NOMBRE: Florencia Valdivia Barros

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia Artificial — 1' 2020

Tarea 2 – Respuesta Parte 1

T1. Al implementar la regla de que si al tener dos nodos en la Open con el mismo valor f, se prefiere para expansión a aquel que tiene el mayor valor g, o, equivalentemente, menor valor h, tiene sentido ya que el algoritmo no sobreestima. De este modo, a menor valor h nos encontramos más cerca del nodo destino, ya que como h es una estimación, esperamos que sea más preciso a medida que nos acercamos a la meta. al igual que si tomamos el mayor valor g, que significaría que estamos probablemente más cercano al nodo destino, debido a que tienen un mayor costo.

T2. Un ejemplo en donde aplicar la regla de quiebre tal que se quebra empates hacia un valor g menor que es exponencialmente más ineficiente, es el cubo rubix. Evidentemente, en la solución de este problema se obtienen varias soluciones hacia el estado objetivo. Ahora si quebramos empates hacia un estado donde se tiene un menor valor g, es probable que se llegue a la solución, ya que no hay solo un camino, pero el costo será mayor. Sabemos que la cantidad de nodos generados a medida que aumenta la profundida aumenta rápidamente en este problema, como por ejemplo a profundidad 10 se esperan más de 244 mil millones de nodos. Por lo tanto, si la open se va expandiendo primero con nodos que tienen un menor valor g, que sería equivalente a un mayor valor h, estaríamos llendo por un camino donde el costo estimado es mayor, por lo tanto se esperaría recorrer más nodos en profundidad, lo que equivale a que el problema se hace exponencialmente más ineficiente.

T3. **Por demostrar**: Si usamos $f = g + w \cdot h$ para ordenar la Open, cualquier solución encontrada es tal que su costo no puede exceder a wc^*, donde c^* es el costo de una solución óptima.

Demostración: Según el lema en que en todo momento de la ejecucion existe un estado s^* en Open en que :

1. Está en un camino óptimo hacia el goal.
2. $g(s) = d(s.inicial, s^*)$

Ahora, en cualquier momento, si s se extrae de Open entonces se tiene que:

$f(s) \leq f(t)$, para todo t perteneciente a Open.

En particular:

$f(s) \leq f(s^*) = g(s^*) + h(s^*)$

Luego si multiplicamos el h por un valor w queda:

$$\begin{aligned} f(s^*) &= g(s^*) + w \cdot h(s^*) \\ &= d(s.inicial, s^*) + w \cdot h(s^*) \end{aligned}$$

Con lo que nos queda:

$$d(s.inicial, s^*) + w \cdot h(s^*) \leq d(s.inicial, s^*) + w \cdot d(s^*, goal) = w \cdot c^*$$

Por lo tanto, tenemos que:

$f(s) \leq w \cdot c^*$

Y si s es un estado objetivo:

$f(s) \leq g(s) + 0 \leq w \cdot c^*$

Con esto demostramos que cualquier solución que encuentro no excede el valor de $w \cdot c^*$.

T4. Se quiere demostrar que:

$$\frac{c(\pi)}{c^*} \leq \frac{c(\pi)}{\text{mín}_{s \in Open} g(s) + h(s)}$$

Si la heurística h es admisible.

Demostración:

Suponga h es admisible. Entonces $\forall s.h(s) \leq h^*(s)$.

Suponga s es extraído de Open, por lo que s es el nodo con menor f donde $f(s) = g(s) + h(s)$.

Por admisibilidad tenemos que:

$$\begin{aligned} f(s) &\leq f^*(s) = g^*(s) + h^*(s) \\ &\leq f^*(s) = dist(s.inicial, s^*) + h^*(s) \leq dist(s.inicial, s^*) + dist(s^*, goal) = c^* \\ &\leq f^*(s) = dist(s.inicial, s^*) + h^*(s) \leq c^* \end{aligned}$$

Con lo anterior se concluye que:

$$f(s) \leq c^*$$

Por lo tanto, si $c(\pi)$ es el costo de una solución al problema tenemos que:

$$\frac{c(\pi)}{c^*} \leq \frac{c(\pi)}{f(s)}$$

Y como s es el nodo extraído de la Open sabemos que es el $\text{mín}_{s \in Open} f(s)$ y $f(s) = g(s) + h(s)$, entonces:

$$\frac{c(\pi)}{c^*} \leq \frac{c(\pi)}{\text{mín}_{s \in Open} g(s) + h(s)}$$

P1. Se muestra una tabla de comparación, con los resultados del problema ejecutando la 'versión mejorada' del algoritmo y la original.

En esta parte se modificó la función fvalue del algoritmo para que calculara $f = g + h \cdot (1 + 10^{-10})$ para que de esta manera no ocurran problemas si dos nodos tienen el mismo valor de f, ya que se eligirá el con menor valor de h.

Problema	Exp. original	Exp. mejorada
1	151817	32470
2	170564	48443
3	198327	66296
4	191259	142928
5	620168	154019
6	440524	179269
7	410133	191088
8	148509	273541
9	301944	330838
10	614465	486106

Table 1: Problema 1.

P2. En esta sección compararemos la diferencia de usar distintos valores para el parámetro weight para los 50 primeros problemas.

Primero que todo, sin duda alguna el weighted A* funciona más rápido y eficiente que A*, y por lo que vimos anteriormente nos aseguramos de que el costo no supera (weight · costo de la solución óptima).

En la siguiente tabla se resumen los datos obtenidos por cada weight, en cuanto al tiempo total, número de expansiones totales y el costo total.

Weight	Tiempo Total	Expansiones totales	Costo Total
1.5	842.86	3544962	2645
2	204.02	1349660	3019
3	54.14	346104	3645
5	42.27	271483	4487

Table 2: Problema de weighted A*.

Como podemos ver en la Tabla 2, a medida que el valor de weight aumenta el algoritmo se demora menos en resolver los 50 problemas. Asimismo, las expansiones totales van disminuyendo, por lo que el número de expansiones requeridas para encontrar la solución óptima en cada problema se hace menor a medida que aumentamos el weight, generalmente. Finalmente, vemos que sacrificamos optimalidad, ya que el costo va aumentando.

NOMBRE: Florencia Valdivia Barros

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia Artificial — 1' 2020

Tarea 2 – Respuesta Parte 2

T1. **Por demostrar:** El costo de una solución óptima desde un estado s en el problema original es mayor o igual al costo de una solución óptima desde el estado α_s sobre el espacio abstracto.

Demostración: Suponga que el costo de una solución óptima del problema original desde un estado s al estado objetivo es $g^*(s)$. Como sabemos que el espacio del problema abstracto es menor que el del original, el costo de $\alpha_l(s)$ al estado objetivo $\alpha_l(s_g)$ tiene un límite más pequeño que el del problema original, pues no sobreestima. Por esta razón, llegar al nodo objetivo tomará un menor o igual costo, que en el problema original. Con esto concluimos que cualquier valor de $\alpha_l(s)$ no será una mejor solución si tiene un costo mayor a $g^*(s)$. Por lo tanto, $\alpha_s \leq g^*(s)$.

T2. Calculando las cardinalidades de los espacios, obtenemos los siguiente:

- Problema original (15-puzle): $15! \times 15 = 1,96 \times 10^{13}$
- Problema $\alpha_{[2,11,15]}(s)$: $(15 \times 14 \times 13) \times 15 = 40.950$

Podemos ver claramente que el espacio abstracto definido por la función α es mucho menor que el espacio original, en particular el espacio original es aproximadamente 478.632.478 más grande que el espacio abstracto.

T3. **Por demostrar:** si h es una heurística admisible y $h^{'}$ es una heurística admisible, entonces $máx\{h, h^{'}\}$ es una heurística admisible.

Demostración: Suponga h y $h^{'}$ son heurísticas admisibles. Entonces se cumple que:

$$h \leq h^* \qquad \forall s \in S.$$
$$h' \leq h^* \qquad \forall s \in S.$$

PD: $máx\{h, h^{'}\}$ es una heurística admisible.

Veamos por casos:

- Suponga $máx\{h, h^{'}\} = h$.
  $\Rightarrow h^{'} \leq h \leq h^*$
  $\Rightarrow$ h domina a $h^{'}$
  Por lo tanto el $máx\{h, h^{'}\}$ es admisible.
- Suponga $máx\{h, h^{'}\} = h^{'}$.
  $\Rightarrow h \leq h^{'} \leq h^*$
  $\Rightarrow h^{'}$ domina a h
  Por lo tanto el $máx\{h, h^{'}\}$ es admisible.

P1. La implementación de esta parte se basa en implementar Puzzle.initialize pdb y Puzzle.pdb 1, donde en la primera se guarda la base de dato en la variable Puzzle.pdb en formato de diccionario {"estado relajdo": profundidad} y en Puzzle.pdb pattern guardamos la pattern correspondiente para después acceder a estas variables en Puzzle.pdb heuristic, y buscar el estado relajado del estado actual en la base de datos para luego comparar su heurística con la de manhattan y retornar la mayor de estas.

P2. En esta parte, se implementa en generate pdb un código para que se creen varias bases de datos. En particular, se generaron 3 pattern distintas, dos con 4 valores y 1 de 5 valores, por lo que tenemos 3 bases de datos.

Luego, en puzzle.py se modificó la función pdb final para que retorne la máxima heuristica combinada entre las tres bases y la de manhattan. Con esta heuristica se ejecuta el algoritmo astar, donde las expansiones totales superan a las descritas en el enunciado.

NOMBRE: Florencia Valdivia Barros

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia Artificial — 1' 2020

Tarea 2 – Respuesta Parte 3

T1. **Por demostrar:** El mecanismo de poda no puede descartar un estado que está en un camino a una solución óptima.

Demostración: Como sabemos, el algoritmo realiza la poda cuando un nodo s tiene un $f(s) = g(s) + h(s) \geq c_{ult}$ donde c_{ult} es el costo de la última solución encontrada. Por lo tanto, al podar soluciones con un costo estimado mayor a la última solución, dado que el algortimo no subestima, no se descartan soluciones en un camino óptimo. En efecto, la poda se realiza para que Open nunca contenga un estado cuyo valor f no ponderado sea mayor o igual que $g(s_{goal})$, por lo que nos aseguramos que no se eliminó ningún estado en camino a la solución óptima.