INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE

Compte-rendu
Intelligence Artificielle

Ludovic Mocquais
Mariétou Sarr

03/04/2022

4 IR B2

Compte-rendu Intelligence Artificielle

Sommaire

I. ALGORITHME A* - APPLICATION AU TAQUIN

1. Familiarisation avec le problème du Taquin 3×3

1.2 a) Quelle clause Prolog permettrait de représenter la situation finale du Taquin 4x4 ?

```
final_state(
            [[1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
            [13, 14, 15, vide]]
    ).
```

b) A quelles questions permettent de répondre les requêtes suivantes :

?- initial_state(Ini), nth1(L,Ini,Ligne), nth1(C,Ligne, d).

Quelle est la position de d dans le jeu ?

?- final_state(Fin), nth1(3,Fin,Ligne), nth1(2,Ligne,P)

Quelle est la pièce à la 3ème ligne et à la 2ème colonne de l'état final du jeu?

c) Quelle requête Prolog permettrait de savoir si une pièce donnée P (ex : a) est bien placée dans U_0 (par rapport à F) ?

?- P=a, final_state(F), initial_state(U0), nth1(L,U0,LigneU), nth1(C,LigneU, P), nth1(L,F,LigneF), nth1(C,LigneF, P).

d) Quelle requête permet de trouver une situation suivante de l'état initial du Taquin 3×3 (3 sont possibles) ?

? - initial_state(Ini),rule(X, 1, Ini, S).

e) Quelle requête permet d'avoir ces 3 réponses regroupées dans une liste ?

?- initial_state(Ini), findall(S, rule(X, 1, Ini, S), ListeRes).

f) Quelle requête permet d'avoir la liste de tous les couples [A, S] tels que S est la situation qui résulte de l'action A en U0 ?

?- initial_state(Ini), findall([X,S], rule(X, 1, Ini, S), ListeRes).

2. Développement des 2 heuristiques

Les tests de nos heuristiques sur la situation initiale U0 et la situation finale F nous donne un coûts suivants :

	Situation Initiale	Situation Finale
heuristique1	4	0
heuristique2	5	0

3. Implémentation de A*

3.2. Algorithme A* adapté aux structures AVL choisies

Les prédicats intermédiaires *affiche_solution, expand, loop_successors* peuvent être testés avec les prédicats de tests suivants :

test_affiche_solution :

```
?- test_affiche_solution.
 left down
true
```

test_expand :

```
?- test_expand(T),write(T).
[[[[b,h,c],[a,vide,d],[g,f,e]],[5,4,1],[[b,h,c],[a,f,d],[g,vide,e]],up],[[[b,h,c
],[a,f,d],[vide,g,e]],[7,6,1],[[b,h,c],[a,f,d],[g,vide,e]],left],[[[b,h,c],[a,f,
d],[g,e,vide]],[7,6,1],[[b,h,c],[a,f,d],[g,vide,e]],right]]
```

test_loop_succ :

```
?- test_loop_succ(T).

Qn
Pfn
[[5,4,1],[[b,h,c],[a,vide,d],[g,f,e]]]
[[5,5,0],[[b,h,c],[a,f,d],[g,vide,e]]]
[[7,6,1],[[b,h,c],[a,f,d],[g,e,vide]]]
[[7,6,1],[[b,h,c],[a,f,d],[vide,g,e]]]
Pun
[[[b,h,c],[a,f,d],[g,e,vide]],[7,6,1],[[b,h,c],[a,f,d],[g,vide,e]],right]
[[[b,h,c],[a,f,d],[g,vide,e]],[5,5,0],nil,nil]
[[[b,h,c],[a,f,d],[vide,g,e]],[7,6,1],[[b,h,c],[a,f,d],[g,vide,e]],left]
[[[b,h,c],[a,vide,d],[g,f,e]],[5,4,1],[[b,h,c],[a,f,d],[g,vide,e]],up]
T = [[[[b, h, c], [a, vide, d], [g, f, e]], [5, 4, 1], [[b, h, c], [a, f, d], [g
, vide|...]], up], [[[b, h, c], [a, f, d], [vide, g, e]], [7, 6, 1], [[b, h, c],
 [a, f|...], [g|...]], left], [[[b, h, c], [a, f, d], [g, e|...]], [7, 6, 1], [[
b, h|...], [a|...], [...|...]], right]] [write]
T = [[[[b, h, c], [a, vide, d], [g, f, e]], [5, 4, 1], [[b, h, c], [a, f, d], [g
, vide, e]], up], [[[b, h, c], [a, f, d], [vide, g, e]], [7, 6, 1], [[b, h, c],
[a, f, d], [g, vide, e]], left], [[[b, h, c], [a, f, d], [g, e, vide]], [7, 6, 1
], [[b, h, c], [a, f, d], [g, vide, e]], right]]
```

Le code est fourni à la fin du fichier aetoile.pl.

3.3. Analyse expérimentale

- *Temps de calcul de A* et influence du choix de l'heuristique*

Situations initiales → Taille séquence optimale / temps de calcul ↘	[[b, h, c], [a, f, e], [g,h,vide]] h1 = 4 h2 = 5 f* = 5	[[a, b, c], [g, h, d], [vide,f,e]] h2 = 2 f*=2	[[b, c, d], [a,vide, g], [f,h,e]] h2 = 10 f* = 10	[[f, g, a], [h,vide, b], [d,c,e]] h2 = 16 f* = 20	[[e, f, g], [d,vide, h], [c,b,a]] h2 = 24 f* = 30	[[a, b, c], [g,vide, d], [h,f,e]] PAS DE SOLUTION
heuristique1	5 / 4ms	2 / 0ms	10 / 8ms			
heuristique2	5 / 2ms	2 / 0ms	10 / 4ms	20 / 23ms		

**Les cases vides sont dues à des exécutions de A* qui plantent. (false)*

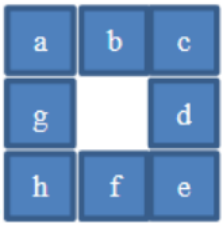
- *Quelle longueur de séquence à envisager pour résoudre le Taquin 4x4 ?*

Ce taquin étant plus grand, on aura certainement des séquences plus longues en fonction de l'état initial donné.
avec U0 ci dessous comme situation initiale, nous avons cette séquence :

```
?- main.
 left down down right up left up right down down right right
true
```

de taille 12.
intitial_state(
[[1, 2, 3, 4],
[5, vide, 7, 8],
[9, 10, 11, 12],
[6, 13, 14, 15]]
).

- *A* trouve-t-il la solution pour la situation initiale suivante ?*



A* ne trouve pas la solution pour cette situation initiale car l'état initial n'est pas connexe avec l'état final. En effet, en partant de cette situation, on arrivera jamais à l'état final. L'arbre des états à développer sera donc vide à un moment.

- *Représentation de l'état du Rubik's Cube*

```
final_state(
        [[[r, r, r],
        [r, r, r],
        [r,r,r] ],
        [[o, o, o],
        [o, o, o],
        [o,o,o] ] ,
        [[b, b, b],
        [b, b, b],
        [b, b, b]],
        [[j, j, j],
        [j, j, j],
        [j, j, j]],
        [[be, be, be],
        [be, be, be],
        [be, be, be] ],
        [[v, v, v],
        [v, v, v],
        [v, v, v] ]]).
```

4. Limitations du programme

Notre algorithme de A* étant programmé en prolog, il n'y a pas vraiment de notions de type. Il est donc assez portable. Il faudrait lui fournir une pile pour sauvegarder les états à développer et un autre pour les états déjà développés.
Le jeu auquel il sera appliqué devra aussi avoir certains prédicats tels que initial_state, rule…
Cependant, comme nous manipulons deux arbres de données frères pour les états à développer, nous pouvons rendre notre algorithme plus portable en créant des fonctions qui regroupent les insertions et les suppressions de données dans Pu et Pf. Nous aurions ainsi un seul insert et un seul suppress / suppress_min.

II. ALGO NEGAMAX - APPLICATION AU TICTACTOE

1. Familiarisation avec le problème du TicTacToe 3×3

- *Quelle interprétation donnez-vous aux requêtes suivantes :*

  ?- situation_initiale(S), joueur_initial(J).

  Retourne le tableau initial de tictactoe, ici, vide, et le premier joueur à jouer.

  ?- situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o).

  Cette requête crée un tableau initial de tictactoe où le joueur O a joué à la troisième ligne et à la deuxième colonne. Une variable _ (<=> n'importe quelle valeur) sera ainsi remplacée par o.

- *Requêtes de tests unitaires pour chaque prédicat (ligne, colonne, diagonale, possible, alignement_gagnant, alignement_perdant)*

Ligne :

?- M = [[a,b,c], [d,e,f], [g,h,i]], ligne(L, M).

```
?- M = [[a,b,c], [d,e,f], [g,h,i]], ligne(L, M).
M = [[a, b, c], [d, e, f], [g, h, i]],
L = [a, b, c] ;
M = [[a, b, c], [d, e, f], [g, h, i]],
L = [d, e, f] ;
M = [[a, b, c], [d, e, f], [g, h, i]],
L = [g, h, i].
```

Colonne :

?- M = [[a,b,c], [d,e,f], [g,h,i]], *colonne*(C, M).

```
?- M = [[a,b,c], [d,e,f], [g,h,i]], colonne(C, M).
M = [[a, b, c], [d, e, f], [g, h, i]],
C = [a, d, g] ;
M = [[a, b, c], [d, e, f], [g, h, i]],
C = [b, e, h] ;
M = [[a, b, c], [d, e, f], [g, h, i]],
C = [c, f, i].
```

Diagonale :

?- M = [[a,b,c], [d,e,f], [g,h,i]], *diagonale*(D, M).

```
?- M = [[a,b,c], [d,e,f], [g,h,i]], diagonale(D, M).
M = [[a, b, c], [d, e, f], [g, h, i]],
D = [a, e, i] ;
M = [[a, b, c], [d, e, f], [g, h, i]],
D = [c, e, g].
```

Possible:

?- M = [o,_,_], possible(M, o).

```
?- M = [o,_,_], possible(M, o).
M = [o, _14, _20].

?- M = [x,_,_], possible(M, o).
false.

?- M = [o,o,o], possible(M, o).
M = [o, o, o].

?- M = [x,_,o], possible(M, o).
false.
```

Alignement gagnant et Alignement perdant:

?- M = [x,x,x], *alignement_gagnant(M,x), alignement_perdant(M,o).*
?- M = [o,x,_], *alignement_gagnant(M,x), alignement_perdant(M,o).*
?- M = [x,x,o], *alignement_gagnant(M,x), alignement_perdant(M,o).*

```
?- M = [x,x,x], alignement_gagnant(M,x), alignement_perdant(M,o).
M = [x, x, x].

?- M = [x,x,_], alignement_gagnant(M,x), alignement_perdant(M,o).
false.

?- M = [x,x,o], alignement_gagnant(M,x), alignement_perdant(M,o).
false.
```

2. Développement de l'heuristique h(Joueur, Situation)

- *Requêtes test de l'heuristique à la situation initiale*

```
?- situation_initiale(U), heuristique(x,U,H1).
U = [[_774, _780, _786], [_798, _804, _810], [_822, _828, _834]],
H1 = 0.
```

- *Requêtes test de l'heuristique à la situation finale gagnante et perdante, nulle*

```
?- M=[[_,_,_], [o,x,o], [x,x,x]], heuristique(x,M,H1).
M = [[_398, _404, _410], [o, x, o], [x, x, x]],
H1 = 10000.

?- M=[[_,_,_], [o,x,o], [x,x,x]], heuristique(o,M,H1).
M = [[_398, _404, _410], [o, x, o], [x, x, x]],
H1 = -10000.

?- M=[[o,o,x], [x,x,o], [o,x,o]], heuristique(o,M,H1).
M = [[o, o, x], [x, x, o], [o, x, o]],
H1 = 0.

?- M=[[o,o,x], [x,x,o], [o,x,o]], heuristique(x,M,H1).
M = [[o, o, x], [x, x, o], [o, x, o]],
H1 = 0.
```

3. *Développement de l'algorithme Négamax*

- *Quel prédicat permet de connaître sous forme de liste l'ensemble des couples [Coord, Situation_Resultante] tels que chaque élément (couple) associe le coup d'un joueur et la situation qui en résulte à partir d'une situation donnée ?*

?- successeurs(J,Etat,Succ).

```
?- Etat= [[_,_,_], [o,x,o], [x,x,x]],successeurs(x,Etat,Succ),write(Succ).
[[[1,1],[[x,_11308,_11314],[o,x,o],[x,x,x]]],[[1,2],[[_11200,x,_11212],[o,x,o],[
x,x,x]]],[[1,3],[[_11098,_11104,x],[o,x,o],[x,x,x]]]]
Etat = [[_10332, _10338, _10344], [o, x, o], [x, x, x]],
Succ = [[[1, 1], [[x, _11308, _11314], [o, x, o], [x, x, x]]], [[1, 2], [[_11200
, x, _11212], [o, x, o], [x, x|...]]], [[1, 3], [[_11098, _11104, x], [o, x|...]
, [x|...]]]].
```

4. Analyse et Expérimentation

4.1.

- *Quel est le meilleur coup à jouer et le gain espéré pour une profondeur d'analyse de 1, 2, 3, 4, 5 , 6 , 7, 8, 9*

Profondeur	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Coup	*[2,2]*	*[2,2]*	*[2,2]*	*[2,2]*	*[2,2]*	*[2,2]*	*[2,2]*	*[1,1]*	*[1,1]*
Gain	*4*	*1*	*3*	*1*	*3*	*1*	*2*	*0*	*0*
Temps d'exécution (en	*1*	*3*	*23*	*136*	*655*	*2502*	*6469*	*12327*	*14489*

ms)									

- *Expliquer les résultats obtenus pour 9 (toute la grille remplie)*

Avec une profondeur de 9, l'algorithme fait une estimation des coups des joueurs jusqu'à la fin de la partie, et trouve qu'en utilisant les meilleurs coups possibles, la partie sera toujours nulle. En simulant le jeu avec x joue d'abord en [1,1] puis o en [2,2], on tombe sur une partie nulle. En inversant les coups des joueurs, le résultat reste le même. Il en est de même pour les coups symétriques à [1,1] ([1,3], [3,1] et [3,3]. L'algorithme choisit donc parmi ceux-là (le premier dans notre cas).

4.2. Comment ne pas développer inutilement des situations symétriques de situations déjà développées ?

Pour ne pas développer inutilement les situations symétriques de situations déjà développées, on peut ajouter une notion de rotation de 90, 180 et 270°. En traitant les successeurs d'un état (loop_successors), on peut comparer leurs rotations à des situations déjà développées (stockées en Q) et les oublier en cas d'égalité.

4.3. Que faut-il reprendre pour passer au jeu du puissance 4 ?

Il faudrait changer les règles de jeu en spécifiant non pas la position où placer le pion mais seulement le numéro de la colonne le pion se plaçant à la position la plus basse.

4.4. Comment améliorer l'algorithme en élaguant certains coups inutiles (recherche Alpha-Beta) ?

On pourrait implémenter l'algorithme de Alpha-Beta prenant en compte la convention Négamax, et ainsi ne pas explorer les branches dont on sait d'avance qu'elles ne rapportent pas de meilleur gain grâce aux valeurs de alpha et bêta obtenue lors de l'exploration d'autres branches.

5. Limitations du programme

Le programme de Négamax est encore plus portable que celui de A*. Il suffit comme précédemment de préciser les règles du jeu ainsi que les situations initiales et finales.

INSA Toulouse
135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Université Fédérale Toulouse Midi-Pyrénées

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE