Práctica 1

Metaheurísticas

Amin Kasrou Aouam

UNIVERSIDAD
DE GRANADA

2021-04-19

Índice

Práctica 1 **3**
Introducción 3
Algoritmos 3
Greedy 3
Búsqueda local 3
Implementación 4
Instalación 4
Ejecución 4
Análisis de los resultados 5
Algoritmo greedy 5
Algoritmo de búsqueda local 6

Práctica 1

Introducción

En esta práctica, usaremos distintos algoritmos de búsqueda para resolver el problema de la máxima diversidad (MDP). Implementaremos:

- Algoritmo *Greedy*
- Algoritmo de búsqueda local

Algoritmos

Greedy

El algoritmo *greedy* añade de forma iterativa un punto, hasta conseguir una solución de tamaño m.

En primer lugar, seleccionamos el elemento más lejano de los demás (centroide), y lo añadimos en nuestro conjunto de elementos seleccionados. A éste, añadiremos en cada paso el elemento correspondiente según la medida del *MaxMin*. Ilustramos el algoritmo a continuación:

Input: A list $[a_i]$, $i = 1, 2, \cdots, m$, that contains the chosen point and the distance
Output: Processed list

```
1  Sel = [ ]
2  centroid ← getFurthestElement()
3  for i ← 0 to m do
4      for element in Sel do
5          closestElements = [ ]
6          closestPoint ← getClosestPoint(element)
7          closestElements.append(closestPoint)
8      end for
9      maximum ← max(closestElements)
10     Sel.append(maximum)
11 end for
12 return Sel
```

Búsqueda local

El algoritmo de búsqueda local selecciona una solución aleatoria, de tamaño *m*, y explora durante un número máximo de iteraciones soluciones vecinas.

Para mejorar la eficiencia del algoritmo, usamos la heurística del primer mejor (selección de la primera solución vecina que mejora la actual). Ilustramos el algoritmo a continuación:

Input: A list $[a_i]$, $i = 1, 2, \cdots, m$, the solution
Output: Processed list

$Solutions = [\,]$
$firstSolution \leftarrow getRandomSolution()$
$Solutions.append(firstSolution)$
$lastSolution \leftarrow getLastElement(neighbour)$
$maxIterations \leftarrow 1000$
for $i \leftarrow 0$ **to** $maxIterations$ **do**
  while $neighbour \leq lastSolution$ **do**
    $neighbour \leftarrow getNeighbouringSolution(lastSolution)$
    $Solutions.append(neighbour)$
    $lastSolution \leftarrow getLastElement(neighbour)$
  end while
  $finalSolution \leftarrow getLastElement(Solutions)$
end for
return $finalSolution$

Implementación

La práctica ha sido implementada en *Python*, usando las siguientes bibliotecas:

- NumPy
- Pandas

Instalación

Para ejecutar el programa es preciso instalar Python, junto con las bibliotecas **Pandas** y **NumPy**.

Se proporciona el archivo shell.nix para facilitar la instalación de las dependencias, con el gestor de paquetes Nix. Tras instalar la herramienta Nix, únicamente habría que ejecutar el siguiente comando en la raíz del proyecto:

```
nix-shell
```

Ejecución

La ejecución del programa se realiza mediante el siguiente comando:

```
python src/main.py <dataset> <algoritmo>
```

Los parámetros posibles son:

dataset	algoritmo
Cualquier archivo de la carpeta data	greedy
	local

También se proporciona un script que ejecuta 1 iteración del algoritmo greedy y 3 iteraciones de la búsqueda local, con cada uno de los *datasets*, y guarda los resultados en una hoja de cálculo. Se puede ejecutar mediante el siguiente comando:

```
python src/execution.py
```

Nota: se precisa instalar la biblioteca XlsxWriter para la exportación de los resultados a un archivo Excel.

Análisis de los resultados

Los resultados obtenidos se encuentran en el archivo *algorithm-results.xlsx*, procedemos a analizar cada algoritmo por separado.

Algoritmo greedy

dataset	desviacion distancia	desviacion tiempo	media distancia	media tiempo
GKD-c_17_n500_m50.txt	0	0	12682.33935	10.4774901866913
MDG-a_31_n2000_m200.txt	0	0	99603	414.753985881805
GKD-c_16_n500_m50.txt	0	0	13179.90735	8.83024668693543
GKD-c_14_n500_m50.txt	0	0	12537.03681	9.76281476020813
GKD-c_15_n500_m50.txt	0	0	12292.98008	9.29908132553101
MDG-a_36_n2000_m200.txt	0	0	98029	357.657149553299
MDG-a_40_n2000_m200.txt	0	0	98648	331.03901219368
MDG-b_10_n500_m50.txt	0	0	589363.679999999	7.89755153656006
GKD-c_11_n500_m50.txt	0	0	12519.44959	7.93594408035278
GKD-c_13_n500_m50.txt	0	0	12298.99796	7.94760537147522
MDG-a_38_n2000_m200.txt	0	0	98389	301.782891273498
GKD-c_12_n500_m50.txt	0	0	13082.06009	7.94936108589172
GKD-c_19_n500_m50.txt	0	0	12289.00789	7.87309288978577
GKD-c_18_n500_m50.txt	0	0	12609.24814	7.86540246009827
MDG-a_34_n2000_m200.txt	0	0	98526	325.084084033966
MDG-a_33_n2000_m200.txt	0	0	98648	745.335252761841
MDG-b_8_n500_m50.txt	0	0	598965.659999999	7.83168935775757
MDG-b_9_n500_m50.txt	0	0	585344.69	7.8765070438385
GKD-c_20_n500_m50.txt	0	0	12871.9537	7.83133149147034
MDG-a_35_n2000_m200.txt	0	0	99456	311.641102790833
MDG-a_32_n2000_m200.txt	0	0	99205	320.019157648086
MDG-b_6_n500_m50.txt	0	0	600801.29	7.85437417030335
MDG-b_7_n500_m50.txt	0	0	585275.39	7.8808856010437
MDG-a_39_n2000_m200.txt	0	0	98190	328.25626206398
MDG-b_5_n500_m50.txt	0	0	585517.960000001	7.86951899528503
MDG-b_4_n500_m50.txt	0	0	604432.120000001	7.81630039215088
MDG-b_1_n500_m50.txt	0	0	589762.07	8.02232670783997
MDG-b_2_n500_m50.txt	0	0	606303.950000001	7.83004355430603
MDG-b_3_n500_m50.txt	0	0	590160.140000001	7.85855078697205
MDG-a_37_n2000_m200.txt	0	0	98453	324.734681606293

Figura 1: Algoritmo greedy

El algoritmo greedy es determinista, por lo tanto la desviación típica es nula, dado que se ejecuta una única vez. El tiempo de ejecución varía considerablemente según el dataset:

- Dataset con n=500: 7-10 segundos
- Dataset con n=2000: 5-12 minutos

La distancia total obtenida, por lo general, es inferior al algoritmo de búsqueda local, aunque no difiere significativamente.

Algoritmo de búsqueda local

dataset	desviacion distancia	desviacion tiempo	media distancia	media tiempo
GKD-c_17_n500_m50.txt	0	0.405745641729275	17788.52922	58.3955753644308
MDG-a_31_n2000_m200.txt	0	41.7978878608236	103523	1356.22949878375
GKD-c_16_n500_m50.txt	0	1.21949469887983	18344.53003	62.8311138153076
GKD-c_14_n500_m50.txt	0	1.77366325104033	18286.45155	62.2634030977885
GKD-c_15_n500_m50.txt	0	1.10164954291249	18092.02378	55.3233214219411
MDG-a_36_n2000_m200.txt	0	82.0373434238616	102691	1241.96534363429
MDG-a_40_n2000_m200.txt	0	11.3801980103746	102898	1237.06790685654
MDG-b_10_n500_m50.txt	0	0.327577234163752	726639.72	210.882393519084
GKD-c_11_n500_m50.txt	0	1.11313891598922	18432.67663	53.3401915232341
GKD-c_13_n500_m50.txt	0	1.07278728990798	17630.96358	44.0156571865082
MDG-a_38_n2000_m200.txt	0	40.9004163332626	103784	1260.28208820025
GKD-c_12_n500_m50.txt	0	0.0308785249015486	17929.50412	50.0223886966705
GKD-c_19_n500_m50.txt	0	0.0730988883717835	18602.8009	59.221225976944
GKD-c_18_n500_m50.txt	0	0.501005792543248	18036.65272	44.6869557698568
MDG-a_34_n2000_m200.txt	0	227.566678765529	104123	1554.23210906982
MDG-a_33_n2000_m200.txt	0	66.3584219906316	103376	1312.43037589391
MDG-b_8_n500_m50.txt	0	0.151493606078811	692479.61	96.0361976623535
MDG-b_9_n500_m50.txt	0	0.113454756390407	688141.110000001	85.6819485823313
GKD-c_20_n500_m50.txt	0	0.0225695163194442	18400.84123	54.0921149253845
MDG-a_35_n2000_m200.txt	0	20.8655615864072	103407	1227.76472640037
MDG-a_32_n2000_m200.txt	0	17.7312685161874	103034	1214.80604545275
MDG-b_6_n500_m50.txt	0	0.096552827775804	684526.73	90.4683658281962
MDG-b_7_n500_m50.txt	0	0.175125610317121	688045.889999999	76.1629154682159
MDG-a_39_n2000_m200.txt	0	10.5258429090711	103179	1203.75519879659
MDG-b_5_n500_m50.txt	0	0.416599993214999	699828.03	116.578805049261
MDG-b_4_n500_m50.txt	0	0.211535014450689	695596.839999999	120.908857345581
MDG-b_1_n500_m50.txt	0	0.562528885964526	706053.110000001	91.2750599384308
MDG-b_2_n500_m50.txt	0	0.203965080122004	697741.41	91.4421387513479
MDG-b_3_n500_m50.txt	0	0.106790837113019	700016.2	144.243008454641
MDG-a_37_n2000_m200.txt	0	10.4723307013733	104123	1276.11965092023

Figura 2: Algoritmo de búsqueda local

El algoritmo de búsqueda local es estocástico, debido a que para la obtención de cada una de las soluciones se utiliza un generador de números pseudoaleatorio. El tiempo de ejecución varía considerablemente según el dataset:

- Dataset con n=500: 1-2 minutos
- Dataset con n=2000: 20-25 minutos

La distancia total obtenida, por lo general, es superior al algoritmo greedy lo cual indica que la búsqueda local obtiene mejores resultados a expensas del tiempo de ejecución.

Debido a nuestras limitaciones computacionales, las ejecuciones de este algoritmo se hicieron con 100 iteraciones máximas.