Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Temas Selectos de Optimización

<u>Implementación vecino más cercano</u>

Alumno: Michael Antonio Noguera Guzmán

Matrícula: 2077402

Catedrático: Iris Abril Martínez Salazar

Semestre: E2023

Grupo: 001

Fecha: 24/04/2023

Introducción

El problema del viajero es un problema bastante sencillo, y el desarrollo de heurísticas para la solución de este problema es en realidad demasiado amplio, por lo que existe muchos métodos interesantes, en este caso se nos da como base el método de solución del **vecino mas cercano**, y se nos ha dado la tarea de encontrar algún tipo de mejora, particularmente. Este problema se ha abordado por una gran cantidad de tiempo, sin embargo, resulta interesante observar de que forma se podría modificar y mejorar el algoritmo de resolución.

Descripción del problema

Este problema es NP-hard, lo que significa que no existe un algoritmo eficiente para resolverlo exactamente para grandes instancias del problema.

Existen varias formas de generar mejoras, en particular a mí me ha interesado generar un poco de exploración en las soluciones, pues sabemos que de base este algoritmo es completamente *greedy*, simplemente va y escoge la mejor opción.

La estrategia de mejor que propongo es generar un poco mas de exploración, que es generar una ruleta en donde todos los nodos tienen un conjunto de probabilidades, estas probabilidades dependerán de la aptitud, en este caso podríamos pensar que las aptitudes entre mas grandes son mas posibles de selección, sin embargo nosotros queremos que entre mas chico sea el costo, esto lo logramos simplemente dividiendo sobre 1, si dividimos sobre 1, entonces entre mas chico es el valor mas grande es el resultado de la división, así podemos lograr lo que buscábamos.

Y posteriormente se genera una lista, en la cual los distintos elementos se irán sumando, es decir, la lista sumara todas las probabilidades hasta tener 1 al final (recordando que en una ruleta la suma de probabilidades da 100%).

Para lograr esto debemos de tener en cuenta algunas cosas:

» Los nodos que ya estén en el recorrido tienen probabilidad de selección de 0.

» Estos nodos ya están cubiertos los nodos en los que i=j.

» La "ruleta" es un numero entre 0 y 1, en el valor del arreglo, en el que el valor de la ruleta sea menor a la probabilidad acumulada este será el valor del nodo seleccionado.

Descripción del algoritmo

Este algoritmo consiste en varias partes importantes.

1.- Lectura del archivo de los nodos: Es necesario generar un archivo que sea capaz de interpretarse por el algoritmo, por lo que este paso se encarga de pasar del archivo de texto a una matriz.

2.- Generación de la matriz de distancias: Es más sencillo de comprender el código y evitar errores si aprovechamos para generar toda la matriz desde un principio, pues así podemos generar una matriz que simplemente se pase de un método a otro.

3.- Selección de un nodo arbitrario: Este método de resolución inicia con la selección arbitraria de un nodo para arrancar la ruta.

4.- Obtención de la aptitud total de todos los nodos: Como se mencionó en la descripción, este método genera un poco más de exploración en las soluciones, esto lo genera en base de una ruleta, y para poder generar dicha ruleta nosotros debemos de obtener la aptitud de todos los nodos.

5.- Obtención de la lista aptitud individual (ruleta): Es necesario generar la ruleta, esta se tiene que generar respetando unas ligeras normas, siendo estas, la primera: los nodos i = j, se tendrán que ignorar para la asignación de aptitud, en este caso decidí que, para evitar errores de distancias, implemente números negativos, y sabemos que este número es imposible de generar en el cálculo de distancias.

6.- Selección del nodo por método de ruleta: Este paso es más sencillo, simplemente lo único que tenemos que hacer es tirar un número al azar, y este número será el encargado de determinar que nodo es el que será seleccionado, este número simplemente será, en el nodo en el que el valor acumulado sea mayor que el número al azar entonces este será el nodo por añadir.

7.- Adición del nodo: Nada que explicar, se añade el nodo a la ruta.

8.- Paso 4, hasta que la ruta sea igual a k=n: hasta que todos los nodos se añadan a la ruta repetimos el paso 4.

9.- Añadir el nodo de inicio: añadir el nodo con el que se inició para poder finalizar.

Implementación

Pondré directamente el código, explicarlo es redundante (A mí opinión).

1.- Lectura del archivo de los nodos:

```
//Lectura de la matriz de nodos
ArrayList<ArrayList<Integer>> matrizObtenida = generarArreglo("5nodes");
```

```
static ArrayList<ArrayList<Integer>> generarArreglo(String archivoObjetivo) {
    ArrayList<ArrayList<Integer>> nodos = new ArrayList<ArrayList<Integer>>();

    String direccion = "C:\\Users\\Antonio Noguera\\workspace-Java Files\\javaFiles\\sr
    File archivo = new File(direccion);

    int lineaEfectiva = 0;

    try {
        Scanner lector = new Scanner(archivo);

        while(lector.hasNextLine()) {
            String linea = lector.nextLine();

            if(lineaEfectiva>2) {

                ArrayList<Integer> fila = new ArrayList<Integer>();

                String[] lineaEncontrada = linea.split(" ");

                for(int i=0;i<lineaEncontrada.length;i++) {
                    fila.add(Integer.parseInt(lineaEncontrada[i]));
                }

                nodos.add(fila);

            }

            lineaEfectiva++;

        }
        lector.close();
    }catch(FileNotFoundException e) {
        System.out.println("Error a la hora de encontrar el archivo");
        e.printStackTrace();
    }

    return nodos;
}
```

2.- Generación de la matriz de distancias:

```
//Generacion de la matriz de distancias
ArrayList<ArrayList<Double>> matrizDeDistancias = generarMatrizDistancia(matrizObtenida);
```

```
static ArrayList<ArrayList<Double>> generarMatrizDistancia(ArrayList<ArrayList<Integer>> nodosMatriz){

    //referentes a la matriz a crear
    ArrayList<ArrayList<Double>> matrizD = new ArrayList<ArrayList<Double>>();

    for (int i = 0; i < nodosMatriz.size(); i++) {

        ArrayList<Integer> nodoA = nodosMatriz.get(i);

        //Fila de la matriz a crear
        ArrayList<Double> fila = new ArrayList<Double>();

        for (int j = 0; j < nodosMatriz.size(); j++) {

            ArrayList<Integer> nodoB = nodosMatriz.get(j);
            if(i!=j) {
                fila.add(distanciaEntreNodos(nodoA,nodoB));
            }else {
                fila.add(-4.2);
            }

        }
        matrizD.add(fila);
    }

    return matrizD;
}
```

Este código requiere a la función (distanciaEntreNodos):

```
 static double distanciaEntreNodos(ArrayList<Integer> nodoA, ArrayList<Integer> nodoB) {
     double distancia=0;

     for(int i=0;i<nodoA.size();i++){    distancia+= Math.pow(nodoA.get(i)-nodoB.get(i),2);  }

     return Math.sqrt(distancia);
 }
```

3.- Selección de un nodo arbitrario:

```
//Gereneación de la ruta
ArrayList<Integer> rutaCreada = new ArrayList<Integer>();

//Seleccion del nodo al azar
rutaCreada.add((int)(Math.round((Math.random()*(matrizDeDistancias.size()-1)))));
```

Este es el método que contiene los siguientes pasos

```java
static int seleccionVecino(ArrayList<Integer> rutaParcial, ArrayList<ArrayList<Double>> matrizDistanc

//Linea que va a seleccionar todo el nodo de la matriz
ArrayList<Double> distanciasDelUltimoNodo = matrizDistancias.get(rutaParcial.get(rutaParcial.size()-1
System.out.println("\nNodo Base: "+(rutaParcial.get(rutaParcial.size()-1)));

//Linea que despliega la funcion de limpieza del nodo(eliminara los nodos que ya estan en la subruta)
ArrayList<Double> listaLimpia = limpiezaDeAspirantes(distanciasDelUltimoNodo,rutaParcial);

//For que va a seleccionar el nodo mas cercano.
int nodoSeleccionado=0;

//Generacion de ruleta

ArrayList<Double> listaAptitudes= new ArrayList<Double>();


//Obtencion aptitud Total
double aptitudTotal = 0;

for(int i=0;i<listaLimpia.size();i++) {
    if(listaLimpia.get(i)>0){
        aptitudTotal+=(1/listaLimpia.get(i));
    }
}

//Generador de Ruleta
double acumAptitudes=0;

for(int i=0;i<listaLimpia.size();i++) {
    if(listaLimpia.get(i)>0) {
        acumAptitudes+=((1/listaLimpia.get(i))/aptitudTotal);
        System.out.printf("Al nodo %d = %.4f\n",i,(1/listaLimpia.get(i))/aptitudTotal);
        listaAptitudes.add(acumAptitudes);
    }else {
        listaAptitudes.add(0.0);
    }
}

double tiroRuleta=Math.random();
System.out.println("Lista de Aptitudes Acum: "+listaAptitudes);

for(int i=0;i<listaLimpia.size();i++) {
    if(tiroRuleta<=listaLimpia.get(i)) {
        nodoSeleccionado=i;
        break;
    }
}

return nodoSeleccionado;
```

4.- Obtención de la aptitud total de todos los nodos:

```
//Linea que va a seleccionar todo el nodo de la matriz
ArrayList<Double> distanciasDelUltimoNodo = matrizDistancias.get(rutaParcial.get(rutaParcial.size()-1));
System.out.println("\nNodo Base: "+(rutaParcial.get(rutaParcial.size()-1)));

//Linea que despliega la funcion de limpieza del nodo(eliminara los nodos que ya estan en la subruta)
ArrayList<Double> listaLimpia = limpiezaDeAspirantes(distanciasDelUltimoNodo,rutaParcial);
```

```
//Obtencion aptitud Total
double aptitudTotal = 0;

for(int i=0;i<listaLimpia.size();i++) {
    if(listaLimpia.get(i)>0){
        aptitudTotal+=(1/listaLimpia.get(i));
    }
}
```

5.- Obtención de la lista aptitud individual (ruleta):

```
//Generador de Ruleta
double acumAptitudes=0;

for(int i=0;i<listaLimpia.size();i++) {
    if(listaLimpia.get(i)>0) {
        acumAptitudes+=((1/listaLimpia.get(i))/aptitudTotal);
        System.out.printf("Al nodo %d = %.4f\n",i,(1/listaLimpia.get(i))/aptitudTotal);
        listaAptitudes.add(acumAptitudes);
    }else {
        listaAptitudes.add(0.0);
    }
}

System.out.println("Lista de Aptitudes Acum: "+listaAptitudes);
```

6.- Selección del nodo por método de ruleta:

```
double tiroRuleta=Math.random();
System.out.println("Valor Obtenido por ruleta: "+tiroRuleta);
System.out.println("Lista de Aptitudes Acum: "+listaAptitudes);

for(int i=0;i<listaLimpia.size();i++) {
    if(tiroRuleta<=listaAptitudes.get(i)) {

        nodoSeleccionado=i;
        break;
    }
}
System.out.println("Nodo seleccionado = "+ nodoSeleccionado);
return nodoSeleccionado;
```

El anterior proceso llama al método limpieza de aspirantes

```
static ArrayList<Double> limpiezaDeAspirantes(ArrayList<Double> listaSucia, ArrayList<Integer> subRuta){

    for(int i=0;i<listaSucia.size();i++) {
        if(subRuta.contains(i)){
            listaSucia.set(i,-4.2);
        }
    }

    return listaSucia;
}
```

7.- Adición del nodo: Nada que explicar, se añade el nodo a la ruta.

8.- Paso 4, hasta que la ruta sea igual a k=n: hasta que todos los nodos se añadan a la ruta repetimos el paso 4.

```
//Seleccion del mas cercano
do {
    rutaCreada.add(seleccionVecino(rutaCreada,matrizDeDistancias));
}while(rutaCreada.size() < matrizDeDistancias.size());
```

9.- Añadir el nodo de inicio: añadir el nodo con el que se inició para poder finalizar.

```
rutaCreada.add(rutaCreada.get(0));
```

Extra-Impresión de la ruta y su costo:

```
//Impresión de la ruta creada

System.out.println("\n\nRuta Obtenida = "+rutaCreada);

obtencionCosto(rutaCreada,matrizDeDistancias);
```

```
static double obtencionCosto(ArrayList<Integer> recorrido, ArrayList<ArrayList<Double>> matrizDistancias) {

    double costo = 0;
    System.out.println("\n\nCosto de la ruta Obtenida:");
    for(int i=0;i<(recorrido.size()-1);i++) {
        ArrayList<Double> nodoInicio = matrizDistancias.get(recorrido.get(i));
        costo += nodoInicio.get(recorrido.get(i+1));
        System.out.printf("%.4f + ",nodoInicio.get(recorrido.get(i+1)));
    }
    System.out.print("= "+costo);

    return costo;
}
```

Presentación de resultados

```
Nodos Empleados:
0.-[1, 12, 10]
1.-[2, 5, 14]
2.-[3, 9, 11]
3.-[4, 7, 5]
4.-[5, 2, 12]

Distancias entre los nodos:
0.-[-4.2, 8.12403840463596, 3.7416573867739413, 7.681145747868608, 10.954451150103322]
1.-[8.12403840463596, -4.2, 5.0990195135927845, 9.433981132056603, 4.69041575982343]
2.-[3.7416573867739413, 5.0990195135927845, -4.2, 6.4031242374328485, 7.3484692283495345]
3.-[7.681145747868608, 9.433981132056603, 6.4031242374328485, -4.2, 8.660254037844387]
4.-[10.954451150103322, 4.69041575982343, 7.3484692283495345, 8.660254037844387, -4.2]

Nodo Base: 1
Al nodo 0 = 0.1928
Al nodo 2 = 0.3072
Al nodo 3 = 0.1660
Al nodo 4 = 0.3340
Valor Obtenido por ruleta: 0.10566247024380215
Lista de Aptitudes Acum: [0.19281003029830843, 0.0, 0.5000055777495073, 0.6660432313144352, 1.0]
Nodo seleccionado = 0

Nodo Base: 0
Al nodo 2 = 0.5468
Al nodo 3 = 0.2664
Al nodo 4 = 0.1868
Valor Obtenido por ruleta: 0.509424230857507
Lista de Aptitudes Acum: [0.0, 0.0, 0.5468403404963733, 0.8132184651364361, 1.0]
Nodo seleccionado = 2

Nodo Base: 2
Al nodo 3 = 0.5344
Al nodo 4 = 0.4656
Valor Obtenido por ruleta: 0.8291361890793227
Lista de Aptitudes Acum: [0.0, 0.0, 0.0, 0.5343721981481256, 0.9999999999999999]
Nodo seleccionado = 4

Nodo Base: 4
Al nodo 3 = 1.0000
Valor Obtenido por ruleta: 0.42566589652055076
Lista de Aptitudes Acum: [0.0, 0.0, 0.0, 1.0, 0.0]
Nodo seleccionado = 3

Ruta Obtenida = [1, 0, 2, 4, 3, 1]

Costo de la ruta Obtenida:
8.1240 + 3.7417 + 7.3485 + 8.6603 + 9.4340 + = 37.308400189660425
```

Por fines de espacio presentare únicamente el archivo de 5 nodos.

Conclusiones

En lo personal para realizar este proyecto lo que terminé desarrollando fue una modificación del método propuesto, y para poder lograr esto tuve que desarrollar el vecino más cercano primero, y este método es peculiar. Pues al final de cuentas siempre vamos a tener solo la misma cantidad de soluciones que nodos, pues siempre va a tomar el menor, siempre lo mismo, por lo que básicamente lo único que este algoritmo nos muestra es siempre la misma ruta, es decir, en términos de exploración, no hay exploración. Por lo que en lo personal no le veo como un método viable de solución.

Es por ello por lo que mi mejora propone algo bastante sencillo, introduce el concepto de exploración y en lo personal observado, podemos apreciar una mejora constante de las soluciones, pase por alto capturar los costos de la solución, sin embargo, la solución estándar dependía siempre del primer nodo, y del siguiente método tenemos un poco mas de variaciones, y tenemos un mejor resultado promedio, es decir tenemos un costo estándar.

Referencias

Reinelt, G. (2003). The traveling salesman: computational solutions for TSP applications (Vol. 840). Springer.

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan y D. B. Shmoys, editors. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester, 1985.

M. R. Garey y D. S. Johnson. Computers and Intractability: A Guide to the Theory of NPCompleteness. Freeman, New York, 1979.

Optima, 58, Mathematical Programming Society Newsletter, Junio 1998.