PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
FACULTAD DE INGENIERÍA
IIC - 2613 Inteligencia Artificial
Profesor: Jorge Baier (jabaier@ing.puc.cl)
Nombre Alumno: Alberto Valdés (anvaldes@uc.cl).

Tarea 2

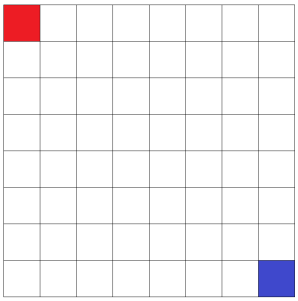
Alberto Valdés.

1. Parte 1

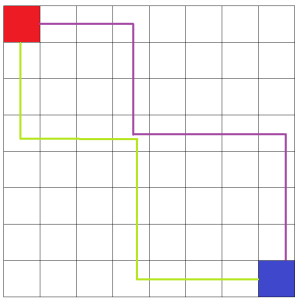
T.1.1) La razón por la que la regla de desempate que se describe tiene sentido, es porque lo que se esta diciendo es que: (Se considera que se elige n_1 versus n_2)

Si dos nodos tienen el mismo valor de f $(f(n_1) = f(n_2))$, *entonces se elige para expandir al nodo que esta más cerca de la meta* $(h(n_1) < h(n_2))$ *o equivalentemente se elige al nodo que esta más lejos del punto inicial* $(g(n_1) > g(n_2))$.

T.1.2) Consideremos un problema en donde estamos trabajando con la Métrica Manhattan. El problema consiste en encontrar un camino desde un punto a otro y en donde no existe ningun obstaculo. Los movimientos posibles son arriba, abajo, derecha, izquierda y diagonal, pero todos estos movimientos solo son posibles entre posiciones adyacentes. Además, cada posición se considerara como un nodo en el problema de busqueda. El problema sería algo como esto, donde el color rojo es el inicio y el color azul es la meta:

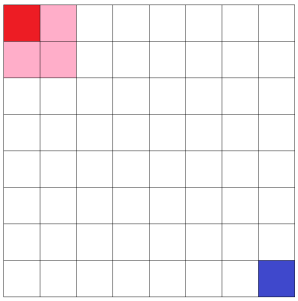


Notemos que los caminos que poseen el mismo costo son muchos, y que también como nos dicen en el enunciado, si disordenamos los pasos realizados en una solución obtenemos otra solución. Ahora pasamos a mostrar un ejemplo de dos posibles soluciones al problema:

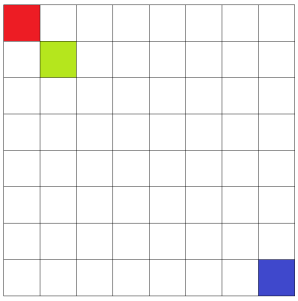


En primer lugar analizaremos como funciona el algoritmo usando la regla que nos recomiendan.

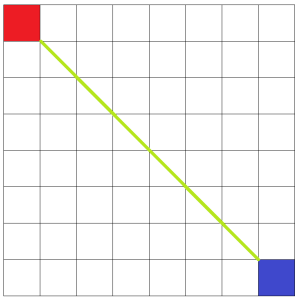
Notemos que si partimos desde el inicio (bloque rojo) entonces los posibles nodos a los que movernos son (colores rosados):



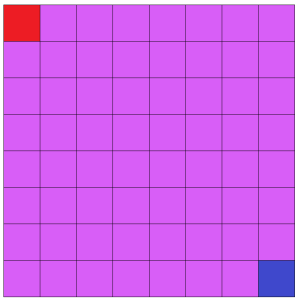
Según la métrica manhattan usando la regla que nos recomiendan nos movemos finalmente al nodo (color verde):



Ahora si repetimos esto hasta llegar a la meta notamos que el recorrido hasta la meta sera:



Ahora si consideramos el caso opuesto a la regla que nos recomiendan, haciendo el mismo analisis llegamos a que para llegar a la meta, debería recorrer todos los nodos antes de llegar. Es decir, los nodos recorridos (color morado) serían:



De esta forma, comparando ambos casos se observa que usando la regla de manera contraria a lo que nos recomiendan, obtendriamos un algoritmo A^* exponencialmente más ineficiente que cuando usamos la regla que nos recomiendan.

P.1.1) Notemos que lo queremos hacer es implementar la regla de empates que nos recomiendan. Para esto lo que utilizaremos será el hecho de que tanto la función g como h tienen como recorrido los enteros positivos que son menores que 1000. De esta forma, lo que haremos será que en el metodo *fvalue* en vez de retornar $f(n) = g(n) + h(n)$ retornaremos $f'(n) = g(n) + h(n) + 0{,}001 \cdot h(n)$, dado a que esto implica que:

Si $f(n_1) > f(n_2)$, entonces como el recorrido de f es en los enteros, entonces:

$$f(n_1) \geq f(n_2) + 1$$

$\Rightarrow$

$$\boxed{g(n_1) + h(n_1) \geq g(n_2) + h(n_2) + 1 \quad (1)}$$

Por otro lado, notemos que como h solo toma valores enteros positivos menores que mil, entonces:

$$1000 > h(n_2) - h(n_1)$$

$\Rightarrow$

$$1 > 0{,}001 \cdot h(n_2) - 0{,}001 \cdot h(n_1)$$

$\Rightarrow$

$$\boxed{1 + 0{,}001 \cdot h(n_1) > 0{,}001 \cdot h(n_2) \quad (2)}$$

Sumando (1) y (2) llegamos a que:

$$g(n_1) + h(n_1) + 0{,}001 \cdot h(n_1) + 1 > g(n_2) + h(n_2) + 0{,}001 \cdot h(n_2) + 1$$

$\Rightarrow$

$$g(n_1) + h(n_1) + 0,001 \cdot h(n_1) > g(n_2) + h(n_2) + 0,001 \cdot h(n_2)$$

$\Rightarrow$

$$\boxed{f'(n_1) > f'(n_2)}$$

De esta forma llegamos a que:

$$\boxed{\text{Si } f(n_1) > f(n_2) \;\Rightarrow\; f'(n_1) > f'(n_2) \quad (3)}$$

Lo que significa que con f' seguimos manteniendo el orden entre nodos que tienen distinto valor de f.

Ahora analizaremos el caso en que $f(n_1) = f(n_2)$.

Sabemos que esto implica que:

$$g(n_1) + h(n_1) = g(n_2) + h(n_2)$$

Ahora si tenemos que $h(n_1) < h(n_2)$, entonces:

$$g(n_1) + h(n_1) + 0{,}001 \cdot h(n_1) < g(n_2) + h(n_2) + 0{,}001 \cdot h(n_2)$$

$\Rightarrow$

$$f'(n_1) < f'(n_2)$$

De esta forma llegamos a que:

$$\boxed{\text{Si } f(n_1) = f(n_2) \text{ y } h(n_1) < h(n_2) \;\Rightarrow\; f'(n_1) < f'(n_2) \quad (4)}$$

Así llegamos a que utilizando (3) y (4) tenemos que si n_1 y n_2 tienen distinto valor de f, entonces f' preserva el orden y si n_1 y n_2 tienen el mismo valor de f entonces si $h(n_1) < h(n_2)$ esto implica que $f'(n_1) < f'(n_2)$, es decir, que le da más prioridad a los nodos que tienen menor valor de h que es justamente la regla que nos piden implementar.

De esta forma, se demostro que si en vez de retornar $f(n)$ en *fvalue* retornamos $f'(n)$ entonces estamos implementando la regla que nos recomendaban.

Ahora, una vez implementada esta regla, reportaremos la diferencia (medida en expansiones) de los primeros 10 problemas al implementar el algoritmo inicial versus el algoritmo mejorado.

Problema	Expansiones Algoritmo Inicial	Expansiones Algoritmo Mejorado
1	151.817	32.470
2	170.564	48.443
3	198.327	66.296
4	191.259	142.928
5	620.168	154.019
6	440.524	179.269
7	410.133	191.088
8	148.509	273.541
9	301.944	330.838
10	614.465	486.106
Total	3.247.710	1.904.998

Notemos que el algoritmo mejorado posee muchas menos expansiones que el algoritmo inicial excepto en los problemas 8 y 9. En el total se observa que la diferencia en expansiones es de más de 1 millon.

T.1.3)

Antes de demostrar lo pedido, enunciaremos dos cosas que vimos en clases.

Lema: En todo momento de la ejecución del algoritmo existe un estado s^* de *Open* tal que:

1. Existe un camino optimo hacia goal $[d(s_{inicial}, s^*) + d(s^*, goal) = c^*]$.

2. $g(s^*) = d(s_{inicial}, s^*)$.

Definición: $h(s)$ es admisible si $h(s) \leq d(s, goal)\ \forall s$.

Ahora pasaremos a demostrar lo pedido en este inciso.

Consideremos el s asociado a una solución encontrada. Entonces tenemos $f(s) \leq f(t)\ \forall t \in Open$.

En particular esta desigualdad se cumple para $t = s^*$, de este modo:

$$f(s) \leq f(s^*) = g(s^*) + \omega \cdot h(s^*) \overset{Lema}{=} d(s_{inicial}, s^*) + \omega \cdot h(s^*) \overset{1 \leq \omega}{\leq} \omega \cdot d(s_{inicial}, s^*) + \omega \cdot h(s^*)$$

$$\overset{h\ admisible}{\leq} \omega \cdot d(s_{inicial}, s^*) + \omega \cdot d(s^*, goal) = \omega \cdot [d(s_{inicial}, s^*) + d(s^*, goal)] = \omega \cdot c^*$$

De este modo demostramos que sea $f(s)$ el costo asociado a una solución encontrada, entonces este debe ser menor o igual a $\omega \cdot c^*$ que es justamente lo que nos pedían demostrar.

P.1.2)

Para la implementación de esta parte lo que haremos será modificar el método *fvalue* para que retorne:

$$f'(n) = g(n) + \omega \cdot h(n) + 0{,}001 \cdot h(n) = g(n) + [\omega + 0{,}001] \cdot h(n)$$

Al igual que en la pregunta **P.1.1** esta implementado el rompimiento de empates y todo esto se debe al termino $0{,}001 \cdot h(n)$.

Ahora pasaremos a mostrar los resultados cuando $\omega = 1{,}5$ para los primeros 50 problemas.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	5928	11823	45	0.56	1.15
2	24043	45589	48	2.47	1.33
3	25297	49135	42	2.81	1.17
4	30202	57469	47	3.64	1.34
5	11498	21593	46	1.44	1.21
6	27259	51675	47	3.06	1.21
7	14823	28123	48	2.42	1.33
8	16095	31301	63	1.39	1.34
9	19964	37967	48	3.41	1.33
10	72669	134928	50	10.33	1.32
11	56462	103653	64	9.20	1.33
12	31314	60507	55	4.62	1.34
13	17203	33045	51	3.30	1.31
14	24215	43849	47	2.73	1.27
15	733	1426	52	0.27	1.37
16	10247	19567	46	2.05	1.35
17	24082	44643	55	3.64	1.34
18	55746	99962	58	8.09	1.38
19	44568	82653	53	6.50	1.29
20	77422	144777	47	11.08	1.27
21	51855	98776	56	6.52	1.40
22	80907	145947	54	11.19	1.29
23	8629	16418	52	1.34	1.30
24	107625	202430	48	14.53	1.33
25	24698	47275	55	2.20	1.28
26	9652	19109	57	1.61	1.33
27	21150	40486	60	3.08	1.36
28	119464	195196	50	14.50	1.32
29	133806	242557	51	16.66	1.31
30	30207	54861	55	3.16	1.34
31	43306	81545	59	5.36	1.31
32	191486	355948	51	24.58	1.31
33	21415	40862	54	1.91	1.35
34	135334	256005	46	17.72	1.28
35	51538	96001	58	6.36	1.32
36	27086	50888	47	2.62	1.27
37	12665	23413	49	2.38	1.32
38	42816	82088	52	4.84	1.24
39	147895	281685	51	19.28	1.31
40	62035	116323	61	7.98	1.30
41	147271	274147	63	19.17	1.40
42	19710	36766	57	1.73	1.33
43	7421	13794	58	1.06	1.32
44	63715	119860	54	8.62	1.29
45	291111	540120	54	40.12	1.29
46	35965	66765	70	4.67	1.35
47	61168	115487	54	8.02	1.29
48	398683	729661	50	55.41	1.32
49	213222	391308	57	28.64	1.27
50	33725	63137	54	4.64	1.29

Tiempo total: 422.92
Expansiones totales: 3185330
Costo total: 2649

Ahora pasaremos a mostrar los resultados cuando $\omega = 2$ para los primeros 50 problemas.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	4000	7596	55	0.39	1.49
2	7656	14034	56	0.69	1.75
3	5760	11013	50	0.55	1.56
4	13131	23003	49	2.33	1.58
5	3615	6526	58	0.34	1.61
6	2255	3716	59	0.23	1.69
7	27910	51015	54	4.08	1.59
8	11660	22117	73	1.05	1.55
9	67487	118713	58	9.30	1.71
10	23708	43061	60	3.06	1.67
11	11395	21021	78	1.66	1.70
12	1363	2722	63	0.19	1.54
13	17581	32522	55	1.47	1.57
14	17549	30280	55	3.02	1.49
15	24906	44067	62	3.08	1.63
16	13002	24271	50	1.72	1.47
17	17020	29437	65	2.34	1.67
18	8813	15334	68	1.59	1.79
19	19362	33680	65	2.72	1.67
20	11330	21260	51	2.17	1.65
21	11627	21493	60	1.38	1.67
22	61741	97562	64	8.38	1.68
23	5147	8913	54	0.45	1.42
24	18542	34265	48	2.84	1.50
25	20224	36625	69	2.56	1.68
26	15838	29719	73	2.94	1.70
27	27862	51132	70	3.88	1.59
28	95162	139815	58	13.80	1.61
29	80741	137208	57	13.23	1.54
30	9030	16948	63	1.88	1.62
31	23459	42134	67	3.34	1.49
32	54123	93104	57	8.17	1.63
33	17300	30485	68	2.12	1.70
34	51637	95482	48	8.55	1.60
35	13677	25546	66	1.30	1.74
36	2008	4023	57	0.53	1.73
37	22694	38156	53	4.33	1.51
38	7994	14150	66	0.73	1.74
39	26456	51430	59	4.22	1.59
40	1311	2645	61	0.47	1.36
41	9642	17947	77	1.69	1.71
42	17606	31966	69	3.11	1.68
43	2056	4199	62	0.39	1.41
44	4817	8862	62	0.47	1.63
45	61815	110499	62	9.36	1.72
46	81333	144550	88	12.16	1.76
47	999	1965	62	0.34	1.72
48	6021	11158	54	1.28	1.69
49	199289	327122	71	29.45	1.65
50	19451	34822	66	2.38	1.74

Tiempo total: 187.69
Expansiones totales: 1279105
Costo total: 3075

Ahora pasaremos a mostrar los resultados cuando $\omega = 3$ para los primeros 50 problemas.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	5110	9541	79	0.62	2.26
2	4204	6969	72	0.42	2.25
3	6610	10943	54	0.59	1.69
4	2796	4849	59	0.58	1.90
5	2826	5296	74	0.62	2.06
6	3474	6058	79	0.77	2.26
7	2082	4098	62	0.23	1.82
8	822	1705	91	0.16	1.94
9	11885	18600	68	1.02	2.00
10	1504	2946	66	0.20	1.83
11	22886	37299	102	3.70	2.22
12	19582	30879	81	2.73	1.98
13	4216	7882	67	0.44	1.91
14	3841	6923	63	0.39	1.70
15	17757	27178	74	2.52	1.95
16	5642	9086	60	1.16	1.76
17	4310	7996	71	0.45	1.82
18	17417	25274	88	2.52	2.32
19	8910	15218	83	1.33	2.13
20	3551	5975	71	0.34	2.29
21	7192	12752	72	1.22	2.00
22	3778	5357	82	0.75	2.28
23	7163	11722	84	1.30	2.21
24	2078	3930	60	0.25	2.00
25	1617	3108	71	0.27	1.82
26	4999	9401	91	0.58	2.12
27	947	1827	90	0.16	2.05
28	14696	24156	72	2.36	2.00
29	21790	35395	75	2.97	2.14
30	4022	6879	81	0.94	2.08
31	1463	2837	79	0.44	1.76
32	8433	14535	75	1.55	2.14
33	4698	8776	76	0.53	1.90
34	10720	19058	54	1.98	2.08
35	1284	2496	80	0.41	2.11
36	4261	7363	65	0.84	1.97
37	1393	2562	63	0.22	1.80
38	6173	11459	84	0.52	2.21
39	26170	45086	79	3.94	2.14
40	2929	5003	75	0.31	1.67
41	6508	11529	87	0.61	1.93
42	7152	12916	87	1.47	2.12
43	9066	14022	86	1.80	1.95
44	15869	27857	78	1.86	2.05
45	20942	31914	76	3.80	2.24
46	11585	20320	102	1.17	2.04
47	2935	5319	70	0.69	1.94
48	5394	9684	70	1.11	2.19
49	3006	5622	83	0.77	1.93
50	36756	56498	84	5.38	2.21
Tiempo total:		60.95			
Expansiones totales:		404444			
Costo total:		3795			

Ahora pasaremos a mostrar los resultados cuando $\omega = 5$ para los primeros 50 problemas.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	1801	3611	95	0.23	2.71
2	10648	15652	100	0.92	3.12
3	1262	2399	80	0.17	2.50
4	2216	4417	77	0.41	2.48
5	3362	5836	74	0.83	2.06
6	2877	5362	87	0.70	2.49
7	1044	1994	72	0.20	2.12
8	4799	8734	111	0.47	2.36
9	17257	23351	76	1.61	2.24
10	850	1615	76	0.33	2.11
11	3601	6881	118	0.83	2.57
12	9588	16399	103	1.05	2.51
13	9087	13990	85	0.80	2.43
14	3890	4935	81	0.50	2.19
15	7903	10749	84	1.56	2.21
16	1995	3472	80	0.38	2.35
17	8557	13467	99	0.77	2.54
18	1355	2548	100	0.36	2.63
19	1491	2727	95	0.39	2.44
20	4725	8837	83	1.11	2.68
21	10353	16065	78	1.42	2.17
22	701	1254	88	0.17	2.44
23	683	1387	96	0.12	2.53
24	936	1882	76	0.12	2.53
25	2349	4284	83	0.23	2.13
26	5747	10157	103	0.53	2.40
27	1622	2977	104	0.41	2.36
28	4841	7645	80	1.08	2.22
29	23450	36622	101	3.30	2.89
30	7759	13496	91	1.61	2.33
31	6818	11741	91	0.86	2.02
32	4997	9031	91	0.52	2.60
33	6502	9357	100	0.53	2.50
34	3865	6092	60	0.42	2.31
35	4104	7160	98	1.09	2.58
36	3700	6663	73	0.88	2.21
37	9992	16772	81	1.95	2.31
38	6532	10717	104	0.59	2.74
39	796	1609	85	0.14	2.30
40	5459	7310	89	1.05	1.98
41	12197	19540	105	2.14	2.33
42	4529	7619	117	0.44	2.85
43	11213	11554	98	0.88	2.23
44	2157	4335	100	0.28	2.63
45	3175	5669	94	0.72	2.76
46	3746	6748	110	0.83	2.20
47	7599	13330	86	1.66	2.39
48	2446	4840	88	0.64	2.75
49	1064	1930	97	0.16	2.26
50	10814	18093	76	1.19	2.00
Tiempo total:		39.56			
Expansiones totales:		268454			
Costo total:		4519			

Notemos que entre los casos analizados, es claro que a mayor valor de ω mayor velocidad de ejecución del algoritmo, pero también tenemos una cota de suboptimalidad más alta. Así se ve claramente que hay un tradeoff al aumentar el valor de ω. Pero, además de lo ya nombrado tenemos que la cantidad de expansiones realizadas va decreciendo a medida que aumentamos ω. Para $\omega = 1{,}5$ realizamos más de 3 millones de expansiones, para $\omega = 2$ realizamos más de 1 millon de expansiones, para $\omega = 3$ realizamos poco más de 400 mil expansiones y finalmente para $\omega = 5$ realizamos poco menos de 270 mil expansiones.

T.1.4)

Para resolver esta pregunta utilizaremos el mismo lema y la misma definición enunciadas en **T.1.3)**.

Notemos que:

$$\min_{s \in Open} [g(s) + h(s)] \leq g(t) + h(t) \quad \forall t \in Open$$

En particular esto se cumple para $t = s^*$, de esta forma:

$$\min_{s \in Open} [g(s)+h(s)] \leq g(s^*)+h(s^*) \overset{Lema}{=} d(s_{inicial}, s^*)+h(s^*) \overset{h\ admisible}{\leq} d(s_{inicial}, s^*)+d(s^*, goal) = c^*$$

$\Rightarrow$

$$\boxed{\min_{s \in Open} [g(s) + h(s)] \leq c^*}$$

$\Rightarrow$

$$\frac{1}{c^*} \leq \frac{1}{\min_{s \in Open}[g(s) + h(s)]}$$

Ahora sea π la solución a un problema de busqueda y sea $c(\pi)$ su costo asociado, entonces $c(\pi) \geq 0$, por lo tanto:

$$\boxed{\frac{c(\pi)}{c^*} \leq \frac{c(\pi)}{\min_{s \in Open}[g(s) + h(s)]}}$$

Demostrando así lo pedido.

P.1.3)

Para implementar *estimate suboptamility* utilizaremos que *BinaryHeap* es iterable y lo que hacemos es iterar sobre open y obtener el menor valor de $h(s) + g(s)$ y por otro lado el valor de $c(\pi)$ lo calculamos en el momento que llegamos a la meta.

Para mostrar que nuestra implementación es correcta, correremos nuestro archivo para $\omega = 2$ para los primeros 10 problemas y mostraremos que coinciden con los resultados puestos en el enunciado de la Tarea.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	4000	7596	55	0.41	1.49
2	7656	14034	56	0.72	1.75
3	5760	11013	50	0.59	1.56
4	13131	23003	49	1.19	1.58
5	3615	6526	58	0.41	1.61
6	2255	3716	59	0.52	1.69
7	27910	51015	54	3.30	1.59
8	11660	22117	73	2.20	1.55
9	67487	118713	58	8.52	1.71
10	23708	43061	60	3.22	1.67

Como se observa, nuestros resultados coinciden con los puestos en el enunciado de la Tarea.

2. Parte 2

T.2.1)

Primero que todo siguiendo la notación usada en el enunciado tenemos que P es el problema original y P' el problema relajado el cual dada la función α_l se tiene que el problema relajado es el problema original pero que se le han agregado nodos y/o arcos. De esta forma tenemos que el grafo asociado a P es subconjunto del grafo asociado a P'.

Consideremos el camino optimo para llegar desde $\alpha_l(s)$ a $\alpha_l(s_g)$. Este camino es el de costo menor de entre todos los caminos que se llegan desde $\alpha_l(s)$ a $\alpha_l(s_g)$. Llamaremos a este costo c'. Ahora como el grafo asociado a P es subconjunto del grafo asociado a P' entonces tenemos los siguientes dos casos:

i. Existe el camino optimo que une a $\alpha_l(s)$ y $\alpha_l(s_g)$ en P.

Como el grafo de P es subconjunto del grafo de P', entonces los caminos que unen a s y s_g es subconjunto de los caminos que unen a $\alpha_l(s)$ y $\alpha_l(s_g)$. Como el mínimo de todos estos caminos tiene costo c' y ese camino existe en P, entonces el costo minimo de los caminos que unen a s y s_g denotado por c es igual a c'.

Por ende, para este caso $c = c'$.

ii. No existe el camino optimo que une a $\alpha_l(s)$ y $\alpha_l(s_g)$ en P.

Como el grafo de P es subconjunto del grafo de P', entonces entre los caminos que unen a s y s_g no se encuentra el camino optimo que une a $\alpha_l(s)$ y $\alpha_l(s_g)$ y además es subconjunto de los caminos que unen a $\alpha_l(s)$ y $\alpha_l(s_g)$. Como el mínimo de todos estos caminos que unen a $\alpha_l(s)$ y $\alpha_l(s_g)$ tiene costo c' y este camino no existe en P, entonces el costo de minimo de los caminos que unen a s y s_g denotado por c es mayor o igual a c'.

Por ende, para este caso $c \geq c'$.

Así llegamos a que uniendo **i** y **ii** concluimos que:

$$\boxed{c \geq c'}$$

Que es justamente lo que nos pedían demostrar.

T.2.2)

Primero que todo, lo que haremos será resolver este problema para una forma general y luego analizaremos el resultado especifico pedido. Sea α_A una función de abstracción donde A es una lista con k elementos que son distintos entre si y que toman valores entre el 1 y el 15.

De esta forma, como el tablero tiene 16 posiciones y los objetos moviendose son estos k elementos más el espacio en blanco, entonces tenemos en realidad $(k+1)$ elementos moviendose.

De esta forma, las posibles composiciones del tablero (que también corresponde a la cantidad de nodos) se puede obtener considerando las siguientes 2 cosas:

(i) La cantidad de formas que se pueden extraer $(k+1)$ elementos de un total de 16 es: $\binom{16}{k+1}$.

(ii) Estamos observando un caso en el que importa el orden, por ende, se debe multiplicar por $(k+1)!$.

De esta forma tenemos que la cantidad de estados son:

$$\boxed{\binom{16}{k+1} \cdot (k+1)!}$$

Ahora, para esta pregunta en especifico nos piden calcular la cardinalidad de $\{\alpha_{[2,11,15]}(s) : s \in \mathcal{S}\}$ y como sabemos por el resultado anterior, tenemos que el caso analizado es el de $k = 3$ y la cantidad de estados es:

$$\binom{16}{4} \cdot (4)! \ = \frac{(16)!}{(12)!\ (4)!} \cdot (4)! \ = \frac{(16)!}{(12)!} = 16 \cdot 15 \cdot 14 \cdot 13 = 43.680$$

Por lo tanto:

$$\boxed{\text{La cardinalidad de } \{\alpha_{[2,11,15]}(s) : s \in \mathcal{S}\} \text{ es } 43.680}$$

Ahora notemos que la cardinalidad de $\mathcal{S}$ es equivalente a analizar la cantidad de casos analizados anteriormente para $k = 15$, pues en ese caso la función de abstracción es $\alpha_{[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}$ y es claro que en ese caso el problema es igual al original.

De esta forma la cardinalidad de $\mathcal{S}$ es:

$$\binom{16}{15+1} \cdot (15+1)! \ = \binom{16}{16} \cdot (16)! \ = (16)! \ \approx 2.09 \cdot 10^{13}$$

De esta manera:

$$\boxed{\text{La cardinalidad de } \mathcal{S} \text{ es aproximadamente } 2.09 \cdot 10^{13}}$$

Notemos que es claro que:

$$\boxed{\text{Cardinalidad de } \mathcal{S} - \ \text{Cardinalidad de } \{\alpha_{[2,11,15]}(s) : s \in \mathcal{S}\} \approx 2.09 \cdot 10^{13}}$$

Y además:

$$\boxed{\frac{\text{Cardinal de } \mathcal{S}}{\text{Cardinalidad de } \{\alpha_{[2,11,15]}(s) : s \in \mathcal{S}\}} \approx 4.79 \cdot 10^{8}}$$

T.2.3) Notemos que sea h una heuristica admisible, entonces:

$$h(s) \leq d(s, goal) \ \forall s \quad (1)$$

Y si h' es una heuristica admisible, entonces:

$$h'(s) \leq d(s, goal) \ \forall s \quad (2)$$

Uniendo (2) y (1) tenemos que:

$$h''(s) = \text{máx}\{h(s), h'(s)\} \leq \text{máx}\{d(s, goal), d(s, goal)\} = d(s, goal) \quad \forall s$$

$\Rightarrow$

$$\boxed{h''(s) = \text{máx}\{h(s), h'(s)\} \leq d(s, goal) \quad \forall s}$$

Por lo tanto máx$\{h(s), h'(s)\}$ es una heuristica admisible.

P.2.1) y **P.2.2)** Para esta parte lo que hicimos fue implementar todo lo pedido, y todo esto esta implementado en el archivo:

`puzzle.py`

que es justamente el archivo que entregamos.

Ahora, nosotros para superar el algoritmo del profesor lo que hicimos fue comparar 9 patrones los cuales corresponden a:

$$[1, 5, 9, 13] \quad [2, 6, 10, 14] \quad [3, 7, 11, 15] \quad [1, 4, 8, 12] \quad [2, 5, 9, 13]$$

$$[1, 3, 12, 15] \quad [5, 6, 9, 10] \quad [1, 2, 13, 14] \quad [4, 7, 8, 11]$$

Notemos que por lo demostrado en **T.2.3)** tenemos que el maximo entre dos heuristicas admisibles, es una heuristica admisible, entonces si lo extrapolamos para 9 casos también se cumple.

Como tenemos 9 patrones, entonces es necesario inicializar 9 tableros y dado a que esto es un poco tedioso, también subiré el archivo:

`test_astar.py`

el cual ya tiene implementado todo esto y para obtener los resultados que pasaremos a mostrar solo es necesario descargar este archivo, el de puzzle, las bases de datos proporcionadas por nosotros y el resto de archivos que subio el profesor.

Los resultados obtenidos fueron:

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	16659	32875	45	13.78	1.00
2	3924	8558	42	3.39	1.00
3	18549	37900	42	17.23	1.00
4	11470	24272	41	11.55	1.00
5	53057	105415	46	45.17	1.00
6	58821	115008	47	41.58	1.00
7	63193	126843	44	44.03	1.00
8	167673	324022	53	128.94	1.00
9	76642	155428	46	47.16	1.00
10	74244	148665	46	42.27	1.00
Tiempo total:		395.09			
Expansiones totales:		544232			
Costo total:		452			

Como se observa para cada uno de los problemas analizados tenemos que obtuve menos exapansiones y por ende en terminos de expansiones totales también obtuvimos un numero menor, por lo tanto nuestro algoritmo es más eficiente.

3. Parte 3

P.3.1)

Implementamos todo lo pedido por enunciado y lo tenemos en el archivo:

`ara.py`

En donde implementamos la clase *Ara*.

Ahora pasaremos a mostrar los resultados que obtuvimos al ejecutar el archivo:

`test_ara.py`

el cual también subiremos.

#prob	#exp	#gen	|sol|	tiempo	maxsubopt
1	3219	5727	105	0.52	3.00
1	7889	14260	79	1.19	2.26
1	14238	26070	61	1.88	1.65
1	21262	39872	53	2.67	1.43
1	27387	51438	45	3.42	1.15
2	2965	5754	136	0.48	4.25
2	15805	21747	74	3.47	2.31
2	28609	45208	58	4.83	1.81
2	55705	89944	50	8.20	1.47
2	71910	117422	42	9.98	1.17
2	80000	132197	42	9.98	1.05
3	1958	3465	100	0.41	3.12
3	32101	45395	70	5.80	2.19
3	38312	54924	50	8.06	1.56
3	56204	87735	42	13.92	1.17
3	80000	133670	42	13.92	1.05
4	1100	2119	81	0.55	2.61
4	13632	24366	57	3.92	1.84
4	30830	52661	49	8.66	1.58
4	52496	90461	47	14.16	1.34
4	55002	94746	43	15.19	1.23
4	80000	141396	43	15.19	1.10
5	1763	3464	116	0.41	3.22
5	3920	7353	74	0.61	2.06
5	7449	13591	58	0.97	1.61
5	36042	63919	50	4.78	1.32
5	40147	69224	46	6.02	1.21
5	80000	143929	46	6.02	1.10

```
 6      1721      3315       115      0.72      3.29
 6      4299      8128        77      1.27      2.20
 6      6205     11032        61      1.67      1.74
 6     14332     26785        53      3.75      1.43
 6     30577     57267        47      6.83      1.21
 6     80000    150289        47      6.83      1.09
 7      1457      2916        76      0.70      2.24
 7      1621      3252        48      0.75      1.41
 7     17386     34019        46      3.66      1.28
 7     72166    139057        44     18.11      1.16
 7     80000    153530        44     18.11      1.10
 8      8819     14018       141      2.70      3.00
 8      9162     14634        91      2.83      1.94
 8     50416     91157        71      8.48      1.51
 8     55544    101036        61      9.44      1.30
 8     80000    147476        61      9.44      1.30
 9      3542      6536       108      0.75      3.18
 9      8631     13727        74      1.30      2.18
 9     80000    131520        74      1.30      2.18
10      4911      7817       112      1.69      3.11
10      6425     10364        66      2.09      1.83
10     80000    134291        66      2.09      1.83
Tiempo total:        269.69
Expansiones totales: 797731
Costo total:         510
```

Como se observa, los resultados son los mismos que obtuvo el profesor, lo cual indica que nuestro algoritmo esta bien implementado.

T.3.1)

Lo que nos piden demostrar es que los nodos que se descartan no pertenecen al camino optimo.

De aquí en más usaremos la notación de que s es el nodo que decidimos no expandir y s^* es el nodo asociado a c_{ult}, es decir, el ultimo nodo encontrado que llega a la meta.

Primero que todo, notemos que si estamos viendo si expandir s es porque todovía no hemos llegado al optimo.

Además, como s^* es el ultimo nodo encontrado y además como h es admisible, entonces:

$$c_{ult} = g(s^*) + \omega \cdot h(s^*) \overset{\text{Todavía no llegamos al optimo}}{>} min\ _{t \in Open}[g(t) + h(t)]$$

$\Rightarrow$

$$\boxed{c_{ult} > min\ _{t \in Open}[g(t) + h(t)] \quad (1)}$$

Ahora como s es tal que no se expande, entonces:

$$\boxed{g(s) + h(s) \geq c_{ult} \quad (2)}$$

Ahora uniendo (1) y (2) llegamos a que:

$$g(s) + h(s) \overset{(2)}{\geq} c_{ult} \overset{(1)}{>} min\ _{t \in Open}[g(t) + h(t)]$$

$\Rightarrow$

$$\boxed{g(s) + h(s) > min\ _{t \in Open}[g(t) + h(t)] \quad (3)}$$

Ahora, notemos que todo nodo para todo nodo v que lleguemos a través de s se cumple que:

$\Rightarrow$

$$\boxed{g(v) + h(v) \geq g(s) + h(s) \quad (4)}$$

Ahora uniendo (4) y (3) llegamos a que:

$$g(v) + h(v) \overset{(4)}{\geq} g(s) + h(s) \overset{(3)}{>} min\ _{t \in Open}[g(t) + h(t)]$$

$\Rightarrow$

$$\boxed{g(v) + h(v) > min\ _{t \in Open}[g(t) + h(t)] \quad (5)}$$

De esta manera, por (5) hemos demostrado que para todos los caminos que llegamos a través de s se da que el costo de ese camino no es el optimo, lo cual es justamente lo que nos pedían demostrar.