Universidad de Buenos Aires
Facultad de Ingeniería

2do Cuatrimestre 2019

Modelos y Optimización I (71.14)

Informe Trabajo Práctico 1

Fecha: 12 de noviembre de 2019

Integrantes:

Apellido y Nombre	Padrón	Correo Electrónico
Ferres, Julian	101483	julianferres@gmail.com
Gamarra Silva, Cynthia	92702	cyntgamarra@gmail.com
Loguercio, Sebastian	100517	seba21log@gmail.com

Índice

1. Parte A

1.1. Análisis

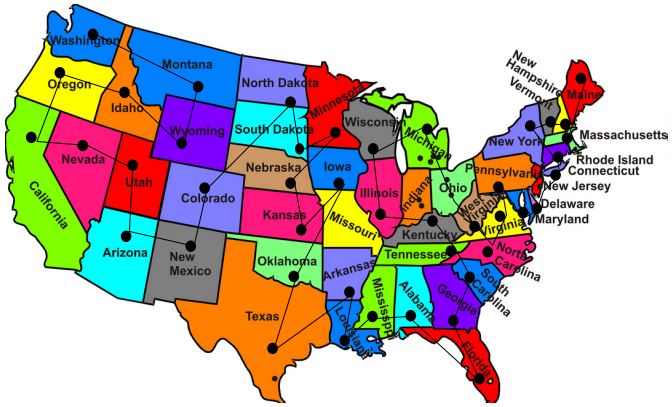
El problema radica en saber que, dado un conjunto de ciudades y las distancias entre ellas, conocer cuál es el orden de recorrido que debe realizar el auto para poder visitar todos los estados. Respondida esta pregunta se estaría cumpliendo con el objetivo de determinar el orden de recorrido a utilizar para minimizar el costo total del viaje.

Con lo dicho anteriormente, el problema a analizar puede ser modelado según el problema combinatorio llamado *Problema del viajante* o *Problema del Agente Viajero* (PAV). En este tipo de problema se tiene un cartero que debe entregar n cartas a sus n direcciones correspondientes, partiendo de la oficina de correos y volviendo a ella. Este recorrido debe realizarlo de forma óptima ya sea minimizando el tiempo o el costo monetario del recorrido.

En nuestro caso, tenemos viajeros que desean viajar una cantidad de estados en los Estados Unidos. Para ello cuentan con un auto y pueden conseguir tanto alimentos como agua para hidratarse durante el viaje.

Teniendo en cuenta los datos aportados por el enunciado podemos utilizar un grafo cuyos vértices serán las ciudades visitadas realizando un camino hamiltoniano.

Un gráfico ilustrativo sería el siguiente:



Respondiendo a la pregunta extra, las capitales que decidieron no visitar son Alaska y Hawaii.

1.2. Objetivo

Determinar el orden de recorrido de Carlos y Andrés de manera de minimizar los gastos del viaje en un período determinado.

1.3. Hipótesis

Las hipótesis impuestas son las siguientes:

- La heladera dura todo el viaje, por lo que habrá que comprar solo una.
- El auto no se queda sin combustible durante el viaje; siempre puede recargar en alguna estación de servicio de alguna capital.

- No hay otros gastos extras que deban realizar más que los especificados.
- Se detienen a pasar la noche sólo en las capitales.
- No hay inflación durante el período del análisis del problema.
- EL viaje en auto comienza en el **punto de partida** y termina en el **punto de llegada**.
- Todo **camino** se puede recorrer en ambas direcciones.
- No se consideran eventualidades como retraso en las salidas, condiciones climáticas inestables, etc.El auto y los **caminos** están disponibles en todo momento.
- Se puede arribar a una **ciudad** desde cualquier otra y viceversa.
- Existe un único **camino** entre 2 ciudades.
- No se quedan sin dinero en el trayecto del viaje a realizar.
- Al detenerse para hidratarse, se toman una sola botella de agua entre ambos.
- El tiempo que se tarda viajando de una cuidad a otra es siempre menor a un día.
- El gasto por comida se hace dos veces por día sin excepción.

1.4. Modelo de programación lineal continua

1.5. Constantes

- d_{ij} = Distancia entre la ciudad i y la ciudad j (en km).

1.5.1. Variables

- U_i = Orden en el que la capital i fue visitada. (entera)
- Y_{ij} = Vale 1 si el camino entre la capital i y la j está en el tour (binaria).
- D = Distancia recorrida (en km).
- DES_j = Vale 1 si descansan 2 dias en la ciudad j (binaria).
- E = Cantidad de veces totales que se detienen a estirarse. (entera)
- H = Vale 1 si compran la heladera (binaria).
- $Agua$ = Precio de todas las botellas de agua (usd).
- $Hidr_1$ = Cantidad de veces que se detienen y se hidratan tomando un agua si compran la heladera (si $H = 1$).
- $Hidr_0$ = Cantidad de veces que se detienen y se hidratan tomando un agua si no compran la heladera (si $H = 0$).
- $Hidr$ = Cantidad de veces que se detienen y se hidratan.

- K_i = Cantidad de Km recorridos hasta ciudad i.
- K_i^I = Cantidad de Km recorridos hasta ciudad i si la ciudad esta entre el recorrido de 0 y 10000Km del viaje.
- K_i^{II} = Cantidad de Km recorridos hasta ciudad i si la ciudad esta entre el recorrido de 10000 y 20000Km del viaje.
- K_i^{III} = Cantidad de Km recorridos hasta ciudad i si la ciudad esta entre el recorrido de 20000 y 30000Km del viaje.
- K_i^{IV} = Cantidad de Km recorridos hasta ciudad i si la ciudad esta entre el recorrido de mas de 30000Km.
- X_i^I = Vale 1 si la ciudad i está en el recorrido entre 0 y 10000Km del viaje (si $K_i \leq 10000$).
- X_i^{II} = Vale 1 si la ciudad i está en el recorrido entre 10000 y 20000Km del viaje (si $10000 \leq K_i \leq 20000$).
- X_i^{III} = Vale 1 si la ciudad i está en el recorrido entre 20000 y 30000Km del viaje (si $20000 \leq K_i \leq 30000$).
- X_i^{IV} = Vale 1 si la ciudad i está en el recorrido mayor a 30000Km del viaje (si $K_i \geq 30000$).
- A_{ij} = Vale 1 si se pasó por la ciudad i antes que j (si $U_i \leq U_j$).
- V_{ijk} = Vale 1 si se hizo el viaje de i a j y además se pasó por j antes que k (es decir, si $A_{jk} = 1$ y $Y_{ij} = 1$).
- AND_i^I = Vale 1 si X_i^I y DES_i valen 1 (es decir, si la ciudad i está en el primer tramo del recorrido y descansaron 2 días en esa ciudad).
- AND_i^{II} = Vale 1 si X_i^{II} y DES_i valen 1 (es decir, si la ciudad i está en el segundo tramo del recorrido y descansaron 2 días en esa ciudad).
- AND_i^{III} = Vale 1 si X_i^{III} y DES_i valen 1 (es decir, si la ciudad i está en el tercer tramo del recorrido y descansaron 2 días en esa ciudad).
- AND_i^{IV} = Vale 1 si X_i^{IV} y DES_i valen 1 (es decir, si la ciudad i está en el ultimo tramo del recorrido y descansaron 2 días en esa ciudad)
- C_1 = Cantidad de días que tardan en hacer el recorrido de 0 a 10000Km.
- C_2 = Cantidad de días que tardan en hacer el recorrido de 10000 a 20000Km.
- C_3 = Cantidad de días que tardan en hacer el recorrido de 20000 a 30000Km.
- C_4 = Cantidad de días que tardan en hacer el recorrido desde 30000Km.

1.5.2. Ecuaciones

- Cantidad de ciudades: $n = 48$
- Distancia recorrida:

$$D = \sum_i \sum_j d_{ij} Y_{ij}$$

- Subtours:

$$\forall i, j \; ciudades :$$
$$U_i - U_j + n \cdot Y_{ij} \leq n - 1$$

- Se sale de todas las ciudades, pasando una vez por cada una:

$$\forall i \; ciudad :$$
$$\sum_j Y_{ij} = 1$$

- Llegadas a todas las ciudades:

$$\forall j \; ciudad :$$
$$\sum_i Y_{ij} = 1$$

- Si viajaron mas de 250Km de corrido, descansan 2 dias en la ciudad:

Siendo M un número suficientemente grande:

$$\forall j \; ciudad :$$
$$250\,km \cdot DES_j \leq \sum_i Yij \cdot d_{ij} \leq (1 - DES_j) \cdot 250\,km + DES_j \cdot M$$

- Cantidad de veces que se detienen a hidratarse:

$$Hidr = Hidr_0 + Hidr_1$$

- Cantidad de veces que se detienen a hidratarse si compran la heladera:

Siendo M un número suficientemente grande:

$$0{,}1 \cdot H \leq Hidr_1 \leq H \cdot M$$

- Cantidad de veces que se detienen a hidratarse si no compran la heladera:

Siendo M un número suficientemente grande:

$$0{,}1 \cdot (1 - H) \leq Hidr_0 \leq (1 - H) \cdot M$$

- Cantidad de veces que se detienen a estirar:

$$E = \tfrac{D}{100}$$

- Cada dos veces que se detienen a estirar, toman una botella de agua:

$$Hidr = \frac{E}{2}$$

- Precio de todas las botellas de agua:

$$Agua = \$2 \cdot Hidr_1 + \$3 \cdot Hidr_0$$

- Definición A_{ij} (se visito i antes que j):

Siendo M un número suficientemente grande:

$$\forall i,\, j\ ciudades:$$
$$-M(1 - A_{ij}) \leq U_j - U_i \leq M \cdot A_{ij}$$

- Definición V_{ijk}:

$$\forall i, j, k\ ciudades\ distintas:$$
$$2 \cdot V_{ijk} \leq A_{jk} + Y_{ij} \leq V_{ijk} + 1$$

- Cantidad de Km recorridos hasta ciudad i:

$$\forall k\ ciudad:$$
$$K_k = \sum_i \sum_j d_{ij} \cdot V_{ijk}\ ;\ con\ i \neq j \neq k$$

- K_i por recorrido:

$$K_i = K_i^I + K_i^{II} + K_i^{III} + K_i^{IV}$$

- Definición X_i:

$$\forall i\ ciudad:$$

$$\begin{aligned} X_i^I + X_i^{II} + X_i^{III} + X_i^{IV} &= 1 \\ K_i^I &\leq X_i^I \cdot 10000Km \\ X_i^{II} \cdot 10000{,}1Km \leq K_i^{II} &\leq X_i^{II} \cdot 20000Km \\ X_i^{III} \cdot 20000{,}1Km \leq K_i^{III} &\leq X_i^{III} \cdot 30000Km \\ X_i^{IV} \cdot 30000{,}1Km \leq K_i^{IV} &\leq X^{IV} \cdot M \end{aligned}$$

- Definicion ANDs:

$$\forall i\ ciudad$$
$$2 \cdot AND_i^I \leq X_i^I + DES_i \leq AND^I + 1$$
$$2 \cdot AND_i^{II} \leq X_i^{II} + DES_i \leq AND^{II} + 1$$
$$2 \cdot AND_i^{III} \leq X_i^{IIi} + DES_i \leq AND^{III} + 1$$
$$2 \cdot AND_i^{IV} \leq X_i^{IV} + DES_i \leq AND^{IV} + 1$$

- Cantidad de días que tardan en cada intervalo del recorrido:

Siendo i las ciudades:

$$C_1 = \sum_i (X_i^I + AND_i^I)$$

$$C_2 = \sum_i (X_i^{II} + AND_i^{II})$$

$$C_3 = \sum_i (X_i^{III} + AND_i^{III})$$

$$C_4 = \sum_i (X_i^{IV} + AND_i^{IV})$$

1.5.3. Funcional

$Habitaciones = \$50 \sum_i \sum_j Y_{ij} + \$50 \sum_j DES_j$
$Nafta = \$2 \cdot D$
$CostoAgua = \$60 \cdot H + Agua$
$Comida = \$30 \cdot C_1 + \$25 \cdot C_2 + \$20 \cdot C_3 + \$15 \cdot C_4$

$Z = Habitaciones + Nafta + CostoAgua + 2 \cdot Comida \longrightarrow Min$

1.6. Modelo en computadora

```
/* Problema del viajante con restricciones*/

#Conjuntos:
set CIUDADES;

#Parámetros o constantes:
#Las distancias en km. de ir desde la ciudad i hasta la j.
param DISTANCIA{i in CIUDADES, j in CIUDADES: i<>j};

# constante M.
param M;

#Variables:
#U_i orden de secuencia en que la ciudad i es visitada (excluyendo
    el punto de partida).
var U{i in CIUDADES: i<>'0'} >=0, integer;

#Y_ij, bivalente que vale 1 si va desde la ciudad i hasta la j (i <>
    j).
var Y{i in CIUDADES, j in CIUDADES: i<>j} >= 0, binary;

# D: Distancia recorrida (en km).
var D >=0;
```

```
# DES_j : Vale 1 si descansan 2 dias en la ciudad j.
var DES{j in CIUDADES} >=0, binary;

# E: Cantidad de veces totales que se detienen a estirarse.
var E >=0;

# H : Vale 1 si compran la heladera.
var H >=0, binary;

# Agua: Precio de todas las botellas de agua.
var Agua >=0;

#Hidr1: Cantidad de veces que se detienen y se hidratan tomando un
   agua si compran la heladera.
var Hidr0 >=0;

#Hidr1: Cantidad de veces que se detienen y se hidratan tomando un
   agua si no compran la heladera.
var Hidr1 >=0;

# Hidr: Cantidad de veces que se detienen y se hidratan.
var Hidr >=0;

# K : Cantidad de Km recorridos hasta ciudad i.
var K{i in CIUDADES} >=0;

# KI: Cant de Km recorridos si la ciudad esta en el primer tramo:
var KI{i in CIUDADES} >= 0;

# KII: Cant de Km recorridos si la ciudad esta en el segundo tramo:
var KII{i in CIUDADES} >= 0;

# KIII: Cant de Km recorridos si la ciudad esta en el tercer tramo:
var KIII{i in CIUDADES} >= 0;

# KIV: Cant de Km recorridos si la ciudad esta en el cuarto tramo:
var KIV{i in CIUDADES} >= 0;


# XI_i : Vale 1 si la ciudad i está en el recorrido entre 0 y 10000
   Km del viaje (si K_i <= 10000).
var XI{i in CIUDADES} >=0, binary;

# XII_i: Vale 1 si la ciudad $i$ está en el recorrido entre 10000 y
   20000 Km del viaje(si 10000 <= K_i <= 20000).
var XII{i in CIUDADES} >=0, binary;
```

```
# XIII_i: Vale 1 si la ciudad $i$ está en el recorrido entre 20000 y
    30000 Km del viaje(si 20000 <= K_i <= 30000).
var XIII{i in CIUDADES} >=0, binary;

# XIV_i : Vale 1 si la ciudad i está en el recorrido mayor a 30000
   Km del viaje (si K_i >= 30000).
var XIV{i in CIUDADES} >=0, binary;

#A_ij, bivalente que vale 1 si si se pasó por la ciudad i antes que
   j(U_i<= U_j, i <> j).
var A{i in CIUDADES, j in CIUDADES: i<>j} >= 0, binary;

#V_ijk, bivalente que vale 1 si se hizo el viaje de i a j(i <> j) y
   además se pasó por j antes que k(es decir, A_jk = 1 y Y_ij = 1).
var V{i in CIUDADES, j in CIUDADES, k in CIUDADES: i<>j and j<>k and
    k<>i} >= 0, binary;

#ANDI_i: Vale 1 si XI_i y DES_i valen 1(es decir, si la ciudad i est
   á en el primer tramo del recorrido y descansaron 2 días en esa
   ciudad).
var ANDI{i in CIUDADES} >=0, binary;

#ANDII_i: Vale 1 si XII_i y DES_i valen 1(es decir, si la ciudad i
   está en el segundo tramo del recorrido y descansaron 2 días en
   esa ciudad).
var ANDII{i in CIUDADES} >=0, binary;

#ANDIII_i: Vale 1 si XII_i y DES_i valen 1(es decir, si la ciudad i
   está en el tercer tramo del recorrido y descansaron 2 días en esa
    ciudad).
var ANDIII{i in CIUDADES} >=0, binary;

#ANDIV_i: Vale 1 si XIV_i y DES_i valen 1(es decir, si la ciudad i
   está en el último tramo del recorrido y descansaron 2 días en esa
    ciudad).
var ANDIV{i in CIUDADES} >=0, binary;

# C1 : Cantidad de días que tardan en hacer el recorrido de 0 y
   10000 Km.
var C1 >=0;

# C2: Cantidad de días que tardan en hacer el recorrido de 10000 y
   20000 Km.
var C2 >=0;

# C3: Cantidad de días que tardan en hacer el recorrido de 20000 y
   30000 Km.
```

```
var C3 >=0;

# C4 : Cantidad de días que tardan en hacer el recorrido desde 30000
    Km.
var C4 >=0;

#Otras variables
var Habitaciones >=0;
var Nafta >=0;
var CostoAgua >=0;
var Comida >=0;

#FUNCIONAL
minimize z: Habitaciones + Nafta + CostoAgua + 2*Comida;

#Costo por las habitaciones
s.t. costoHab :Habitaciones = 50 * sum{i in CIUDADES, j in CIUDADES:
    i<>j }(Y[i,j]) + 50 * sum{j in CIUDADES }(DES[j]) ;

#Nafta gastada según la cantidad de kilómetros hecho
s.t. costoNafta :Nafta = 2 * D ;

#Costo total del agua
s.t. costAgua :CostoAgua = 60 * H + Agua;

#Costo total por la comida
s.t. comida :Comida = 30 * C1 + 25 * C2 + 20 * C3 + 15 * C4 ;

#Distancia recorrida
s.t. cantDeKm :D = sum{i in CIUDADES, j in CIUDADES: i<>j } (Y[i,j]
   * DISTANCIA[i,j]);

#Secuencia para evitar subtours
s.t. orden{i in CIUDADES, j in CIUDADES: i<>j and i<>'0' and j
   <>'0'}:
U[i] - U[j] + card(CIUDADES) * Y[i,j] <= card(CIUDADES) - 1;

#Se sale de todas las ciudades, pasando una vez por cada una:
s.t. saleDeI{i in CIUDADES} :sum{j in CIUDADES:i<>j} (Y[i,j]) = 1;

#Llegadas a todas las ciudades:
s.t. llegaAJ{j in CIUDADES}:sum{i in CIUDADES:i<>j} (Y[i,j]) = 1;

#Si viajaron mas de 250Km de corrido, descansan 2 dias en la ciudad.
    Siendo M un número suficientemente grande, cambiarlo por un
   valor para correrlo:
```

```
s.t. viajeMasDe250{j in CIUDADES}: 250 * DES[j] <= sum{i in CIUDADES
   :i<>j} (Y[i,j] * DISTANCIA[i,j]);
s.t. viajeMasDe250_1{j in CIUDADES}: sum{i in CIUDADES:i<>j} (Y[i,j]
    * DISTANCIA[i,j]) <= (1 - DES[j] ) * 250 + M * DES[j];


#Cantidad de veces que se detienen a hidratarse:
s.t. cantHidratacion:Hidr = Hidr0 + Hidr1;

#Cantidad de veces que se detienen a hidratarse si compran la
   heladera. Siendo M un número suficientemente grande:
s.t. hidraConHelad: 0.01*H <= Hidr1;
s.t. hidraConHelad_1: Hidr1 <= H * M;

#Cantidad de veces que se detienen a hidratarse si no compran la
   heladera: Siendo M un número suficientemente grande:
s.t. hidraSinHelad: 0.01*(1 - H )<= Hidr0 ;
s.t. hidraSinHelad_1: Hidr0 <= (1 - H ) * M;

#Cantidad de veces que se detienen a estirar:
s.t. cantEstirarse: 100 * E = D;

#Cada dos veces que se detienen a estirar, toman una botella de agua
   :
s.t. estirarse2veces: 2*Hidr = E;

#Precio de todas las botellas de agua:
s.t. precioAgua: Agua = 2 * Hidr1 + 3 * Hidr0;

#Definición A_ij (se visito i antes que j. Siendo $M$ un número
   suficientemente grande:
s.t. visitaIantesJ{i in CIUDADES, j in CIUDADES: i<>j}: -M*(1 - A[i,
   j]) <= U[j] - U[i];
s.t. visitaIantesJ_1{i in CIUDADES, j in CIUDADES: i<>j}: U[j] - U[i
   ] <= M * A[i,j];

#Definición V_ijk:
s.t. ciudDistA{i in CIUDADES, j in CIUDADES, k in CIUDADES: i<>j and
    j<>k and i<>k} :2 * V[i,j,k]= A[i,j] + Y[i,j];

#Cantidad de Km recorridos hasta ciudad i, para todo k:
s.t. kmHastaI{k in CIUDADES}: K[k] = sum{i in CIUDADES, j in
   CIUDADES: i<>j and j<>k and i<>k} (V[i,j,k] * DISTANCIA[i,j]);

#Ki por recorrido:
s.t. kmKI{i in CIUDADES}: K[i] = KI[i] + KII[i] + KIII[i] + KIV[i];
```

```
#Definicion X_i. Para toda i ciudad:
s.t. xIguales1{i in CIUDADES}:XI[i] + XII[i] + XIII[i] + XIV[i] = 1;

s.t. xTramo1_1{i in CIUDADES}: KI[i] <= XI[i] * 10000;

s.t. xTramo2{i in CIUDADES}: XII[i] * 10000.001 <= KII[i];
s.t. xTramo2_2{i in CIUDADES}: KII[i] <= XII[i] * 20000;

s.t. xTramo3{i in CIUDADES}: XIII[i] * 20000.001 <= KIII[i];
s.t. xTramo3_3{i in CIUDADES}: KIII[i] <= XIII[i] * 30000;

s.t. xTramo4{i in CIUDADES}: XIV[i] * 30000.001 <= KIV[i];
s.t. xTramo4_4{i in CIUDADES}: KIV[i] <= XIV[i] * M;


#Definiciones ANDs:
s.t. andI{i in CIUDADES}: 2*ANDI[i] = XI[i] + DES[i];

s.t. andII{i in CIUDADES}: 2*ANDII[i] = XII[i] + DES[i];

s.t. andIII{i in CIUDADES}: 2*ANDIII[i] = XIII[i] + DES[i];

s.t. andIV{i in CIUDADES}: 2*ANDIV[i] = XIV[i] + DES[i];


#Cantidad de días que tardan en cada intervalo del recorrido, Siendo
    i las ciudades:
s.t. recorridoC1:C1 = sum{i in CIUDADES} (XI[i] + ANDI[i]);
s.t. recorridoC2:C2 = sum{i in CIUDADES} (XII[i] + ANDII[i]);
s.t. recorridoC3:C3 = sum{i in CIUDADES} (XIII[i] + ANDIII[i]);
s.t. recorridoC4:C4 = sum{i in CIUDADES} (XIV[i] + ANDIV[i]);

solve;

end;
```

1.7. Resolución

Se pudo llegar a una solución de manera rápida mediante `glpk` recortando el problema a 5 ciudades. La salida al ejecutar `glpsol -m a.mod -o a.sol -d datos_x_ciudades.dat` es la siguiente:

```
GLPSOL: GLPK LP/MIP Solver, v4.65
Parameter(s) specified in the command line:
 -m a.mod -o a.sol -d datos_x_ciudades.dat --tmlim 120
Reading model section from a.mod...
```

```
217 lines were read
Reading data section from datos_x_ciudades.dat...
15 lines were read
Generating z...
Generating costoHab...
Generating costoNafta...
Generating costAgua...
Generating comida...
Generating cantDeKm...
Generating orden...
Generating saleDeI...
Generating llegaAJ...
Generating viajeMasDe250...
Generating viajeMasDe250_1...
Generating cantHidratacion...
Generating hidraConHelad...
Generating hidraConHelad_1...
Generating hidraSinHelad...
Generating hidraSinHelad_1...
Generating cantEstirarse...
Generating estirarse2veces...
Generating precioAgua...
Generating visitaIantesJ...
Generating visitaIantesJ_1...
Generating A_0j...
Generating A_i0...
Generating ciudDistA...
Generating ciudDistA2...
Generating xIguales1...
Generating xTramo1_1...
Generating xTramo2...
Generating xTramo2_2...
Generating xTramo3...
Generating xTramo3_3...
Generating xTramo4...
Generating xTramo4_4...
Generating andI...
Generating andI2...
Generating andII...
Generating andII2...
Generating andIII...
Generating andIII2...
Generating andIV...
Generating andIV2...
Generating recorridoC1...
Generating recorridoC2...
Generating recorridoC3...
```

```
Generating recorridoC4...
Model has been successfully generated
GLPK Integer Optimizer, v4.65
282 rows, 184 columns, 899 non-zeros
150 integer variables, 146 of which are binary
Preprocessing...
12 hidden packing inequaliti(es) were detected
61 hidden covering inequaliti(es) were detected
7 constraint coefficient(s) were reduced
250 rows, 155 columns, 716 non-zeros
130 integer variables, 126 of which are binary
Scaling...
 A: min|aij| =  1.000e+00  max|aij| =  1.000e+06  ratio =  1.000e+06
GM: min|aij| =  1.256e-01  max|aij| =  7.961e+00  ratio =  6.338e+01
EQ: min|aij| =  1.583e-02  max|aij| =  1.000e+00  ratio =  6.315e+01
2N: min|aij| =  1.562e-02  max|aij| =  1.367e+00  ratio =  8.746e+01
Constructing initial basis...
Size of triangular part is 249
Solving LP relaxation...
GLPK Simplex Optimizer, v4.65
250 rows, 155 columns, 716 non-zeros
      0: obj =   4.306492052e+04 inf =   1.301e+02 (30)
     42: obj =   2.531928459e+04 inf =   1.511e-14 (0)
*    69: obj =   1.865566905e+04 inf =   2.426e-14 (0)
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Long-step dual simplex will be used
+    69: mip =     not found yet >=              -inf        (1; 0)
+   191: >>>>>   2.875433804e+04 >=   2.360855118e+04  17.9% (13; 1)
+   227: >>>>>   2.875162677e+04 >=   2.401307134e+04  16.5% (6; 10)
+   277: >>>>>   2.739039851e+04 >=   2.734211047e+04   0.2% (6; 13)
+   282: mip =   2.739039851e+04 >=     tree is empty   0.0% (0; 35)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.7 Mb (734425 bytes)
Model has been successfully processed
Writing MIP solution to 'a.sol'...
```

El resultado de la solución con los valores de las variables generado en el archivo `a.sol` es la siguiente:

```
Problem:    a
Rows:       282
Columns:    184 (150 integer, 146 binary)
Non-zeros:  899
Status:     INTEGER OPTIMAL
Objective:  z = 27390.39851 (MINimum)
```

No.	Column name		Activity	Lower bound	Upper bound
1	U[2]	*	4	0	
2	U[1]	*	0	0	
3	U[3]	*	1	0	
4	U[4]	*	3	0	
5	Y[0,1]	*	1	0	1
6	Y[0,2]	*	0	0	1
7	Y[0,3]	*	0	0	1
8	Y[0,4]	*	0	0	1
9	Y[1,0]	*	0	0	1
10	Y[1,2]	*	0	0	1
11	Y[1,3]	*	1	0	1
12	Y[1,4]	*	0	0	1
13	Y[2,0]	*	1	0	1
14	Y[2,1]	*	0	0	1
15	Y[2,3]	*	0	0	1
16	Y[2,4]	*	0	0	1
17	Y[3,0]	*	0	0	1
18	Y[3,1]	*	0	0	1
19	Y[3,2]	*	0	0	1
20	Y[3,4]	*	1	0	1
21	Y[4,0]	*	0	0	1
22	Y[4,1]	*	0	0	1
23	Y[4,2]	*	1	0	1
24	Y[4,3]	*	0	0	1
25	D		13199.2	0	
26	DES[0]	*	1	0	1
27	DES[1]	*	1	0	1
28	DES[2]	*	1	0	1
29	DES[3]	*	1	0	1
30	DES[4]	*	1	0	1
31	E		131.992	0	
32	H	*	1	0	1
33	Agua		131.992	0	
34	Hidr0		0	0	
35	Hidr1		65.996	0	
36	Hidr		65.996	0	

37	KI[0]		0	0	
38	KI[1]		0	0	
39	KI[2]		0	0	
40	KI[3]		0	0	
41	KI[4]		0	0	
42	KII[0]		0	0	
43	KII[1]		0	0	
44	KII[2]		0	0	
45	KII[3]		0	0	
46	KII[4]		0	0	
47	KIII[0]		0	0	
48	KIII[1]		0	0	
49	KIII[2]		0	0	
50	KIII[3]		0	0	
51	KIII[4]		0	0	
52	KIV[0]		30000	0	
53	KIV[1]		30000	0	
54	KIV[2]		30000	0	
55	KIV[3]		30000	0	
56	KIV[4]		30000	0	
57	XI[0]	*	0	0	1
58	XI[1]	*	0	0	1
59	XI[2]	*	0	0	1
60	XI[3]	*	0	0	1
61	XI[4]	*	0	0	1
62	XII[0]	*	0	0	1
63	XII[1]	*	0	0	1
64	XII[2]	*	0	0	1
65	XII[3]	*	0	0	1
66	XII[4]	*	0	0	1
67	XIII[0]	*	0	0	1
68	XIII[1]	*	0	0	1
69	XIII[2]	*	0	0	1
70	XIII[3]	*	0	0	1
71	XIII[4]	*	0	0	1
72	XIV[0]	*	1	0	1
73	XIV[1]	*	1	0	1
74	XIV[2]	*	1	0	1
75	XIV[3]	*	1	0	1
76	XIV[4]	*	1	0	1
77	A[1,2]	*	0	0	1
78	A[1,3]	*	0	0	1
79	A[1,4]	*	0	0	1
80	A[2,1]	*	0	0	1
81	A[2,3]	*	0	0	1
82	A[2,4]	*	0	0	1
83	A[3,1]	*	0	0	1

84	A[3,2]	*	0	0	1
85	A[3,4]	*	0	0	1
86	A[4,1]	*	0	0	1
87	A[4,2]	*	0	0	1
88	A[4,3]	*	0	0	1
89	A[1,0]	*	0	0	1
90	A[2,0]	*	0	0	1
91	A[3,0]	*	0	0	1
92	A[4,0]	*	0	0	1
93	A[0,1]	*	1	0	1
94	A[0,2]	*	1	0	1
95	A[0,3]	*	1	0	1
96	A[0,4]	*	1	0	1
97	V[0,1,2]	*	0	0	1
98	V[0,1,3]	*	0	0	1
99	V[0,1,4]	*	0	0	1
100	V[0,2,1]	*	0	0	1
101	V[0,2,3]	*	0	0	1
102	V[0,2,4]	*	0	0	1
103	V[0,3,1]	*	0	0	1
104	V[0,3,2]	*	0	0	1
105	V[0,3,4]	*	0	0	1
106	V[0,4,1]	*	0	0	1
107	V[0,4,2]	*	0	0	1
108	V[0,4,3]	*	0	0	1
109	V[1,0,2]	*	0	0	1
110	V[1,0,3]	*	0	0	1
111	V[1,0,4]	*	0	0	1
112	V[1,2,0]	*	0	0	1
113	V[1,2,3]	*	0	0	1
114	V[1,2,4]	*	0	0	1
115	V[1,3,0]	*	0	0	1
116	V[1,3,2]	*	0	0	1
117	V[1,3,4]	*	0	0	1
118	V[1,4,0]	*	0	0	1
119	V[1,4,2]	*	0	0	1
120	V[1,4,3]	*	0	0	1
121	V[2,0,1]	*	1	0	1
122	V[2,0,3]	*	1	0	1
123	V[2,0,4]	*	1	0	1
124	V[2,1,0]	*	0	0	1
125	V[2,1,3]	*	0	0	1
126	V[2,1,4]	*	0	0	1
127	V[2,3,0]	*	0	0	1
128	V[2,3,1]	*	0	0	1
129	V[2,3,4]	*	0	0	1
130	V[2,4,0]	*	0	0	1

131	V[2,4,1]	*	0	0	1
132	V[2,4,3]	*	0	0	1
133	V[3,0,1]	*	0	0	1
134	V[3,0,2]	*	0	0	1
135	V[3,0,4]	*	0	0	1
136	V[3,1,0]	*	0	0	1
137	V[3,1,2]	*	0	0	1
138	V[3,1,4]	*	0	0	1
139	V[3,2,0]	*	0	0	1
140	V[3,2,1]	*	0	0	1
141	V[3,2,4]	*	0	0	1
142	V[3,4,0]	*	0	0	1
143	V[3,4,1]	*	0	0	1
144	V[3,4,2]	*	0	0	1
145	V[4,0,1]	*	0	0	1
146	V[4,0,2]	*	0	0	1
147	V[4,0,3]	*	0	0	1
148	V[4,1,0]	*	0	0	1
149	V[4,1,2]	*	0	0	1
150	V[4,1,3]	*	0	0	1
151	V[4,2,0]	*	0	0	1
152	V[4,2,1]	*	0	0	1
153	V[4,2,3]	*	0	0	1
154	V[4,3,0]	*	0	0	1
155	V[4,3,1]	*	0	0	1
156	V[4,3,2]	*	0	0	1
157	ANDI[0]	*	0	0	1
158	ANDI[1]	*	0	0	1
159	ANDI[2]	*	0	0	1
160	ANDI[3]	*	0	0	1
161	ANDI[4]	*	0	0	1
162	ANDII[0]	*	0	0	1
163	ANDII[1]	*	0	0	1
164	ANDII[2]	*	0	0	1
165	ANDII[3]	*	0	0	1
166	ANDII[4]	*	0	0	1
167	ANDIII[0]	*	0	0	1
168	ANDIII[1]	*	0	0	1
169	ANDIII[2]	*	0	0	1
170	ANDIII[3]	*	0	0	1
171	ANDIII[4]	*	0	0	1
172	ANDIV[0]	*	1	0	1
173	ANDIV[1]	*	1	0	1
174	ANDIV[2]	*	1	0	1
175	ANDIV[3]	*	1	0	1
176	ANDIV[4]	*	1	0	1
177	C1		0	0	

```
   178 C2                           0             0
   179 C3                           0             0
   180 C4                          10             0
   181 Habitaciones               500             0
   182 Nafta                  26398.4             0
   183 CostoAgua              191.992             0
   184 Comida                     150             0

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 4.00e+00 on row 59
        max.rel.err = 4.00e+00 on row 59
        SOLUTION IS INFEASIBLE

End of output
```

1.8. Análisis de la solución

Antes de realizar el análisis sobre la solución propuesta, cabe aclarar que se adjunto en el informe la resolución para cinco ciudades, debido que al archivo salida del mismo se volvía demasiado extenso si agregabamos más ciudades. El resultado óptimo de z logrado con cinco ciudades fue de **z = $27390.39851**.

Se debe mencionar que nos encontramos frente a un problema NP cuya resolución para la cantidad de ciudades propuestas demandaba demasiados recursos y tiempo de cómputo. Por este motivo nos vemos obligados a realizar el problema considerando menos capitales.

Ejecutando el programa con cada vez más ciudades nos vimos ante el dato de que frente a menos de 10 ciudades el programa terminaba relativamente rápido y utilizaba pocos recursos.

Al aumentar el número de capitales a 14 o más el programa tardaba demasiado o no terminaba (debido a que la memoria de la computadora no resistia) por lo que nos vimos obligados a utilizar menos capitales.

Esto nos hizo comprender la velocidad con la que la complejidad de estos problemas crece, y lo excasos que parecen los recursos apenas aumentando las ciudades.

2. Parte B

2.1. Análisis

El problema consiste en buscar dividir un predio de 8000 metros cuadrados entre dos categorias: Gold y Silver. Cada metro cuadrado de la categoría Gold se lo asigna a una persona. En cambio a cada metro cuadrado Silver se le asignan dos personas. Antes de coomenzar a vender al público general se dan 100 pases Gold al dueño del predio y dos pases Silver al ayuntamiento de la ciudad. Además está garantizado que se vendieron 500 pases Silver.

Además se le deben dar paquetes de merchandising a todos los espectadores. Un paquete cubre 20 personas con pase Silver u 8 con Gold.

2.2. Objetivo

Determinar la cantidad de pases Gold, pases Silver y paquetes de merchandising a vender para obtener la mayor ganancia posible en un periodo determinado.

2.3. Hipótesis

- Todos los pases se venden.
- Las entradas de protocolo son Silver.
- La cantidad de entradas de protocolo para el ayuntamiento son exactamente dos.
- Los 100 pases Gold al dueño del predio no generan ganancia.
- Es necesario que todos las personas tengan merchandising.

2.4. Modelo de programación lineal continua

2.4.1. Variables

M_{gold} = Cantidad de metros cuadrados dedicados a ubicaciones Gold.

M_{silv} = Cantidad de metros cuadrados dedicados a ubicaciones Silver.

S_{gold} = Cantidad de pases Gold a vender.

S_{silv} = Cantidad de pases Silver a vender.

P = Cantidad de paquetes de merchandisign comprados.

P_{gold} = Cantidad de paquetes comprados destinados a ubicaciones Gold.

P_{silv} = Cantidad de paquetes comprados destinados a ubicaciones Silver.

2.4.2. Ecuaciones

- 8000 metros cuadrados en total disponibles:

$$M_{gold} + M_{silv} \leq 8000\, m^2$$

- Relación pases Gold y metros cuadrados:

$$S_{gold} = \frac{1\, pase}{m^2} \cdot M_{gold}$$

- Relación pases Silver y metros cuadrados:

$$S_{silv} = \frac{2\, pases}{m^2} \cdot M_{silv}$$

- 100 pases Golds al dueño del predio:

$$S_{gold} \geq 100\, pases$$

- 500 pases Silver vendidos para pagar la reserva del predio y 2 pases de protocolo para el ayuntamiento:

$$S_{silv} \geq 500\, pases + 2\, pases$$

- Paquetes de merchandising:

$$P = P_{gold} + P_{silv}$$

- Se pueden comprar hasta 800 paquetes:

$$P \leq 800\, paquetes$$

- Un paquete cubre hasta 20 personas que compraron un pase Silver:

$$P_{silv} \geq \frac{1\, paquete}{20\, pases} \cdot S_{silv}$$

- Un paquete cubre hasta 8 personas que compraron un pase Gold:

$$P_{gold} \geq \frac{1\, paquete}{8\, pases} \cdot S_{gold}$$

2.4.3. Funcional

$Ingresos = \frac{\$1500}{pases\ Gold} \cdot S_{gold} + \frac{\$700}{pases\ Silver} \cdot S_{silv}$

$Egresos = \frac{\$700}{pases\ Silver} \cdot 2\, pases\ protoc. + \frac{\$1500}{pases\ Gold} \cdot 100\, pases\ dueño + \frac{\$800}{paquete} \cdot P$

$Z = Ingresos - Egresos \longrightarrow Max.$

2.5. Modelo en computadora

Se utilizó el software `glpksol`, v4.57 para realizar la corrida del modelo propuesto, en el mismo, se utilizó el siguiente codigo:

2.5.1. Codigo utilizado para la simulación del modelo

```
#Resolución Parte B-Contenido del archivo B.mod
/*Declaración de variables*/
var Mgold >= 0;
var Msilv >= 0;
var Sgold >= 0;
var Ssilv >= 0;
var P>=0;
var Pgold, integer, >=0;
var Psilv, integer, >=0;
var Ingresos>=0;
var Egresos>=0;

/*Definición del funcional*/
maximize z: Ingresos-Egresos;

/*Restricciones*/

/*Equivalencia entre metros y pases*/
s.t. pasesMetrosG1: Sgold<=Mgold;
s.t. pasesMetrosG2: Sgold>=Mgold;
s.t. pasesMetrosS1: Ssilv<=2*Msilv;
s.t. pasesMetrosS2: Ssilv>=2*Msilv;

/*Metros totales disponibles*/
s.t. metrosTotales: Mgold+Msilv<=8000;

/*100 pases Gold minimos (por regalo al dueño)*/
s.t. minGold: Sgold>=100;

/*pases Silver minimos (reserva + ayuntamiento)*/
s.t. minSilver: Ssilv>=502;

/*Paquetes de merchandasing*/
s.t. Ptotales1: P>=Pgold+Psilv;
s.t. Ptotales2: P<=Pgold+Psilv;

/* Cantidad maxima de paquetes */
s.t. Pmax: P<=800;

/*Demanda minima de Psilv*/
s.t. demMinSilv: 20*Psilv>=Ssilv;

/*Demanda minima de Pgold*/
s.t. demMinGold: 8*Pgold>=Sgold;
```

```
/*Ingresos*/
s.t. ing1: 1500*Sgold+700*Ssilv>=Ingresos;
s.t. ing2: 1500*Sgold+700*Ssilv<=Ingresos;

/*Egresos*/
s.t. egr1: 800*P+700*2+1500*100<=Egresos;
s.t. egr2: 800*P+700*2+1500*100>=Egresos;

end;
```

Contenido del archivo B.mod utilizado para simular el modelo

2.6. Resolución

2.6.1. Archivo solución generado

Se adjunta a continuación el resultado de la corrida sobre glpk del modelo:

```
   Problem: B
Rows: 17
Columns: 9 (2 integer, 0 binary)
Non-zeros: 35
Status: INTEGER OPTIMAL
Objective: z = 10382600 (MAXimum)

   No. Row name Activity Lower bound Upper bound
------ ------------ ------------- ------------- -------------
     1 z 1.03826e+07
     2 pasesMetrosG1
                                0 -0
     3 pasesMetrosG2
                                0 -0
     4 pasesMetrosS1
                                0 -0
     5 pasesMetrosS2
                                0 -0
     6 metrosTotales
                             7974 8000
     7 minGold 104 100
     8 minSilver 15740 502
     9 Ptotales1 0 -0
    10 Ptotales2 0 -0
    11 Pmax 800 800
    12 demMinSilv 0 -0
    13 demMinGold 0 -0
```

```
    14 ing1 0 -0
    15 ing2 0 -0
    16 egr1 -151400 -151400
    17 egr2 -151400 -151400

   No. Column name Activity Lower bound Upper bound
------ ------------ ------------ ------------ -------------
     1 Mgold 104 0
     2 Msilv 7870 0
     3 Sgold 104 0
     4 Ssilv 15740 0
     5 P 800 0
     6 Pgold * 13 0
     7 Psilv * 787 0
     8 Ingresos 1.1174e+07 0
     9 Egresos 791400 0

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

Notemos que en el mismo se detalla que la cantidad de Pases gold a vender debe ser de 104 y la cantidad de pases Silver de 15740.

La solución del sistema arroja ingresos por $11174000 y egresos por $791400, lo que deja una ganancia de $10382600.

2.7. Análisis de la solución

En la misma se puede observar que el programa nos recomienda fuertemente no vender mas Pases gold que los que son regalados al dueño del predio.

Estimamos que el modelo se comporta de la esa manera por los siguientes hechos:

- Un pase silver ocupa la mitad de metros cuadrados que un pase gold
- Pese a que el ingreso por un pase Gold es mas del doble que el de un Pase Silver, hay que comprar mas paquetes de merchandasing.
- La relación de paquetes de merchandasing entre un pase y otro es de 20 a 8, por lo que esto podría ser un punto determinante en la desición del modelo, ya que podría hacer que la ganancia del pase gold no sea suficiente debido a estos gastos adicionales.