Universidad de Granada
E.T.S.I Informática y Telecomunicación

UNIVERSIDAD DE GRANADA

Práctica 1.a:
Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema de la Máxima diversidad

Problema escogido: Problema de la máxima diversidad.
Algoritmos considerados: de búsqueda local y Greedy.

José Manuel Rodríguez Calvo
DNI: 77147756Y
jmrodriguez@correo.ugr.es
Grupo:
Horario: Jueves de 17:30 a 19:30

Curso 2019 - 2020
Tercer Curso del Grado en Ingeniería Informática

Índice

Descripción del problema

El problema sobre el que trata este documento es el de la máxima diversidad. Este problema es definido como un problema de optimización combinatoria, en el que dado un conjunto de datos, un contexto del problema, algunas restricciones y sobre cierta regla como por ejemplo máximo, mínimo, media,... tiene que llegar al mejor valor bajo ese espacio de trabajo. En particular el problema sobre el que se va a trabajar se define a continuación siguiendo el esquema descrito que se detalla anterior:

Como conjunto de datos, se dispone de 30 archivos de prueba sobre el que testeamos nuestro algoritmo, estos datos se representan en una matriz simétrica de pesos sobre cierto punto (x, y).
El contexto sobre el que se trabaja es que los datos de la matriz son los pesos, en este caso las distancias que existe entre el elemento (x) y elemento (y).
La restricción más evidente sobre este problema es que no se puede repetir elementos seleccionados.
La regla en la que se define este problema es la de maximizar, por tanto se busca maximizar las distancias entre los puntos.

La solución al problema es un subconjunto de elementos que serán seleccionados bajo un orden específico.

Descripción de la solución

Dado que el archivo proporcionado viene dado en un formato que no da mucha información para poder realizar las operaciones, se procede a analizar dichos datos y estructurarlos para su posterior utilización. Este estructuración se lleva a cabo al principio y da como resultado dos datos:

Los datos que se obteniente de este archivo es el número de elementos que se tienen como solución y un matriz simétrica que como se aclara en el punto anterior es una matriz con las distancias entre los puntos dados.

0	2	3	5
2	0	2	3
3	2	0	2
5	3	2	0

Por tanto por ejemplo entre el punto [0] y el punto [3] la distancia es de 5 y entre el punto [3] y [0] la distancia continúa siendo 5. Para la simetría y con conocimiento del problema la distancia entre un mismo punto es igual a 0.

Otras de las funciones complementarias aunque necesarias para el correcto funcionamiento, es la función objetiva que dada una solución representada como un conjunto de elementos con un orden, pertenecientes a los posibles elementos de la matriz de datos.

Esta función va recorriendo la solución, y a la vez manteniendo el valor anterior posible y consulta a la matriz con dichos valores sus distancias y las va acumulando:

```
beforePosition=sum=0
for actualPosition in (0,..,sizeOf(Solution)
        sum+=matrix[solution[beforePosition]][solution[actualPosition]]
        beforePosition=actualPosition
end
```

Para utilizar este código correctamente se basa en que la distancia entre sí mismo es 0 y por consiguiente en la primera interacción, resultará la distancia 0.

Otras de las funciones importantes para el correcto funcionamiento del software es la función de bloquear Elementos en la matriz, su utilización es cuando se escoge un elementos posteriormente ese elemento debe ser bloqueado de la matriz para que no sea elegido posteriormente. Para esto la columna es puesta a 0.

```
for element in forbidenElements:
        matrix[:][element]=0
end
```

Como solución que se devuelve en ambos ejemplos es un conjunto de elementos seleccionados bajo un orden dado de selección.

Descripción de la solución del método de búsqueda

La función principal sobre la que se basa la solución es la encargada de recuperar la mejor solución alternativa y si es mejor que la suma de las distancias para ese intervalo, cambia la solución inicial, para finalmente retornar dicha solución

```
createFirstSolution
for i=0; i< FirstSolution.len; i++
        (weight, BestwayAlternative)=createAlternativeBestSolution()
        if weight > objetive(FirstSolution[i:])
           FirstSolution=swap(FirstSolution[i:], BestwayAlternative)
end
```

Para crear la mejor alternativa a dicha solucion, usamos la siguiente funcion:

createAlternativeBestSolution

```
neighboursDistances, neighboursWay=calculateNeigbours()

maximDistance=argmax(neighboursDistances)

return
neighboursDistances(maximDistance),neighboursWay(maximDistance)
```

calculateNeigbours

```
createMatrix(matrix, preselected)
for index in matrix.row.length-2
  if index not in preselected:
    way[actualpos]=greedy(matrix,index)
   way[actualpos].appendTop(initialvalue)
   sumdistances[actualpos]=objetive( way[actualpos])
```

```
    preselected.push(index)

    COUNTOBJETIVE++
  end

end
```

En la función “calculateNeighbours” el bucle es hasta matrix.row.length-2 debido a que el último elemento seleccionado es igual al seleccionado por el algoritmo greedy y por tanto el último sobre el que se puede calcular la suma de distancias es el penúltimo elemento.

Los elemento preseleccionados son elementos prohibidos, para que no se repita, se bloquean dichos elementos en la matriz, para ello se utiliza la función auxiliar mencionada en el apartado anterior.

Descripción de la solución del método Greedy

Para el algoritmo como idea inicial se tenía pensado utilizar la función de bloquear matriz aunque, al realizar varias pruebas esta resultó poco eficiente y por tanto opte por el código que se detalla a continuación:

```
while no_solutions != 1:
        bestSolution=calculatenext()
        solution.push(bestSolution)
        no_solutions++
end
```

calculatenext

```
maximSolution=argmax(matrix[row_selected])
if maximSolution in solution:
   return calculatenext()
else:
    return calculatenext
```

Procedimientos

El código ha sido diseñado a partir de las transparencias de la asignatura, utilizando python v 3.8.2 y para el cálculo matemático de las matrices he usado numpy.
Para lanzar el programa se puede lanzar todos los ejemplos con el archivo core.py <SEED_START> donde seed_start es la semilla para inicializar el programa y poner el primer elemento a seleccionar, que será el primer elemento de la solución.
Sin embargo si la intención es lanzar uno de los será necesario con especificar el archivo (best_first.py) <SEED_START> <FILENAME> donde seed_start es analogamente igual como core.py y el fichero del archivo es la ruta relativa.
El código se puede encontrar también en el siguiente enlace.

Experimentos y Análisis

Datos

Algoritmo Greedy			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
GKD-c_11_n500_m50	790,4248	95,96	0,00
GKD-c_12_n500_m50	756,6337	96,09	0,00
GKD-c_13_n500_m50	396,9346	97,95	0,00
GKD-c_14_n500_m50	773,6527	96,02	0,00
GKD-c_15_n500_m50	744,8793	96,16	0,00
GKD-c_16_n500_m50	736,9383	96,26	0,00
GKD-c_17_n500_m50	757,6833	96,08	0,00
GKD-c_18_n500_m50	767,9928	96,05	0,00
GKD-c_19_n500_m50	763,3498	96,08	0,00
GKD-c_20_n500_m50	741,4325	96,22	0,00
MDG-b_1_n500_m50	22887,8800	97,06	0,00
MDG-b_2_n500_m50	23952,1400	96,93	0,00
MDG-b_3_n500_m50	23762,4300	96,94	0,00
MDG-b_4_n500_m50	21535,6000	97,22	0,00
MDG-b_5_n500_m50	24889,8800	96,79	0,00
MDG-b_6_n500_m50	23466,5100	96,97	0,00
MDG-b_7_n500_m50	22082,8400	97,16	0,00
MDG-b_8_n500_m50	23823,9500	96,94	0,00
MDG-b_9_n500_m50	29054,6600	96,25	0,00
MDG-b_10_n500_m50	24766,0500	96,80	0,00
MDG-a_31_n2000_m200	997	99,13	0,00
MDG-a_32_n2000_m200	926	99,19	0,00
MDG-a_33_n2000_m200	951	99,17	0,00
MDG-a_34_n2000_m200	981	99,14	0,00
MDG-a_35_n2000_m200	1023	99,10	0,00
MDG-a_36_n2000_m200	1019	99,11	0,00
MDG-a_37_n2000_m200	988	99,13	0,00
MDG-a_38_n2000_m200	1001	99,12	0,04
MDG-a_39_n2000_m200	1009	99,12	0,03
MDG-a_40_n2000_m200	975	99,15	0,03

Media Desv:	**97,44**
Media Tiempo:	**0,00**

Algoritmo BL			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
GKD-c_11_n500_m50	1023,0001	94,78	624,52
GKD-c_12_n500_m50	1002,3052	94,82	616,51
GKD-c_13_n500_m50	1005,8187	94,81	607,85
GKD-c_14_n500_m50	1021,8649	94,75	609,53
GKD-c_15_n500_m50	1015,5247	94,77	612,24
GKD-c_16_n500_m50	1017,1518	94,83	841,27
GKD-c_17_n500_m50	1012,5538	94,76	849,26
GKD-c_18_n500_m50	1007,5857	94,82	660,99
GKD-c_19_n500_m50	1022,5134	94,75	655,00
GKD-c_20_n500_m50	1015,1053	94,82	669,27
MDG-b_1_n500_m50	46412,7200	94,03	9,47
MDG-b_2_n500_m50	39260,0000	94,97	10,03
MDG-b_3_n500_m50	36370,7800	95,32	8,24
MDG-b_4_n500_m50	44959,0400	94,20	8,33
MDG-b_5_n500_m50	44466,9100	94,27	7,60
MDG-b_6_n500_m50	42352,3200	94,54	7,66
MDG-b_7_n500_m50	42779,9900	94,50	7,50
MDG-b_8_n500_m50	44177,5500	94,33	7,41
MDG-b_9_n500_m50	41978,6400	94,58	7,49
MDG-b_10_n500_m50	45178,6000	94,17	7,40
MDG-a_31_n2000_m200	1259	98,90	1797,45
MDG-a_32_n2000_m200	1303	98,86	1679,58
MDG-a_33_n2000_m200	1233	98,92	1883,93
MDG-a_34_n2000_m200	1278	98,88	1669,07
MDG-a_35_n2000_m200	1298	98,86	1704,15
MDG-a_36_n2000_m200	1293	98,87	1760,74
MDG-a_37_n2000_m200	1126	99,01	1913,66
MDG-a_38_n2000_m200	1250	98,91	1782,20
MDG-a_39_n2000_m200	1301	98,86	1776,36
MDG-a_40_n2000_m200	1220	98,93	1907,84

Media Desv:	**96,06**
Media Tiempo:	**823,42**

Análisis

Como análisis posterior, se puede indicar que el tiempo obtenido para el algoritmo greedy es muy bajo mientras que el algoritmo de búsqueda local, al tener que ir comprobando y reconstruyendo la solución para obtener la suma es muy lento. Esto sería aceptable si la solución fuese mucho mejor aunque solo consigue un promedio de 200 más que el algoritmo greedy. Por tanto bajo estos valores obtenidos, y en un entorno profesional me decantaría por un algoritmo greedy cuando el coste de la cpu es elevado y el número de valores es muy alto, en cambio bajo un entorno donde se requiera precisión y el tiempo no sea un factor tan importante el algoritmo de búsqueda local ofrece mejor resultado.