[]Magister en Inteligencia Artificial
Facultad de Ingeniería y Ciencias

Trabajo Sesión Nº 2
Técnica de Búsqueda y Heurísticas

Nombre Eduardo Carrasco
Profesor Jordi Pereira

2021

1. Preguntas Cortas:

a. Considere el problema denominado “el juego del 15” (puede ver su definición en Wikipedia: https://es.wikipedia.org/wiki/Juego del 15. Plantee una posible “heurística” admisible para usar un algoritmo A∗ para resolver el problema.

Uno de los temas no vistos en el curso sobre la programación por restricciones es qué método utiliza un programa para decidir la variable o condición en la que subdividir el problema en dos (la regla de ramificación).

Una regla habitual en este tipo de software consiste en escoger la variable que tiene el menor dominio (menor número de valores posibles o rango más pequeño si el problema usa variables continuas) y dividir el problema en dos sub-problemas según la siguiente condición: el valor de la variable debe ser igual o menor a un valor prefijado (por ejemplo el promedio entre el valor menor y el mayor del dominio), el valor de la variable debe ser superior a ese valor prefijado:

El juego del 15 consiste en una grilla de 4x4 con fichas marcadas con números del 1 al 15 y un espacio vacío (completando 16), el objetivo es obtener desde una posición *“random”* hacia un estado objetivo ordenando los números en forma ascendente, desde izquierda a derecha y desde arriba abajo.

Figura 1: El juego del 15.

Objeto evitar atravesar caminos sin sentido durante la búsqueda de la solución y determinar el nodo para la expansión se utiliza una función de evaluación $f(n)$.
Por otra parte podemos señalar una función heurística $h(n)$, que representa un costo estimado del camino más barato desde el nodo n al nodo objetivo.
Para el ejemplo del juego del 15, tenemos aproximadamente $1{,}04 * 10^{13}$ estados (considerando que existen restricciones con estados repetidos), por lo cual tenemos que buscar una función heurística admisible que nos permita resolver el problema usando un algoritmo A*.

Con el propósito de encontrar el camino más corto al objetivo, exploramos, los estados que tienen el menor número estimado de pasos (menor valor heurístico), el cual podemos estimar simplemente sumando los estados que se encuentran en la posición incorrecta (para el caso de la figura 1, son 4), por lo cual $h1 = 4$.

Si consideramos el método planteado en el enunciado (escoger la variable que tiene el menor dominio), estaríamos frente a un método de búsqueda de primero el mejor (*"greedy best-first search"*) que selecciona la heurística obtenida $h1$ como función de evaluación $f(n)$, por lo tanto $h(n) = f(n)$.

En el caso de utilizar el algoritmo A* como método de búsqueda, cuya función de evaluación es $f(n) = g(n) + h(n)$, donde $g(n)$ representa el coste del camino desde el nodo de inicio al nodo n, se puede señalar que la heurística determinada es admisible, puesto que no sobreestima el costo de alcanzar el objetivo, por ende garantiza el camino óptimo entre el estado inicial y el estado objetivo.

Podemos de igual manera plantear otra heurística para determinar la $h(n)$ basada en la distancia manhattan, esta es la suma de las distancias de las piezas a sus posiciones objetivo (vertical y horizontal), por lo cual, esta $h2(n)$ (de la figura 1) sería determinado de la siguiente manera:

$$h2 = 0 + 0 + 0 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 0 = 3$$

Tanto $h1$ y $h2$ son heurísticas admisibles, no subestimando el costo de alcanzar el objetivo.

b. Explique por qué cree que es conveniente escoger la variable con menor dominio:

Al escoger la variable con menor dominio expandiremos el nodo más cercano que nos dará (supuestamente) una solución más óptima; por lo cual, la heurística sólo considerará el costo del camino con menor dominio desde n al nodo objetivo.

Esto puede no siempre ser el mejor camino, ya que, algunas veces, el expandir un nodo cercano al estado n, puede implicar abrir una serie de nodos y en algunas ocasiones devolverse por un nodo ya explorado o aumentar el costo al expandir nodos que presentan caminos menos eficientes para alcanzar el estado objetivo.

En un ejemplo de la vida real, sería tomar un camino siempre por la vía con menos kilómetros, manejando hacia una ciudad de destino, lo cual puede significar llegar al mismo destino pero avanzando por ramificaciones que aumentan el costo total.

c. Proponga una regla alternativa (no tiene por qué ser eficiente aunque sería interesante identificar qué condiciones cree que podrían ser interesantes).

Una regla alternativa sería utilizar una base de datos de todos los estados posibles: 16 variables que tuvieran un dominio del [0,15], que

ninguna pudiera ser igual a la otra; que además, conociéramos el estado objetivo.
De igual manera, utilizar una heurística determinada para medir la cantidad de pasos necesarios para llegar al objetivo y con ello, comparar el millón de estados propuestos y posibles utilizando técnicas de *machine learning* que nos permitieran resolver el problema.
No sería lo más eficiente puesto que el computador tendría un espacio de estado de todos los estados posibles, pero, no habría un proceso de búsqueda normal, sino más bien, una serie de capaz (filtros) que determinarían el mejor camino dentro de una gran cantidad de posibilidades.
Lo interesante de este método de resolución es que al establecerse de manera gráfica la resolución de un problema (árbol de búsqueda), podemos observar que realmente es aplicable y no tan sólo con el tradicional método de redes neuronales, sino también con árboles de decisión, podando hojas de igual manera.

d. ¿Cree que una red neuronal podría crear una regla para un algoritmo $A*$ que asegure siempre admisibilidad y consistencia para el problema estudiado?

Una red neuronal podría en cierta forma determinar una adecuada heurística, pero es muy fácil hacer trampas con ellas por el determinado número de capas; por lo cual, no se asegura contar con una solución óptima, pudiendo quedar con *overfitting*, al aprenderse todos los datos en una determinada situación y no ser eficiente ante una situación o problema nuevo.
De acuerdo a lo expuesto y considerando que utilizar redes neuronales no asegura optimalidad, entonces tampoco se puede asegurar la admisibilidad y en consecuencia, como todas las heurísticas consistentes son admisibles, no se puede asegurar consistencia.

2. Preguntas Cortas:

a. Proponga un modelo en minizinc para el problema de intercambio de riñones considerando el caso que cada paciente tiene un número de donantes voluntarios que puede variar entre 1 y 3 y que tendría su propia lista de compatibilidades. (Nota: Es posible resolver este problema sólo cambiando los datos y no cambiando el modelo, plantee los dos casos o sólo plantee cómo cambiar los datos si cree que sería más simple que proponer un modelo alternativo).

Para el ejemplo, modificando el modelo propuesto entregado en clases, se deben efectuar las siguientes modificaciones.
El n° de pacientes = 3.
Y la tabla de compatibilidades propuesta es:

1. Paciente 1, es compatible con 2, 5 y 8.
2. Paciente 2, es compatible con 1, 2 y 5.
3. Paciente 3, es compatible con 2, 5 y 4.

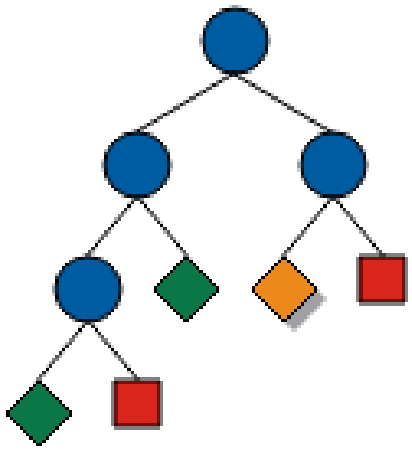
```
% se deben ingresar dos variables:
% paciente: indica el número de pacientes ej: 3.
% compatible: indica la tabla de compatibilidad en función de los pacientes. ej: [{2,5,8},{1,2,5},{2,5,4}]

include "globals.mzn";
%se define la variable de cantidad de pacientes
int: paciente;
% vector de cantidad de personas en una variable de compatibilidad
array[1..paciente] of set of int: compatibilidad;

% personas que no pueden recibir un riñón.
set of 1..paciente: non_compatibilidad = { p | p in 1..paciente where card(compatibilidad[p]) = 0 };

% Enuncia el dominio de cada variable, compatibilidad podada (pruned).
array[1..paciente] of set of int:
compatibilidad_pruned = [ %si hay comptabilidad igual a 0, entonces no hay compatibilidad.
if card(compatibilidad[p]) = 0 then {p} else (compatibilidad[p] diff non_compatibilidad) union {p} endif
                | p in 1..paciente];

% selecciona cual riñón queda con cada persona
array[1..paciente] of var 1..paciente: x;
var 0..paciente: z = sum([bool2int(x[i] != i) | i in 1..paciente]);


solve :: int_search(
     x,
     first_fail,
     indomain_median,
     complete)
   maximize z;

% permitir ingreso de personas compatibles sin poda, también restringe a que no pueda donarse a si mismo.
constraint
  forall(p in 1..paciente) (
    x[p] in compatibilidad_pruned[p]
  )
  /\
  alldifferent(x);

            output [
              "z: " ++ show(z) ++ "\n"
            ]
            ++
```

```
            [
              "Paciente: Donante\n"
            ]
            ++
            [
              if fix(x[i] = i) then
                 show_int(3, i) ++ ":  -\n"
              else
                 show_int(3, i) ++ ": "++  show_int(3, x[i]) ++ "\n"
              endif
              | i in 1..paciente
            ]
;
```

```
Running kidney_exchange_model.mzn, additional arguments
-D compatibilidad=[{2,5,8},{1,2,5},{2,5,4}];
-D paciente=3;
z: 0
        Paciente: Donante
        1: -
        2: -
        3: -
        ----------
        z: 2
        Paciente: Donante
        1:      2
        2:      1
        3:      -
        ----------
        ==========
Finished in 273msec
```

En las soluciones propuestas por MiniZinc podemos observar que existe una solución óptima donde los 3 pacientes son asignados con un riñón diferente (z=3), siendo el paciente 1 asignado con el riñón del paciente 2, el paciente 2 asignado con el riñón del paciente 3 y el paciente 3 asignado con el riñón del paciente 1.

Al efectuar el modelado del árbol de búsqueda, se obtiene el siguiente gráfico:



La solución óptima es representada en color naranjo.