UNIVERSITAT DE BARCELONA

Pràctica 1. Elementary Search Algorithms

Noah Márquez Vara
Gemma Vallès Fusta

4 octubre 2023

1 OBJECTIUS DE LA PRÀCTICA

L'objectiu principal de la pràctica és implementar un algorisme de cerca informada, anomenat A^*. Aquest algorisme ha de ser capaç d'arribar a la configuració objectiu de fer un *check-mate* en un taulell d'escacs en el que només es poden moure les dues peces blanques, i el rei negre es manté estàtic a la posició [0, 4].

Els algorismes de cerca informada, a diferència dels de cerca no-informada, utilitzen informació addicional. Aquesta informació addicional ens permet determinar el recorregut en una quantitat menor de temps, amb una major probabilitat d'haver trobat el camí òptim per arribar a l'objectiu.

2 ALGORISME IMPLEMENTAT (A^*)

Una gran particularitat de l'algorisme de l'A^* és que evita expandir nodes que ja són cars, ja que per a cada node es calcula:

$$f(n) = h(n) + g(n)$$

on:

- $\boldsymbol{f(n)}$ representa el cost total estimat del camí més barat des del node inicial fins al node objectiu que passa pel node n. És el valor utilitzat per prioritzar els nodes a la cua de cerca. Valors més baixos de f(n) indiquen nodes més prometedors.
- $\boldsymbol{h(n)}$ representa l'estimació del cost heurístic des del node n fins al node objectiu. En altres paraules, és una estimació del cost o la distància des del node n fins al node objectiu sense tenir en compte cap camí real. Aquesta funció heurística és específica del problema i hauria de ser admissible, és a dir, mai sobreestima el cost real per assolir l'objectiu. *(ens ve ja donada una funció que calcula l'heurística)*
- $\boldsymbol{g(n)}$ representa el cost del camí des del node inicial fins al node n. És el cost real d'arribar al node n des del node inicial. En altres paraules, és el cost conegut per arribar a aquest node. En el nostre codi és la profunditat actual (augmenta 1 per cada fill que visitem).

A^* continua expandint els nodes fins que arriba a l'objectiu o determina que no existeix cap camí.

3 QÜESTIÓ PLANTEJADA

Now, initialize the board with the white king on position [7,7] and rerun your code. Does your algorithm work in precisely the same way? Did you have to change anything?

En inicialitzar el tauler amb el rei blanc en la posició [7,7] podem observar que el nostre codi segueix funcionant perfectament i aconsegueix resoldre el problema en un temps d'execució baix. Així doncs, hem comprovat que el nostre codi pot resoldre el problema en una posició inicial diferent sense necessitat de canviar el codi.

Com a prova addicional també podem comprovar que el codi segueix funcionant correctament al canviar la torre blanca en la posició inicial a [7,7] i mantenint el rei blanc a la seva posició original [7,4].

4 PROVES REALITZADES

Per a comprovar que el nostre algorisme (A^*) ha estat implementat de forma òptima hem realitzat diverses proves [1].

1. Rendiment

Primer de tot, hem comparat el rendiment del nostre algorisme comparat amb el Depth First Search i Breadth First Search que tenim en el codi. Les variables que hem decidit comparar són el temps d'execució i els moviments necessàris per a arribar a l'objectiu.

En les dues taules següents s'indica l'informació corresponent, en una el rei blanc està en la posició [7,4] i en l'altra en [7,7].

Algorisme	**Temps (s)**	**# moviments**
Depth First Search	307	8
Breadth First Search	254	6
A* Search	0,0009	6

Taula 4.1: Peces en les posicions: [0, 4, 12], [7, 0, 2], [7, 4, 6]

Algorisme	**Temps (s)**	**# moviments**
Depth First Search	186	8
Breadth First Search	107	6
A* Search	0,0019	6

Taula 4.2: Peces en les posicions: [0, 4, 12], [7, 0, 2], [7, 7, 6]

La variació del DFS optimitzada triga notablement més que la versió original, per això no l'hem inclòs en la comparació. Tot i així, les raons principals per les que considerem que pot trigar més són:

1. La sobrecàrrega introduïda calculant permutacions d'estats en el mètode *worthExploring* (tot i que només són dues permutacions) i convertir-les a *string*.
2. El mecanisme per revisar els estats que s'han trobat a nivells més profunds.

2. Múltiples casos

A més a més, també hem afegit un codi addicional que ens permet crear taulers on la posició del rei i la torre blanca són aleatòries per a així poder comprovar que el nostre algorisme funciona per més casos dels ja provats.

A continuació mostrarem en una taula els resultats obtinguts en provar l'algorisme en 10 posicions aleatòries:

[1] És important mencionar que per la nostra implementació hem modificat (mínimament) la funció *reconstructPath*, deixem comentada la funció original per si es vol executar el BFS correctament.

Posició rei blanc	Posició torre blanca	Temps (s)	# moviments
[5,3]	[5,4]	0,0019	5
[7,1]	[6,1]	0,0039	6
[2,4]	[4,0]	0.0019	1
[7,7]	[7,1]	0.0019	6
[4,6]	[1,2]	0,0019	3
[6,3]	[1,2]	0.0039	5
[0,0]	[4,5]	0.0069	6
[4,6]	[0,3]	0,0039	3
[6,1]	[7,4]	0,0039	6
[7,0]	[7,4]	0,0049	7

Taula 4.3: Temps i moviments segons la posició inicial del tauler.

5 Heurística

Una heurística és una estimació de quant pròxim es troba un node de l'objectiu del problema. També cal ressaltar que una heurística és específica del problema concret i indispensable per a l'algorisme (A^*).

Com hem vist a teoria, l'$A*$ és òptima en arbres si l'heurística h utilitzada és admissible; i en el cas dels grafs, si la h és consistent.

1. La nostra heurística

En el nostre cas específic, per calcular la nostra heurística primer es calcula la distància de Manhattan del rei blanc fins a la posició que volem assolir, que aquí és (2, 4), però no podem acabar aquí ja que hem de tenir en compte que el rei es pot moure en diagonal, és per això que un cop tenim la distància que hem de recórrer vertical i horitzontalment, ens quedem amb la més baixa d'ambdues i li sumem la diferència entre la fila i la columna.

```
#Distància de Manhattan
fila = abs(posicioRei[0] - 2)
columna = abs(posicioRei[1] - 4)

#Sumar la menor amb la diferència
hRei = min(fila, columna) + abs(fila - columna)
```

Ara només falta calcular la distància de la torre fins a l'objectiu i sumar-ho amb la del rei. Si la torre està en la fila zero i no entre les columnes 3-5, el cost serà 0 perquè ja està en la posició que volem. Si no està a la primera fila i està entre les columnes 3-5, el cost serà 2 ja que s'ha de moure vertical i horitzontalment per assolir la posició objectiu. Finalment en qualsevol altre cas tindrem cost 1 ja que només s'haurà de moure verticalment fins a la primera fila o horitzontalment fora de les columnes 3 o 5.

```
if posicioTorre[0] == 0 and (posicioTorre[1] < 3 or posicioTorre[1] > 5):
  hTorre = 0
elif posicioTorre[0] != 0 and posicioTorre[1] >= 3 and posicioTorre[1] <= 5:
  hTorre = 2
```

```
else:
  hTorre = 1
```

Finalment sumem els dos costos i ja tenim l'heurística:

```
return hRei + hTorre
```

2. És òptima?

Un cop explicada la nostra heurística, ara falta provar que és òptima, és a dir, admissible i consistent.

Una heurística *h(n)* és admissible si per a tot node n es compleix:

$$h(n) \leq h^*(n)$$

on $h^*(n)$ és el cost real d'assolir l'estat objectiu des de n. En altres paraules, és admissible si mai sobreestima el cost d'arribar a l'objectiu.

Podem observar que això es compleix en la nostra heurística, ja que utilitzem una modificació de la distància de Manhattan per a tenir en compte que el rei es pot moure diagonalment, obtenint així el camí més curt possible. La distància de Manhattan és normalment admissible per al moviment basat en quadrícula si no es permet el moviment en diagonal. No obstant això, els canvis fets per al moviment en diagonal garanteix que segueixi sent admissible.

L'heurística de la torre es basa en configuracions específiques del tauler, de manera que la seva admissibilitat es compleix ja que en cap dels casos estima tenir un cost de més de 2, que és el màxim necessari per aconseguir el *check-mate* en aquest tauler.

Una heurística és consistent si compleix que per cada node n, tot successor n' de n generat per qualsevol acció d'a compleix que:

$$h(n) \leq c(n, a, n') + h(n')$$

És a dir, si per cada node n i el seu successor n' generats per qualsevol acció a, el cost estima d'arribar a l'objectiu des de n no és més gran que el cost d'arribar a n' més el cost d'arribar a l'objectiu des de n'. $c(n, a, n')$ és el cost de l'acció a que ens porta de l'estat n a l'estat n'.

Analitzant la nostra implementació de l'algorisme de l'$A*$ per determinar si l'heurística és consistent veiem que:

- La funció explora els estats en funció del seu cost estimat f, on f és la suma del valor heurístic h i la profunditat de l'estat.
- Per a cada estat, la funció calcula el valor heurístic dels seus estats successors mitjançant la funció heurística h.
- Si l'estat successor n' s'ha visitat abans, la funció comprova si el seu nou cost estimat f és menor que la suma de $h(n')$ i la profunditat de n'. Si és així, s'omet l'estat successor.

A partir d'aquest procés, la consistència de l'heurística es pot deduir de la següent manera:

1. L'heurística del rei blanc és la distància modificada de Manhattan a la posició (2,4). A mesura que el rei s'acosta a aquesta posició, el seu valor heurístic disminuirà o es mantindrà igual. El cost de moure el rei un pas sempre és 1. Per tant, el valor heurístic de l'estat actual sempre serà inferior o igual al valor heurístic de l'estat successor més 1.
2. Per a la torre, el valor heurístic es determina en funció de la seva posició al tauler. Donats els valors definits (0, 1 o 2), el canvi en el valor heurístic quan la torre es mou és com a màxim 2. Com que el cost de moure la torre també és 1, el valor heurístic per a l'estat actual serà inferior a o igual al valor heurístic de l'estat successor més 1.

Tenint en compte aquestes observacions i el comportament de la funció *AStarSearch*, l'heurística sembla ser consistent.

3. Una altra possible heurística

Una vegada hem comprovat que la nostra heurística és òptima, ens hem preguntat quina altra heurística seria òptima en el nostre problema. En el cas de la torre no tenim gaire marge de millora ja que només és basa en els 3 casos que es poden donar, però per calcular el cost del rei tenim alternatives. Com que el rei es pot moure diagonalment, ja hem vist que la distància de Manhattan no és totalment òptima i s'ha de modificar, és per això que la nostra nova heurística utilitzarà la distància Euclidiana, la qual és més òptima.

El codi és el següent:

```
def h'(self,state):
    if state[0][2] == 2:
        posicioRei = state[1]
        posicioTorre = state[0]
    else:
        posicioRei = state[0]
        posicioTorre = state[1]
    # With the king we wish to reach configuration (2,4),
    # calculate Euclidean distance
    hRei = math.sqrt((posicioRei[0] - 2)**2 + (posicioRei[1] - 4)**2)
    # with the tower we have 3 different cases
    if posicioTorre[0] == 0 and (posicioTorre[1] < 3 or posicioTorre[1] > 5):
        hTorre = 0
    elif posicioTorre[0] != 0 and posicioTorre[1] >= 3 and posicioTorre[1] <= 5:
        hTorre = 2
    else:
        hTorre = 1
    # In our case, the heuristics is the real cost of movements
    return hRei + hTorre
```

Un cop tenim la nova heurística, ara només falta comprovar que sigui òptima i comparar-la amb la que ja teníem per veure si una és significativament millor que l'altra o si són similars.

Per comprovar que és òptima només cal analitzar la distància Euclidiana, perquè ja hem comprovat la resta de l'algorisme en l'heurística anterior. Com que la distància Euclidiana és per definició la distància mínima entre dos punts, compleix els requeriments per a ser òptima

Per a comparar les dues heurístiques inicialitzarem el tauler en les mateixes posicions per ambdues heurístiques i analitzarem el temps d'execució i els estats visitats en cada cas.

En les taules següents hem fet la comparativa:

Heurística inicial		
Tauler inicial	**Temps (s)**	**# estats visitats**
[[7, 0, 2], [7, 4, 6]]	0.0039	86
[[7, 0, 2], [7, 7, 6]]	0.0029	87
[[7, 0, 6], [7, 4, 2]]	0.0029	106
[[3, 0, 6], [7, 7, 2]]	0.0039	89
[[1, 6, 6], [2, 6, 2]]	0.0020	45

Taula 5.1: Temps i estats visitats en el cas de l'heurística inicial.

Heurística Euclidiana		
Tauler inicial	**Temps (s)**	**# estats visitats**
[[7, 0, 2], [7, 4, 6]]	0.0039	80
[[7, 0, 2], [7, 7, 6]]	0.0039	98
[[7, 0, 6], [7, 4, 2]]	0.0029	115
[[3, 0, 6], [7, 7, 2]]	0.0019	74
[[1, 6, 6], [2, 6, 2]]	0.0019	44

Taula 5.2: Temps i estats visitats en el cas de l'heurística Euclidiana.

Segons els resultats obtinguts podem veure que les dues heurístiques són similars, a vegades una és lleugerament més eficient que l'altra segons el cas concret, però de mitjana els resultats són molt similars.

6 CONCLUSIONS

En conclusió, la nostra implementació de l'algorisme A^* va demostrar ser una solució robusta i eficient per al problema d'escacs en qüestió. Amb el rei negre en una posició fixa i amb ple control sobre els moviments del rei blanc i de la torre, el nostre algorisme va navegar eficaçment pel tauler d'escacs per arribar a un estat d'escac i mat. Aquest projecte no només va mostrar el poder d'A^* per guiar la presa de decisions intel·ligents, sinó que també va destacar la importància de les funcions heurístiques per estimar els camins més prometedors.

A més, aquest exercici va servir com una excel·lent oportunitat d'aprenentatge, aprofundint en la nostra comprensió dels conceptes algorísmics i les seves aplicacions al món real. Va subratllar les complexitats inherents als escacs com a domini de la intel·ligència artificial, posant èmfasi en la necessitat de la planificació estratègica i la precisió algorítmica.