PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Escuela de Ingeniería
IIC2613 – Inteligencia Artificial
2023-2

Control Largo 1 (PAUTA)

Lunes 25 de septiembre

Aspectos generales

Formato y plazo de entrega

El formato de entrega será en papel, subiendo sus respuestas para las preguntas de desarrollo a Canvas en forma de fotografías (se recomienda utilizar una aplicación para escanear con el teléfono), además, deberás reportar tu asistencia al control antes de abandonar la sala al cuerpo docente, verificando que lo marquen dentro de la lista de asistentes.

Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo para este control debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente levantando tu mano para ser atendido. Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

Razonamiento Lógico para la IA

Pregunta 1 (2 ptos.)

Dado el siguiente programa:

```
padre(bastian, ignacio).   % bastian es el padre de ignacio
madre(constanza, ignacio). % constanza es la madre de ignacio
% se omiten más hechos que describen un árbol genealógico
progenitor(X, Y) :- padre(X, Y).
progenitor(X, Y) :- madre(X, Y).
```

1. Escribe el predicado `hermanx(X,Y)` que se satisface cuando `X` es hermano o hermana de `Y`. ¡Asegúrate que este predicado no deduzca que una persona es hermana de sí misma! (1 pto.)

   ```
   hermanx(X,Y):- progenitor(P,X), progenitor(P,Y), X != Y.
   ```

2. Escribe el predicado `primx(X,Y)`, que se satisface cuando `X` es primo hermano o prima hermana de `Y`. Como arriba, asegúrate que `primo` no captura pares de valores que no corresponden. (1 pto.)

   ```
   primx(X,Y):- progenitor(XX,X), progenitor(YY,Y), hermanx(XX,YY).
   ```

Pregunta 2 (2 ptos.)

Como bien sabes, un número natural n es un número primo si n es divisible solo por 1 y por n. El objetivo de esta pregunta es que escribas un programa para encontrar todos los primos menores o iguales a `m`. Suponiendo que tu programa ya tiene una regla:

```
numero(1..m).
```

1. Escribe el predicado `compuesto`, tal que `compuesto(N)` se satisface si existe un número `M` mayor que 1 y menor que `N`, que divide a `N`. *Ayuda:* Usa la expresión `L\R` para calcular el módulo de la división entre `L` y `R`. (`L\R` es el equivalente de `L % R` en Python.) (1 pto.)

   ```
   compuesto(N):- 1 < M, M < N, N \ M = 0, numero(N), numero(M).
   ```

2. Escribe el predicado `primo`, tal que `primo(N)` se satisface si `N` es primo. (1 pto.)

   ```
   primo(N):- not compuesto(N), numero(N).
   ```

Pregunta 3 (3 ptos.)

El problema de las N reinas consiste en ubicar N reinas en un tablero de ajedrez de $N \times N$, de tal forma que ninguna reina ataque a otra. Una reina ataca a otra reinas si ambas comparten una columna, una fila, o una diagonal. Supón que te entregan el siguiente código base para modelar este problema.

```
reina(1..n).
columna(X) :- reina(X).
fila(Y)    :- reina(Y).
endiagonal(X1, Y1, X2, Y2) :- |X1 - X2| = |Y1 - Y2|,
                              fila(X1), fila(X2),
                              columna(Y1), columna(Y2), .
```

En el modelo de tu programa, cuando aparece el átomo `ubicada(r,x,y)`, significa que la reina `r` está en la posición $(\mathtt{x}, \mathtt{y})$. Escribe código que exprese lo siguiente:

Definimos el predicado auxiliar `ubicada`:

```
% Cada reina se puede ubicar en únicamente una posición
1{ubicada(R, X, Y): fila(X), columna(Y)}1 :- reina(R).
```

1. En cada fila hay exactamente una reina. (1 pto.)

   ```
   :- ubicada(R1, X, Y1), ubicada(R2, X, Y2), R1 != R2.
   ```

2. No hay dos reinas en una misma columna. (1 pto.)

   ```
   :- ubicada(R1, X1, Y), ubicada(R2, X2, Y), R1 != R2.
   ```

3. No hay dos reinas en una misma diagonal. (1 pto.)

   ```
   :- ubicada(R1, X1, Y1), ubicada(R2, X2, Y2), endiagonal(X1, Y1, X2, Y2), R1 != R2.
   ```

Pregunta 4 (3 ptos.)

Las siguientes reglas describen el funcionamiento de una huerta con riego automático.

```
% riega(R,P) el regador R riega (apunta a) a la planta P
riega(r1, durazno).
riega(r2, tomate).
riega(r2, oregano).
riega(r3, oregano).
regador(r1).
regador(r2).
regador(r3).
planta(durazno).
planta(tomate).
planta(oregano).

% una planta está mojada cuando un regador que la riega
% está encendido y éste no está defectuoso
mojada(P) :- planta(P), encendido(R), riega(R, P), not defectuoso(R).

% hay exactamente dos regadores encendidos
2 { encendido(R): regador(R)} 2.
```

Escribe el código necesario para que clingo pueda "descubrir" cuáles son los regadores que podrían estar defectuosos **dado que se ha observado que el orégano no está mojado y que se sabe que a lo más 1 regador puede estar defectuoso**. Puedes suponer que la siguiente regla está definida también en el programa:

```
observacion :- not mojada(oregano).

% Criterio: Regador riega oregano y se encuentra encendido, pero este no está mojado
criterio(R) :- regador(R), encendido(R), riega(R, oregano), observacion.

% A lo más 1 regador puede estar defectuoso
1 {defectuoso(R): criterio(R)} 1.
```

Búsqueda

Pregunta 5 (2 ptos.)

Indica si la afirmación es verdadera o falsa y justifica tu respuesta con claridad.

1. Si h_1 y h_2 son heurísticas admisibles, entonces $(h_1 + h_2)/2$ es una heurística admisible. (1 pto.)

   Verdadero. Si $C_r(A.B)$ es el costo real de llegar desde A hasta B, tenemos que

   $$h1(A, B) \leq C_r(A, B)$$
   $$h2(A, B) \leq C_r(A, B)$$

   $$h1(A, B) + h2(A, B) \leq C_r(A, B) + C_r(A, B)$$
   $$\frac{h1(A, B) + h2(A, B)}{2} \leq C_r(A, B)$$

   Por lo que es admisible.

2. Si N es el número de caminos que hay desde el estado inicial hasta cierto estado s del espacio de búsqueda, entonces A^* podría expandir s N veces. (1 pto.)

   Falso. Un nodo es expandido solo una vez dentro de la búsqueda de A*, además, el algoritmo nos garantiza que su expansión será llevada a cabo solo cuando el costo mínimo de llegar a aquel nodo ha sido alcanzado.

Pregunta 6 (2 ptos.)

Indica si la afirmación es verdadera o falsa y justifica tu respuesta con claridad.

1. El número máximo de veces que A^* puede extraer un mismo nodo de la lista Open corresponde al número de nodos en el espacio de búsqueda. (1 pto.)

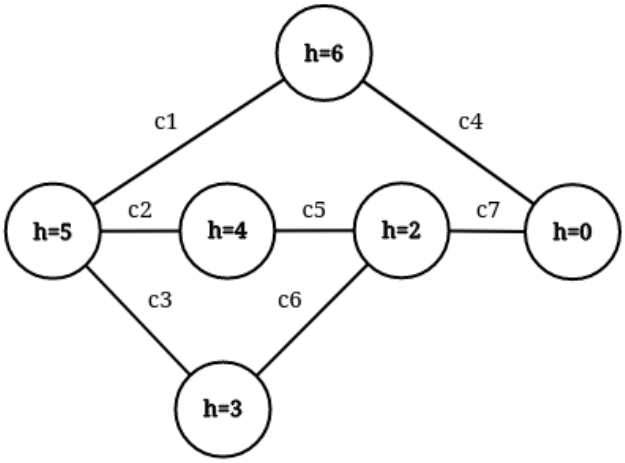
   Falso. Dado que A* es óptimo, un nodo es extraído únicamente cuando el costo de llegada hasta él es el óptimo, esto debido a que lo extraemos de una lista ordenada *Open*, en donde actualizamos su costo hasta llegar al mínimo.

2. Si n es un nodo objetivo que acaba de entrar en la Open, entonces $g(n)$ puede ser mayor que el costo el camino óptimo desde el estado inicial hasta n. (1 pto.)

   Verdadero. El costo es óptimo cuando el nodo es extraído, pero puede ser insertado a *Open* con un costo mayor y actualizado a lo largo de la ejecución del algoritmo.

Pregunta 7 (3 ptos.)

La figura ilustra un espacio de búsqueda de 6 estados; los nodos corresponden a estados y los arcos a acciones que transforman un estado en otro. Las etiquetas de los arcos corresponden al costo de la acción respectiva. Una función heurística h está definida para cada estado y su valor es mostrado en la etiqueta de cada nodo. El estado objetivo es aquel en que el valor de su heurística es 0. El estado inicial es aquél donde la heurística vale 6.



Escribe todas las relaciones que se tienen que cumplir entre costos del grafo, expresados en forma de desigualdad, para que, al ejecutar A* sobre ese grafo, el estado marcado con $h = 2$ sea expandido dos veces. La primera vez, desde el estado marcado con $h = 3$ y la segunda, desde el estado marcado con $h = 4$.

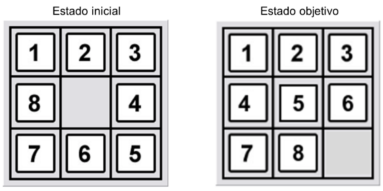
Ayuda: Para partir, te debes asegurar que en la segunda iteración del loop principal, el estado marcado con $h = 0$ es expandido después que el estado marcado con $h = 5$; para que eso ocurra, debe cumplirse una relación entre el valor f de estos dos nodos. Escribe todas las restricciones que faltan.

Una solución posible para el problema es:

- $c_1 + 5 < c_4$ (h = 5, antes h = 0)
- $c_3 + c_1 + 3 < c_4$ (h = 3, antes h = 0)
- $c_2 + c_1 + 4 < c_4$ (h = 4, antes h = 0)
- $c_3 + c_1 + 3 < c_2 + c_1 + 4$ (h = 3, antes h = 4)
- $c_2 + c_1 + 4 < c_6 + c_3 + c_1 + 2$ (h = 4, antes h = 2)

Pregunta 8 (3 ptos.)

Considera el problema del *puzzle de 8* que hemos visto en clases, que es tal que en el estado final las piezas se encuentran ordenadas (como se muestra en la figura). En clases argumentamos que la heurística que suma las distancias Manhattan entre la posición actual y final de cada pieza es una heurística admisible.



Ahora considera la siguiente pieza clave de información:

> *"Cuando un par de piezas se encuentran sobre la misma fila o sobre la misma columna que su objetivo, pero se encuentran invertidas, entonces es necesario ejecutar al menos dos acciones más, adicionales a la distancia Manhattan para estas dos piezas."*

Argumenta que la afirmación es verdadera, y explica cómo podrías aplicar ese conocimiento para incorporarlo a la heurística Manhattan sin cambiar su admisibilidad. Justifica que tu heurística sigue siendo admisible.

Si dos piezas están en la misma fila o columna que su posición objetivo pero están invertidas, se necesitarán al menos dos movimientos adicionales para llegar a su posición objetivo. Esto se debe a que (1) una pieza debe moverse fuera de la fila o columna, (2) luego la otra pieza debe moverse a su posición objetivo, y (3) finalmente la primera pieza debe moverse de nuevo a su posición objetivo, concluyendo los pasos necesarios en al menos dos.

Podemos aplicar este conocimiento a la heurística de Manhattan sumando 2 por cada par de piezas invertidas en la misma fila o columna. La nueva heurística sería: $h(n) = h_{manhattan}(n) + 2 * \#inversiones$.

Esta heurística sigue siendo admisible porque nunca sobreestima el costo para llegar al objetivo. La distancia de Manhattan es una estimación exacta del costo si todas las piezas están en su fila o columna objetivo y no están invertidas. Al agregar un costo de 2 por cada par de piezas invertidas, estamos contando el costo adicional para resolver estas inversiones. Por lo tanto, la heurística sigue siendo una subestimación del costo real y por lo tanto es admisible.