Inteligencia Artificial

Guía 2.1. Prolog

4° Licenciatura En Sistemas de Información

2020

Universidad: UADER FCyT Concepción del Uruguay

Profesor: Lopez De Luise Daniela, Bel Walter

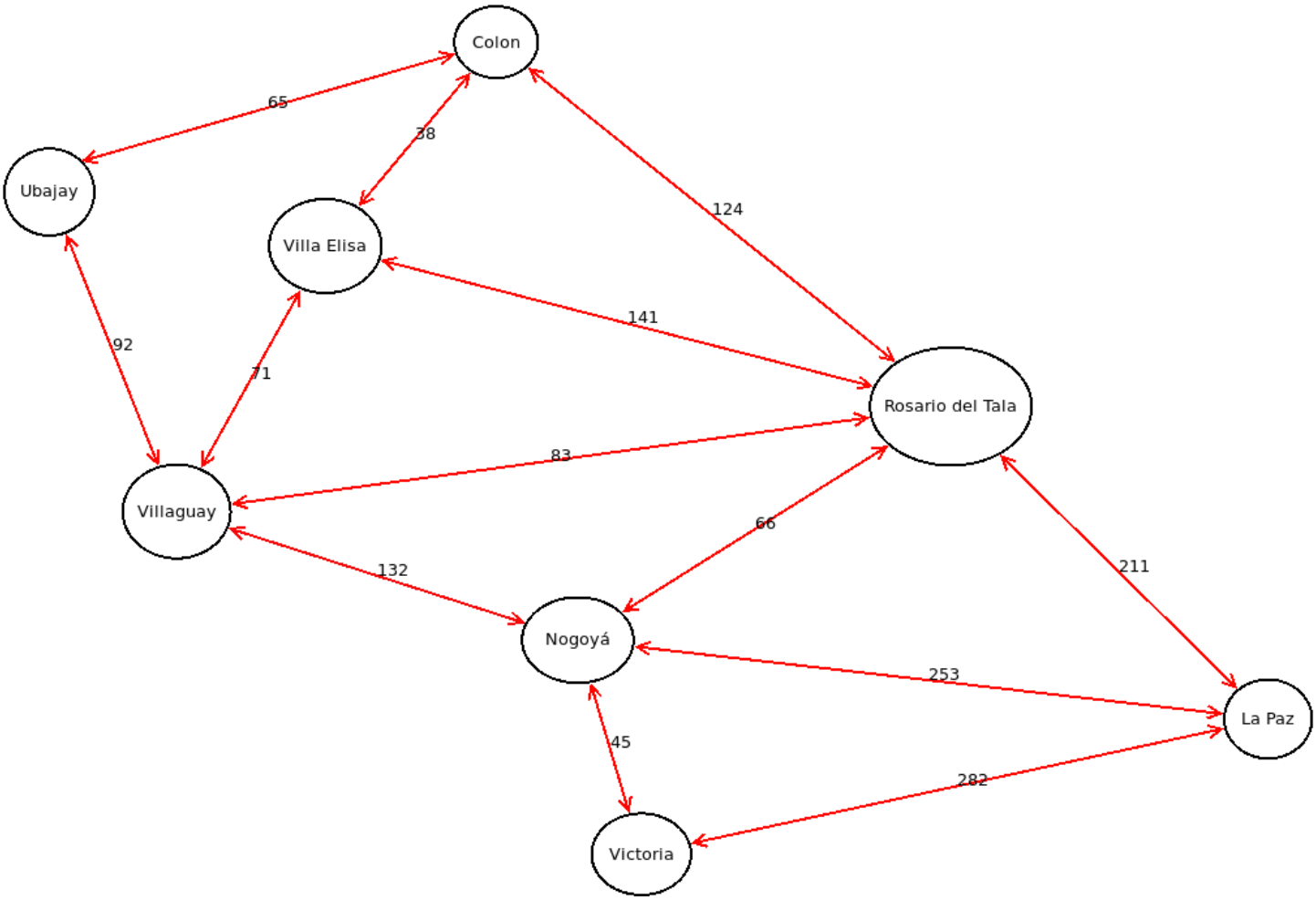
Alumnos: Exequiel Gonzalez, Cepeda Leandro

1. Sea el viajante de comercio que vive en Colón pero debe pasar por Ubajay, Villaguay, Rosario del Tala, Villa Elisa, Nogoyá, La Paz y Victoria. Luego debe volver a su casa. Todos los tramos los hace por las rutas del mapa. Hacer la tabla de distancias de cada ciudad al resto (siempre pasando por las rutas rojas)

Para el caso va a utilizar una escala de costo en km ∈ Z.

	Colon	Ubajay	Villaguay	Rosario del Tala	Villa Elisa	Nogoyá	La Paz	Victoria
Colon	0	65	108	124	38	187	271	231
Ubajay	65	0	92	168	59	232	241	277
Villaguay	108	92	0	83	71	146	163	191
Rosario del Tala	124	168	83	0	145	66	211	111
Villa Elisa	38	59	71	145	0	204	233	249
Nogoyá	187	232	146	66	204	0	253	45
La Paz	271	241	163	211	233	253	0	282
Victoria	231	277	191	111	249	45	282	0

2. Esquematizar la red de nodos del problema



3. Hacer el modelo matemático del problema del viajante

Sea x_{ij} igual 1, si existe el camino de ir de la i a la ciudad j, y 0 en otro caso, para el conjunto de ciudades 0,..., n. Sean ui para i = 1,..., n variables artificiales y sea c_{ij} la distancia desde la ciudad i a la ciudad j. Entonces el modelo de programación lineal en enteros puede ser escrito como:

$$
\begin{aligned}
&\min \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} c_{ij} x_{ij} \\
&0 \leq x_{ij} \leq 1 && i, j = 0, \ldots n \\
&x_{ij} \text{ integer} && i, j = 0, \ldots n \\
&\sum_{i=0, i \neq j}^{n} x_{ij} = 1 && j = 0, \ldots, n \\
&\sum_{j=0, j \neq i}^{n} x_{ij} = 1 && i = 0, \ldots, n \\
&u_i - u_j + n x_{ij} \leq n - 1 && 1 \leq i \neq j \leq n.
\end{aligned}
$$

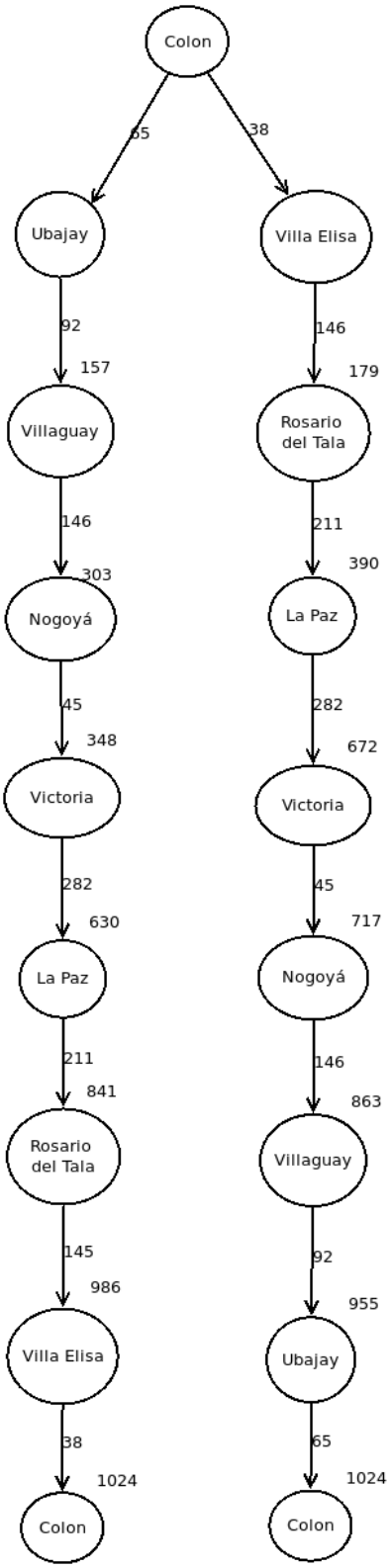
El primer conjunto de igualdades asegura que cada ciudad 0,..., n de salida llegue exactamente a una ciudad, y el segundo conjunto de igualdades aseguran que desde cada ciudad 1,..., n se salga exactamente hacia una ciudad (ambas restricciones también implican que exista exactamente una salida desde la ciudad 0.) La última restricción obliga a que un solo camino cubra todas las ciudades y no dos o más caminos disjuntos cubran conjuntamente todas las ciudades. Para probar esto se muestra en (1) que toda solución factible contiene solamente una secuencia cerrada de ciudades, y en (2) que para cada uno de los recorridos que cubren todas las ciudades, hay valores para todas las variables u_i que satisfacen las restricciones.

Para probar que cada solución factible contiene solamente una secuencia cerrada de ciudades, es suficiente mostrar que cada sub-ruta en una solución factible pasa a través de la ciudad 0 (note que las igualdades aseguran que solamente puede haber un recorrido de ese tipo). Por tanto, si sumamos todas las desigualdades correspondiente a x_{ij}=1 para cada sub-ruta de k pasos que no pasan a través de la ciudad 0, obtenemos nk <= (n-1)k, lo cual es una contradicción.

Ahora, mostramos que para cada recorrido que cubre todas las ciudades, hay valores de las variables u_i que satisfacen las restricciones.

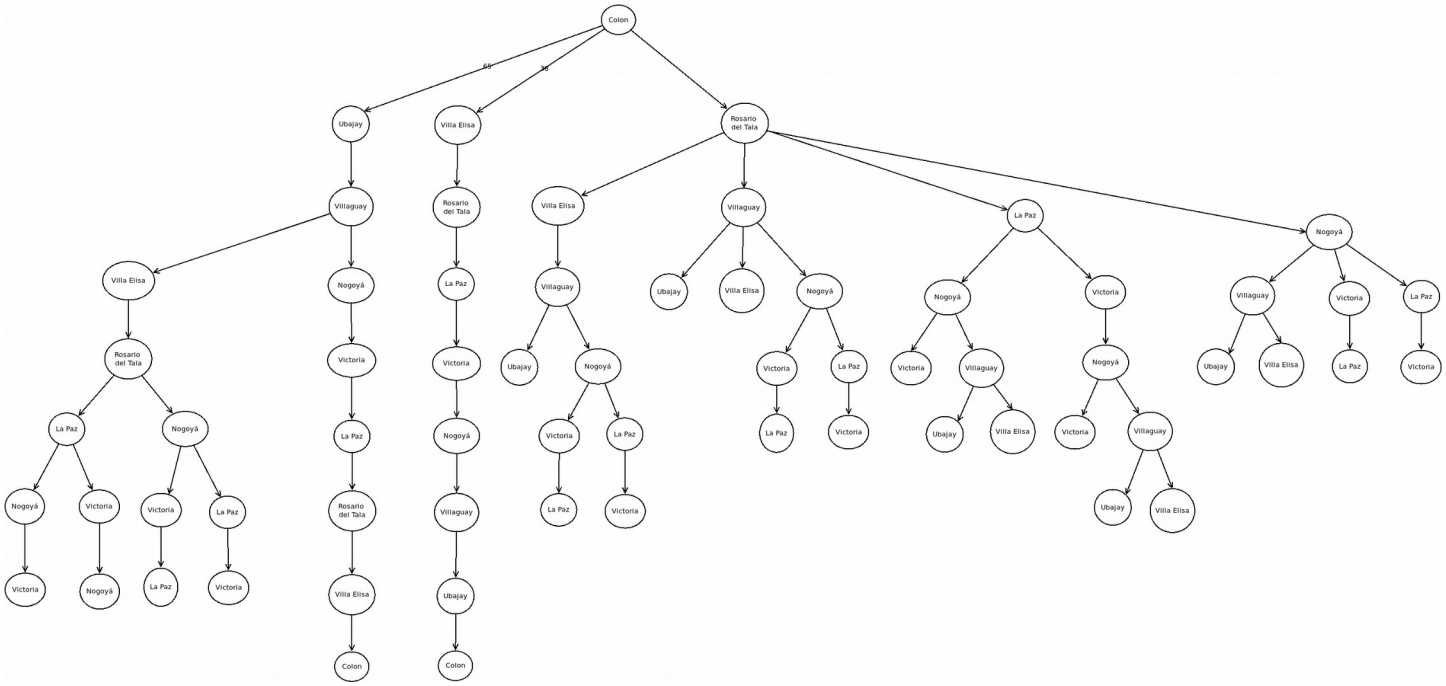
Sin pérdida de generalidad, se define el recorrido con origen y fin en la ciudad 0. Escoger $u_i = t$ si la ciudad i es visitada en el paso t (i, t = 1, 2,..., n). Entonces $u_i - u_j <= n-1$ dado u_i no puede ser mayor que n y u_j no puede ser menor que 1; por lo tanto las restricciones se satisfacen siempre que $x_{ij} = 0$. Para $x_{ij} = 1$, $u_i - u_j + nx_{ij} = (t) - (t+1) + n = n-1$, se satisfacen las restricciones.

4. Hacer el árbol correspondiente al problema



5. Resolver manualmente con Branch & Bound

Representamos algunas de las ramificaciones que no cumplen con el problema del viajante, dichas ramas no se representan en su totalidad por cuestiones de visualizacion.



6. Implementar con Prolog y resolver. Justificar.

ej6.pl

```
% this is a prolog rule file
% This is a declarative solver which generates solutions for a asymmetric travelling salesman problem

% solution(+Path, +RoadNetwork, -SolutionCost, -SolutionPath).
% Path is a list of cities in reverse order of being visited
% RoadNetwork is an adjacency list of the cities in the road network
% SolutionCost is the cost of a tour
% SolutionPath is the tour for which the SolutionCost was calculated

% solution(
%     [colon],
%     [
%         (colon,     [ (ubajay, 65),       (tala, 124),        (villaelisa, 38)   ]),
%         (ubajay,    [ (colon, 65),        (villaguay, 92)     ]),
%         (villaguay, [ (ubajay, 92),       (tala, 83),         (villaelisa, 70),   (nogoya, 146)]),
%          (tala,      [ (colon, 124),       (villaguay, 83),    (villaelisa, 145),  (nogoya, 66), (lapaz, 211)
]),
%         (villaelisa,[ (colon, 38),        (villaguay, 71),    (tala, 145)   ]),
%         (nogoya,    [ (villaguay, 146),   (tala, 66),         (lapaz, 253),  (victoria, 45)   ]),
%         (lapaz,     [ (tala, 211),        (nogoya, 253),      (victoria, 282) ]),
%         (victoria,  [ (nogoya, 45),       (lapaz, 282)    ])
%     ],
%     SolutionCost,
%     SolutionPath
% ).

solution(Path, RoadNetwork, SolutionCost, SolutionPath):-
    length(RoadNetwork,1),
    member(Start,Path),
    member((Start,[]),RoadNetwork),
    SolutionCost = 0,
    SolutionPath = [Start,Start].

solution(Path, RoadNetwork, SolutionCost, SolutionPath):-
    Costs = [],
    solution(Path, RoadNetwork, Costs, SolutionCost, SolutionPath).
```

```
solution(Path, RoadNetwork, Costs, SolutionCost, SolutionPath):-
    length(RoadNetwork,Length),
    length(Path,Length),
    [End|_] = Path,
    last(Path,Start),
    member((End,Roads), RoadNetwork),
    member((Start,Cost),Roads),
    sumlist([Cost|Costs], SolutionCost),
    reverse([Start|Path], SolutionPath).


solution(Path, RoadNetwork, Costs, SolutionCost, SolutionPath):-
    length(RoadNetwork,CityLength),
    length(Path,PathLength),
    PathLength < CityLength,
    [City|_] = Path,
    member((City,Roads), RoadNetwork),
    member((NewCity,NewCost),Roads),
    member((NewCity,_),RoadNetwork),
    is_set([NewCity|Path]),
    solution([NewCity|Path],RoadNetwork, [NewCost|Costs], SolutionCost, SolutionPath).
```

Resultado:

```
?- consult("ej6.pl").
true.

?- solution(
|        [colon],
|        [
|            (colon,     [ (ubajay, 65),       (tala, 124),        (villaelisa, 38)   ]),
|            (ubajay,    [ (colon, 65),        (villaguay, 92)     ]),
|            (villaguay, [ (ubajay, 92),       (tala, 83),         (villaelisa, 70),   (nogoya, 146)]),
|            (tala,      [ (colon, 124),       (villaguay, 83),    (villaelisa, 145),  (nogoya, 66), (lapaz, 211)  ]),
|            (villaelisa,[ (colon, 38),        (villaguay, 71),    (tala, 145)   ]),
|            (nogoya,    [ (villaguay, 146),   (tala, 66),         (lapaz, 253),  (victoria, 45)   ]),
|            (lapaz,     [ (tala, 211),        (nogoya, 253),      (victoria, 282) ]),
|            (victoria,  [ (nogoya, 45),       (lapaz, 282)    ])
|        ],
|        SolutionCost,
|        SolutionPath
|    ).
SolutionCost = 1024,
SolutionPath = [colon, ubajay, villaguay, nogoya, victoria, lapaz, tala, villaelisa, colon] ;
SolutionCost = 1024,
SolutionPath = [colon, villaelisa, tala, lapaz, victoria, nogoya, villaguay, ubajay, colon] ;
false.
```

Justificación: como se puede observar el resultado obtenido es el esperado, tal y como se muestra en el árbol que representa la solución al problema planteado en el ejercicio 4 de la guiá.