Heurísticos en Optimización Combinatorial

A. Moujahid

Grupo de Inteligencia Computacional
Universidad del País Vasco UPV(EHU)
Curso 2014-2015

Contenido

Índice

Problema de optimización: Formulación general

$$\begin{array}{ll} \text{minimizar} & f_0(\mathbf{x}), \\ \text{sujeto a:} & f_i(\mathbf{x}) \geq b_i \quad i = 1, \ldots, m \end{array}$$

donde el vector $x = (x_1, ..., x_n)$ es la variable de optimización del problema, $f_0 : \Re^n \rightarrow \Re$ es la función objetivo, y $f_i : \Re^n \rightarrow \Re$ son las funciones de restriciones delimitadas por las constantes b_i

Distintas aproximaciones

- Programación lineal: optimizar una función lineal sujeta a restricciones lineales (Algotitmo simplex)
- Programación convexa: optimizar una función convexa sujeta a restricciones convexas
- Programación no lineal: cuando la función f o cualquiera de las funciones g_i y h_j son no lineales (Algoritmos numéricos).

Ejemplo1

Producto	Mano de obra (hora/unidad)	Materia prima (Kg/unidad)	Beneficio (Euro/unidad)
Producto A	1	4	20
Producto B	2	3	30
Recursos disponibles	40	120	

¿Se trata de determinar el número de cado uno de los productos A y B que hay que producir para maximizar el beneficio teniendo en cuenta los recursos disponibles?

Modelo de programación lineal

$x_1 =$ número de productos tipo A producidos por hora
$x_2 =$ número de productos tipo B producidos por hora

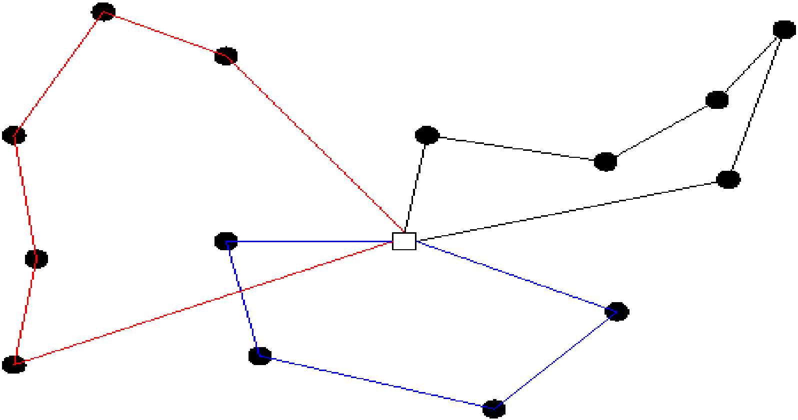
$$
\begin{array}{ll}
\text{maximizar} & 20x_1 + 30x_2 \\
\text{sujeto a} & x_1 + 2x_2 \leq 40 \\
 & 4x_1 + 3x_2 \leq 120 \\
 & x_1, x_2 \geq 0
\end{array}
$$

Índice

El problema del viajante de comercio

- Dadas *n* ciudades y conocido el coste de trasladarse de una ciudad a otra, se trata de encontrar la gira que visitando todas las ciudades, una y sólo una vez, tenga asociada un coste mínimo
- Referenciado por vez primera en 1932 como TSP (*Travelling Salesman Problem*)
- Buen número de problemas de optimización en Inteligencia Artificial se relacionan con la búsqueda de la permutación óptima (http://www.tsp.gatech.edu/index.html)

El problema del viajante de comercio (Fuente:Wikepedia)



El problema del viajante de comercio: Formulación

- **Solución**: permutación de las ciudades,

$$\pi^* = (\pi_1^*, \ldots, \pi_n^*)$$

- **Espacio de búsqueda**: conjunto de todas las permutaciones,

$$\Omega = \{(\pi_1, \pi_2, ..., \pi_n) | \pi_i \in \{1, 2, ..., n\}, \pi_i \neq \pi_j \forall i \neq j\}$$

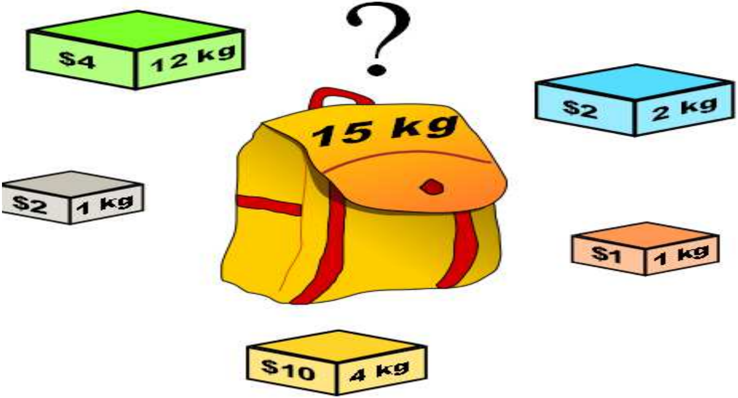
- **Función objetivo**:

$$F(\pi_1, \pi_2, ..., \pi_n) = \sum_{i=1}^{n-1} c_{\pi_i \pi_{i+1}} + c_{\pi_n \pi_1}$$

El problema de la mochila

Dado un conjunto finito de items x_i, $i = 1, ..., n$, cada uno de los cuales tiene asocido un peso w_i y una ganancia b_i, seleccionar el subconjunto de items a incluir en una mochila –capaz de soportar un peso máximo finito C– cuya inclusión proporcione una ganancia máxima.

El problema de la mochila (Fuente:Wikepedia)



El problema de la mochila: Formulación

- **Solución**: vector binario $(x_1, ..., x_n)$ de tamaño n tal que:

$$x_j = \begin{cases} 1 & \text{si el item } j \text{ se introduce en la mochila} \\ 0 & \text{en caso contrario} \end{cases}$$

- **Espacio de búsqueda**: conjunto de los vectores,

$$\Omega = \{(x_1, ..., x_n) | x_i \in \{0, 1\}, \sum_i^n w_i x_i \leqslant C\}$$

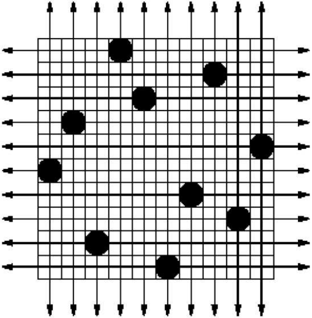
- **Función objetivo**:

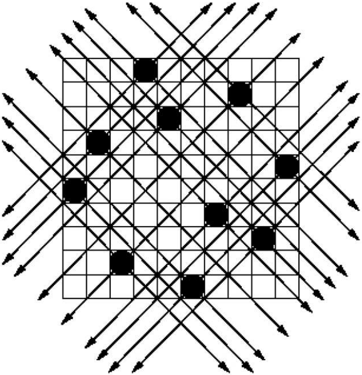
$$F(x_1, ..., x_n) = \sum_i^n b_i x_i, con (x_1, ..., x_n) \in \Omega$$

El problema de n reinas

Se trata de colocar las n reinas en un tablero de ajedrez $n \times n$, de tal manera que ninguna ataque al resto, es decir, dos reinas no pueden estar en la misma fila, columna y diagonal.

El problema de *n* reinas





El problema de n reinas: Formulación

- **Solución**: permutación de n elementos, es decir,

$$\pi = (\pi_1, \ldots, \pi_n)$$

donde π_i representa la posición de la columna en la fila i.

- **Espacio de búsqueda**: conjunto de todas las permutaciones,

$$\Omega = \{(\pi_1, \pi_2, ..., \pi_n) | \pi_i \in \{1, 2, ..., n\}, \pi_i \neq \pi_j \forall i \neq j\}$$

- **Función objetivo**: $F(\pi_1, \pi_2, ..., \pi_n) = \sum_{i,j=1, i \neq j}^{n} \delta \pi_i \pi_j$

$$\delta(\pi_i, \pi_j) = \begin{cases} 1 & \text{si } |i-j| = |\pi_i - \pi_j| \\ 0 & \text{si } |i-j| \neq |\pi_i - \pi_j| \end{cases}$$

Índice

Heurísticos: definición

- Simon (1963) define heurístico como

  un proceso que puede resolver un problema dado pero no ofrece garantías de hacerlo

- Hoy en día:

  procedimiento simple, a menudo basado en el sentido común, que se supone que va a ofrecer una buena solución (no necesariamente la óptima) de un modo fácil y rápido a problemas difíciles

Heurísticos

- **Justificación de uso**: problemas NP para los cuales no existe un algoritmo de resolución polinomial con el tamaño del problema
- **Ventajas**:
  - Flexibilidad frente al encorsetamiento de la investigación operativa clásica
  - Más comprensibles e intuitivos (en general) que los métodos exactos
- **Inconvenientes**:
  - Imposible conocer la cercanía con respecto del óptimo global de la solución obtenida

Heurísticos: condiciones en las que se aconseja su uso

- No existe un método exacto de resolución, o el mismo requiere de mucho gasto computacional y/o de memoria
- No es necesario encontrar la solución óptima; basta con una solución suficientemente buena
- Los datos son poco fiables y por tanto no tiene sentido el tratar de encontrar el óptimo global para dichos datos
- Existen limitaciones de tiempo (y/o de memoria) en proporcionar la respuesta
- Se va a utilizar como solución inicial para un algoritmo exacto de tipo iterativo

Búsquedas heurísticas

- **Heurísticas constructivas** añaden componentes individuales a la solución inicial hasta que se obtiene una solución final factible,
- **Heurísticas basadas en la mejora de una solución** parten de una solución para en cada paso buscar en la vecindad de la misma una solución mejor,
- **Heurísticas basadas en poblaciones** trabajan con poblaciones de individuos que evolucionan iterativamente.

Índice

Búsqueda local

- Explorar repetidamente la vecindad de una solución en busca de una solución mejor
- Cuando no se encuentra una solución que mejora la actual, se dice que la solución es localmente óptima

Búsqueda local: Características

- Es el algoritmo heurístico más sencillo
- Está basado en el concepto de localidad
- Se mantiene en todo momento una posible solución al problema
- A cada paso se elige una solución cercana a la solución actual que la mejore
- El algoritmo termina cuando ninguna solución cercana mejora la actual

Algoritmo de búsqueda local en pseudocódigo

Seleccionar una solución inicial $e_0 \in \Omega$
Repetir
    Elegir $e \in V(e_0)$ tal que $f(e) < f(e_0)$
    Asignar e a e_0
hasta $f(e) \geq f(e_0) \forall e \in V(e_0)$

Aspectos a determinar en la búsqueda local

- Solución inicial
- Conjunto de soluciones vecinas de cada solución e
- Elección de la solución vecina a cada paso

Ventajas de la búsqueda local

- Sencillez
- Tiempo computacional

Inconvenientes de la búsqueda local

- El algortimo termina en un óptimo local
- La solución final puede depender de la inicial

Mejoras de la búsqueda local

- Repetir la búsqueda comenzando en soluciones distintas (Algoritmos multiarranque)
- Aceptar una degradación temporal de la solución actual (Simulated Annealing)

Métodos multiarranque: Características

- El esquema general de un método multiarranque consiste en iterar dos pasos:
  - Generar una solución inicial
  - Aplicar una búsqueda local a la solución generada
- Se diferencian en como se llevan a cabo los pasos anteriores y en el criterio de parada elegido
- Algunos representantes son:
  - GRASP (greedy randomized adaptive search procedure)
  - ILS (iterated local search)

Métodos multiarranque: Multiple Restart Hill Climbing

- El algoritmo MRHillClimbing genera varias soluciones
- A continuación implementa una búsqueda local (hillClimbing).

Métodos multiarranque: Iterated Local Search (ILS)

- El algoritmo ILS genera una solución y a continuación implementa un HillClimbing.
- La solución obtenida se le aplica una perturbación para generar una solución nueva.

El proceso se repite hasta que se cumple una condición de parada.

Métodos multiarranque: Greedy Randomized Adaptative Search Procedure (GRASP)

Cada iteración del algoritmo GRASP consta de:

- Fase de construcción de la solución inicial utilizando un algoritmo voraz (greedy),
- Fase de búsqueda local mediante un hillClimbing sobre la solución obtenida en la fase anterior.

Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones GRASP se devuelve como la solución aproximada.

Pseudocódigo genérico del GRASP

Procedure grasp( )
InputInstance()
While GRASP stopping criterion not satisfied
    ConstructGreedyRandomizedSolution(Solution)
    LocalSearch(Solution)
    UpdateSolution(Solution,BestSolutionFound)
    Return(BestSolutionFound)
End grasp

Simulated Annealing (Enfriamiento estadístico)

- Se realiza una búsqueda local partiendo de una sola solución, pero permitiendo una degradación temporal de la función objetivo.
- Se parte de una temperatura inicial, la cual se va decrementando en cada iteración para reducir la probabilidad ($e^{\frac{-\Delta E}{T}}$) de aceptar soluciones muy distintas de la actual.

Simulated Annealing

- Para temperaturas altas, el término $e^{\frac{-\Delta E}{T}} \longrightarrow 1$, implicando la aceptación de la mayoría de las modificaciones. En este caso el algoritmo se comporta como un paseo aleatorio simple en el espacio de búsqueda.
- Para temperaturas bajas, el término $e^{\frac{-\Delta E}{T}} \longrightarrow 0$, implicando el rechazo de la mayoría de las modificaciones. En este caso el algoritmo se comporta como una mejora iterativa clásica.
- Para valores intermedios de la temperatura, el algoritmo autoriza de manera intermitente movimientos que degradan la función objetivo.

Simulated Annealing: Pseudocódigo

Procedure SimulatedAnnealing( )
$s \leftarrow s_0$; $e \leftarrow f(s)$
$s_{best} \leftarrow s$; $e_{best} \leftarrow e$
currentTemp $\leftarrow$ initialTemp
While $currentTemp > finalTemp$
    for 1 to maxIntIter
        $s_{new} \leftarrow neighbour(s)$; $e_{new} \leftarrow f(s_{new})$
        $\Delta E = e_{new} - e_{best}$
        if $\Delta E < 0$ **then** $s_{best} \leftarrow s_{new}$; $e_{best} \leftarrow e_{new}$
        else if $e^{\frac{-\Delta E}{currentTemp}} > random(0, 1)$ **then** $s \leftarrow s_{new}$; $e \leftarrow e_{new}$
    Update currentTemp by factor A
    Return s_{best}
End SimulatedAnnealing

Índice

El problema del viajante de comercio

Ejemplo de matriz de costos, correspondiente a un TSP con $n = 6$ ciudades:

$$\begin{pmatrix} 0 & 6 & 2 & 1 & 4 & 10 \\ 6 & 0 & 3 & 4 & 3 & 1 \\ 2 & 3 & 0 & 2 & 8 & 3 \\ 1 & 4 & 2 & 0 & 5 & 6 \\ 4 & 3 & 8 & 5 & 0 & 9 \\ 10 & 1 & 3 & 6 & 9 & 0 \end{pmatrix}$$

TSP: Algoritmo heurístico constructivo voraz

```
1.   Escoger una ciudad al azar como ciudad de partida
2.   Mientras queden ciudades por visitar, escoger --de
     entre las no visitadas-- la que se encuentre a
     menor distancia de la última visitada
3.   Mostrar la solución
```

Pseudocódigo de un algoritmo heurístico constructivo voraz para el TSP

TSP: Algoritmo heurístico constructivo voraz

El heurístico constructivo voraz aplicado a la matriz de costes anterior

$$\begin{pmatrix} 0 & 6 & 2 & 1 & 4 & 10 \\ 6 & 0 & 3 & 4 & 3 & 1 \\ 2 & 3 & 0 & 2 & 8 & 3 \\ 1 & 4 & 2 & 0 & 5 & 6 \\ 4 & 3 & 8 & 5 & 0 & 9 \\ 10 & 1 & 3 & 6 & 9 & 0 \end{pmatrix}$$

proporciona la solución (se supone que la ciudad 4 es seleccionada como ciudad de partida):

$$4 - 1 - 3 - 2 - 6 - 5$$

El costo asociado a dicha gira es:

$$c_{41} + c_{13} + c_{32} + c_{26} + c_{65} + c_{54} = 1 + 2 + 3 + 1 + 9 + 5 = 21$$

TSP: Algoritmo heurístico constructivo voraz

- Con el heurístico anterior:
  - No hay garantía de que se alcance el óptimo global
  - El algoritmo es determinista
- *Versión estocástica* del algoritmo heurístico constructivo voraz: de entre las no visitadas seleccionar con una probabilidad inversamente proporcional a la distancia con respecto a la última ciudad visitada

TSP: Algoritmo heurístico de mejora iterativa

```
1. Seleccionar al azar una permutación de ciudades π
2. Repetir
            Obtener la permutación π' vecina de π
            con menor costo
            Hacer π := π'
    Hasta que el costo de ninguna permutación vecina
    sea menor que el costo de la solución actual
```

Algoritmo de mejora iterativa de una solución para el TSP. Ascenso (descenso) por la colina (*hillclimbing*)

Heurísticos en Optimización Combinatorial

A. Moujahid

Grupo de Inteligencia Computacional
Universidad del País Vasco UPV(EHU)
Curso 2014-2015