* **Simulación y gestión de Pedidos**

**Introducción al problema**

Se debe confeccionar una propuesta de acción para la empresa VALLON. La misma se encarga de hacer deliverys en tiempo real, y se enfrenta a la posible situación de una campaña geo localizada durante el tiempo de 1 hora y de acuerdo a sus datos históricos se presentan tres posibles escenarios de impacto:

* Bajo: con cantidad de ordenes uniformes (20,25) con probabilidad de 0,2
* Moderado: con cantidad de ordenes uniformes (30,40) con probabilidad de 0,4
* Alto: con cantidad de ordenes uniformes (45,60) con probabilidad de 0,4

Las decisiones ante los distintos escenarios presentados son las siguientes:

* Mantener el mismo número de picker 5
* Extender el número de picker hasta 10

Por cada orden ingresada dentro de la hora propuesta el ingreso es de $100 y el costo de cada picker por hora es de $100.

**Formulación del Modelo**

Los parámetros del problema planteado son los siguientes:

* N° de ordenes generadas
* N° de ítems por orden
* Tiempos Interpicks
* Ingreso por orden generada ($100)
* Costo por hora de cada picker ($100)
* N° de pickers [5,10]

Los parámetros 1, 2 y 3 son las **variables aleatorias** que simularemos con el programa que se detalla a continuación. Una vez obtenida la tabla de pagos generada por cada escenario y para cada posible solución, utilizaremos el criterio del Valor Económico Esperado para hacer la elección de la decisión que genere una mayor ganancia.

El programa generado funciona modelando el procesamiento de ordenes bajo distintos escenarios (de demanda Alta, Baja y Media con sus respectivas probabilidades de ocurrencia).

El programa simula la cantidad de órdenes encargadas bajo cada escenario de forma aleatoria siguiendo una distribución binomial con los límites establecidos según el escenario bajo el cual se está modelando. Esto se realiza por medio de la siguiente función:

Luego se simulan la cantidad de ítems que componen cada una de las órdenes por medio de la siguiente función:

def generate\_orders(n\_orders, items\_vals, items\_cumprobs):

ordenes=[]\*n\_orders #Lista que guarda los ítems de cada orden

for i in range(0,n\_orders):

ordenes.append(simulate\_empirical(items\_vals,items\_cumprobs))

return ordenes

Luego se simulan los tiempos interpick por medio de la siguiente función:

def simulate\_order\_time(n\_items, picker, pickers\_dist):

time\_tot\_to\_process=0 #variable que va a acumular el tiempo que tarda en procesar la orden sumando los tiempos entre cada item que la componen

for i in range (0,n\_items):

j=simulate\_empirical(pickers\_dist[picker]['times'], pickers\_dist[picker]['cumprobs'])

time\_tot\_to\_process = time\_tot\_to\_process + j #se acumulan los tiempos de cada item

return time\_tot\_to\_process

Ambas funciones se alimentan de los datos relevados por la empresa y que son input del problema planteado y utilizan la función simulate.empirical la cual se alimenta de las funciones get.pdf y get\_cdf para tomar de forma aleatoria valores de la lista de posibles dado como input a las funciones provenientes de los datos relevados por la empresa

Una vez que se obtienen las órdenes y los tiempos interpick, se procesan las órdenes por medio de la siguiente función:

def process\_orders(orders, pickers\_dist, n\_pickers):

pickers\_stat = [0.0]\*n\_pickers

collected\_rev = 0.0

for order in orders:

free\_picker = get\_picker(pickers\_stat)

tiempo\_de\_procesamiento = simulate\_order\_time(order, free\_picker, pickers\_dist)

pickers\_stat[free\_picker] = pickers\_stat[free\_picker] + tiempo\_de\_procesamiento

if tiempo\_de\_procesamiento <= 3600:

collected\_rev = collected\_rev + 100

last\_to\_finish = max(pickers\_stat)

return last\_to\_finish, collected\_rev

Donde en base a los parámetros de los tiempos interpick y la cantidad de ítems que posee cada orden y teniendo en cuenta la cantidad de pickers que se están simulando se simula el procesamiento de las órdenes y se colecta el revenue generado en base a si cumple con procesarla en su totalidad en menos del tiempo estipulado en la calidad de servicio demandada por el cliente (3600 seg)

Una vez definidas las funciones previamente mencionadas, se define la función simulate\_process, la cual genera las 1000 simulaciones para cada uno de los escenarios de demanda con las cantidades de pickers de 5 a 10, en total 18 diferentes escenarios.

La función simulate\_process se define de la siguiente manera:

def simulate\_process(pickers\_data, outcome, number\_pickers, n\_sims):

items\_vals, items\_probs = get\_pdf(pickers\_data.items\_per\_order)

items\_cumprobs = get\_cdf(items\_probs)

pickers\_dist = {}

for picker in pickers\_data.picker\_times:

times, probs = get\_pdf(pickers\_data.picker\_times[picker])

cumprobs = get\_cdf(probs)

pickers\_dist[picker] = {'times': times, 'cumprobs': cumprobs}

total\_times = []

total\_revs = []

for i in range(n\_sims):

cant\_orders = simulate\_uniform(outcome[0],outcome[1])

ordenes = generate\_orders(cant\_orders, items\_vals, items\_cumprobs) #se genera un alista de int, donde cada número representa los ítems que componen la orden

tiempos,revenues=process\_orders(ordenes, pickers\_dist, number\_pickers) #devuelve dos valores, uno con el tiempo que se tardó en procesar todos los pedidos y el revenue generado.

total\_times.append(tiempos)

total\_revs.append(revenues)

return total\_times,total\_revs #retorna una lista con el tiempo que tardó en procesar todas las órdenes y el revenue generado

Las funciones antes mencionadas son llamadas y sus inputs entregadas a partir del main, el cual se define de la siguiente manera:

def main():

random.seed(0)

max\_number\_pickers = 10 # No tocar este valor.

number\_of\_simulations = 1000

# Definicion de los posibles escenarios como una lista de tuplas, donde cada tupla representa los valores Unif(a,b)

# Impacto bajo: Unif(20,25), posicion 0 de la lista scenarios.

# Impacto moderado: Unif(30,40), posicion 1 de la lista scenarios.

# Impacto alto: Unif(45,60), posicion 2 de la lista scenarios.

scenarios = [(20,25),(30,40),(45,60)]

# Se leen los datos de archivo y se procesan. No deberian modificar esto. Es algo simple de hacer (pero molesto!)

data = PickersData(max\_number\_pickers)

for outcome in scenarios:

# Para cada escenario, el posible numero de pickers

print("Escenario con demanda uniforme: " + str(outcome))

for n\_pickers in range(5,max\_number\_pickers + 1):

print("Cantidad de pickers: " + str(n\_pickers))

times,revs = simulate\_process(data, outcome, n\_pickers, number\_of\_simulations)

analyze\_results(times,revs) #analiza los resultados de cada uno de los escenarios indicando los tiempos promedios de procesamiento y el porcentaje de ordenes procesadas dentro del tiempo máximo (lo cual sirve para ver el nivel de servicio en esas condiciones) y el revenue promedio generado por ese escenario.

**Resultados obtenidos en la experimentación**

En la siguiente tabla se muestran todos los resultados generados por el promedio de las 1000 simulaciones para cada escenario y posible decisión:



Por ultimo tomamos el revenue por cada escenario y decisión tomada, y calculamos el Valor Económico Esperado como:

EMV (# pickers) = prob Baja x (Rev Baja – Costo #pickers) + prob Mod x (Rev Mod – Costo #pickers) + prob Alta x (Rev Alta – Costo #pickers)



Consideramos que la mejor combinación de pickers para los escenarios estimados es permanecer con los **5 pickers sin contratar ninguno adicional** ya que arrojo el máximo valor económico esperado.

* **Problema de la Dieta**

**Introducción al problema**

A partir de la herramienta CPLEX se busca resolver una variante del problema de la dieta donde se quiere evaluar el efecto que tiene en el costo y los diferentes nutrientes el hecho de incorporar flexibilidad los parámetros permitiendo dietas que estén dentro de un rango de los parámetros de referencia. Para ello, se introduce un parámetro *k* que expresa la flexibilidad (en porcentaje) para todos los requerimientos.

**Formulación del Modelo**

El objetivo del modelo es, para diferentes valores de k, minimizar el costo sujeto a restricciones alimenticias de la dieta: calorías, proteínas y calcio. Se busca analizar la evolución de los costos mínimos obtenidos para cada k, junto con el de las calorías, calcio y las proteínas.

***Variables de decisión:***

xi = cantidad de porciones del alimento i

ci = costo del alimento i

aij =contribución del alimento i del nutriente j

***Función objetivo y restricciones:***

s.t.

Para realizar el modelo contábamos con dos data sets: *diet1* y *diet2*. En el presente trabajo utilizamos *diet1:*

**Tabla 1 – Información data set y aportes de los requerimientos nutricionales respecto a su target**



**Consideraciones y dificultades implementación del modelo**

En principio se programaron las funciones requeridas y se corrió el modelo para un conjunto de valores de k reducido agregando prints en las funciones para ir verificando los datos. Se excluyeron los gráficos en esta primera instancia.

Luego, al corroborar que funcionaba correctamente, se incluyeron todos los valores de k entre 0.01 y 0.2 y se corrió nuevamente el modelo sumando los gráficos finales.

**Resultados obtenidos en la experimentación**

A continuación presentamos una tabla de resultados del modelo con la cantidad de porciones de cada alimento y el costo mínimo obtenido para cada valor de k. Lo primero que podemos resaltar es que las variables X2, X3 y X6 son nulas para todos los escenarios mientras que las cantidades de X4 y X5 son decrecientes a medida que aumenta el k y las de X1 son crecientes hasta el k=0,17.

**Tabla 2 – Resultados del modelo**



De la tabla de datos inicial, esperábamos a priori, que el modelo asigne porciones pequeñas a las variables de mayor costo unitario (X2, X3, X5 y X6), hecho que vemos reflejado en esta tabla. De estas variables, el modelo asigna porciones solo a X5 y descarta el resto de las variables con costo elevado dado que este alimento presenta un aporte global menor a X2, X3 y X6 (ver Tabla 1), pero a medida que el k aumenta entendemos que el costo empieza a tener más peso, por lo que le decrecen las porciones que el modelo le asigna al alimento 5.

Como se muestra en el gráfico a continuación, a medida que el k aumenta el costo total de la dieta disminuye. En principio, esta conclusión de los datos entendemos que es correcta dado que al aumentar la flexibilidad a los requerimientos, resulta más factible hallar una solución que, a menor costo, cumpla con las restricciones alimenticias requeridas.

**Gráfico 1 – Evolución del Costo vs. k**



Con respecto a los requerimientos nutricionales, vemos que las Proteínas van creciendo hasta el k=0,17 donde empiezan a caer, esto puede deberse a que la variable X1 presenta el mismo comportamiento y a que el modelo no asigna porciones al alimento 5 y reduce porciones al alimento 4 en el k=0,18, siendo estos dos los de mayor aporte proteico.

**Gráfico 2 – Evolución del Costo y las Proteínas vs. k**



En el caso de las Calorías y el Calcio, vemos que las mismas decrecen a medida que aumenta k, dado que el X1 va cobrando mayor participación en la dieta (siendo éste el alimento con menor aporte calórico y de calcio). Asimismo X4 y X5 que aportan más a éstos nutrientes van perdiendo participación en las porciones en la dieta a medida que aumenta k.

**Gráfico 3 – Evolución del Costo y las Calorías vs. k**



**Gráfico 4 – Evolución del Costo y el Calcio vs. k**

