Universidad Veracruzana

Inteligencia Artificial

Programación para la Inteligencia Artificial

Actividad 3

Leonardo Flores Torres

17 de noviembre de 2022

1. Me estoy cambiando de casa y debo llevar a mi casa nueva a mi perro, mi gato y mi hamster que, sobra decirlo, no se llevan muy bien entre ellos. Mi mini auto solo me permite llevar a una mascota conmigo. De manera que, por ejemplo, puedo llevarme al gato, dejando solos al hamster y al perro, pero no puedo dejar juntos a éste último y al gato, ni al gato y al perro. Escribir un programa en Prolog para encontrar los movimientos válidos para pasar todas mis mascota de una casa a otra. Implemente una solución al problema mediante una búsqueda en el espacio de soluciones del problema. [25 puntos]

   Solución:

   Para resolver este problema se tomó como referencia la implementación en [1], ya que mantiene una redacción de código limpia a la par que sencilla mientras que se incluye una explicación detallada acerca de los elementos considerados para la solución.

```
% goal/1 denotes the desired state/4 which I want to be in after successfully
↪  moving out from my old house with all my pets. Whereas, state/4 allows me to
↪  know which pets are in which house at a given time.
goal(state(newhome, newhome, newhome, newhome)).

% change/2 denotes the allowed change in position for all involved parties, i.e.,
↪  move from the old house to the new one, and vice versa.
change(newhome, oldhome).
change(oldhome, newhome).

% move/2 modifies the current state me and my pets are in. Thus, if a move is
↪  made me and/or one of my pets will change positions between houses.
% Possible moves:
% a) Me and me alone move between houses.
move(state(X, Dog, Kitty, Rat), state(Y, Dog, Kitty, Rat)) :- change(X, Y).
% b) My dog and me.
move(state(X, X, Kitty, Rat), state(Y, Y, Kitty, Rat)) :- change(X, Y).
% c) My cat and me.
move(state(X, Dog, X, Rat), state(Y, Dog, Y, Rat)) :- change(X, Y).
% d) My little rat and me.
move(state(X, Dog, Kitty, X), state(Y, Dog, Kitty, Y)) :- change(X, Y).

% illegal/1 declares which moves shall not be made in order to avoid leaving
↪  alone those pets that cannot be so.
```

```
% a) My dog and my cat.
illegal(state(X, Y, Y, _)) :- change(X, Y).
% b) My cat and my little rat.
illegal(state(X, _, Y, Y)) :- change(X, Y).

% Solution
solution([Loc1|Locations], [Loc1|Locations]) :- goal(Loc1).
solution([Loc1|Locations], LS) :- move(Loc1, Loc2),
    not(illegal(Loc2)),
    not(member(Loc2, Locations)),
    solution([Loc2,Loc1|Locations], LS).

print_states([]) :-
    write('End'), nl,
    write('---').
print_states([State|States]) :-
    write(State), nl,
    print_states(States).

% print_states(States) :-
%     print_states_aux(States, 0).
% print_states_aux([], _) :-
%     write('End'), nl,
%     write('---').
% print_states_aux(States, Iter) :-
%     reverse(States, [St|Sts]),
%     Iter1 is Iter + 1,
%     write(Iter1), write(' >>> '), write(St), nl,
%     print_states_aux(Sts, Iter1).
```

Otra implementación leída fue [2] en donde se resuelve el problema de manera similar, aunque en vez de guardar el estado de los movimientos en una estructura como lo es `state/2` se usa una lista, también en [3]. Ya sea de la manera elegida ó usando una lista, la solución a la que se llega a partir de esta manera es similar sólamente cambiando la forma de guardar los datos y ajustando la representación de los mismos de la manera adecuada para trabajar con listas.

La solución para este problema, en las que puedo mover mis mascotas de una casa, se incluye a continuación:

```
?- [movingout].
true.

?- solution([state(oldhome, oldhome, oldhome, oldhome)], Sol), reverse(Sol,
↪  Sol1),
print_states(Sol1).
state(oldhome,oldhome,oldhome,oldhome)
state(newhome,oldhome,newhome,oldhome)
state(oldhome,oldhome,newhome,oldhome)
state(newhome,newhome,newhome,oldhome)
state(oldhome,newhome,oldhome,oldhome)
state(newhome,newhome,oldhome,newhome)
```

```
state(oldhome,newhome,oldhome,newhome)
state(newhome,newhome,newhome,newhome)
End
---
Sol = [state(newhome, newhome, newhome, newhome), state(oldhome, newhome,
↪  oldhome, newhome), state(newhome, newhome, oldhome, newhome), state(oldhome,
↪  newhome, oldhome, oldhome), state(newhome, newhome, newhome, oldhome),
↪  state(oldhome, oldhome, newhome, oldhome), state(newhome, oldhome, newhome,
↪  oldhome), state(oldhome, oldhome, oldhome, oldhome)],
Sol1 = [state(oldhome, oldhome, oldhome, oldhome), state(newhome, oldhome,
↪  newhome, oldhome), state(oldhome, oldhome, newhome, oldhome), state(newhome,
↪  newhome, newhome, oldhome), state(oldhome, newhome, oldhome, oldhome),
↪  state(newhome, newhome, oldhome, newhome), state(oldhome, newhome, oldhome,
↪  newhome), state(newhome, newhome, newhome, newhome)] ;
state(oldhome,oldhome,oldhome,oldhome)
state(newhome,oldhome,newhome,oldhome)
state(oldhome,oldhome,newhome,oldhome)
state(newhome,oldhome,newhome,newhome)
state(oldhome,oldhome,oldhome,newhome)
state(newhome,newhome,oldhome,newhome)
state(oldhome,newhome,oldhome,newhome)
state(newhome,newhome,newhome,newhome)
End
---
Sol = [state(newhome, newhome, newhome, newhome), state(oldhome, newhome,
↪  oldhome, newhome), state(newhome, newhome, oldhome, newhome), state(oldhome,
↪  oldhome, oldhome, newhome), state(newhome, oldhome, newhome, newhome),
↪  state(oldhome, oldhome, newhome, oldhome), state(newhome, oldhome, newhome,
↪  oldhome), state(oldhome, oldhome, oldhome, oldhome)],
Sol1 = [state(oldhome, oldhome, oldhome, oldhome), state(newhome, oldhome,
↪  newhome, oldhome), state(oldhome, oldhome, newhome, oldhome), state(newhome,
↪  oldhome, newhome, newhome), state(oldhome, oldhome, oldhome, newhome),
↪  state(newhome, newhome, oldhome, newhome), state(oldhome, newhome, oldhome,
↪  newhome), state(newhome, newhome, newhome, newhome)] ;
false.
```

Se tiene que partir de un estado inicial, en este caso

`state(oldhome,oldhome,oldhome,oldhome)`,

el cual representa el caso en el que todavía las mascotas y el dueño se encuentran en el viejo hogar antes de mudarse. Además, se tiene que declarar la meta que en este caso coincide con mover a todas las mascotas al nuevo hogar junto con el dueño especificada por el hecho

`goal(state(newhome,newhome,newhome,newhome))`.

Para este caso en particular se tiene una estructura similar al caso del mundo de los bloques visto en clase [4], la regla `move/2` es similar en estructura a la regla donde se define el sucesor para los bloques, en realidad es aqui en donde para ambos casos se declaran los movimientos permitidos. De hecho, en el mundo de los bloques también se define una meta a alcanzar; una manera en la que se desea que los bloques estén acomodados al final. Hay ciertos movimientos que no deben realizarse, por ejemplo,

el dejar a la rata y al gato solos, o al gato y al perro solos, estos movimientos no permitidos se definen en en la regla `illegal/1`. Finalmente, la solución se busca al usar `solution/2` como query en Prolog, donde se trata de hacer coincidir las ubicaciones del dueño y sus mascotas con la meta final, `goal/1`.

---

2. Aplique el algoritmo primero el mejor a un problema de su elección (diferente al visto en clase). Justifique la elección de sus predicados sucesor y meta. Justifique su función de costo y heurística. [20 puntos]

   Solución:

   El algoritmo de primero el mejor puede ser usado en muchas situaciones para encontrar la solución a un problema, por ejemplo, para problemas de búsquedas en árboles de desición. Si se toma la estrategia de búsqueda en árboles en amplitud mediante un modelo de búsqueda *greedy* (tomando la mejor opción, sin reconsiderar) obtenemos este algoritmo.

   Supongamos que un estudiante del curso de Programación para la Inteligencia Artificial se siente confundido con los contenidos que se están viendo. El estudiante, bien preparado para tales ocasiones en las que se siente totalmente perdido, saca de su mochila un pequeño juguete para niños que le trae confort. El típico rompecabezas de 8 piezas que seguido veía en los super mercados pero nunca entendió cual era el chiste de dichoso juguete hasta ahora. Al verlo se dá cuenta que las piezas están revueltas, y su toc le dice que debe ordenarlas como sigue:

   $$\begin{array}{ccc} 1 & 3 & 4 \\ 8 & & 2 \\ 7 & 6 & 5 \end{array} \quad \rightarrow \quad \begin{array}{ccc} 1 & 2 & 3 \\ 8 & & 4 \\ 7 & 6 & 5 \end{array} \tag{1}$$

   "Si tomo el 2, lo muevo al espacio vacío, luego tomo el 4 y lo muevo al espacio donde estaba el dos, y tomo el 3 y lo muevo a donde estaba el 4, listo ¡Ya está!" Pensó el estudiante mientras seguía sin entender la clase. Dejó su juguete sobre su banca para salir un momento por café, como todo estudiante de maestría, pero al regresar se encontró con que alguien había movido las piezas a la siguiente configuración:

   $$\begin{array}{ccc} 8 & 3 & 5 \\ 4 & 1 & 6 \\ 2 & 7 & \end{array} \tag{2}$$

   "¿Habrá sido el profesor por no ponerle atención?" Se dijo para sus adentros, y procedió a resolverlo igualmente.

   Dejando a un lado entretenida historia se procederá con la explicación. A continuación se muestra una parte del archivo `busquedaPrimeroMejor.pl`. Dicho archivo se vió en clase [4], y se modificó aquí para incluir lo necesario para resolver el rompecabezas. Para este caso, un nodo dentro del árbol de búsqueda es una configuración de las posiciones de los espacios (números), con coordenadas x y y mostrados como `X/Y`. Las distancias entre posiciones se calculan como la distancia Manhattan, y para este caso en particular todos los arcos tienen el mismo peso igual a 1.

Es importante recordar que aquí un nodo representa un estado de como están ordenadas las piezas en el juguete, o sea, cada movimiento lleva a una configuración distinta y dicha configuración es representada por un nodo. El objetivo es minimizar la cantidad de movimientos realizados para llegar a la configuración deseada.

```
% ------------------------------------------------------
% Eight puzzle


s([Vacio|Fichas], [Ficha|Fichas1], 1) :-
    cambio(Vacio, Ficha, Fichas, Fichas1).
% cambio/4 realiza cambios de piezas unicamente entre un espacio vacio y un
↪  espacio y uno no vacio.
cambio(Vacio, Ficha, [Ficha|Ts], [Vacio|Ts]) :-
    mandist(Vacio, Ficha, 1).
cambio(Vacio, Ficha, [T1|Ts], [T1|Ts1]) :-
    cambio(Vacio, Ficha, Ts, Ts1).

% Distancia Manhattan entre dos espacios, D.
mandist(X/Y, X1/Y1, D) :-
    dif(X, X1, Dx),
    dif(Y, Y1, Dy),
    D is Dx + Dy.

dif(A, B, D) :-
    D is abs(A - B).

% La heuristica h/2 es la suma de las distancias a cualquier espacio desde el
↪  espacio de origen mas 3 veces el puntaje de la secuencia.
h([Vacio|Fichas], H) :-
    meta([Vacio1|CuadrosMeta]),
    totdist(Fichas, CuadrosMeta, D),
    secpuntaje(Fichas, S),
    H is D + 3*S.

totdist([], [], 0).
totdist([Ficha|Fichas], [Cuadro|Cuadros], D) :-
    mandist(Ficha, Cuadro, D1),
    totdist(Fichas, Cuadros, D2),
    D is D1 + D2.

secpuntaje([Primera|OtrasFichas], S) :-
    secpuntaje([Primera|OtrasFichas], Primera, S).
secpuntaje([Ficha1,Ficha2|Fichas], Primera, S) :-
    puntaje(Ficha1, Ficha2, S1),
    secpuntaje([Ficha2|Fichas], Primera, S2),
    S is S1 + S2.
secpuntaje([Last], Primera, S) :-
    puntaje(Last, Primera, S).

puntaje(2/2, _, 1) :- !.
puntaje(1/3, 2/3, 0) :- !.
puntaje(2/3, 3/3, 0) :- !.
puntaje(3/3, 3/2, 0) :- !.
```

```
48 puntaje(3/2, 3/1, 0) :- !.
49 puntaje(3/1, 2/1, 0) :- !.
50 puntaje(2/1, 1/1, 0) :- !.
51 puntaje(1/1, 1/2, 0) :- !.
52 puntaje(1/2, 1/3, 0) :- !.
53 puntaje(_, _, 2).
54
55 meta([2/2, 1/3, 2/3, 3/3, 3/2, 3/1, 2/1, 1/1, 1/2]).
56
57 muestrasol([]).
58 muestrasol([P|L]) :-
59     muestrasol(L), nl,
60     write('+-----+'),
61     muestrapos(P), nl,
62     write('+-----+').
63
64 muestrapos([S0, S1, S2, S3, S4, S5, S6, S7, S8]) :-
65     member(Y, [3,2,1]), nl,
66     member(X, [1,2,3]),
67     member(Ficha-X/Y,
68         [' '-S0, 1-S1, 2-S2, 3-S3, 4-S4, 5-S5, 6-S6, 7-S7, 8-S8]),
69     write(' '), write(Ficha),
70     fail
71     ;
72     true.
73
74 % Estados con distintas configuraciones iniciales del rompecabezas, se muestran
   ↪  sus numeros como referencia, y sus posiciones en x/y.
75 % Numeros: 0,   1,   2,   3,   4,   5,   6,   7,   8.
76 estado1([2/2, 1/3, 3/2, 2/3, 3/3, 3/1, 2/1, 1/1, 1/2]).
77 estado2([2/1, 1/2, 1/3, 3/3, 3/2, 3/1, 2/2, 1/1, 2/3]).
78 estado3([2/2, 2/3, 1/3, 3/1, 1/2, 2/1, 3/3, 1/1, 3/2]).
79 estado4([3/1, 2/2, 1/1, 2/3, 1/2, 3/3, 3/2, 2/1, 1/3]).
```

La manera en como se ordenan las posiciones de los espacios es mediante una lista donde el primer elemento de la lista contiene la posición de la pieza vacía, el segundo elemento la posición de la pieza con el número 1, el tercer elemento la posición de la pieza con el número 2, y así hasta el último elemento que contiene la posición de la pieza de número 8. La configuración a la que se desea llegar está declarada en el hecho `meta/1`.

La regla `totdist/3` calcula la distancia total de todas las piezas fuera de lugar con respecto al orden deseado final declarado en `meta/1`. De manera similar, la regla `secpuntaje/2` y `secpuntaje/3` calculan dan un valor al grado de ordenamiento respecto a que tan ordenadas se encuentran las en comparación con la configuración final deseada. El puntaje se computa como la suma de todos los puntajes individuales de las piezas mediante las siguientes consideraciones: una pieza con número en el centro tiene un puntaje de 1; una pieza no en el centro tiene un puntaje de 0 si está seguida de su sucesor; una pieza no en el centro tiene un puntaje de 2 si no está seguida de su sucesor. Estas condiciones para asignar puntajes dependiendo de la posición de las piezas están declaradas en las reglas `puntaje/3`.

La función heurística `h/2` elegida es la suma de las distancias de las piezas desde donde están hasta donde se supone que deberían estar en el estado meta sumándosele 3 veces el puntaje de la secuencia. Aunque esta función heurística es buena en el sentido que dirige la búsqueda hacia la configuración meta resulta que no garantiza encontrar siempre la solución más corta [5].

Al hacer un query en Prolog para encontrar la solución a la configuración inicial mostrada en la ecuación (1) obtenemos los siguientes pasos:

```
?- [busquedaPrimeroMejor].
true.

?- estado1(Pos), primeroMejor(Pos, Sol), muestrasol(Sol).

+-----+
1 3 4
8   2
7 6 5
+-----+
+-----+
1 3 4
8 2
7 6 5
+-----+
+-----+
1 3
8 2 4
7 6 5
+-----+
+-----+
1   3
8 2 4
7 6 5
+-----+
+-----+
1 2 3
8   4
7 6 5
+-----+
Pos = [2/2, 1/3, 3/2, 2/3, 3/3, 3/1, 2/1, 1/1, ... / ...],
Sol = [[2/2, 1/3, 2/3, 3/3, 3/2, 3/1, 2/1, ... / ...|...], [2/3, 1/3, 2/2, 3/3,
↪  3/2, 3/1, ... / ...|...], [3/3, 1/3, 2/2, 2/3, 3/2, ... / ...|...], [3/2,
↪  1/3, 2/2, 2/3, ... / ...|...], [2/2, 1/3, 3/2, ... / ...|...]] .
```

El caso de la ecuación (2) después de que el estudiante regresó a ver su rompecabezas es más complicado de resolver, esta configuración fue vista en [6]. Claramente no habría sido tan trivial llegar a la configuración deseada para el desafortunado estudiante.

```
?- estado4(Pos), primeroMejor(Pos, Sol), muestrasol(Sol).

+-----+
8 3 5
```

```
4 1 6
2 7
+-----+
+-----+
8 3 5
4 1 6
2   7
+-----+
+-----+
8 3 5
4   6
2 1 7
+-----+
+-----+
8 3 5
4 6
2 1 7
+-----+
+-----+
3 5
8 4 6
2 1 7
+-----+
+-----+
3   5
8 4 6
2 1 7
+-----+
+-----+
3 4 5
8   6
2 1 7
+-----+
+-----+
3 4 5
8 6
2 1 7
+-----+
+-----+
3 4 5
2 8 6
1 7
+-----+
+-----+
3 4 5
2 8 6
1   7
+-----+
+-----+
3 4 5
2 8 6
1 7
+-----+
+-----+
```

```
3 4 5
2 8
1 7 6
+-----+
+-----+
3 4
2 8 5
1 7 6
+-----+
+-----+
3   4
2 8 5
1 7 6
+-----+
+-----+
3 4
2 8 5
1 7 6
+-----+
+-----+
2 3 4
8 5
1 7 6
+-----+
+-----+
2 3 4
1 8 5
7 6
+-----+
+-----+
2 3 4
1 8 5
7   6
+-----+
+-----+
2 3 4
1 8 5
7 6
+-----+
+-----+
2 3 4
1 8
7 6 5
+-----+
+-----+
2 3
1 8 4
7 6 5
+-----+
+-----+
2   3
1 8 4
7 6 5
+-----+
```

```
+-----+
2 3
1 8 4
7 6 5
+-----+
+-----+
1 2 3
8 4
7 6 5
+-----+
+-----+
1 2 3
8   4
7 6 5
+-----+
Pos = [3/1, 2/2, 1/1, 2/3, 1/2, 3/3, 3/2, 2/1, ... / ...],
Sol = [[2/2, 1/3, 2/3, 3/3, 3/2, 3/1, 2/1, ... / ...|...], [1/2, 1/3, 2/3, 3/3,
↪  3/2, 3/1, ... / ...|...], [1/3, 1/2, 2/3, 3/3, 3/2, ... / ...|...], [2/3,
↪  1/2, 1/3, 3/3, ... / ...|...], [3/3, 1/2, 1/3, ... / ...|...], [3/2, 1/2, ...
↪  / ...|...], [3/1, ... / ...|...], [... / ...|...], [...|...]|...] .
```

---

3. Revise el artículo de Quinlan, Induction of Decision Trees, Machine Learning 1: 81-106, 1986; con el objetivo de identificar aquellos aspectos que podrían mejorar nuestra implementación básica de ID3. Elija uno de ellos y agregue la mejora a nuestro programa. Reporte en una página la mejora elegida, el diseño experimental para verificar los efectos de la mejora y los resultados obtenidos. [35 puntos]

   Solución:

   La mejora que se eligió agregar al programa fue utilizar el criterio de la razón de ganancia en vez de utilizar la ganancia únicamente. A continuación se muestra un extracto del programa `id3Improved.pl` donde se muestran las modificaciones pertinentes. La razón de ganancia se agregó en la regla `eligeAtr/5` como `GainRatio`, para lo cual se tuvo que computar `splitInformation`. Se siguió la estructura de la regla `informacionResidual/3` para escribir la regla de `splitInformation/3`.

```
eligeAtr(Ejs,Atrs,Atr,Vals,RestoAtrs) :-
    length(Ejs,NumEjs),
    contenidoInformacion(Ejs,NumEjs,I), !,
    findall((Atr-Vals)/GainRatio, % Cambio de Gain por GainRatio
            (member(Atr,Atrs),
             vals(Ejs,Atr,[],Vals),
             separaEnSubConjs(Vals,Ejs,Atr,Parts),
             informacionResidual(Parts,NumEjs,IR),
             % Modificacion para calcular splitInformation
             splitInformation(Parts, NumEjs, SI),
             Gain is I - IR,
             % Modificacion para calcular GainRatio
             GainRatio is Gain / (SI + 1)),
```

```
            Todos),
    write(Todos),
    maximo(Todos,(Atr-Vals)/_),
    eliminar(Atr,Atrs,RestoAtrs), !.

% informacionResidual = Sum (Sv / S) E(Sv) donde v in A
% splitInformation = - Sum (Si / S) log2(Si / S) donde i = 1,...,c
% gainRatio = Gain / splitInformation

splitInformation([], _, 0) :- !.
splitInformation([Part|Parts], NumEjs, SI) :-
    length(Part, NumEjsPart),
    splitInformation(Parts, NumEjs, S),
    SI is S - (NumEjsPart / NumEjs) * log(NumEjsPart / NumEjs) / log(2).

separaEnSubConjs([],_,_,[]) :- !.
separaEnSubConjs([Val|Vals],Ejs,Atr,[Part|Parts]) :-
    subconj(Ejs,Atr=Val,Part), !,
    separaEnSubConjs(Vals,Ejs,Atr,Parts).

informacionResidual([],_,0) :- !.
informacionResidual([Part|Parts],NumEjs,IR) :-
    length(Part,NumEjsPart),
    contenidoInformacion(Part,NumEjsPart,I), !,
    informacionResidual(Parts,NumEjs,R),
    IR is R + I * NumEjsPart/NumEjs.
```

La razón de ganancia está dada por

$$\text{gainRatio}(S, A) = \frac{\text{gain}(S, A)}{\text{splitInformation}(S, A)}, \tag{3}$$

donde la ganancia es

$$\text{gain}(S, A) = E(S) - \sum_{v \in A} \frac{|S_v|}{|S|} E(S_v), \tag{4}$$

y el término que nos interesa es el que se encuentra el denominador el cual está dado por

$$\text{splitInformation}(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}. \tag{5}$$

Como se vió en clase, esta expresión tiene un problema cuando hay atributos que tienen el mismo valor en todos sus ejemplos, haciendo así que $|S_i| \approx |S|$ ya que el denominador en $\text{gainRatio}(S, A)$ podría diverger. Una manera en como se pudiera solucionar este problema es deliveradamente sumando una unidad al denominador

$$\text{gainRatio}(S, A) = \frac{\text{gain}(S, A)}{\text{splitInformation}(S, A) + 1}. \tag{6}$$

De hecho, no se puede encontrar una solución para el archivo del zoológico con esta mejora sin sumar una unidad al denominador ya que se da el problema mencionado de la división sobre cero.

Primero se probó `id3.pl`, sin la mejora, con el archivo `tenis.csv`; se obtuvo lo siguiente (las listas resultantes deliveradamente se muestran ordenadas en todos los resultados de los queries ya que sería complicado de leer si no fuese el caso):

```
?- [id3].
true.

?- id3("tenis.csv").
[(cielo-[lluvioso,nublado,soleado])/0.246749819774439,
(temperatura-[fresca,templada,alta])/0.029222565658954647,
(humedad-[normal,alta])/0.15183550136234136,
(viento-[fuerte,debil])/0.04812703040826927]
[(temperatura-[fresca,templada])/0.01997309402197478,
(humedad-[normal,alta])/0.01997309402197478,
(viento-[fuerte,debil])/0.9709505944546686]
[(temperatura-[fresca,templada,alta])/0.5709505944546686,
(humedad-[normal,alta])/0.9709505944546686,
(viento-[fuerte,debil])/0.01997309402197478]
cielo=lluvioso
    viento=fuerte ⇒ [no/2]
    viento=debil ⇒ [si/3]
cielo=nublado ⇒ [si/4]
cielo=soleado
    humedad=normal ⇒ [si/2]
    humedad=alta ⇒ [no/3]
true.
```

En comparación, el programa mejorado, `id3Improved.pl`, da los siguientes resultados:

```
?- [id3Improved].
true.

?- id3("tenis.csv").
[(cielo-[lluvioso,nublado,soleado])/0.09573570973892705,
(temperatura-[fresca,templada,alta])/0.011429991978842776,
(humedad-[normal,alta])/0.07591775068117068,
(viento-[fuerte,debil])/0.024242569171122667]
[(temperatura-[fresca,templada])/0.01013373652194515,
(humedad-[normal,alta])/0.01013373652194515,
(viento-[fuerte,debil])/0.4926306104204075]
[(temperatura-[fresca,templada,alta])/0.2263944779441339,
(humedad-[normal,alta])/0.4926306104204075,
(viento-[fuerte,debil])/0.01013373652194515]
cielo=lluvioso
    viento=fuerte ⇒ [no/2]
    viento=debil ⇒ [si/3]
cielo=nublado ⇒ [si/4]
cielo=soleado
    humedad=normal ⇒ [si/2]
    humedad=alta ⇒ [no/3]
true.
```

De antemano se sabía que el árbol resultante para este archivo no cambiaría ya que

en clase se había visto que este era un archivo con atributos bien portados porque así fue diseñado. Lo que si puede observarse son los cambios en valores de sus ganancias `Gain`'s con respecto a las razones de ganancia `GainRatio`'s, respectivamente. De manera similar, se buscó el árbol resultante para el archivo `zoo.csv` usando la versión no mejorada:

```
?- [id3].
true.

?- id3("zoo.csv").
[(name-[...])/2.390559682294039,
(hair-[0,1])/0.7906745736101792,
(feathers-[1,0])/0.7179499765002912,
(eggs-[1,0])/0.8301384483633478,
(milk-[0,1])/0.9743197211096903,
(airbone-[1,0])/0.46970260950477294,
(aquatic-[1,0])/0.3894874837982223,
(predator-[0,1])/0.09344704054083186,
(toothed-[0,1])/0.8656941534932372,
(backbone-[0,1])/0.6761627418829199,
(breathes-[0,1])/0.6144940279390552,
(venomous-[1,0])/0.13308962953512316,
(fins-[1,0])/0.4666135671503897,
(legs-[5,8,6,2,0,4])/1.3630469031539394,
(tail-[1,0])/0.5004604482515027,
(domestic-[1,0])/0.05066877984551832,
(catsize-[0,1])/0.3084903449142815]
name=wren ⇒ [2/1]
name=worm ⇒ [7/1]
name=wolf ⇒ [1/1]
name=wasp ⇒ [6/1]
name=wallaby ⇒ [1/1]
name=vulture ⇒ [2/1]
name=vole ⇒ [1/1]
name=vampire ⇒ [1/1]
name=tuna ⇒ [4/1]
name=tuatara ⇒ [3/1]
name=tortoise ⇒ [3/1]
name=toad ⇒ [5/1]
name=termite ⇒ [6/1]
name=swan ⇒ [2/1]
name=stingray ⇒ [4/1]
name=starfish ⇒ [7/1]
name=squirrel ⇒ [1/1]
name=sparrow ⇒ [2/1]
name=sole ⇒ [4/1]
name=slug ⇒ [7/1]
name=slowworm ⇒ [3/1]
name=skua ⇒ [2/1]
name=skimmer ⇒ [2/1]
name=seawasp ⇒ [7/1]
name=seasnake ⇒ [3/1]
name=sealion ⇒ [1/1]
name=seal ⇒ [1/1]
```

```
name=seahorse ⇒ [4/1]
name=scorpion ⇒ [7/1]
name=rhea ⇒ [2/1]
name=reindeer ⇒ [1/1]
name=raccoon ⇒ [1/1]
name=pussycat ⇒ [1/1]
name=puma ⇒ [1/1]
name=porpoise ⇒ [1/1]
name=pony ⇒ [1/1]
name=polecat ⇒ [1/1]
name=platypus ⇒ [1/1]
name=pitviper ⇒ [3/1]
name=piranha ⇒ [4/1]
name=pike ⇒ [4/1]
name=pheasant ⇒ [2/1]
name=penguin ⇒ [2/1]
name=parakeet ⇒ [2/1]
name=ostrich ⇒ [2/1]
name=oryx ⇒ [1/1]
name=opossum ⇒ [1/1]
name=octopus ⇒ [7/1]
name=newt ⇒ [5/1]
name=moth ⇒ [6/1]
name=mongoose ⇒ [1/1]
name=mole ⇒ [1/1]
name=mink ⇒ [1/1]
name=lynx ⇒ [1/1]
name=lobster ⇒ [7/1]
name=lion ⇒ [1/1]
name=leopard ⇒ [1/1]
name=lark ⇒ [2/1]
name=ladybird ⇒ [6/1]
name=kiwi ⇒ [2/1]
name=housefly ⇒ [6/1]
name=honeybee ⇒ [6/1]
name=herring ⇒ [4/1]
name=hawk ⇒ [2/1]
name=hare ⇒ [1/1]
name=hamster ⇒ [1/1]
name=haddock ⇒ [4/1]
name=gull ⇒ [2/1]
name=gorilla ⇒ [1/1]
name=goat ⇒ [1/1]
name=gnat ⇒ [6/1]
name=girl ⇒ [1/1]
name=giraffe ⇒ [1/1]
name=fruitbat ⇒ [1/1]
name=frog ⇒ [5/2]
name=flea ⇒ [6/1]
name=flamingo ⇒ [2/1]
name=elephant ⇒ [1/1]
name=duck ⇒ [2/1]
name=dove ⇒ [2/1]
name=dolphin ⇒ [1/1]
```

```
name=dogfish ⇒ [4/1]
name=deer ⇒ [1/1]
name=crow ⇒ [2/1]
name=crayfish ⇒ [7/1]
name=crab ⇒ [7/1]
name=clam ⇒ [7/1]
name=chub ⇒ [4/1]
name=chicken ⇒ [2/1]
name=cheetah ⇒ [1/1]
name=cavy ⇒ [1/1]
name=catfish ⇒ [4/1]
name=carp ⇒ [4/1]
name=calf ⇒ [1/1]
name=buffalo ⇒ [1/1]
name=boar ⇒ [1/1]
name=bear ⇒ [1/1]
name=bass ⇒ [4/1]
name=antelope ⇒ [1/1]
name=aardvark ⇒ [1/1]
true.
```

El resultado de usar la versión mejorada en el archivo `zoo.csv` se muestra a continuación. Es de importancia mencionar que se cortaron los contenidos de los nombres de los animales en el output a `name-[...]` lo cual no significa que sean los mismos nombres en todas las instancias en que se sustituyó, solamente se hizo con motivos visuales. Claramente es mejor el árbol producido por la versión mejorada, y deja de haber valores tan altos para la ganancia al tomar la razón de ganancia en su lugar. El árbol encontrado al usar la versión no mejorada evidentemente no es confiable ya que toma al nombre de los animiales para producir el árbol y no los demás atributos.

```
?- [id3Improved].
true.

?- id3("zoo.csv").
[(name-[...])/0.3129656352536219,
(hair-[0,1])/0.3985193700897312,
(feathers-[1,0])/0.41791087419370476,
(eggs-[1,0])/0.4193749004253327,
(milk-[0,1])/0.4934964234475974,
(airbone-[1,0])/0.26224747236079304,
(aquatic-[1,0])/0.20079938221203444,
(predator-[0,1])/0.04692467035277209,
(toothed-[0,1])/0.4397541374313168,
(backbone-[0,1])/0.4033992195312441,
(breathes-[0,1])/0.3536677511822206,
(venomous-[1,0])/0.09510599800201057,
(fins-[1,0])/0.2821395061374844,
(legs-[5,8,6,2,0,4])/0.44928532086243317,
(tail-[1,0])/0.27455031912253797,
(domestic-[1,0])/0.03260754038982052,
(catsize-[0,1])/0.15517496414910664]
[(name-[...])/0.3468369933781861,
```

```
(hair-[1,0])/0.1625776658086195,
(feathers-[1,0])/0.47870397138568,
(eggs-[0,1])/0.059888635754324786,
(airbone-[1,0])/0.30761861371544025,
(aquatic-[0,1])/0.2421252966366888,
(predator-[0,1])/0.0786209249795597,
(toothed-[0,1])/0.4518493433427184,
(backbone-[0,1])/0.4684500943427063,
(breathes-[1,0])/0.37590485104307575,
(venomous-[1,0])/0.07548664319897222,
(fins-[0,1])/0.42988564191959355,
(legs-[5,8,6,4,2,0])/0.5388863369933956,
(tail-[0,1])/0.409761977030914,
(domestic-[1,0])/0.03769374430672337,
(catsize-[1,0])/0.05627296510141827]
[(name-[...])/0.16703843262486698,
(hair-[1,0])/0.08673509875673362,
(feathers-[0])/0.0,
(eggs-[1])/0.0,
(airbone-[1,0])/0.1633364609915221,
(aquatic-[0,1])/0.4192556570921077,
(predator-[0,1])/0.2373047915415823,
(toothed-[0])/0.0,
(backbone-[0])/0.0,
(breathes-[1,0])/0.4192556570921077,
(venomous-[1,0])/0.042339512048454646,
(fins-[0])/0.0,
(tail-[0])/0.0,
(domestic-[1,0])/0.0232431187531189,
(catsize-[0])/0.0]
[(name-[...])/0.3915173738220576,
(hair-[0])/ -2.220446049250313e-16,
(feathers-[0])/ -2.220446049250313e-16,
(eggs-[1])/ -2.220446049250313e-16,
(airbone-[0])/ -2.220446049250313e-16,
(aquatic-[0,1])/0.46326608332742664,
(predator-[0,1])/0.06109040127713122,
(toothed-[1,0])/0.30991353570669133,
(backbone-[1,0])/0.3717301612139897,
(breathes-[1,0])/0.3717301612139897,
(venomous-[1,0])/0.08047211751995782,
(fins-[0])/ -2.220446049250313e-16,
(tail-[1,0])/0.26276105344011635,
(domestic-[0])/ -2.220446049250313e-16,
(catsize-[1,0])/0.19222449298941552]
[(name-[...])/0.24707250536060574,
(hair-[0])/0.0,
(feathers-[0])/0.0,
(eggs-[1])/0.0,
(airbone-[0])/0.0,
(predator-[0,1])/0.042339512048454646,
(toothed-[1,0])/0.4192556570921077,
(backbone-[1,0])/0.4192556570921077,
(breathes-[1,0])/0.4192556570921077,
```

```
(venomous-[1,0])/0.042339512048454646,
(fins-[0])/0.0,
(tail-[1,0])/0.042339512048454646,
(domestic-[0])/0.0,
(catsize-[0])/0.0]
[(name-[...])/0.24030724958799854,
(hair-[0])/0.0,
(feathers-[0])/0.0,
(eggs-[0,1])/0.11555731781351165,
(airbone-[0])/0.0,
(aquatic-[0,1])/0.2822747746770287,
(predator-[0,1])/0.05446862743551582,
(toothed-[0,1])/0.4192556570921077,
(backbone-[0,1])/0.4192556570921077,
(breathes-[1,0])/0.22311252189907485,
(venomous-[1,0])/0.09734423813981084,
(fins-[0,1])/0.48295511255633944,
(tail-[0,1])/0.4192556570921077,
(domestic-[1,0])/0.024944586956176348,
(catsize-[1,0])/0.0831094684903519]
[(name-[...])/0.25876971183495706,
(hair-[0])/0.0,
(feathers-[0])/0.0,
(eggs-[0,1])/0.12447120023429868,
(airbone-[0])/0.0,
(aquatic-[1,0])/0.003208440465220612,
(predator-[0,1])/0.1565609880859559,
(toothed-[1,0])/0.496279555055255,
(backbone-[1,0])/0.496279555055255,
(breathes-[1,0])/0.010197421035043602,
(venomous-[1,0])/0.06451917367404496,
(tail-[1,0])/0.496279555055255,
(domestic-[0])/0.0,
(catsize-[0])/0.0]
milk=0
    legs=5 ⇒ [7/1]
    legs=8 ⇒ [7/2]
    legs=6
    breathes=1 ⇒ [6/8]
    breathes=0 ⇒ [7/2]
    legs=4
    aquatic=0 ⇒ [3/2]
    aquatic=1
        breathes=1 ⇒ [5/4]
        breathes=0 ⇒ [7/1]
    legs=2 ⇒ [2/20]
    legs=0
    fins=0
        tail=1 ⇒ [3/3]
        tail=0 ⇒ [7/4]
    fins=1 ⇒ [4/13]
milk=1 ⇒ [1/41]
true.
```

4. Revisen el artículo de Riccardo Buscaroli et al., A Prolog application for reasoning on maths puzzles with diagrams, Journal of Experimental & Theoretical Artificial Intelligence, 2022. Preparen una presentación conjunta sobre el problema que plantea el paper y el uso de Prolog para resolverlo. Discutan en las conclusiones la pertinencia, o no, del uso de este lenguaje para el problema en cuestión. [20 puntos]

Referencias

[1] Fabian Quijosaca. Prolog: Acertijo del granjero, la cabra, el lobo y la col. `https://dev.to/foqc/prolog-acertijo-del-granjero-la-cabra-el-lobo-y-la-col-4bd1`, 2020. Visitado: 2022-11-13.

[2] Sorin Lerner & Ranjit Jhala. What is prolog good for? `https://cseweb.ucsd.edu/classes/wi09/cse130/misc/prolog/goat_etc.html`, 2009. Visitado: 2022-11-12.

[3] Randy Latimer. Prolog program 2: A day at the river. `https://www.tjhsst.edu/~rlatimer/assignments2004/farmerPrologBak.html`, 2004. Visitado: 2022-11-12.

[4] Alejandro Guerra-Hernandez. Programación para la inteligencia artificial. `https://www.uv.mx/personal/aguerra/pia/`, 2022. Visitado: 2022-10-07.

[5] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2012.

[6] Josh Richard. An application using artificial intelligence. `https://www.d.umn.edu/~jrichar4/8puz.html`. Visitado: 2022-16-12.