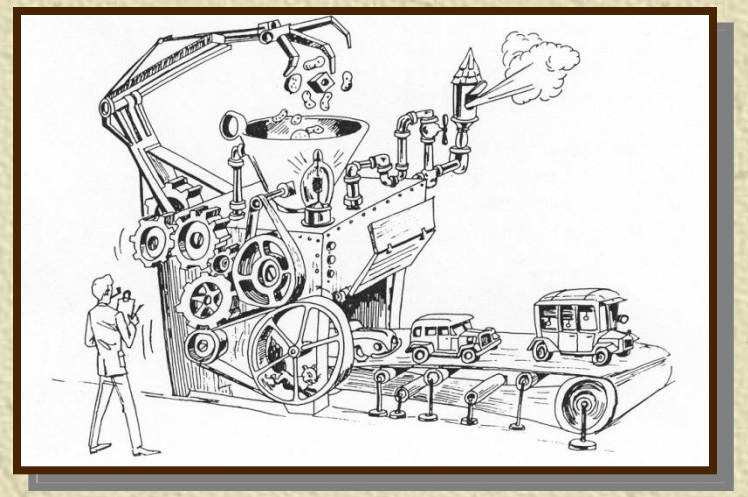
OTROS TIPOS DE MODELOS DE PROGRAMACIÓN DETERMINISTA

ESPERANZA AYUGA TÉLLEZ

PROGRAMACIÓN

- ENTERA
- NO LINEAL
- DINÁMICA



PROGRAMACIÓN ENTERA

PROGRAMACIÓN ENTERA INTRODUCCIÓN

- No siempre es admisible que las variables de un PL tomen valores continuos:
  - Decisiones dicotómicas (si-no)
  - Decisiones que deben tomarse en unidades discretas
- Problema de Programación entera:

  Cuando en un problema existen variables que deben tomar valores discretos y la función objetivo y las restricciones son lineales.
- Problema de Programación binaria o 0-1:

  Cuando los valores que pueden tomar las variables discretas son tan sólo 0 o 1.

PROGRAMACIÓN ENTERA INTRODUCCIÓN

- La PE tiene gran cantidad de aplicaciones en todos los campos.
- Hay problemas que no pueden resolverse con las técnicas actuales por:
  - Disponibilidad de tiempo de ordenador
  - Capacidad de memoria
- Para evitar esto parece sensato calcular la solución de un PE redondeando la solución continua.
- Pero el redondeo no es aconsejable debido a:
  - La solución redondeada no es necesariamente óptima. En muchos casos, ni siquiera estará cera del óptimo.
  - La solución redondeada puede no ser factible.

PROGRAMACIÓN ENTERA
INTRODUCCIÓN

- EJEMPLO

  Max (z) = $y_1 + y_2$

  sujeto a:

  $-2\ y_1 + 2\ y_2 \geq 1$

  $-8\ y_1 + 10\ y_2 \leq 13$

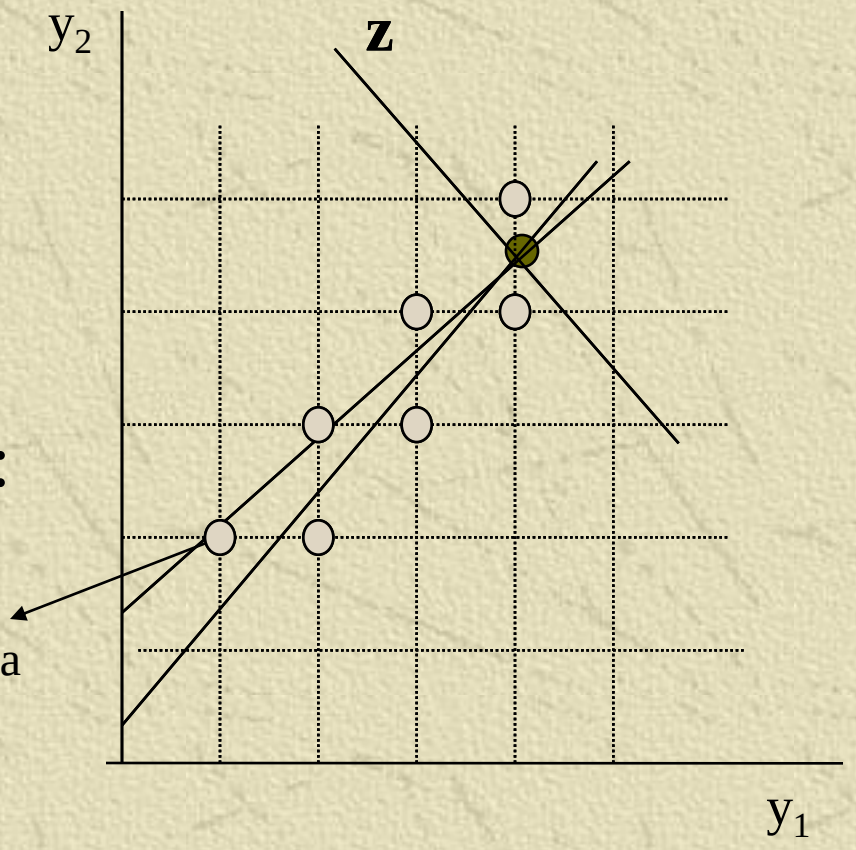
  y_1, y_2 {0,1,2,...}

  La solución continua es:

  $y_1 = 4$

  $y_2 = 4{,}5$ Solución Entera

  $z = 8{,}5$



PROGRAMACIÓN ENTERA PURA (PE)

- Todas las variables toman valores enteros.
- Programación Entera:

  $\text{Max } (z) = c^T y$

  sujeto a:

  $Ay < b; \qquad y_j \{0,1,2,...\} \quad j = 1,2,... n$

- Programación Entera Binaria:

  $\text{Max } (z) = c^T y$

  sujeto a:

  $Ay < b; \qquad y_j \{0,1\} \qquad j = 1,2,... n$

PROGRAMACION ENTERA MIXTA (MIP)

- Algunas variables de decisión están restringidas a tomar valores enteros, mientras que otras pueden tomar valores continuos.
- Un problema de MIP puede expresarse como:

$$\text{Max } (z) = c^T x + d^T y$$

sujeto a:

$$Ax + By < b; \qquad y_j \{0,1,2,...\} \; j = 1,2,... \; n$$

- En este caso:
  - x: vector de variables que toman valores continuos.
  - y: contiene n variables enteras

TIPOS DE ALGORITMOS DE PROGRAMACIÓN ENTERA

- Algoritmos de PE Pura (IP)
  - **Enumerativos:**
    - Branch and Bound
    - Balas, etc
  - **De planos cortantes:**
    - Gomory
    - Otros
- Algoritmos de PE Mixta (MIP)
  - **Enumerativos:**
    - Branch and Bound
    - Balas, etc
  - **De descomposición dual:**
    - Benders
    - Otros

ALGORITMOS ENUMERATIVOS

- Estos algoritmos obtienen la solución en base a enumerar, implícita o explícitamente, todas las soluciones posibles y escogiendo la mejor de todas ellas.
- ENUMERACIÓN EXPLÍCITA:
  - Calcular todas las posibles soluciones y escoger la mejor de ellas.
  - Este método tiene graves inconvenientes:
    - Ejemplo: En PE 0-1 el número de posibles soluciones es $2^{n^o var.enteras}$

      4 variables: $2^4 = 16$ soluciones o nodos

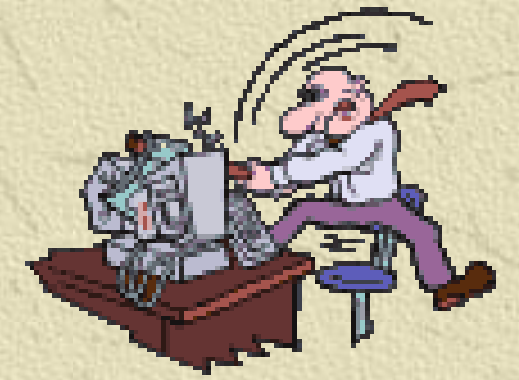
      10 variables: $2^{10} = 1.024$ nodos

      20 variables $2^{20} = 1.048.276$ nodos
  - En problemas complejos, un ordenador no sería capaz de enumerar todas las posibles soluciones.

ALGORITMOS ENUMERATIVOS

ENUMERACION IMPLÍCITA:

- Aplican un conjunto de reglas para evitar enumerar soluciones infactibles o peores que la mejor solución factible que se haya localizado hasta el momento.
- La familia de algoritmos enumerativos más importante es la de los algoritmos de Branch and Bound (BB).
- Prácticamente todos los códigos comerciales de PE están basados en un algoritmo del tipo BB.



ALGORITMOS DE BRANCH AND BOUND

- Tienen su origen en un trabajo de Land y Doig de 1960:

"An automatic method of solving discrete programming problems". *Econometrica*


ECONOMETRICA

BRANCH

BOUND

ALGORITMOS DE BRANCH AND BOUND

- Esta metodología se ha sofisticado posteriormente pero, la idea básica es muy sencilla.

- EJEMPLO:

  $\text{Max } (z) = x_1 + 3\, x_2$

  sujeto a:

  $x_2 \leq 1{,}87; \qquad 22\, x_1 + 34\, x_2 \leq 105$

  $x_1 \in \{0,1,2,...\} \qquad x_2 \in \{0,1,2,...\}$

  La solución continua del problema es:

  $x_1 = 1{,}88; \quad x_2 = 1{,}87; \quad z = 7{,}49$

ALGORITMOS DE BRANCH AND BOUND

- *Bound*:
  - Asociamos a esta solución el nodo 0.
  - Cualquier solución entera tendrá un valor de la función objetivo menor o igual que z = 7,49
  - Esto se debe a que al poner la condición de integralidad el problema se hace más restrictivo.
- *Branch:*
  - A partir del nodo 0 se generan 2 problemas añadiendo a uno de ellos $x_1 \geq 2$ (nodo1) y $x_1 \leq 1$ (nodo2).
  - Es decir, buscamos la solución a cada lado de la variable que está más cercana a tomar un valor entero.

ALGORITMOS DE BRANCH AND BOUND

- Las soluciones a ambos problemas son:

nodo 1 ($x_1 \geq 2$): $x_1 = 2$; $x_2 = 1{,}79$; $z = 7{,}38$

nodo 2 ($x_1 \leq 1$): $x_1 = 1$; $x_2 = 1{,}87$; $z = 6{,}61$

- No tenemos soluciones enteras, por tanto, debemos seguir.
- En el nodo 1 el valor de la función objetivo es mayor. Seguimos ramificando el nodo 1:
  - nodo 3 ($x_2 \geq 2$)
  - nodo 4 ($x_2 \leq 1$)

nodo 3 ($x_2 \geq 2$): INFACTIBLE

nodo 4 ($x_2 \leq 1$): $x_1 = 3{,}23$; $x_2 = 1$; $z = 6{,}23$

ALGORITMOS DE BRANCH AND BOUND

- *Branch:*
  - En el nodo 4 el valor de z es inferior al del nodo 2. Debemos seguir por el nodo 2.
- *Bound:*
  - A partir del nodo 2 se generan 2 nuevos nodos:

    nodo 5 ($x2 \geq 2$) nodo 6 ($x2 \leq 1$)

    nodo 5 ($x_2 \geq 2$): INFACTIBLE

    nodo 6 ($x_2 \leq 1$): $x_1 = 1$; $x_2 = 1$; $z = 4$
  - Ya tenemos una solución entera, pero el valor de z es menor que en el nodo 4.
- *Branch:*
  - Como el valor de z en el nodo 6 es menor que en el 4, ramificamos por el 4.

ALGORITMOS DE BRANCH AND BOUND

- Bound:
  - Del nodo 4 surgen dos nuevos nodos:

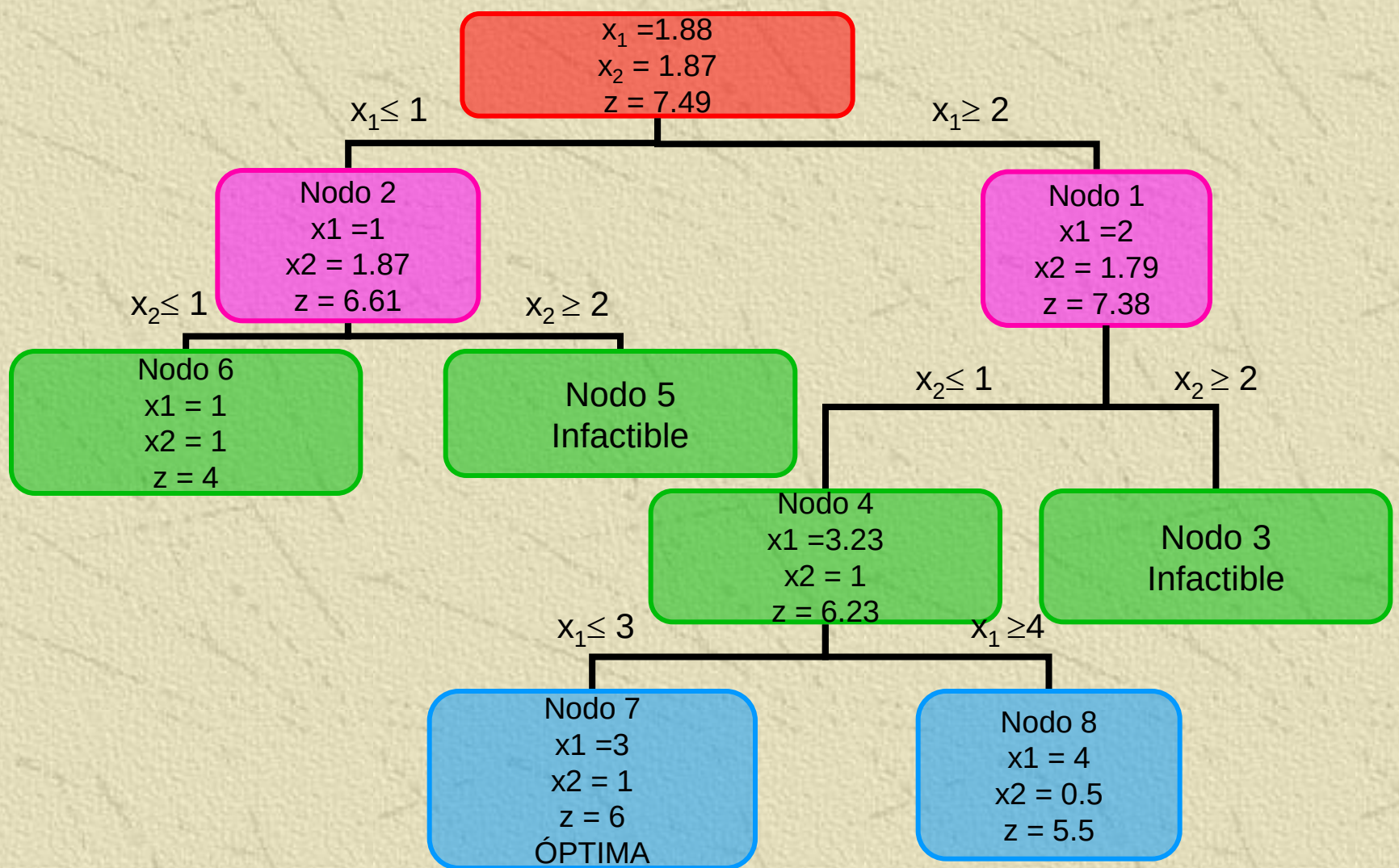
    nodo 7 ($x_1 \leq 3$) nodo 8 ($x_1 \geq 4$)

    nodo 7 ($x_1 \leq 3$): $x_1 = 3$; $x_2 = 1$; $z = 6$

    nodo 8 ($x_1 \geq 4$): $x_1 = 4$; $x_2 = 0,5$; $z = 5,5$
- La solución del nodo 7 es entera y mejor que la del nodo 6.
- La solución del nodo 8 es continua y peor que la 7.
- No queda ninguna posibilidad de mejorar el valor de la función objetivo.
- Por tanto tenemos la siguiente solución entera:

  $x_1 = 3$ $x_2 = 1$ $z = 6$

ALGORITMOS DE BRANCH AND BOUND



PROGRAMACIÓN NO LINEAL

Maximizar $Z(X)$ → **Función no lineales**

Sujeto a:

$$g_1(X) \leq b_1$$
$$g_2(X) \leq b_2$$

$$g_m(X) \leq b_m$$

Ejemplo: selección de cartera:

Minimizar varianza del portafolio $V(X) = \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha j \sigma_{ij}$

Sujeto a:

$$R(X) = \sum_{j=1}^{n} \alpha_j X_j \geq R$$

$$\sum_{i=1}^{n} \alpha_i = 1$$

$$\alpha_i \geq 0$$

$$\sum_{i=1}^{n} p_i \alpha_i \leq B$$

Donde R es un rendimiento mínimo aceptable para el decisor

P_i: precio de cada acción

B= dinero presupuestado

Ejemplo:

un individuo desea invertir en bolsa y ha considerado 3 posibles tipos de valores X, Y, Z y quiere determinar qué parte de su presupuesto dedicará a cada uno de ellos para ganar al menos el 12%. No desea que ningún tipo de valor supere el 75% de la cartera y quiere minimizar el riesgo de la misma.

$R(X)=30\%$ $\sigma^2_x=3$

$R(Y)=20\%$ $\sigma^2_y=2$

$R(Z)=8\%$ $\sigma^2_z=1$

$\sigma_{xy}=2$, $\sigma_{xz}=-1$, $\sigma_{yz}=-0.8$

Solución:

Minimizar $3x^2+2y^2+z^2+2xy-xz-0.8yz$

(x, y, z porciones en que se invertirá X ,Y y Z)

Sujeto a: $1.3x+1.2y+1.08z \geq 1.12$

$x \leq 0.75$

$y \leq 0.75$

$z \leq 0.75$

$x+y+z=1$

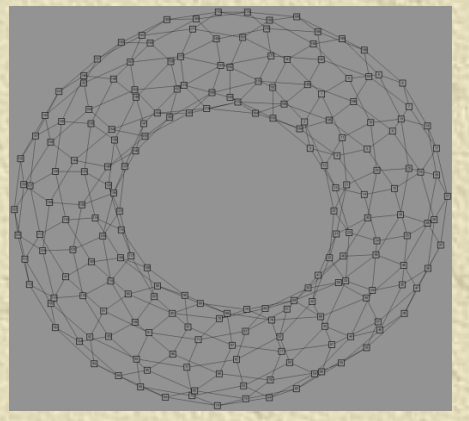
Solución por solver EXCEL: x=0.50 y=0.30 z=0.20,

Varianza 1.12, rendimiento=1.22

PROGRAMACIÓN NO LINEAL

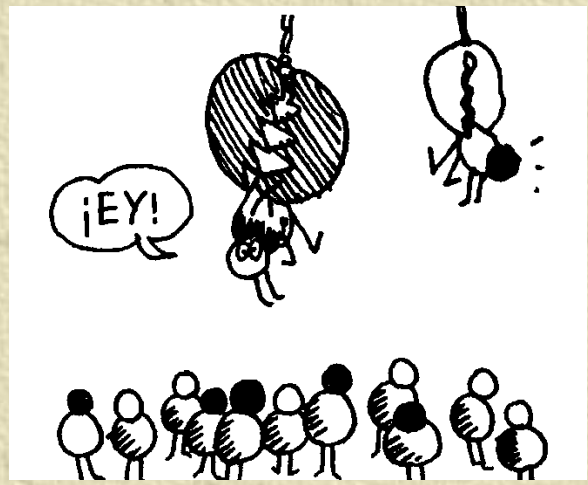
Clasificación de métodos

- Deterministicos (Duros)
- Heuristicos

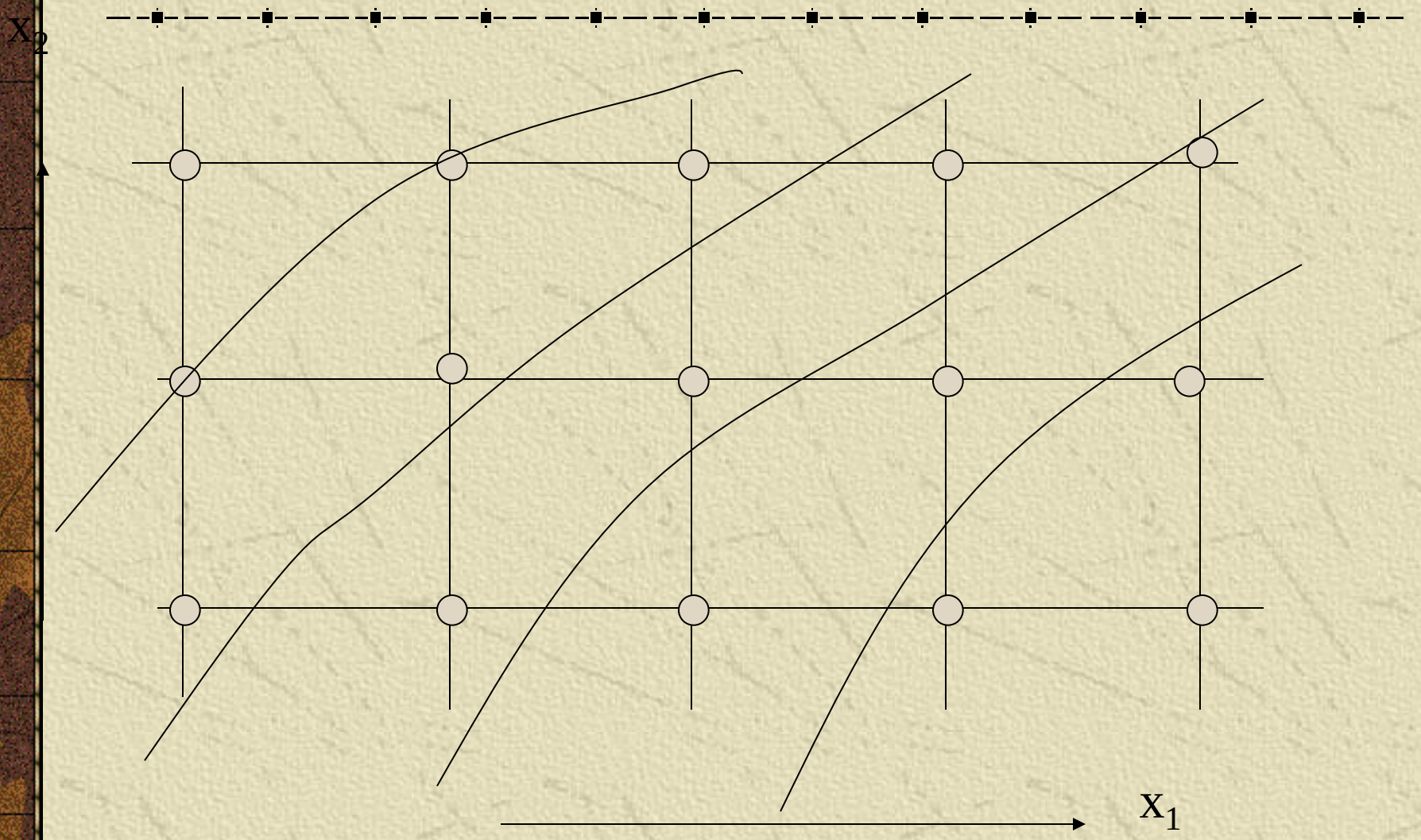


Métodos determinísticos

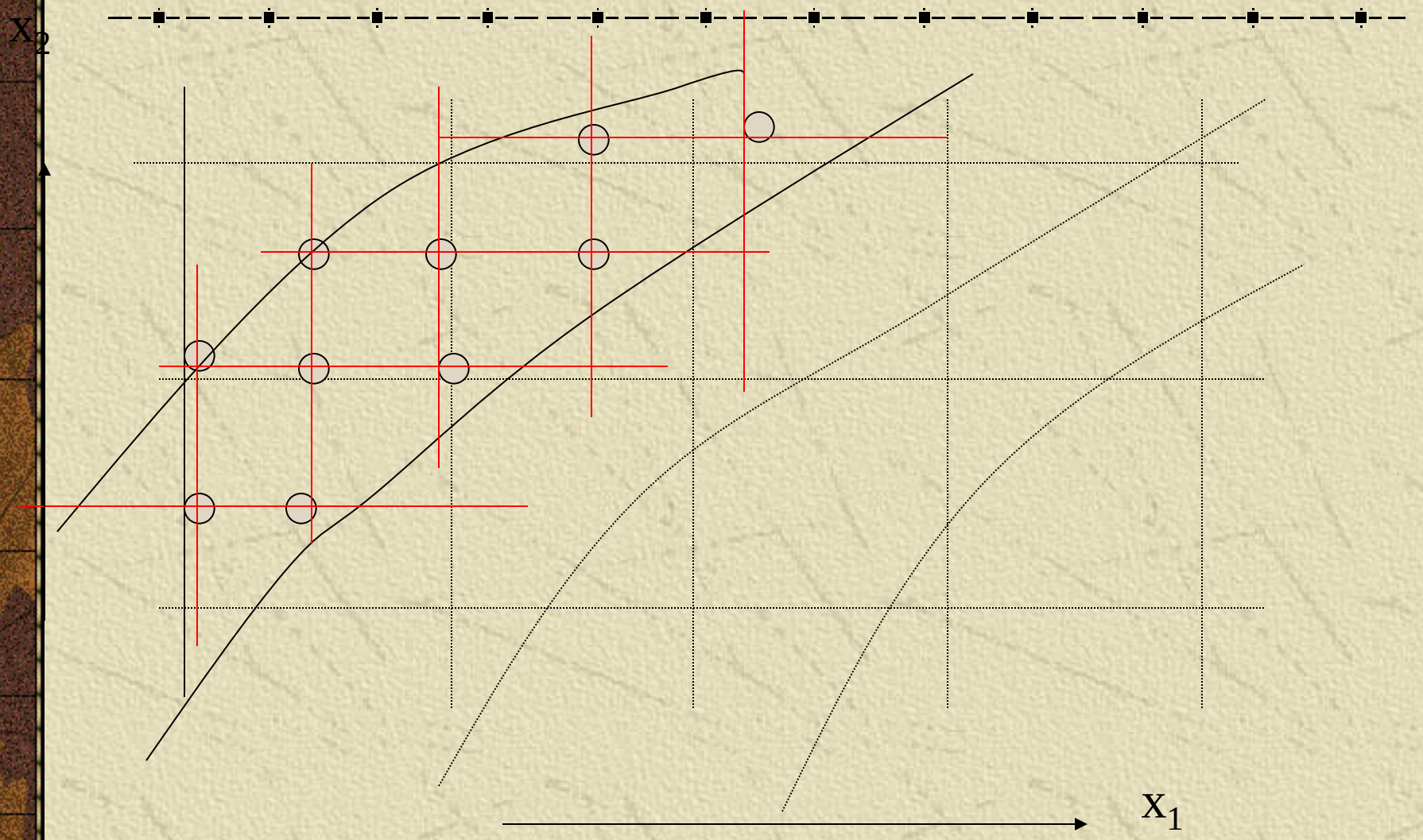
1. Método de la malla uniforme
2. Método de la muestra aleatoria
3. Prueba y error
4. Gradiente



1. Malla Uniforme



1. Malla Uniforme



2. Muestra aleatoria

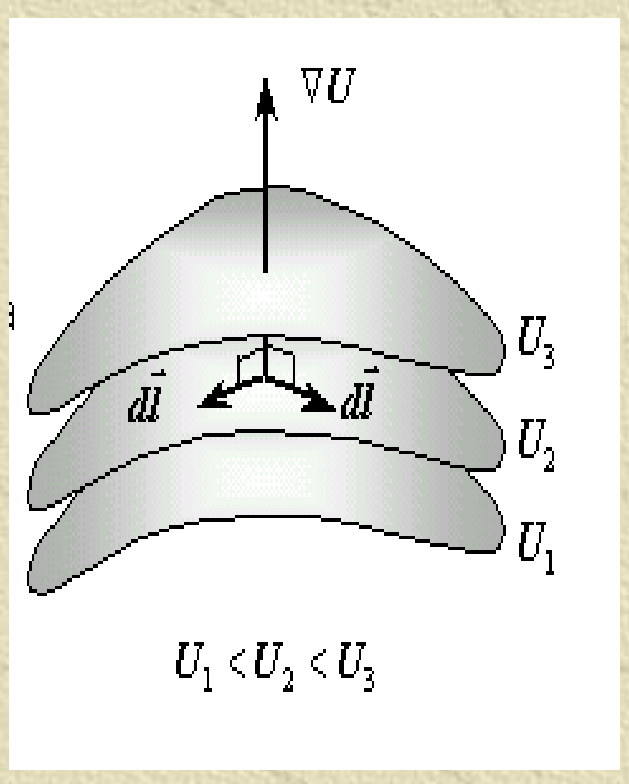
- Selecciona al azar varios valores de las variables de decisión. Así se aproxima a la función de distribución de la Función Objetivo y se calcula la probabilidad de que el mejor de las alternativas sea superado por otro.

3. Prueba error

- Haciendo pruebas “A mano”.
- Usar análisis del experto para la búsqueda

4. Gradiente

- Hace uso de la información sobre el gradiente de la función objetivo en un punto "actual" y se avanza en la dirección del vector gradiente hasta un punto en el cual se repite el proceso.



- Para pequeños cambios de cada x_j, Δx_j, se dan cambios $(\Delta z)_j$, aproximándose la derivada parcial según esa variable por:

$$\frac{\partial z(x)}{\partial x_j} \approx \frac{(\Delta z)_j}{\Delta x_j}$$

- Dado ρ=parámetro de búsqueda , Se multiplica el vector gradiente así determinado para avanzar en es dirección:

$$\Delta \vec{x} = \rho \nabla \vec{z}$$

- Y obtener un nuevo punto: $\vec{x}' = \vec{x} + \Delta \vec{x}$

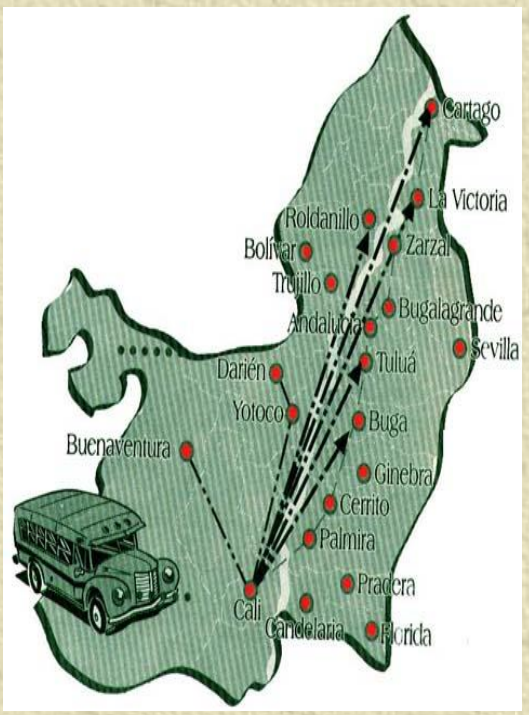
- El algoritmo termina cuando todos los $(\Delta z)_j$ son negativos o 0.

Métodos Heurísticos

Técnicas para resolver problemas de gran dificultad basados en principios generales de Inteligencia Artificial (IA). Casi todos se han desarrollado para resolver problemas combinatoriales, pero se usan en PNL también

Problemas combinatoriales

- Transporte
- Selección de rutas
- Selección de proyectos
- Problemas de expansión
- Problemas de Control
- Priorización



Objetivo de las técnicas heurísticas combinatoriales

Desarrollar técnicas eficientes para encontrar un mínimo o un máximo valor de una función de muchas variables discretas independientes y con gran cantidad de soluciones, sin tener que conocer los valores del objetivo de todas esas soluciones factibles.

Métodos Meta-heurísticos

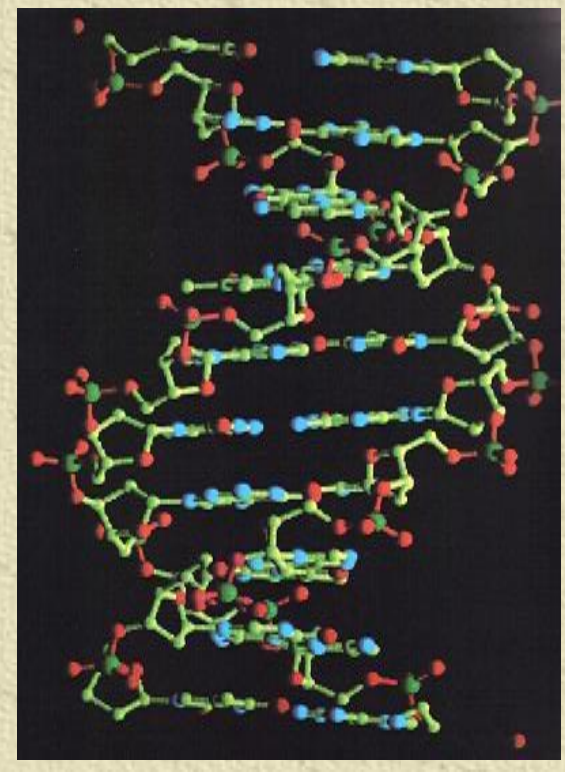
- Generalmente, comienzan con una configuración conocida del sistema.
- Aplican luego una operación estándar de re-asignar a todas las partes del sistema en turno, hasta que se descubre una configuración que mejore la función objetivo.
- La configuración re-asignada se convierte en una nueva configuración del sistema
- el proceso continua hasta que no pueda obtenerse mejoras.

Otras características...

- Esta búsqueda puede quedarse en un óptimo local por eso es recomendable hacer el proceso varias veces comenzando desde diferentes configuraciones generadas aleatoriamente y salvar los mejores resultados.

- Generalmente usan reglas de transición probabilísticas, no determinísticas

Métodos Meta-heurísticos de Optimización más conocidos



- Algoritmos genéticos
- Enfriamiento Simulado
- Búsqueda Tabú

PROGRAMACIÓN DINÁMICA

Programación dinámica: Introducción

- Recordemos el 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

- Recordar la 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.

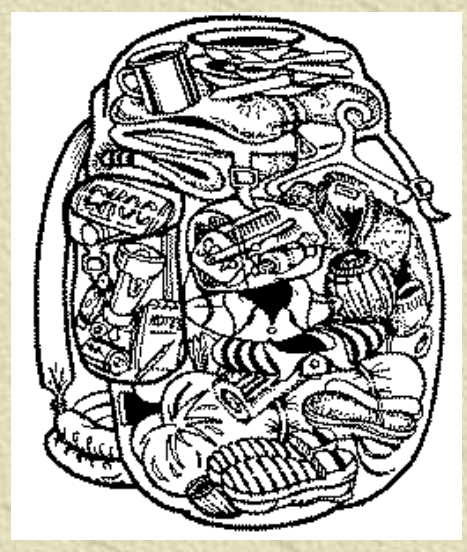
El problema de la mochila 0-1

- 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 $c\text{-}p_j \geq 0$).

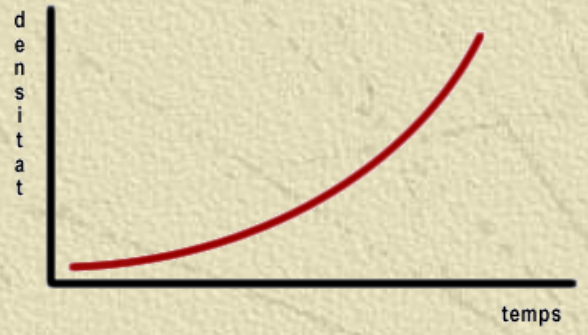
  - Además, $\vec{g}_0(c) = 0$ para cualquier capacidad de c

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.

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

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

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

$$s.a.$$

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

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

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

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

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

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

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

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

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

s.a.

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

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

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

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

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

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

Formulación con var. binarias

Restricciones disyuntivas

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

K de N alternativas deben darse

$$f_1(x) \le \delta_1 \overline{f_1}, \quad f_2(x) \le \delta_2 \overline{f_2}, \quad f_n(x) \le \delta_{n2} \overline{f_n}$$

$$\sum_{j=1}^{N} \delta_j = N - K, \delta \in \{0,1\}$$

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

$$x \Rightarrow y \qquad \Rightarrow \qquad y \le x$$

La formulación

Traducción de los elementos básicos a expresiones matemáticas

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

¡ **PRACTIQUEMOS!**