Metaheurísticas
Práctica 1: Problemas, instancias y soluciones. Búsqueda aleatoria.

Convocatoria de junio 2014 (curso académico 2013/2014)

Pedro Antonio Gutiérrez Peña

21 de febrero de 2014

Resumen

Esta práctica sirve de introducción a las metaheurísticas, familiarizando al alumno con los conceptos básicos sobre representación de problemas y soluciones. El alumno deberá discernir la diferencia entre problema e instancia, y deberá ser capaz de cargar instancias para cada uno de los problemas presentados. La entrega se hará utilizando las tareas en Moodle habilitadas al efecto. El día tope para la entrega es el **20 de marzo a las 23.55h**. En caso de que dos alumnos entreguen códigos copiados, no se puntuarán ninguno de los dos. Comprueba que los comportamientos de los programas son similares a los esperados en los ejemplos de ejecución.

1. Introducción

El trabajo a realizar en la práctica consiste en implementar las estructuras básicas para abordar cuatro problemas distintos susceptibles de resolverse utilizando metaheurísticas. En este caso, nos limitaremos a generar soluciones aleatorias de cada problema, evaluarlas y devolver la mejor solución encontrada utilizando esta búsqueda aleatoria. Las cuatro secciones siguientes introducen los cuatro problemas, mientras que la última sección introduce los entregables para esta práctica (código fuente y memoria con resultados).

2. MaxMin Diversity Problem (MMDP)

2.1. Descripción

El problema de la *máxima mínima diversidad* (*MaxMin Diversity Problem*, MMDP), consiste en seleccionar un determinado número de elementos (m) de un conjunto de n elementos de tal forma que la menor de las distancias entre los elementos seleccionados sea máxima. La definición de distancia entre los elementos depende de las aplicaciones específicas. El problema MMDP se puede formular como un problema cuadrático binario:

$$\begin{array}{rc} \text{Maximizar:} & z_{\text{MMDP}}(\mathbf{x}) = \text{mín}_{i \neq j} \left(d_{ij} : x_i = 1 \wedge x_j = 1 \right), \\ \text{sujeto a:} & \sum_{i=1}^{n} x_i = m, \\ & x_i, x_j \in \{0, 1\}, \\ & 1 \leq i \leq n, 1 \leq j \leq n, \end{array} \tag{1}$$

donde d_{ij} sería la distancia entre el nodo i y el nodo j. El valor $x_i \in \{0, 1\}$ indica si el elemento i-ésimo ha sido incluido en la selección (0 si no está incluido y 1 si lo está).

2.2. Instancias

Para esta práctica, se utilizarán dos conjuntos de instancias:

- GKD-Ia: instancias del problema más pequeñas (con n entre 10 y 30, y con diferentes valores de m). Estas instancias son más sencillas de resolver.
- GKD-Ic: instancia más grande ($n = 500$, $m = 50$) requiriendo un mayor coste computacional.

Estos dos conjuntos de instancias junto con los mejores valores conocidos para ambos pueden descargarse desde el Moodle.

2.3. Formato de los ficheros

El formato de los ficheros es el mismo para ambos conjuntos. Su estructura es la siguiente:

- Una primera línea donde se indica el valor de n y de m.
- A continuación, en cada línea se indica el índice de dos elementos y la distancias entre ellos. Los índices van de 0 a $n - 1$.
- Para estas instancias, existe una distancia para cada par de elementos.

Un ejemplo de fichero de entrada es el siguiente:

```
5 2
0 1 166.47234
0 2 170.18475
0 3 174.55453
0 4 153.28670
1 2 145.12186
1 3 144.42723
1 4 170.98466
2 3 176.58110
2 4 162.99632
3 4 168.48360
```

2.4. Tareas a realizar

A partir del esqueleto de código suministrado, el alumno deberá realizar las siguientes tareas:

1. Definir las estructuras de datos más indicadas para almacenar las instancias en memoria del MMDP.
2. Implementar un procedimiento para cargar una instancia del MMDP en dichas estructuras de datos, a partir del nombre del fichero que contiene información sobre el mismo.
3. Definir la estructura de una solución para el problema MMDP ($\mathbf{x}$).
4. Implementar un procedimiento para calcular la función objetivo a partir de una solución del problema MMDP ($z_{\mathrm{MMDP}}(\mathbf{x})$).
5. Implementar un método que genere una solución aleatoria **válida** de una instancia del problema MMDP.
6. Implementar un programa que genere 1000 soluciones aleatorias para una instancia del problema MMDP, que imprima el valor de la función objetivo de cada solución y que finalmente imprima la mejor solución encontrada.

Algunas aclaraciones:

- Se deben permitir líneas vacías, siempre que estén fuera de las cabeceras.
- Se proporciona un esqueleto del programa para que completes las clases necesarias. Deberás introducir una clase por cada tipo de problema e implementar los métodos correspondientes.
- El entorno de programación a utilizar es Eclipse, en su versión June. Eclipse[1] es un IDE (*Integrated Development Environment*) escrito en Java para desarrollar proyectos Java. Sin embargo, el *plugin* CDT permite desarrollar en C++.
- Las instrucciones para cargar el esqueleto desde Eclipse, se encuentran en la plataforma Moodle.
- El programa a desarrollar recibe tres argumentos (que pueden aparecer en cualquier orden):
  - Argumento `f`: Indica el nombre del fichero que contiene la instancia del problema a cargar. Sin este argumento, el programa no puede funcionar.
  - Argumento `p`: indica el tipo problema al que corresponde la instancia a cargar (en esta primera parte será siempre `MMDP`, pero debes ir contemplando y teniendo en mente que en las siguiente subsecciones ampliaremos a `CWP` y `CPH`). Si el usuario no especifica ningún tipo, se tomará por defecto el `MMDP`.
  - Argumento `s`: Indica la semilla para los números aleatorios que se debe utilizar. Si el usuario no especifica semilla, se extraerá a partir de la fecha.
- Para procesar la secuencia de entrada, se recomienda utilizar la función `getopt()` de la librería `libc`.

El siguiente sería un ejemplo de ejecución del programa para un problema MMDP:

```
i02gupep@NEWTS:~/workspace-mh/practica1/Release$ ./practica1 -f ../instancias/MMDP/GKD-
    Ia_1_n10_m2.txt -s 80 -p MMDP
Semilla 80. Problema MMDP. Instancia ./instancias/MMDP/GKD-Ia_1_n10_m2.txt...
============================
Dimensiones:
m=2, n=10
Matriz de conectividad:
0 166.472 170.185 174.555 153.287 145.122 144.427 170.985 176.581 162.996
166.472 0 168.484 175.632 135.436 153.113 147.681 195.461 156.552 184.358
170.185 168.484 0 183.435 159.962 150.259 131.151 169.923 168.09 193.298
174.555 175.632 183.435 0 187.822 122.499 196.629 182.584 144.857 187.116
153.287 135.436 159.962 187.822 0 167.544 162.239 176.992 196.174 127.36
145.122 153.113 150.259 122.499 167.544 0 140.804 108.779 131.51 165.057
144.427 147.681 131.151 196.629 162.239 140.804 0 162.186 199.05 173.735
170.985 195.461 169.923 182.584 176.992 108.779 162.186 0 164.319 188.754
176.581 156.552 168.09 144.857 196.174 131.51 199.05 164.319 0 243.973
162.996 184.358 193.298 187.116 127.36 165.057 173.735 188.754 243.973 0
============================
Solución generada (iteracion 0):
**************************
Elementos escogidos:
0 0 0 0 0 1 0 0 1 0
**************************
Función objetivo: 131.51

...

Solución generada (iteracion 999):
**************************
Elementos escogidos:
1 0 0 1 0 0 0 0 0 0
**************************
```

[1] http://www.eclipse.org/

```
Función objetivo: 174.555

==> Valores extremos generados: máximo=243.973 mínimo=108.779
```

3. Travelling Salesman Problem (TSP)

El problema del *viajante de comercio* (*Travelling Salesman Problem*, TSP), consiste en encontrar el recorrido más corto que une un conjunto de m ciudades, de forma que ninguna ciudad se visita dos veces y que el recorrido empieza y termina en la misma ciudad. La definición de distancia entre los elementos depende de las aplicaciones específicas. El problema puede representarse con un grafo, cuyos nodos representan las ciudades y cuyos arcos representan la distancia entre las mismas (en ese caso, el TSP equivaldría a encontrar un ciclo hamiltoniano en el grafo). Consideramos la versión **simétrica**, que implica que la distancia desde la ciudad i a la ciudad j es la misma que desde la ciudad j a la ciudad i.

El problema TSP se puede formular como un problema de programación entera, representando la solución como una matriz de valores binarios. Consideramos un valor $x_{ij} \in \{0, 1\}$ que indica si el camino de i a j ha sido incluido en la selección (0 si no está incluido y 1 si lo está). La definición sería:

$$\begin{aligned} \text{Minimizar:} \quad & z_{\text{TSP}}(\mathbf{x}) = \textstyle\sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} \cdot x_{ij}, \\ \text{sujeto a:} \quad & \textstyle\sum_{j=1}^{m} x_{ij} = 1, 1 \leq i \leq m, \\ & \textstyle\sum_{i=1}^{m} x_{ij} = 1, 1 \leq j \leq m, \\ & \textstyle\sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} = m, \end{aligned} \tag{2}$$

donde d_{ij} sería la distancia entre el nodo i y el nodo j. Definiremos $d_{ii} = \infty$ para $1 \leq i \leq m$.

Sin embargo, este planteamiento no es el más óptimo. Utilizaremos permutaciones para representar las soluciones, es decir, $\mathbf{x}$ será una permutación de los identificadores de las ciudades, de forma que x_i será el identificador de la ciudad que se visita en la posición i-ésima al realizar el recorrido. De esta forma, las ciudades se visitarán en el orden indicado por la permutación. El problema entonces se reduce a:

$$\text{Minimizar:} \quad z_{\text{TSP}}(\mathbf{x}) = \textstyle\sum_{i=1}^{m-1} d_{x_i x_{i+1}} + d_{x_n x_1}. \tag{3}$$

Un ejemplo de los recorridos correspondientes a dos posibles soluciones se puede consultar en la Figura 1.

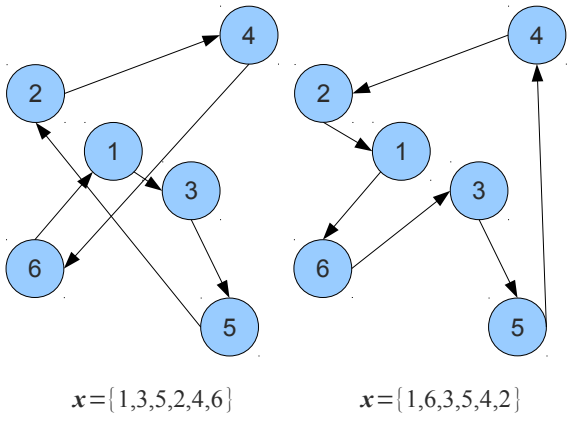

Figura 1: Ejemplo de dos soluciones TSP.

3.1. Instancias

Para esta práctica, se utilizará un solo conjunto de instancias con cinco problemas TSP[2]. Los ficheros incluyen los siguientes problemas:

- `att48`: 48 ciudades (capitales de los Estados Unidos). El viaje de mínimo recorrido es de 10628.
- `dantzig42`: conjunto de 42 ciudades. El mínimo recorrido es de 699.
- `fri26`: 26 ciudades. El mínimo recorrido tiene distancia 937.
- `gr17`: conjunto de 17 ciudades con un recorrido mínimo de 2085.
- `p01`: conjunto pequeño de 15 ciudades.

Estas instancias pueden descargarse del Moodle.

3.2. Formato de los ficheros

El formato de los ficheros es el mismo para ambos conjuntos. Su estructura es la siguiente:

- Una primera línea donde se indica el valor de m.
- A continuación, aparecerá la matriz de distancias de las distintas ciudades, d_{ij}. Esta matriz tendrá m filas y m columnas, de forma que el elemento d_{ij} será igual a la distancia entre la ciudad i y la ciudad j.

Un ejemplo de fichero de entrada es el siguiente:

```
15
0 29 82 46 68 52 72 42 51 55 29 74 23 72 46
29 0 55 46 42 43 43 23 23 31 41 51 11 52 21
82 55 0 68 46 55 23 43 41 29 79 21 64 31 51
46 46 68 0 82 15 72 31 62 42 21 51 51 43 64
68 42 46 82 0 74 23 52 21 46 82 58 46 65 23
52 43 55 15 74 0 61 23 55 31 33 37 51 29 59
72 43 23 72 23 61 0 42 23 31 77 37 51 46 33
42 23 43 31 52 23 42 0 33 15 37 33 33 31 37
51 23 41 62 21 55 23 33 0 29 62 46 29 51 11
55 31 29 42 46 31 31 15 29 0 51 21 41 23 37
29 41 79 21 82 33 77 37 62 51 0 65 42 59 61
74 51 21 51 58 37 37 33 46 21 65 0 61 11 55
23 11 64 51 46 51 51 33 29 41 42 61 0 62 23
72 52 31 43 65 29 46 31 51 23 59 11 62 0 59
46 21 51 64 23 59 33 37 11 37 61 55 23 59 0
```

3.3. Tareas a realizar

A partir del esqueleto de código suministrado, el alumno deberá realizar las siguientes tareas:

1. Definir las estructuras de datos más indicadas para almacenar las instancias en memoria del TSP.
2. Implementar un procedimiento para cargar una instancia del TSP en dichas estructuras de datos, a partir del nombre del fichero que contiene información sobre el mismo.
3. Definir la estructura de una solución para el problema TSP ($\mathbf{x}$).
4. Implementar un procedimiento para calcular la función objetivo a partir de una solución del problema TSP ($z_{\text{TSP}}(\mathbf{x})$).

[2] `http://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html`

5. Implementar un método que genere una solución aleatoria **válida** de una instancia del problema TSP.

6. Implementar un programa que genere 1000 soluciones aleatorias para una instancia del problema TSP, que imprima el valor de la función objetivo de cada solución y que finalmente imprima la mejor solución encontrada.

El siguiente sería un ejemplo de ejecución del programa para un problema TSP:

```
i02gupep@NEWTS:~/workspace-mh/practica1/Release$ ./practica1 -f ../instancias/TSP/gr17.
    txt -s 80 -p TSP
Semilla 80. Problema TSP. Instancia ./instancias/TSP/gr17.txt...
===========================
Dimensiones:
m=17
Matriz de distancias:
0 633 257 91 412 150 80 134 259 505 353 324 70 211 268 246 121
633 0 390 661 227 488 572 530 555 289 282 638 567 466 420 745 518
257 390 0 228 169 112 196 154 372 262 110 437 191 74 53 472 142
91 661 228 0 383 120 77 105 175 476 324 240 27 182 239 237 84
412 227 169 383 0 267 351 309 338 196 61 421 346 243 199 528 297
150 488 112 120 267 0 63 34 264 360 208 329 83 105 123 364 35
80 572 196 77 351 63 0 29 232 444 292 297 47 150 207 332 29
134 530 154 105 309 34 29 0 249 402 250 314 68 108 165 349 36
259 555 372 175 338 264 232 249 0 495 352 95 189 326 383 202 236
505 289 262 476 196 360 444 402 495 0 154 578 439 336 240 685 390
353 282 110 324 61 208 292 250 352 154 0 435 287 184 140 542 238
324 638 437 240 421 329 297 314 95 578 435 0 254 391 448 157 301
70 567 191 27 346 83 47 68 189 439 287 254 0 145 202 289 55
211 466 74 182 243 105 150 108 326 336 184 391 145 0 57 426 96
268 420 53 239 199 123 207 165 383 240 140 448 202 57 0 483 153
246 745 472 237 528 364 332 349 202 685 542 157 289 426 483 0 336
121 518 142 84 297 35 29 36 236 390 238 301 55 96 153 336 0
===========================
Solución generada (iteracion 0):
**************************
Viaje a realizar:
15 1 3 12 0 13 2 6 9 11 16 4 14 5 10 8 7 **************************
Función objetivo: 4869
...
Solución generada (iteracion 999):
**************************
Viaje a realizar:
3 5 13 15 7 11 0 2 16 10 12 8 1 9 4 14 6 **************************
Función objetivo: 4277


==> Valores extremos generados: máximo=5996 mínimo=3315
```

4. CutWidth Problem (CWP)

4.1. Descripción

Dado un grafo no dirigido, el problema del minimizado del *cutwidth* (*CutWidth Problem*, CWP) consiste en encontrar una ordenación lineal del grafo de tal forma que el máximo número de aristas cortadas entre dos vértices consecutivos sea mínimo. Supongamos los siguientes elementos:

- $G = \{V, E\}$ representa el grafo que estamos tratando con $n = |V|$ vértices y $m = |E|$ aristas.
- V es un conjunto de n vértices, $V = \{1, 2, \ldots, n\}$.
- E es el conjunto de aristas, $E = \{(i, j)\}$, $i, j \in V$. Por ejemplo, $E = \{(1, 2), (1, 3)\}$ implica que existe una arista desde el nodo 1 hasta el nodo 2 y otra desde el nodo 1 hasta el nodo 3.

- $\mathbf{x}$ sería una posible ordenación lineal de V (permutación de V) que asignaría un número de orden (entero) a cada vértice de forma que x_i es el identificador del vértice que se queda en la i-ésima posición, según la permutación.
- El corte del vértice i-ésimo con respecto a la ordenación $\mathbf{x}$ se representa con $c(\mathbf{x}, i)$ y es el número de aristas que salen de i o de vértices anteriores (según la ordenación $\mathbf{x}$) a i y van a parar a vértices posteriores a i.

El problema del *cutwidth* se puede formular de la siguiente manera:

$$\text{Minimizar:} \quad z_{\text{CWP}}(\mathbf{x}) = \max_{i \in V} c(\mathbf{x}, i), \tag{4}$$
$$c(\mathbf{x}, i) = |\{(j,k) \in E : x_s = j, x_r = i, x_t = k, s \leq r < t\}|, i, j, k \in V.$$

El valor de corte del último vértice colocado en la ordenación $\mathbf{x}$ (vértice asociado al valor máximo de $\mathbf{x}$) siempre será cero, ya que no puede existir ningún x_i mayor. La figura 2 muestra un ejemplo de cálculo tanto de los distintos valores $c(\mathbf{x}, i)$ para una solución dada, como del valor de bondad de la solución $z(\mathbf{x})$.

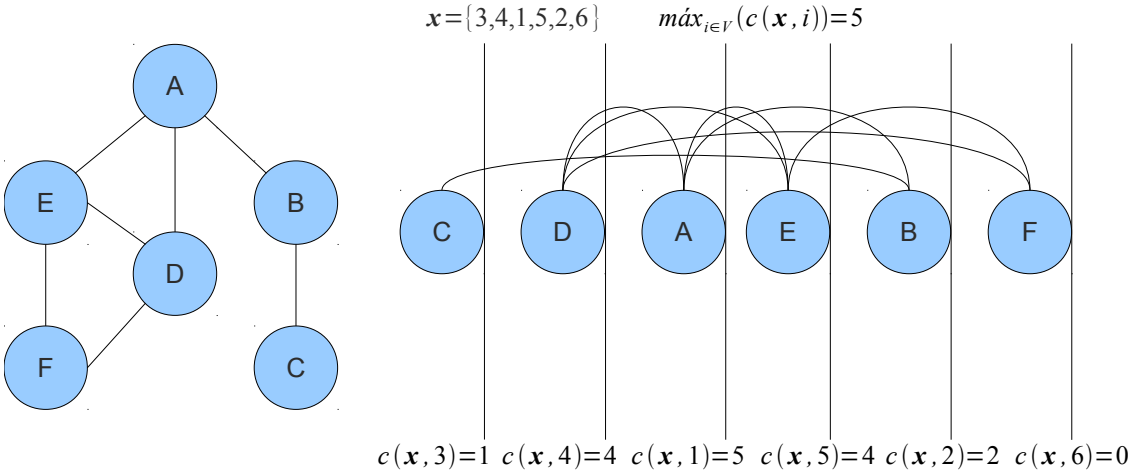

Figura 2: Ejemplo de cálculo de los distintos valores de corte ($c(\mathbf{x}, i)$) y de la función objetivo ($z(\mathbf{x})$).

4.2. Instancias

En este caso, se utilizará un único conjunto de instancias denominado Harwell-Boeing.

4.3. Formato de los ficheros

El formato de los ficheros es el siguiente:

- Una primera línea donde se indica el número de vértices (por duplicado) y el número de aristas.
- El resto de líneas describen la matriz de conectividad del grafo, mediante una línea por arista expresada con los dos índices de los vértices que conecta. Los índices van de 1 a n.

Un ejemplo de fichero de entrada es el siguiente (el cuál describe el grafo de la figura 2):

```
6 6 7
1 2
1 4
```

```
1 5
4 5
4 6
5 6
2 3
```

4.4. Tareas a realizar

Las tareas a realizar para este problema son:

1. Definir las estructuras de datos más indicadas para almacenar las instancias en memoria del CWP.
2. Implementar un procedimiento para cargar una instancia del CWP en dichas estructuras de datos, a partir del nombre del fichero que contiene información sobre el mismo.
3. Definir la estructura de una solución para el problema CWP ($\mathbf{x}$).
4. Implementar un procedimiento para calcular la función objetivo a partir de una solución del problema CWP ($z_{\text{CWP}}(\mathbf{x})$).
5. Implementar un método que genere una solución aleatoria **válida** de una instancia del problema CWP.
6. Implementar un programa que genere 1000 soluciones aleatorias para una instancia del problema CWP, que imprima el valor de la función objetivo de cada solución y que finalmente imprima la mejor solución encontrada.

El siguiente sería un ejemplo de ejecución del programa para un problema CWP:

```
i02gupep@NEWTS:~/workspace-mh/practica1/Release$ ./practica1 -f ../instancias/ -s 80 -p
    CWP
Semilla 80. Problema CWP. Instancia ../instancias/CWP/ejemplo.txt...
============================
Dimensiones:
n=6, m=7
Matriz de conectividad:
0 1 0 1 1 0
1 0 1 0 0 0
0 1 0 0 0 0
1 0 0 0 1 1
1 0 0 1 0 1
0 0 0 1 1 0
============================
Solución generada (iteracion 0):
**************************
Ordenación realizada:
3 2 0 5 4 1
**************************
Función objetivo: 5
...
Solución generada (iteracion 999):
**************************
Ordenación realizada:
1 2 5 0 3 4
**************************
Función objetivo: 4


==> Valores extremos generados: máximo=6 mínimo=3
```

5. Capacited p-Hub (CPH)

5.1. Descripción

El problema del *capacited p-hub* (CPH) consiste en un conjunto de n centros que podrían tener el rol de cliente o de concentrador (también denominados *hubs*), de manera que:

- Se establece un número máximo de centros que podrían ser concentradores igual a p.
- Todos los nodos son clientes, incluso los propios concentradores (que siempre estarán conectados a sí mismos).
- Cada cliente solo puede solicitar servicios de un concentrador.
- El objetivo es seleccionar los p centros que actuarán como concentradores y decidir el concentrador de cada cliente, intentando minimizar la suma de las distancias de los clientes y los concentradores.
- Además, se incluye la restricción de que cada concentrador sólo puede servir una determinada cantidad de recursos y que cada cliente tiene una demanda que debe ser satisfecha. Una solución factible debería asegurarse de que todos los concentradores pueden servir la demanda de recursos de los clientes que tiene.

El modelo matemático sería el siguiente:

$$
\begin{array}{rll}
\text{Minimizar:} & z_{\text{CPH}}(\mathbf{X}, \mathbf{y}) = \sum_j \sum_i d_{ij} \cdot x_{ij}, & \\
\text{Sujeto a:} & \sum_j x_{ij} = 1, & 1 \leq i \leq n, \\
& x_{ij} \leq y_j, & 1 \leq i \leq n, 1 \leq j \leq n, \\
& \sum_j y_j = p, & \\
& \sum_i f_i \cdot x_{ij} \leq c, & 1 \leq j \leq n, \\
& x_{ij} \in \{0, 1\}, & \\
& y_j \in \{0, 1\}, &
\end{array}
\tag{5}
$$

donde:

- Una solución está compuesta por una matriz cuadrada $\mathbf{X}$ de $n \times n$ elementos y un vector $\mathbf{y}$ de n elementos.
- El elemento x_{ij} de $\mathbf{X}$ es igual a 1 si el concentrador j presta servicios al cliente i y es igual a 0 en caso contrario.
- y_j es igual a 1 si el nodo j es concentrador y 0 si el nodo es cliente.
- d_{ij} es la distancia entre el nodo i y el nodo j. Si las coordenadas de ambos nodos son $\mathbf{p}_i = (p_{i\text{x}}, p_{i\text{y}})$ y $\mathbf{p}_j = (p_{j\text{x}}, p_{j\text{y}})$, la distancia entre ambos se define como:

$$
d_{ij} = \sqrt{(p_{i\text{x}} - p_{j\text{x}})^2 + (p_{i\text{y}} - p_{j\text{y}})^2}. \tag{6}
$$

- f_i es la demanda del nodo i cuando actúa como cliente.
- c es la capacidad genérica de cada concentrador (misma capacidad para todos los concentradores).

Aunque este sea el planteamiento del CPH desde el punto de vista matemático, se recomienda utilizar la representación introducida en las diapositivas de la asignatura.

5.2. Instancias

Se van a considerar dos conjuntos de instancias:

- `phub_50_5`: formado por instancias pequeñas, con un total de 50 nodos entre los que hay que escoger 5 concentradores.
- `phub_100_10`: instancias más grandes, con un total de 100 nodos de los que vamos a escoger 10 concentradores.

5.3. Formato de los ficheros

El formato de los ficheros es el mismo para ambos conjuntos:

- En la primer línea se incluirá el número total de nodos (n), el número de nodos concentradores (p) y la capacidad por concentrador (c), que será la misma para todos.
- Luego tendremos una línea por nodo, cada línea tendrá cuatro valores:
  - El primer valor es el número de nodo (de 1 a n).
  - El segundo y tercer valor son las coordenadas del nodo (en un espacio euclídeo de dos dimensiones).
  - El último valor es la demanda del nodo cuando actúa como cliente.

Finalmente, un ejemplo de este tipo de ficheros es el siguiente:

```
10 2 120
1 3 60 4
2 68 49 8
3 81 53 9
4 62 94 12
5 1 57 10
6 7 98 14
7 24 94 14
8 59 95 12
9 19 13 1
10 70 86 3
```

5.4. Tareas a realizar

Las tareas a realizar para este problema son:

1. Definir las estructuras de datos más indicadas para almacenar las instancias en memoria del CPH.
2. Implementar un procedimiento para cargar una instancia del CPH en dichas estructuras de datos, a partir del nombre del fichero que contiene información sobre el mismo.
3. Definir la estructura de una solución para el problema CPH ($\mathbf{x}$).
4. Implementar un procedimiento para calcular la función objetivo a partir de una solución del problema CPH ($z_{\text{CPH}}(\mathbf{x})$). Si la solución no es factible, devuelva un valor muy alto (mayor valor `float` posible).
5. Implementar un método que genere una solución aleatoria **válida** de una instancia del problema CPH. Una estrategia simple para cumplir la restricción de la capacidad máxima de un nodo, es siempre elegir como concentrador para un cliente, aquel que tenga su capacidad actual al mínimo.

6. Implementar un programa que genere 1000 soluciones aleatorias para una instancia del problema CPH, que imprima el valor de la función objetivo de cada solución y que finalmente imprima la mejor solución encontrada.

El siguiente sería un ejemplo de ejecución del programa para un problema CPH:

```
i02gupep@NEWTS:~/workspace-mh/practica1/Release$ ./practica1 -f ../instancias/CPH/
    phub_50_5_1.txt -s 80 -p CPH
Semilla 80. Problema CPH. Instancia ../instancias/CPH/phub_50_5_1.txt...
===========================
Dimensiones:
n=50, p=5, c=120
Matriz de distancias:
0 86.3308 42.8019 67.424 54.6443 76.4003 78.4474 107.703 100.846 57.8705 42.0595 93.1075
     44.2719 21 25.4951 35.1283 56.8858 16.9706 60.1415 101.242 10.8167 54.3783 58.5235
    72.1803 82.1523 60.8276 58.6686 72.8354 43.1393 67.1863 70.214 15.2643 36.4966
    61.0082 102.157 14.5602 69.6348 64.3817 23.3238 83.7257 19.105 52.4023 94.255 10.198
     62.0322 50.3289 69.9714 53.4603 52.0865 4.12311
86.3308 0 76.844 23.0868 47.676 18.4391 60.075 23.6008 16.6433 51.4781 70.214 7.07107
    62.426 97.2008 63.6003 75.0267 51.8941 70.5762 37.6563 19.3132 75.6439 49.8197
    74.8465 27.7308 32.249 71.7844 28.1603 30.8869 47.5184 43.4166 42.0595 86.0233
    72.0069 80.6536 16.2788 77.4919 21.2132 46.1736 63.1585 6.40312 94.4034 81.8841 15
    78.3901 62.2977 87.4643 47.8853 72.7805 42.8019 85.6154
...
===========================
Solución generada (iteracion 0):
***************************
Concentradores:
7 0 3 4 1 0 0 0 2 1 4 2 0 3 3 5 2 3 3 2 0 2 1 3 4 1 3 0 4 2 3 2 9 1 2 0 4 1 0 4 8 1 0 3
    6 4 2 3 4 3 ***************************
Función objetivo: 2616.32
...
Solución generada (iteracion 999):
***************************
Concentradores:
3 7 4 1 3 2 4 4 5 1 8 4 3 2 0 4 0 1 2 0 3 2 0 1 2 3 0 1 4 1 3 3 3 0 4 9 0 3 0 2 2 2 1 2
    3 3 4 6 1 4 ***************************
Función objetivo: 2478.76


==> Valores extremos generados: máximo=3103.66 mínimo=1761.62
```

6. Entregables

Los ficheros a entregar serán los siguientes:

- Memoria de la práctica en un fichero `pdf` que describa el programa generado y las tablas de resultados.
- Fichero ejecutable de la práctica y código fuente.

6.1. Memoria de la práctica

La memoria de la práctica deberá incluir, al menos, el siguiente contenido:

- Portada con el número, título de la práctica, asignatura, titulación, escuela, universidad, curso académico, nombre, DNI y correo electrónico del alumno.
- Índice del contenido de la memoria con numeración de las páginas.
- Breve descripción de los cuatro problemas tratados (**máximo 2 carillas**).

- Descripción del esquema de representación de soluciones para cada problema y descripción en pseudocódigo de la función objetivo, etc... (**máximo 4 carillas**).
- Experimentos y análisis de resultados:
  - Descripción de las instancias de los problemas empleadas.
  - Resultados obtenidos para cada problema (mínimo y máximo de las 1000 ejecuciones y elección de la mejor solución).
  - Análisis del valor de la función objetivo para la mejor solución de la instancia más pequeña de cada problema, con objeto de comprobar si el programa funciona correctamente.
- Referencias bibliográficas u otro tipo de material distinto del proporcionado en la asignatura que se haya consultado para realizar la práctica (en caso de haberlo hecho).

Aunque lo importante es el contenido, se valorará también la presentación, incluyendo formato, estilo y estructuración del documento. La presencia de demasiadas faltas ortográficas puede disminuir la nota obtenida.

6.2. Ejecutable y código fuente

Junto con la memoria, se deberá incluir el fichero ejecutable preparado para funcionar en las máquinas de la UCO (en concreto, probar por `ssh` en `ts.uco.es`). Además se incluirá todo el código fuente necesario. No incluir los ficheros correspondientes a las instancias de los problemas. Para facilitar la creación de las tablas, utilizad el *script* `procesaTodasInstancias.sh`, que recoge los valores máximo y mínimo obtenido para cada instancia, además del coste computacional. Un ejemplo de aplicación del *script* es el siguiente:

```
1 i02gupep@NEWTS:~/workspace-mh/practica1/Release$ ./procesaTodasInstancias.sh ../
      instancias/ informe.csv
2 Procesando el fichero ../instancias/TSP/gr17.txt...
3 Procesando el fichero ../instancias/TSP/fri26.txt...
4 Procesando el fichero ../instancias/TSP/p01.txt...
5 Procesando el fichero ../instancias/TSP/dantzig42.txt...
6 Procesando el fichero ../instancias/TSP/att48.txt...
7 Procesando el fichero ../instancias/CPH/phub_100_10_4.txt...
8 Procesando el fichero ../instancias/CPH/phub_50_5_5.txt...
9 ...
```