Tarea 4 Computo Evolutivo

Integrantes:
Rodríguez Miranda Alexia 316293611
Rueda Tokuhara Daiki Benedek 420003533

Nombre de equipo: Neural

Para el problema del TSP utilizamos una representación de soluciones como permutaciones.
Investiga y ejemplifica lo siguiente:

1. ¿En qué consiste el operador k-opt y cómo se podría usar como operador de mutación?

El operador k-opt es una técnica utilizada para mejorar soluciones mediante la optimización local. Consiste en eliminar hasta *k* aristas de una solución y luego reconectar los nodos de una manera diferente, intentando obtener una mejor solución (es decir, un recorrido más corto).

Por ejemplo, en el 2-opt, si tenemos un recorrido A-B-C-D, se eliminan las aristas B-C y D-A, luego se reconectan las ciudades de manera diferente, como B-D y C-A, creando un nuevo recorrido.

Al tener una solucion, cuando utilizamos un k-opt en el TSP, primero elegimos una k para cambiar ese numero de aristas, y reconectamos a los nodos involucrados de manera diferente para crear una nueva permutacion y si esta nueva solucion es mas corta, la reemplazamos o simplemente la dejamos para la nueva generacion, dependiendo de nuestro algoritmo genetico.

---

2. Si utilizamos el operador de cruza uniforme (aplicado directamente a las permutaciones), ¿Cuál podría ser una estrategia para reparar la solución?

Cuando utilizamos el operador de cruza uniforme nos puede pasar que el hijo se genere con una solucion invalida, ya que los elementos de la permutacion

pueden repetirse, entonces para solucionarlo y no quedar con el mismo padre y que la mutacion haga algo haremos:

1. Guardaremos el valor del elemento que cambiaremos y el valor del elemento a cambiar dentro de esa permutacion.
2. Si estos dos no son iguales, entonces busca en la permutacion la posicion donde este el mismo elemento que cambiaremos.
3. Intercambia estos dos valores para que sea una solucion valida y la cruza uniforme sea apropiada.

Ejemplo:
Tenemos dos padres:

- Padre 1: (A, B, C, D, E)
- Padre 2: (D, B, A, E, C)

Usamos cruza uniforme para generar un hijo:

- Hijo: (A, B, C, D, C) — Esta solución no es válida, ya que hay una repetición de la ciudad "C" y falta la ciudad "D".

Vemos que la ciudad "C" está duplicada y la ciudad "E" está ausente, entonces de nuestro padre 1, identificamos que estamos cambiando la “E” por una “C”, intercambiamos de lugar estos dos valores y reparamos al hijo.

- Hijo reparado: (A, B, E, D, C).

---

3. Investiga y propón un operador de cruza, diferente al visto en clase, para permutaciones.

El PBX (Position Based Crossover) o cruza de orden basado en posicion, es un operador que selecciona posiciones específicas en uno de los padres y copia los elementos y completa con los elementos restantes del otro padre.
Entonces:

1. Elegir posiciones específicas aleatoriamente en el primer padre.
2. Copiar los valores en esas posiciones al hijo.
3. Completar las posiciones vacías con los valores del segundo padre en el orden que aparecen.

Ejemplo:

- Padre 1: (A, B, C, D, E)

- Padre 2: (E, D, C, A, B)

Seleccionamos las posiciones 2 y 4:

- Copiamos: (-, B, -, D, -).
- Rellenamos con elementos del Padre 2: E, C, A.

Hijo: (E, B, C, D, A)

Pseudocódigo para las implementaciones del inciso 1

Selección de padres por el método de la ruleta

- poblacion: lista de individuos.
- num_seleccionados: número de padres a seleccionar

```
Seleccion_ruleta(poblacion, num_seleccionados){
    para cada individuo de la poblacion:
        fitness = fitness del individuo
        total_fitness = suma de todos los fitness
        lista probabilidades le añadimos (fitness/total_fitness)
    seleccionados = elegir el mismo numero de num_seleccionados al azar entre la lista de probabilidades
    Devolver los seleccionados
}
```

Operador de cruza de n puntos

```
Cruza_n_puntos(padre1, padre2, n_puntos){
    Si los dos padres no tienen la misma longitud:
        error

    longitud = obtener dimensiones de ambos padres
    puntos_cruce = ordenar la lista con un numero random entre la longitud y n_puntos
```

```
    hijo1 = padre1
    hijo2 = padre2

    for de i hasta longitud de puntos_cruce
        if el indice es par
            inicio = puntos_cruce[i]
            fin = puntos_cruce[i+1]

        else
            intercambia los hijos del padre1 y padre2
    return hijo1 e hijo2
}
```

Mutación flip

```
Mutacion_flip(individuo, tasa_m){
    copia_individuo = individuo
    for i hasta longitud(individuo)
        if numero_random < tasa_m
            if individuo[i] == 0:
                copia_individuo[i] = 1
            else:
                0
    return copia_individuo
}
```

Reemplazo generacional con elitismo

```
seleccionar_elitistas(poblacion, num_elitistas){
    for cada individuo en poblacion:
```

```
        fit = funcion fitness del individuo
        fitness = todas las fit

    individuos_con_fitness = crear lista de tuplas de la poblacion con el fitness
    total
    ordenar individuos_con_fitness en orden descendente

    declarar lista elite
    for individuo en individuos_con_fitness hasta num_elitistas
        elite = añadir el individuo

    return elite
}
```