Tema 4. Métodos Determinístas.

PROGRAMACIÓN DINÁMICA

Imagen: http://tgsjuan.blogspot.com.es/2011/01/5-programacion-dinamica.html

Introducción

La técnica de Programación Dinámica, PD (Dynamic Programming) proporciona un procedimiento sistemático para determinar la combinación óptima de decisiones.

En contraste con las otras técnicas de programación (lineal, no lineal,…) no existe una formulación matemática estándar de resolución de un problema mediante PD.

Cada situación o problema a resolver con este enfoque requiere adaptar el sistema matemático y las ecuaciones a emplear.

Se requiere, por tanto, cierto grado de ingenio y conocimiento de la estructura general de los problemas de PD para reconocer cuándo y cómo un problema puede ser resuelto mediante esta técnica.

Prof. Harvey M. Wagner, UNC http://www.ime.unicamp.br/~andreani/MS515/capitulo7.pdf

Programación dinámica: Introducción

- Un caso típico es el conocido como "problema de la mochila":
  - Se tienen *n* objetos fraccionables y una mochila.
  - El objeto *i* tiene peso p_i y una fracción x_i ($0 \le x_i \le 1$) del objeto *i* produce un beneficio $b_i x_i$.
  - El objetivo es llenar la mochila, de capacidad *C*, de manera que se maximice el beneficio.

$$\text{maximizar} \quad \sum_{1 \le i \le n} b_i x_i$$

$$\text{sujeto a} \quad \sum_{1 \le i \le n} p_i x_i \le C$$

$$\text{con} \quad 0 \le x_i \le 1,\ b_i > 0,\ p_i > 0,\ 1 \le i \le n$$

Programación dinámica: Introducción

- Una variante: la "mochila 0-1"
  - x_i sólo toma valores 0 ó 1, indicando que el objeto se deja fuera o se mete en la mochila.
  - Los pesos, p_i, y la capacidad son números naturales.
    Los beneficios, b_i, son reales no negativos.

- Ejemplo:

  n=3 C=15

  (b_1,b_2,b_3)=(38,40,24)

  (p_1,p_2,p_3)=(9,6,5)

Programación dinámica: Introducción

- Si se parte de un “plan avaricioso” o “estrategia voraz”:
  - Tomar siempre el objeto que proporcione mayor beneficio por unidad de peso.
  - Se obtiene la solución:

    $(x_1,x_2,x_3)=(0,1,1)$, con beneficio 64
  - Sin embargo, la solución óptima es:

    $(x_1,x_2,x_3)=(1,1,0)$, con beneficio 78

- Por tanto, la estrategia voraz no calcula la solución óptima del problema de la mochila 0-1.

R. Bellman: *Dynamic Programming*, Princeton University Press, 1957.

- Técnica de programación dinámica
  - Se emplea típicamente para resolver problemas de optimización.
  - Permite resolver problemas mediante una secuencia de decisiones.

    Como el esquema voraz
  - A diferencia del "esquema voraz", se producen varias secuencias de decisiones y solamente al final se sabe cuál es la mejor de ellas.
  - Se basa en el **principio de optimalidad de Bellman**:

    "Cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema, también debe ser óptima respecto al subproblema que resuelve."

Programación dinámica: Introducción

- Supongamos que un problema se resuelve tras tomar un secuencia $d_1, d_2, \ldots, d_n$ de decisiones.
- Si hay d opciones posibles para cada una de las decisiones, una técnica de fuerza bruta exploraría un total de d^n secuencias posibles de decisiones (explosión combinatoria).
- La técnica de programación dinámica evita explorar todas las secuencias posibles por medio de la resolución de subproblemas de tamaño creciente y almacenamiento en una tabla de las soluciones óptimas de esos subproblemas para facilitar la solución de los problemas más grandes.

Problemas típicos

- Problema del transporte
- Problema de flujo con coste mínimo en red
- Problema de asignación
- Problema de la mochila (knapsack)
- Problema del emparejamiento (matching)
- Problema del recubrimiento (set-covering)
- Problema del empaquetado (set-packing)
- Problema de partición (set-partitioning)
- Problema del coste fijo (fixed-charge)
- Problema del viajante (TSP)
- Problema de rutas óptimas

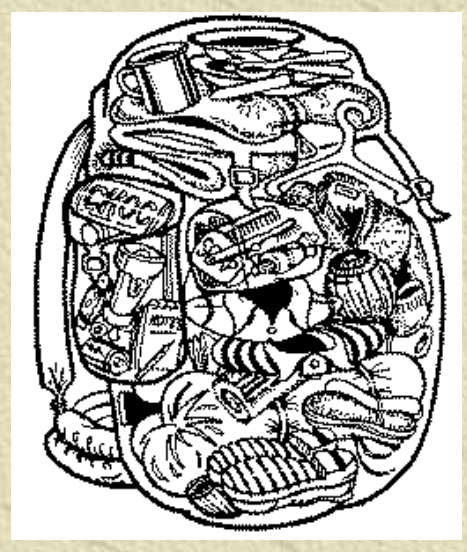
El problema de la mochila 0-1

- Sea *mochila*(*k*,*l*,*P*) el problema:

$$\text{maximizar} \quad \sum_{i=k}^{l} b_i x_i$$

$$\text{sujeto a} \quad \sum_{i=k}^{l} p_i x_i \leq P$$

$$\text{con} \quad x_i \in \{0,1\}, \quad k \leq i \leq l$$



  - El problema de la mochila 0-1 es *mochila* (1,*n*,*C*).

El problema de la mochila 0-1

- Ecuación de recurrencia hacia adelante:
  - Si $\vec{g}_j(c)$ es el beneficio (o ganancia total) de una solución óptima de *mochila*(*j*,*n*,*c*), entonces

    $$\vec{g}_j(c) = \max\left\{\vec{g}_{j+1}(c); \vec{g}_{j+1}(c-p_j)+b_j\right\}$$

    dependiendo de que el objeto *j*-ésimo entre o no en la solución (nótese que sólo puede entrar si $c\text{-}p_j \geq 0$).
  - Además, $\vec{g}_{n-1}(c) \geq 0$ para cualquier capacidad de c

Ambas ecuaciones permiten calcular, $\vec{g}_1(c)$ que es el valor de una solución óptima de *mochila*(1,*n*,*C*).

(Nótese que la ecuación de recurrencia es hacia adelante pero el cálculo se realiza hacia atrás.)

El problema de la mochila 0-1

- Ecuación de recurrencia hacia atrás:
  - Si $\vec{g}_j(c)$ es el beneficio (o ganancia total) de una solución óptima de *mochila*(1,*j*,*c*), entonces

$$\vec{g}_j(c) = \max\left\{\vec{g}_{j-1}(c); \vec{g}_{j-1}(c-p_j)+b_j\right\}$$

    dependiendo de que el objeto *j*-ésimo entre o no en la solución (nótese que sólo puede entrar si $c\text{-}p_j \geq 0$).

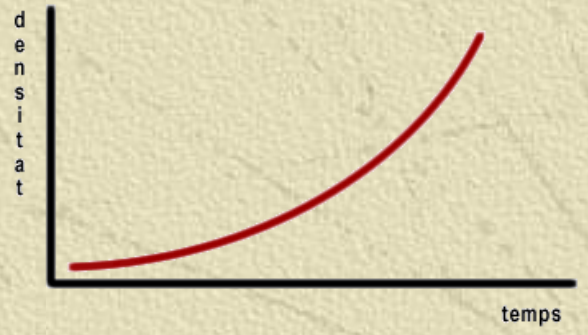
  - Además, $\vec{g}_0(c) = 0$ para cualquier capacidad de c

Ambas ecuaciones permiten calcular, $\vec{g}_n(c)$ que es el valor de una solución óptima de *mochila* (1,*n*,*C*).

(Ahora la recurrencia es hacia atrás pero el cálculo se realiza hacia adelante.)

El problema de la mochila 0-1

- Problema: ineficiencia
  - Un problema de tamaño *n* se reduce a dos subproblemas de tamaño (*n*-1).
  - Cada uno de los dos subproblemas se reduce a otros dos…
  - Por tanto, se obtiene un algoritmo exponencial.



El problema de la mochila 0-1

- Sin embargo, el número total de sub-problemas a resolver no es tan grande:

  La función $\vec{g}_j(c)$ tiene dos parámetros:

  - el primero puede tomar *n* valores distintos y
  - el segundo, *C* valores.

  ¡Luego sólo hay *nC* problemas diferentes!

- Por tanto, la solución recursiva está generando y resolviendo el mismo problema muchas veces.

El problema de la mochila 0-1

- Para evitar la repetición de cálculos, las soluciones de los subproblemas se deben almacenar en una tabla.
  - Matriz $n \times C$ cuyo elemento (j,c) almacena $\vec{g}_j(c)$
  - Para el ejemplo anterior:

    $n=3 \qquad C=15$

    $(b_1,b_2,b_3)=(38,40,24)$

    $(p_1,p_2,p_3)=(9,6,5)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_1 = 9$	0	0	0	0	0	0	0	0	0	38	38	38	38	38	38	38
$p_2 = 6$	0	0	0	0	0	0	40	40	40	40	40	40	40	40	40	78
$p_3 = 5$	0	0	0	0	0	24	40	40	40	40	40	64	64	64	64	78

$$\vec{g}_j(c) = \max\left\{\vec{g}_{j-1}(c); \vec{g}_{j-1}(c-p_j)+b_j\right\}$$

El problema de la mochila 0-1

- Consideraciones finales
  - Cada componente de la tabla g se calcula en tiempo constante, luego el coste de construcción de la tabla es $O(nC)$.
  - El algoritmo test se ejecuta una vez por cada valor de j, desde n descendiendo hasta 0, luego su coste es $O(n)$.
  - Si C es muy grande, entonces esta solución no es buena.
  - Si los pesos p_i o la capacidad C son reales, esta solución no sirve.

Problema del transporte

Minimizar el coste total de transporte entre los centros de origen y los de destino, satisfaciendo la demanda, y sin superar la oferta

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$s.a.$$

$$\sum_{i=1}^{m} x_{ij} = b_j , j = 1..n$$

$$\sum_{j=1}^{n} x_{ij} = a_i , i = 1..m$$

$$x_{ij} \geq 0, x_{ij} \in Z$$

x_{ij}: unidades a enviar de origen i a destino j
c_{ij}: coste unitario de transporte de i a j

a_i: unidades de oferta en el punto origen i
b_j: unidades de demanda en el punto destino j

Se supone oferta total igual a demanda total

Flujo con coste mínimo en red

Embarcar los recursos disponibles a través de la red para satisfacer la demanda a coste mínimo

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$s.a.$$

$$\sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i, j = 1..m$$

$$x_{ij} \geq 0, x_{ij} \in Z$$

x_{ij}: unidades enviadas de i a j (flujo)
c_{ij}: coste unitario de transporte de i a j

b_i:recursos disponibles en un nodo i

- oferta: $b_i>0$
- demanda: $b_i<0$
- transbordo: $b_i=0$

Se supone oferta total igual a demanda total

Problema de asignación

Minimizar el coste total de operación de modo que:

- cada tarea se asigne a una y sólo una máquina
- cada máquina realice una y sólo una tarea

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

s.a.

$$\sum_{i=1}^{m} x_{ij} = 1, j = 1..n$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1..m$$

$$x_{ij} \in \{0,1\}$$

x_{ij}: 1 si la tarea i se hace con la máquina j
c_{ij}: coste de realizar la tarea i con máquina j

n tareas
m máquinas

Si hay más máquinas que tareas se formula con desigualdades, y se resuelve con tareas ficticias

Problema de la mochila

Escoger un grupo de productos que maximice el valor total sin exceder el espacio disponible

$$Max \sum_{j=1}^{n} c_j x_j$$

$$s.a.$$

$$\sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \in \{0,1\}$$

n objetos

a_j: espacio que ocupa el objeto j
c_j: valor del objeto j

b: volumen de la mochila

x_j: 1 si se escoge el objeto j

Problema de emparejamiento

Distribuir un conjunto por parejas de tal forma que el valor sea máximo. Si hay elementos sin pareja: emparejamiento imperfecto. Si están en dos conjuntos, emparejamiento bipartito.

$$Max \sum_{i=1}^{2n-1} \sum_{j=1+1}^{2n} c_{ij} x_{ij}$$

s.a.

$$\sum_{k=1}^{i-1} x_{ki} + \sum_{j=i+1}^{2n} x_{ij} = 1, i = 1..2n$$

$$x_{ij} \in \{0,1\}$$

x_{ij}=1 si los elementos i y j son pareja

c_{ij}: valor de la pareja i-j

i<j

Problema de recubrimiento

Minimizar el coste de las actividades que en su conjunto cubren todas las características al menos una vez

$$Min \sum_{j=1}^{n} c_j x_j$$

$$s.a.$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1..m$$

$$x_j \in \{0,1\}$$

m características
n actividades

x_j=1 si la actividad j se realiza

c_j: coste unitario de la actividad j

a_{ij}=1 si la característica i está en la actividad j

A: matriz de incidencia

Problema de empaquetado

Maximizar el beneficio total de forma que hay que elegir conjuntos completos de actividades, y que no se realice una actividad dos veces

$$Min \sum_{j=1}^{n} c_j x_j$$

$$s.a.$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1, i = 1..m$$

$$x_j \in \{0,1\}$$

m actividades
n conjuntos de actividades

x_j=1 si se elige el subconjunto j

c_j: beneficio por realizar el conjunto j

a_{ij}=1 si el conjunto j incluye la actividad i

A: matriz de incidencia

Problema de partición

Si en el problema de recubrimiento o en el de empaquetado las desigualdades se cambian por igualdades

$$Min \sum_{j=1}^{n} c_j x_j$$

$$s.a.$$

$$\sum_{j=1}^{n} a_{ij} x_j = 1, i = 1..m$$

$$x_j \in \{0,1\}$$

m actividades
n conjuntos de actividades

x_j=1 si se elige el subconjunto j

c_j: beneficio por realizar el conjunto j

a_{ij}=1 si el conjunto j incluye la actividad i

A: matriz de incidencia

Problema del coste fijo

Decidir la cantidad de cada producto de modo que se minimicen los costes de producción y se satisfaga la demanda

$$Min \sum_{j=1}^{n} c_j x_j + \sum_{k=1}^{m} f_k y_k$$

s.a.

$$\sum_{j=1}^{n} x_{ij} \geq b_j$$

$$\sum_{j=1}^{n} a_{kj} x_j \leq M_k y_k, k = 1..m$$

$$x_{ij} \geq 0, y_k \in \{0,1\}$$

x_{ij}: unidades del producto j
c_j: coste unitario de producción de j

y_k=1 si se usa la instalación k
f_k: coste de arranque de la instalación k
a_{kj}=1 si el producto j usa la instalación k

b_j: demanda del producto j
M: número lo suficientemente grande

Problema del viajante

Encontrar un circuito que visite exactamente una vez cada ciudad empezando en la primera y que tenga longitud mínima

$$Min \sum_{(i,j)\in A} c_{ij}\, x_{ij}$$

$$s.a.$$

$$\sum_{i/(i,\ j)\in A} x_{ij} = 1 \ \ ,\forall j \in V$$

$$\sum_{j/(i,\ j)\in A} x_{ij} = 1, \forall i \in V$$

$$x_{ij} \in \{0,1\}$$

x_{ij}=1 si de i va directamente a j

c_{ij}: distancia entre i y j

A: conjunto de arcos

V: conjunto de nodos

$$\sum_{(i,j)\in A/\, i\in U,\, j\in V-U} x_{ij} \geq 1, \forall U \subset V \,/\, 2 \leq |U| \leq |V| - 2$$

$$\sum_{(i,j)\in A/\, i\in U,\, j\in U} x_{ij} \leq |U| - 1, \forall U \subset V \,/\, 2 \leq |U| \leq |V| - 2$$

Programación Dinámica: Problema del viajante

$$Min \sum_{k=1}^{n} \sum_{(i,j)\in A} c_{ij}\, x_{ijk}$$

$$s.a.$$

$$\sum_{i/(i,\,j)\in A} \sum_{k=1}^{n} x_{ijk} = 1, \forall j \in V$$

$$\sum_{j/(i,\,j)\in A} \sum_{k=1}^{n} x_{ijk} = 1, \forall i \in V$$

$$\sum_{(i,j)\in A} x_{ijk} = 1$$

$$\sum_{i/(i,\,j)\in A} x_{ijk} = \sum_{r/(j,r)\in A} x_{jrk+1}, \forall j \in V, \forall k$$

$$x_{ijk} \in \{0,1\}$$

Problema de rutas

Minimizar el coste total, visitando todos los clientes

N: clientes
M: vehículos

x_{ijk}=1 si el vehículo k visita j después de i
c_{ij}: coste unitario de transporte de i a j
d_{ij}: distancia de i a j
t_{ij}: tiempo de i a j

q_i: demanda
s_i: tiempo de descarga
δ_i: prioridad
Q_k: capacidad
r_o^k, d_o^k: período tiempo disponible
c_k: coste fijo por uso

Programación Dinámica: Problema de las rutas

$$Min \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij} \sum_{k=1}^{m} x_{ijk} + \sum_{k=1}^{m} c_k \sum_{j=1}^{n} x_{ojk}$$

s.a.

$$\sum_{i=0}^{n}\sum_{k=1}^{m} x_{ijk} = 1, j = 1..n$$

$$\sum_{i=0}^{n} x_{ijk} - \sum_{i=0}^{n} x_{jik} = 0, \forall j, \forall k$$

$$\sum_{i=1}^{n} q_i \sum_{j=0}^{n} x_{ijk} \leq Q_k, \forall k$$

$$\sum_{i=0}^{n}\sum_{j=0}^{n} t_{ij} x_{ijk} + \sum_{i=1}^{n} s_i \sum_{j=0}^{n} x_{ijk} \leq d_0^k - r_0^k, \forall k$$

$$\sum_{j=1}^{n} x_{ojk} \leq 1, k = 1..m$$

$$\sum_{i\in S}\sum_{j\in S}\sum_{k=1}^{m} x_{ijk} \leq |S| - 1, 2 \leq |S| \leq N - 2$$

Formulación con var. binarias

Restricciones disyuntivas

$$f(x) \le 0 \quad \text{ó} \quad g(x) \le 0 \quad \Rightarrow \quad f(x) \le \delta \overline{f}, \; g(x) \le (1-\delta)\overline{g}$$

K de N alternativas deben darse

$$f_1(x) \le \delta_1 \overline{f_1}, \; f_2(x) \le \delta_2 \overline{f_2}, \; f_n(x) \le \delta_{n2} \overline{f_n} \qquad \sum_{j=1}^{N} \delta_j = N - K, \delta \in \{0,1\}$$

Restricciones condicionales

$$f(x) > 0 \Rightarrow g(x) \le 0 \quad \text{equiv. a} \quad f(x) \le 0 \;\; \text{ó} \;\; g(x) \le 0$$

Decisiones contingentes

$$x \Rightarrow y \quad \Rightarrow \quad y \le x$$

La formulación

Traducción de los elementos básicos a expresiones matemáticas

Es un **Arte** que **mejora** con la **práctica**…

¡ **PRACTIQUEMOS!**

Ejemplo: (y ejercicios resueltos en

https://www.u-cursos.cl/forestal/2009/1/EF046/1/material_docente/objeto/481171)

Un Ingeniero Forestal para ir desde su oficina hasta un aserradero junto a una explotación maderera tiene que pasar por 3 provincias y varias ciudades de las mismas, antes de llegar a su destino. Quiere determinar: Cuál es la ruta de mínimo coste, y cuál es ese coste. Las posibles rutas y el costo asociado por Kms. de distancia en $, se ven en el siguiente esquema:

Enlaces con casos de aplicación:

http://www.nii.ac.jp/pi/n9/9_31.pdf

Progress in Informatics, No. 9, pp.31–34, (2012)

Special issue: Theoretical computer science and discrete mathematics

Research Paper

A dynamic programming algorithm for lot-sizing problem with outsourcing

Ping ZHAN[1]

[1]*Department of Communication and Business, Edogawa University*

ABSTRACT

Lot-sizing problem has been extensively researched in many aspects. In this manuscript, we give a dynamic programming algorithm scheme for lot-sizing problems with outsourcing.

KEYWORDS

Mixed integer problem, Lot-sizing, Dynamic programming, Minimum cost network flow

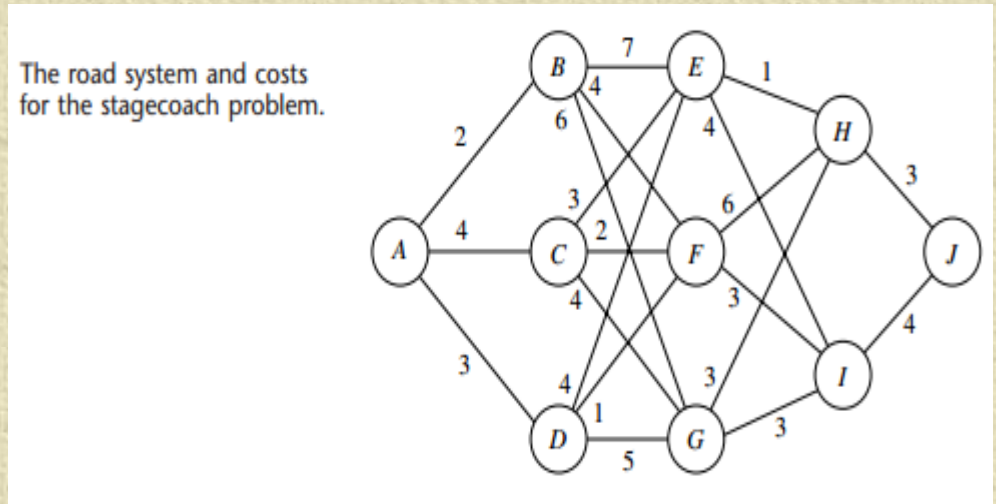
Enlaces con casos de aplicación:

Prof. Harvey M. Wagner, UNC http://www.ime.unicamp.br/~andreani/MS515/capitulo7.pdf

Problema de la diligencia ("camino más corto"): Un mítico buscador de fortunas de Missouri (*A*) decidió ir hacia el oeste para unirse a la fiebre del oro en California (*J*) en el siglo XIX.

En la figura aparecen las posibles rutas y paradas hasta llegar al destino *J*. En cada segmento de unión aparecen los costes c_{ij} asociados a cada ruta.

Para optimizar, los c_{ij} pueden equivaler a las distancias, en este caso.



The road system and costs for the stagecoach problem.