelecciados=[]
    torneo=[]
    listaFitness=[]
    individuoSeleccionado=[]
    indice=0

    if(numeroIndividuosASeleccionar >= len(listaDeIndividuos)
    or numeroIndividuosASeleccionar<=0 or numeroIndividuosASeleccionar%2==1):
        numeroIndividuosASeleccionar=int(len(listaDeIndividuos)/2)
        if(numeroIndividuosASeleccionar%2==1):
            numeroIndividuosASeleccionar-=1

    for i in range(numeroIndividuosASeleccionar):
        for j in range(k):
            num=int(len(listaAuxiliar))
            aleatorio = random.randrange(num)
            individuoSeleccionado=listaAuxiliar.pop(aleatorio)
            torneo.append(individuoSeleccionado)
            listaFitness.append(individuoSeleccionado.getCalidad())
        indice=listaFitness.index(max(listaFitness),0,k)
        individuoSeleccionado=torneo.pop(indice)
        listaAuxiliar.extend(torneo)
        del torneo[:]
        del listaFitness[:]
        listaDeSelecciados.append(individuoSeleccionado)

    return listaDeSelecciados
```

Esta es la implementación que hemos realizado para pasar a python el método de selección por torneo visto en clase.
Se le pasan por parámetros una población de individuos, el número de individuos que se quieren seleccionar, y el tipo de torneo. El método nos devuelve una lista con los individuos seleccionados.

Selección por ruleta aleatoria

```
def seleccionPorRuleta(listaDeIndividuos, numeroIndividuosASeleccionar):
    res=listaDeIndividuos[:]
    listaFitness=calcularListaFitness(listaDeIndividuos)
    listaFitnessAcumulativa=calcularListaFitnessAcumulativa(listaFitness)

    aux=[]
    indice=0
    total=0

    while(total<numeroIndividuosASeleccionar):
        r=random.randrange(listaFitnessAcumulativa[(len(listaFitnessAcumulativa)-1)]-1)
        indice=0
        for i in range(len(listaFitness)):
            if r > listaFitnessAcumulativa[i]:
                indice=indice+1
            else:
                break

        if aux.count(indice) <1:
            aux.append(indice)
            total=total+1

    for i in aux:
        res.append(listaDeIndividuos[i])

    return res
```

Esta es la implementación que hemos realizado para pasar a python el método de selección por ruleta aleatoria visto en clase.
Se le pasan por parámetros una población de individuos y el número de individuos que se quieren seleccionar. El método nos devuelve una lista con los individuos seleccionados.
Nota: En la función seleccionPorRuleta se llama a dos funciones que no tienen relevancia para la explicación.

Estrategia de reemplazo

Para que haya una mayor intensificación hemos elegidos estas estrategias de reemplazo elitista, solo nos quedamos con los mejores y descartamos los que tienen peor función de valoración.

Implementación del algoritmo genético

El funcionamiento del algoritmo es el siguiente:

1. Se inicia una población inicial. Para ello, tenemos un metodo llamado primerArranque, cuya función es cargar las pistas del archivo txt y generar padres para la primera población. Este paso solo se realiza en la primera iteración.
2. Entramos en bucle.
    a. Seleccionamos unos padres mediante las estrategias de selección para obtener sus hijos
    b. Cruzamos todos los padres entre ellos de forma al azar, usando una estrategia de cruzamiento
    c. Mutamos a los hijos generados
    d. Evaluamos a los individuos.
        i. Si hemos encontrado a un individuo que tiene fitness igual a 22 (la solución), el programa se para y devuelve la solución
        ii. Si no, evalúa al mejor individuo de esta generación y lo guarda. Lo compara con el mejor individuo que hayamos encontrado, y si es mejor que el que hemos encontrado anteriormente, lo sustituye.
        iii.
3. Devuelve el mejor individuo que hayamos encontrado.

Tutorial de funcionamiento.

Para arrancar el programa, es necesario crear una carpeta llamada “Number Mind” (respetar mayusculas y espacios). Dentro de esa carpeta, descomprimimos el contenido del archivo zip.
Una vez descomprimido, inicializamos una terminal y nos situamos en esa carpeta (mediante el uso de los comandos cd y ls).

Una vez concluido la instalación, ejecutamos en la terminal:

```
$ python nuevoMain.py
```

Una vez ejecutada, nos muestra unos mensajes y un menú.

- Si introducimos la **opción 0**, se inicia el modo de Lanzamiento Parametrizado, en el cual, tienes que introducir por consola todos los datos que el programa necesita, como la población, el número de iteraciones, etc.

Nota: El lanzador es un lanzador muy básico, y no contempla errores, por ello, en el lanzamiento parametrizado, si introduces un número de forma errónea, el programa fallará.

- Si introducimos la **opción 1**, **2**, o **3**; se ejecutan unos lanzamientos ya establecidos o predeterminados. Con estas opciones, hemos encontrado las mejores soluciones con la calidad más alta. Se encuentran detallados en el programa.

Solución Encontrada

Para encontrar la solución, realizamos diferentes ejecuciones en diferentes equipos.
Mi compañero y yo decidimos que las mejores opciones para la ejecución eran:

- Intercambio aleatorio, Recombinación en un punto y Selección por Torneo.
- Intercambio uniforme, Mutación uniforme, y Selección por ruleta.
- Intercambio aleatorio, Recombinación en un punto, y Selección por Torneo.

Esta decisión se lleva a cabo tras haber probado diferentes configuraciones, y con esas 3 obtuvimos los mejores resultados, que obtenemos códigos secreto de fitness 17.

Realizamos las 3 ejecuciones en 3 ordenadores de sobremesa con prestaciones de alta gama, y lanzamos las ejecuciones cuya duración media superaba las 6 horas.

Para ello, introdujimos una **población de 4000 individuos**, **dando 4000 iteraciones**.
Tras más de 7 horas de ejecución, en el equipo que ejecutaba la primera configuración aquí descrita, **encontró un código secreto de fitness 19**. Las otras ejecuciones sacaron un fitness 14 y 16 respectivamente.

El código secreto encontrado con fitness igual a 19 es:

[3, 1, 3, 3, 0, 8, 2, 5, 5, 0, 1, 5, 0, 8, 3, 3]