Práctica 1

Metaheurísticas

Amin Kasrou Aouam

UNIVERSIDAD
DE GRANADA

2021-04-19

Índice

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 3 iteraciones de ambos algoritmos, 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
MDG-b_2_n500_m50.txt	0	0.261434902138248	2803.22	8.64382664362589
MDG-b_3_n500_m50.txt	4.54747350886464E-13	0.092981462003912	2775.63	8.58431116739909
MDG-b_1_n500_m50.txt	0	0.014719699327197	3256.56	8.49988547960917
MDG-a_32_n2000_m200.txt	0	10.4844260886423	38	368.712572574615
MDG-b_5_n500_m50.txt	0	0.305653196036549	2386.75	10.4136736392975
MDG-b_4_n500_m50.txt	0	0.107096459461325	3040.24	10.2561395168304
MDG-b_6_n500_m50.txt	0	0.020252303863778	3210.92	10.1704465548197
MDG-a_35_n2000_m200.txt	0	2.47489197218353	39	375.158727327983
MDG-b_7_n500_m50.txt	0	0.227942396037436	2649.76	10.3995771408081
MDG-a_39_n2000_m200.txt	0	11.929401254362	55	398.635855356852
GKD-c_20_n500_m50.txt	0	0.019726730000708	543.75469	8.2245346705118
MDG-a_37_n2000_m200.txt	0	11.4559195620061	49	368.42397403717
MDG-b_8_n500_m50.txt	0	0.008579120395212	3153.74	8.25128229459127
MDG-b_9_n500_m50.txt	0	0.024371430519978	3165.65	8.2448894182841
MDG-a_36_n2000_m200.txt	0	0.932687807894657	42	351.645670334498
MDG-a_31_n2000_m200.txt	0	1.15553431614969	29	350.185215870539
MDG-a_38_n2000_m200.txt	0	2.20556947767629	50	354.702796697617
GKD-c_19_n500_m50.txt	0	0.025057819092256	532.7204	8.24011325836182
MDG-a_40_n2000_m200.txt	0	6.10863331203448	44	357.595341444016
GKD-c_18_n500_m50.txt	0	0.051769909303781	541.95701	8.52847909927368
GKD-c_13_n500_m50.txt	0	0.024419886862672	545.62264	8.44420949618022
MDG-a_33_n2000_m200.txt	0	12.9084497110929	46	366.965148766835
GKD-c_12_n500_m50.txt	0	0.167566060643811	571.8175	8.64895057678223
MDG-a_34_n2000_m200.txt	0	23.1242921844201	55	388.232962608337
GKD-c_11_n500_m50.txt	0	0.088092102502906	538.16704	10.253582239151
MDG-b_10_n500_m50.txt	0	0.0186921766728	2649.1	10.3257246017456
GKD-c_14_n500_m50.txt	0	0.110239644357362	550.139	10.3202366828919
GKD-c_15_n500_m50.txt	0	0.957262844699994	528.57953	9.61564723650614
GKD-c_17_n500_m50.txt	0	0.014424194829711	528.68647	8.29386266072591
GKD-c_16_n500_m50.txt	0	0.695329083112121	590.55134	10.3096911112467

Figura 1: Algoritmo greedy

El algoritmo greedy es determinista, por lo tanto la desviación típica es prácticamente nula (varía en el tiempo de ejecución únicamente). El tiempo de ejecución varía considerablemente según el dataset:

- Dataset con n=500: 8-10 segundos
- Dataset con n=2000: 6-7 minutos

Por lo tanto, el algoritmo o la implementación de éste no escalan al aumentar el número de casos.

Las distancias obtenidas son considerablemente peores que las del algoritmo de búsqueda local, aunque hay que tener en cuenta que en la implementación del algoritmo de búsqueda local la distancia del primer elemento no es 0, lo cual tiene afecta el resultado final.

Algoritmo de búsqueda local

dataset	desviacion distancia	desviacion tiempo	media distancia	media tiempo
MDG-b_2_n500_m50.txt	7.27595761418343E-12	0.168021228339694	47345.36	2.83932169278463
MDG-b_3_n500_m50.txt	7.27595761418343E-12	0.025255362191039	47474.27	2.66395171483358
MDG-b_1_n500_m50.txt	0	0.007538183973697	47609.13	2.31797631581624
MDG-a_32_n2000_m200.txt	0	1.16041012417128	1384	10.2383976777395
MDG-b_5_n500_m50.txt	0	0.216158379425281	47863.03	3.54463561375936
MDG-b_4_n500_m50.txt	7.27595761418343E-12	0.012292366510675	45475.33	2.36432353655497
MDG-b_6_n500_m50.txt	7.27595761418343E-12	0.012571578970968	45678.01	2.4174648920695
MDG-a_35_n2000_m200.txt	0	1.02277578317777	1378	12.599294980367
MDG-b_7_n500_m50.txt	0	0.257731309973307	47067.47	2.75673413276672
MDG-a_39_n2000_m200.txt	0	1.17586511290857	1439	11.0350430011749
GKD-c_20_n500_m50.txt	1.136868377216616E-13	0.010784625643396	821.09563	2.14565992355347
MDG-a_37_n2000_m200.txt	0	2.25604561143329	1464	11.9650530815125
MDG-b_8_n500_m50.txt	7.27595761418343E-12	0.00609258787756	46608.2	1.86693509419759
MDG-b_9_n500_m50.txt	0	0.014247752178129	46474.81	2.07033443450928
MDG-a_36_n2000_m200.txt	0	0.015155490336137	1430	9.2965792020162
MDG-a_31_n2000_m200.txt	0	0.024554971250424	1447	10.5715458393097
MDG-a_38_n2000_m200.txt	0	0.03879754996687	1434	8.90016746520996
GKD-c_19_n500_m50.txt	0	0.017351144266703	832.92263	2.23581210772196
MDG-a_40_n2000_m200.txt	0	0.405820610204352	1427	9.96694374084473
GKD-c_18_n500_m50.txt	0	0.011885495384292	815.90189	1.88125189145406
GKD-c_13_n500_m50.txt	0	0.008685320652168	816.19612	2.44805328051249
MDG-a_33_n2000_m200.txt	0	0.498958963706008	1379	10.718853076299
GKD-c_12_n500_m50.txt	0	0.019958522738798	809.7311	1.68720134099325
MDG-a_34_n2000_m200.txt	0	1.16620424911878	1495	11.0765051841736
GKD-c_11_n500_m50.txt	1.136868377216616E-13	0.020400119473588	835.56987	2.49059414863586
MDG-b_10_n500_m50.txt	7.27595761418343E-12	0.008104870163731	46687.65	3.08750780423482
GKD-c_14_n500_m50.txt	0	0.021773678784178	825.99836	2.42206994692484
GKD-c_15_n500_m50.txt	0	0.310386397007616	833.65199	3.11498006184896
GKD-c_17_n500_m50.txt	1.136868377216616E-13	0.001865194754757	848.13592	2.79607351620992
GKD-c_16_n500_m50.txt	0	0.012765496344886	833.58529	2.90375844637553

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 es prácticamente constante con cada dataset:

- Dataset con n=500: 1-3 segundos
- Dataset con n=2000: 8-12 segundos

Por lo tanto éste escala bien al aumentar el número de casos. Aún así, al realizar ciertas pruebas aumentando el número de iteraciones máximas, el rendimiento del algoritmo emperoaba considerablemente. Por este motivo, las ejecuciones de este algoritmo se han hecho con 100 iteraciones máximas.