Metaheurística - Práctica 1.a

Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema de la Asignación Cuadrática

3º Grado Ingeniería Informática, Grupo 3 (Miércoles)
Salvador Corts Sánchez, 75935233C
salvacorts@correo.ugr.es

Contents

1 Descripción del problema

El problema de asignación cuadrática (en inglés, quadratic assignment problem, QAP) es uno de los problemas de optimización combinatoria más conocidos. En él se dispone de n unidades y n localizaciones en las que situarlas, por lo que el problema consiste en encontrar la asignación óptima de cada unidad a una localización. La nomenclatura "cuadrático" proviene de la función objetivo que mide la bondad de una asignación, la cual considera el producto de dos términos, la distancia entre cada par de localizaciones y el flujo que circula entre cada par de unidades. El QAP se puede formular como:

$$QAP = \underset{\pi \in \prod_N}{min} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)} \right)$$

donde:

- π es una solución al problema que consiste en una permutación que representa la asignación de la unidad i a la localización $\pi(i)$.
- f_{ij} es el flujo que circula entre la unidad i y la j.
- d_{kl} es la distancia existente entre la localización k y la l.

2 Consideraciones comunes a los algoritmos utilizados

Esta práctica ha sido diseñada como una librería de metaheurísticas per se. Es decir, existe un tipo de objeto **Solution** y un tipo de objeto **Solver** del cual heredarán los objetos que implementan las diversas metaheurísticas. Cada metaheurística deberá implementar la función *Solve* que devuelve un objeto **Solution**.

Clase Solver

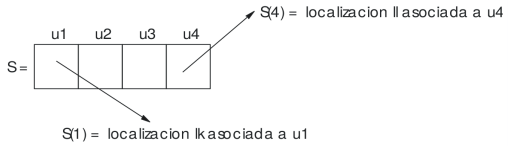
Esta clase debe ser heredada por las metaheurísticas a implementar. Su representación consta de dos matrices:

- **Distancias**: Matriz de distancias entre un punto i y otro j.
- **Frecuencias**: Matriz de flujo entre un objeto i y otro j.

Tiene una función virtual llamada *Solve* que ha de ser implementada por los objetos que hereden de **Solver**. Es la interfaz común a todos los objetos de tipo Solver para obtener una Solución.

Clase Solution

Sirve para representar una solución, la cual, se implementa como un vector donde cada posición i representa un objeto y alberga la localización j donde debe ser colocado dicho objeto i.



Existe una función *CalcCost* que calcula el coste de dicha solución como:

$$cost = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} f_{ij} d_{\pi(i)\pi(j)}$$

donde:

- π es la solución al problema.
- f_{ij} es el flujo que circula entre la unidad i y la j.
- d_{kl} es la distancia existente entre la localización k y la l.

Dado que el cálculo del coste de la solución es bastante costoso, $O(n^2)$, Esta función debe llamarse manualmente al menos una vez para obtener el coste y que este se guarde en la representación de la clase.

3 Algoritmo Greedy

Se basa en el cálculo de los potenciales de flujo y distancia definidos como:

$$\hat{f}_i = \sum_{j=1}^{n} f_{ij} \qquad \hat{d}_i = \sum_{j=1}^{n} d_{ij}$$

El algoritmo irá seleccionando la unidad i libre con mayor $\hat{f}_i$ y le asignará la localización j libre con menor $\hat{d}_j$. Su implementación en pseudocódigo es la siguiente:

```
# Calcula los potenciales
dp = fp = vector(n)
for i = 0 to n do
    f̂_i = d̂_i = 0
    for j = 0 to n do
        f̂_i = f̂_i + f_ij
        d̂_i = d̂_i + d_ij
    end
    dp_i = d̂_i
    fp_i = f̂_i
end

# Calcula la mejor combinación. π es la representación de la solución
locAssigned = unitAssigned = vector(n){0}
for i = 0 to n do
    bestf̂ = −∞;   bestf̂_index = 0
    bestd̂ = ∞;   bestd̂_index = 0
    for j = 0 to n do
        f̂_i = fp_j;   d̂_i = dp_j
        if f̂_i > bestf̂ and unitAssigned_j ≠ 1 then
            bestf̂ = f̂_i;   bestf̂_index = j
        end
        if d̂_i < bestd̂ and locAssigned_j ≠ 1 then
            bestd̂ = d̂_i;   bestd̂_index = j
        end
    end
    π(bestf̂_index) = bestd̂_index
    unitAssigned_{bestf̂_index} = locAssigned_{bestd̂_index} = 1
end
```

4 Algoritmo de Búsqueda Local

Vamos a utilizar una **búsqueda local del primer mejor**. Cuando se genera una solución vecina que mejora a la actual, se toma esta como solución y se pasa a la siguiente iteración. Se detiene la búsqueda cuando no se genera ningún vecino mejor que la solución actual.

Como se comentó anteriormente, el proceso de cálculo del coste de la solución es de orden cuadrático por lo que realizar dicho calculo con cada vecino es sumamente costoso; En su lugar, vamos a considerar una **factorización** teniendo en cuenta solo los cambios realizados por el movimiento de intercambio para generar el vecino.

Búsqueda Local con *Don't Look Bits*

b