Investigación Operativa

Resumen y notas de clases – Temas Segundo parcial

Ultima modificación el domingo 25 de junio de 2006 a las 23:32:26

Modelos de inventario o stock

Tamaño de lote óptimo para minimizar costos CTE (Costo total esperado)

Objetivo : minimización de CTE de la gestión de stock.

Costos

K = costo o producción } costo administrativo **independiente de la cantidad** (papeleo, Preparación, etc)

b = costo unitario del producto $/unidad

c1 = costo por almacenamiento (deposito, seguro, refrigeración,etc) $/u prod, u de tiempo

c2 = costo por agotamiento $/u prod, u de tiempo ; (perdida de clientes, etc)

Stock en el tiempo

Sin agotamiento : el lote se repone al llegar a cero

Con agotamiento : hay un gap de entrega inmediata, y luego hay demanda insatisfecha por ordenes pendientes.

Notación

Q = Tamaño del lote

q* ó $\mathbf{q_0}$ = Tamaño del lote óptimo (minimiza el CTE)

CTE = costo total esperado

CTE* ó $\mathbf{CTE_0}$ = CTE óptimo (mínimo)

S = nivel de stock, puede o no coincidir con Q

S=S(t) } gráfico en el tiempo

Ti = tiempo entre 2 lotes ; $Ti=\frac{T}{n}$

Ti* ó Ti0 = tiempo óptimo entre lotes

T = tiempo de gestión

D = demanda total en T

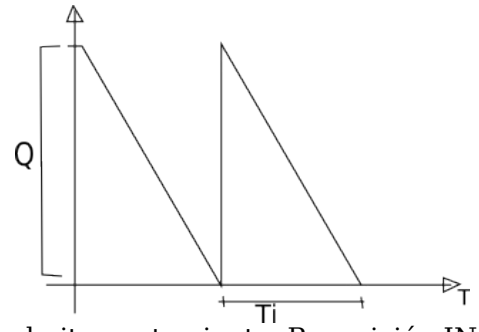
n = cantidad de veces que se produce/compra un lote

$$n \cdot q = D \;\; ; \;\; n \cdot T_i = T$$

$$n = \frac{D}{Q} \;\; ; \;\; n = \frac{T}{t}$$

Modelo de Wilson

D = demanda conocida, constante



NO admite agotamiento. Reposición INSTANTANEA.

$$CTE_i = k + bq + \frac{1}{2} c_1 T_i q$$

$$CTE = n \cdot CTE_i \rightarrow CTE = \frac{D}{q}\left(k + bq + \frac{1}{2} c_1 T_i q\right) = \frac{DK}{q} + b\,D + \frac{1}{2} c_1 + q$$

Optimización

$CTE(q) = \frac{Dk}{q} + bD + \frac{1}{2} c_1 T\, q$ } costo esperado para lote de tamaño q

$q^* = \sqrt{\frac{2\mathrm{kD}}{Tc_1}}$ } Tamaño del lote óptimo

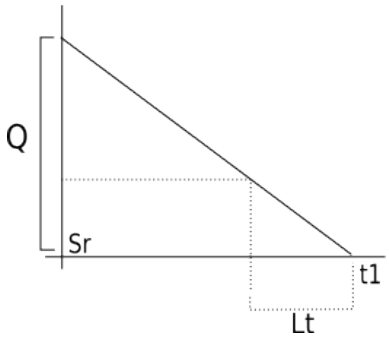
$CTE^* = \sqrt{2\mathrm{kD}\, Tc_1} + bD$ } costo total esperado óptimo

$T_i^* = \sqrt{\frac{2\mathrm{kT}}{Dc_1}}$ } tiempo entre lotes

Nota: Como el CTE crece mas lento hacia la derecha, si no puedo usar q*, conviene pedir/producir algo > a q*.

$S_r = L_t \cdot d$ } stock de reposición , indica cuando tengo que hacer otro pedido.

"d" es la demanda diaria, Lt es lead time.



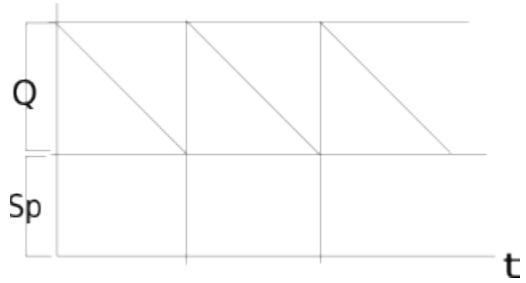
Modelo con stock de protección o seguridad

D = demanda conocida, constante

No admite agotamiento.

Reposición instantánea.

Sp = stock de protección o seguridad.



$q^* = \sqrt{\frac{2kD}{Tc_1}}$ } Tamaño del lote óptimo ; ídem Wilson

$CTE^* = \sqrt{2k\,D\,T\,c_1} + bD + c_1 S_p T$ } costo lote óptimo

Modelo con agotamiento

Demanda conocida, reposición instantánea, **admite agotamiento. Matemáticamente, este modelo siempre conviene mas que el tener stockeado, porque reduce el costo.**

Costos involucrados

K , b , c1, c2 = costo por agotamiento (medido en $ x u prod x u tiempo)

Ti1 = tiempo antes de agotar stock (aca corre costo c1)

Ti2 = periodo de agotamiento, tiempo desde agotar stock hasta reponer (aca corre costo c2)

$$T_{i2} = \frac{Tq}{D} - \frac{ST}{D} = \frac{T}{D}\cdot(q-S) \quad ; \quad T_i = T_{i1} + T_{i2} \quad ; \quad T_i^* = \sqrt{\frac{2kT}{Dc_1}}\sqrt{\frac{c_1+c_2}{c_2}}$$

$$n = \frac{D}{q} \qquad n\cdot T_i = T \qquad T_i = \frac{T_q}{D}$$

Para el periodo i, CTE

$$CTE_i = k + bq + \frac{1}{2}c_1 S T_{i1} + \frac{1}{2}c_2 (q-S) T_{i2}$$

Para todo el tiempo, el CTE es $CTE = n\cdot CTE_i$

Lote óptimo: $q^* = \sqrt{\frac{2kD}{Tc_1}}\cdot\sqrt{\frac{c_1+c_2}{c_2}}$

Stock óptimo : $S^* = \frac{c_2}{c_1+c_2}\cdot q^*$

CTE óptimo : $CTE^* = \sqrt{2\,k\,D\,T\,c_1}\cdot\sqrt{\frac{c_2}{c_1+c_2}} + bD$

$S_r = L_t d - (q^* - S^*)$

Redes / grafos

$x=\left(x_{1,}x_{2,}...,x_{n}\right)$; $\tau=$aplicacion entre los vertices

$x\to x$; $G=\left(x,\tau\right)$ } red de orden n

Grafo no conexo : no se relaciona con el resto de la red (x7 en el ejemplo)

Grafo conexo : si para dos cualquiera de sus vértices existe una cadena que los una.

Componente fuertemente conexo: enlaces de ida y vuelta, ejemplo x5 y x6

Camino es una sucesión de vértices entre los cuales hay arcos de forma tal que el extremo final de un arco es el inicial del siguiente.

Longitud del camino : El número de arcos del camino.

Cadena : camino en un grafo no dirigido.

Árbol : un grafo conexo sin ciclos.

Tipos de camino:

- sencillo : no pasa 2 veces por el mismo arco
- elemental : no pasa 2 veces por el mismo vértice
- circuito: camino cerrado, el extremo final del ultimo arco coincide con el inicial del primero.

Semi grado interior : cantidad de arcos que llegan a un vértice. (suma por columna de la matriz adyacencias)

Semi grado exterior : cantidad de arcos que parten desde un vértice. (suma por fila de la matriz adyacencias)

Red orientada : tienen sentido (flechas), tiene vértices y arcos ; camino es una sucesión de arcos.

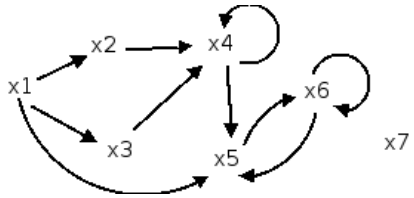
Red no orientada : no tiene sentido (solo lineas), tiene vértices y aristas ; cadena : sucesión de aristas, la cadena cerrada forma un ciclo.

Matriz de adyacencias

La matriz de adyacencias muestra las conexiones, va un 1 en cada conexión, 0 en desconexión.

La matriz es de n x n, en este caso de ejemplo, 7x7

Ejemplo de red:



Matriz:

	x1	x2	x3	x4	x5	x6	x7
x1		1	1		1		
x2			1	1			
x3				1			
x4				1	1		
x5						1	
x6					1	1	
x7							

Si elevo la matriz al cuadrado, me da la cantidad de caminos de longitud 2 que hay ; si hago a la potencia n, obtengo la cantidad de caminos de longitud n.

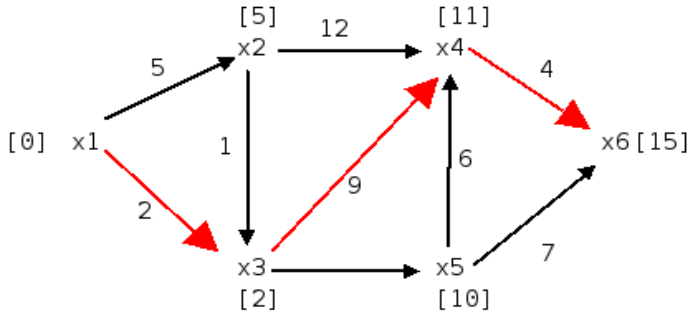
Algoritmo de Ford

Encuentra el camino de longitud ponderada mínima entre dos vértices de una red.

Longitud ponderada es una cantidad asociada a los arcos de una red que se necesita minimizar.

Ejemplo:

	x1	x2	x3	x4	x5	x6
x1		5	2			
x2			1	12		
x3				9	8	
x4						4
x5				6		7
x6						



En este ejemplo : óptimo mínimo = [15]

Los valores entre [] son del algoritmo. Son la menor distancia ponderada desde el vértice anterior.

Luego vuelvo, armando con los menores. Ej : x6[15]-4 = [11]x4 y asi hasta volver al [0] de x1

Algoritmo de Ford-Fulkerson

Se aplica a redes de transporte, para encontrar el flujo máximo de la red. El objetivo es encontrar el flujo máximo que puede circular por la red de transporte.

Dicho en otras palabras, la idea es encontrar conductos subutilizados entre la fuente y el destino, que pueden aumentarse hasta que una restricción de capacidad detenga el aumento.

Para aplicar el algoritmo, la red debe cumplir:

- una entrada al sistema
- una salida del sistema
- capacidad de cada arco c(u)
- 0<=f(u)<=c(u) ; f(u) = flujo en el arco, c(u) = capacidad del arco
- en cada vértice de la red, la suma de lo que llega es igual a la suma de lo que sale

Si f(u) = c(u), entonces el arco esta saturado.

Si f(u) < c(u), entonces hay capacidad ociosa.

El método es así:

- primero, genero un flujo completo (no que tenga por lo menos un arco saturado en cada camino de x0 a Z)
- ver si es el óptimo, para esto usamos un sistema de marcas:

  Siempre marcamos x0 con [+] y luego aplicamos

  - si xi (el actual) esta marcado, entonces marcamos xj (el siguiente) con la marca [+i], solo si f(u) < c(u), o sea, solo si no esta saturado.
  - Si xj (el siguiente) esta marcado, entonces marcamos xi con [-j], solo si f(u) > 0, o sea, si hay flujo.

  Termina cuando es imposible seguir, si no alcanzamos Z, lo obtenido en el paso anterior es óptimo, si con la marca alcanzamos Z (o sea, marcamos Z con algún [], entonces es posible hacer una modificación que mejore el flujo obtenido en el paso anterior.

Para mejorar, primero tomo la cadena, del final al principio, usando los índices [i]

Entonces, si en la cadena la conexión es:

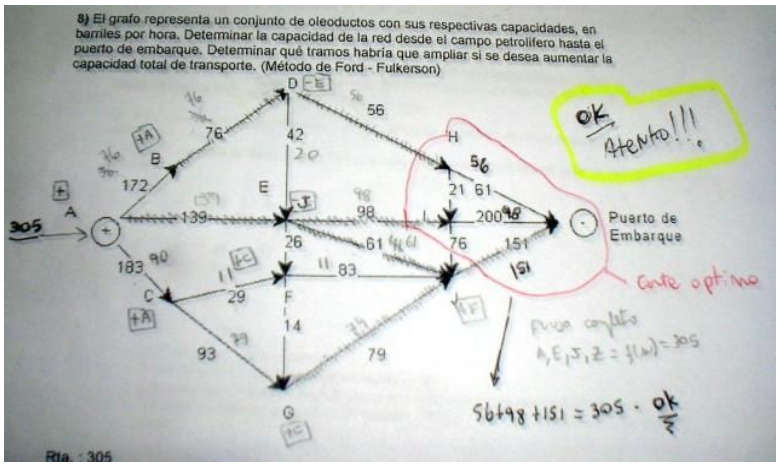
[+] -> [+], al flujo le sumo 1

[+] -> [-], al flujo le resto 1

[-] -> [+], le sumo 1

Así obtengo el flujo optimizado.

Ejemplo:



Claude Berge

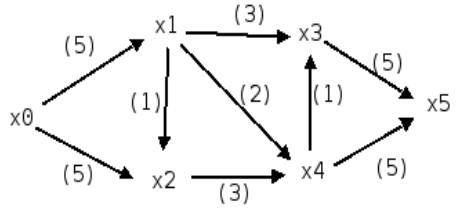
Llamamos **corte** a una forma de atravesar la red a través de los arcos de manera tal que el vértice final pertenezca al corte y el vértice inicial no.

El teorema de Ford-Fulkerson dice que el **flujo máximo** que puede circular por una red de transporte es igual a la capacidad mínima de un corte. (o sea, el "cuello de botella" que da el flujo máximo de la red).

Corte óptimo: debe cumplir que **todos los arcos que ingresan deben estar saturados,** y los **arcos que salen deben ser vacíos.** Corta los vértices marcados. Permite verificar que este bien resuelto el problema.

Nota importante : en una red de transporte no se aceptan múltiples salidas, si hay varias, se unen en un único nodo que hace de salida.

Programación lineal aplicada al flujo máximo en redes



$x_{0,}x_1$ Flujo por arco $x_{0,}x_1$

$x_{01}=x_{13}+x_{12}+x_{14}$

$0 \le x_{0,}x_1 \le 5$

$0 \le x_{1,}x_2 \le 1$

etc

$\begin{matrix} MAX\ x_0 \\ MAX: z=x_{3,5}+x_{4,5} \end{matrix}$ } funcional (tomo uno de los 2 funcionales)

$x=\left(x_{1,}x_{2,}\ldots,x_n\right)$

n-1 aristas que vinculan todos los vértices.

Se crea una matriz cuadrada y simétrica que vincula los vértices.

	x1	x2	...	xn
x1	0			
x2		0		
...			0	
xn				0

Algoritmo de Kruskal MST (Minimum Spanning Tree)

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol expandido mínimo en un grafo conexo (o sea, están todos los vértices interconectados entre si) y valuado (cada unión tiene un valor de "peso").

Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el peso total de todas las aristas del árbol es el mínimo.

El algoritmo funciona así:

- Buscar el valor mínimo de la matriz, e ir seleccionando los mínimos siguientes conectados. (cuando hay "empate", se empieza por cualquiera)
- Seleccionar una cadena , sin formar ciclo (cadena cerrada).

  Cuando una arista cierra ciclo, se descarta y se sigue con la siguiente.
- Así obtenemos el árbol de expansión mínima.

Se obtienen n-1 aristas que recorren **todos** los vértices.

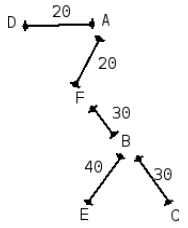
Ejemplo:

Partiendo de esta matriz de un grafo conexo y valuado:

	A	B	C	D	E	F
A	0	~~40~~	~~50~~	20	~~140~~	20
B		0	30	~~50~~	40	30
C			0	~~80~~	~~70~~	~~50~~
D				0	~~110~~	~~90~~
E					0	~~130~~
F						0

Y aplicando el algoritmo, arribamos a este grafo, de 6 vértices, 5 aristas, que conecta todos los vértices.

20 DA + 20 AF + 30 FB + 40 BE + 20 BC = 140



Procesos estocásticos

En matemáticas de probabilidades, un proceso estocástico es un proceso aleatorio que evoluciona con el tiempo.

Matemáticamente, un proceso estocástico se define como un conjunto de variables aleatorias Xt cuya distribución varía de acuerdo a un parámetro, generalmente el tiempo.

La variable tiempo t toma valores en un subconjunto de números enteros o reales no negativos.

Las variables aleatorias Xt toman valores en un conjunto que se denomina espacio de estados.

Cadenas de Markov

Una cadena de Markov es una serie de eventos, en la cual la probabilidad de que ocurra un evento depende del evento **inmediato anterior**. En efecto, las cadenas de este tipo tienen memoria. “Recuerdan” el último evento y esto condiciona las posibilidades de los eventos futuros. Esta dependencia del evento anterior distingue a las cadenas de Markov de las series de eventos independientes, como tirar una moneda al aire o un dado.

Es un proceso estadístico de tiempo discreto, donde hay ciertos elementos que pasan de un estado a otro con cierta probabilidad y la transición de un estado depende de dichos estados y no de la forma en que se llega a ellos.

O sea, la probabilidad del valor actual depende del valor que salio en el inmediato anterior.

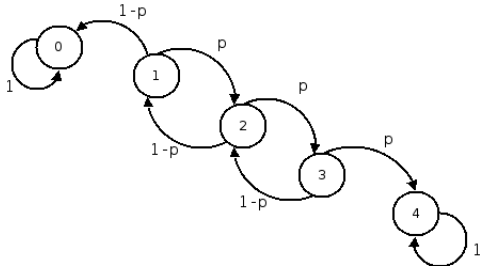
Si se vuelca en una matriz de transición los estados y las probabilidades nos encontramos en una matriz estocástica donde cada fila suma uno.

El producto de la matriz estocástica es otra matriz estocástica. Lo mismo la potencia.

Ejemplo 1:

Un jugador gana $ 2 por jugada, gana $ 1 con prob p, pierde $1 con 1-p

	0	1	2	3	4
0	1	0	0	0	0
1	1-p	0	p	0	0
2	0	1-p	0	p	0
3	0	0	1-p	0	p
4	0	0	0	0	1



[DEBUG : meter ejemplo de la fotocopia del laboratorio]

Programación dinámica

Es una optimización en etapas, sirve para cuando puedo dividir un proceso en etapas.

Vamos a considerar problemas para los cuales es posible identificar cierto número de pasos secuencial es que llamaremos "proceso de decisión en n etapas".

Las "decisiones" son las opciones que tenemos para completar las etapas.

Una secuencia de decisiones se denomina "política".

La condición del proceso en una etapa dada se denomina "estado" en esa etapa.

Cuando se toma una decisión se produce una transición del estado actual a otro estado asociado con la etapa siguiente.

Por ahora nos ocuparemos de la programación dinámica finita si tiene n etapas finitas y si los estados asociados con cada etapa también son finitos.

Es determinística si el resultado de cada decisión se conoce exactamente.

El objetivo es determinar una "política óptima".

Variantes:

- Determinista
- Aleatoria con cantidad de periodos finitos
- Aleatoria con cantidad de periodos infinitos

Decisiones en etapas -> política

Una política óptima esta formada por políticas también óptimas.

Principio de optimalidad de R. Bellman

$$x_1 + x_2 + ... + x_n \leq b$$

$$Z = f_1(x_1) + f_2(x_2) + ... + f_n(x_n) \rightarrow optimizar$$

Notar que fn son funciones no lineales.

La respuesta usual de estos problemas es del tipo “si usted llega a ... situación, haga ...”

Para explicarlo, es mejor ver este ejemplo:

Ejemplo de programación dinámica

[meter ejemplo]