COMPTE RENDU A* ET NEGAMAX

Pour:
ORGANISTA CALDERÓN José Daniel - *organist@insa-toulouse.fr,*

Enseignants:
Patrick Esquirol - *esquirol@insa-toulouse.fr*

Vendredi 01 Avril

Institut national des Sciences appliquées de Toulouse
GEI
INFORMATIQUE ET RÉSEAUX
2022

INSA TOULOUSE

SOMMIER

INSA TOULOUSE

ALGORITHME A* - APPLICATION AU TAQUIN

1) Familiarisation avec le problème du Taquin 3×3

a) Quelle clause Prolog permettrait de représenter la situation finale du Taquin 4x4 ?
Peut être représenté pour une matrix 4 x 4

```
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).
```

- **initial_state(Ini):** Laisse le variable Ini comme la Situation initiale (une matrice 3x3),
- **nth1(L,Ini,Ligne):** l'indice (L) de tous les Lignes (Ligne) dans l'état initial (Ini),
- **nth1(C,Ligne, d):** l'indice (C) de l'élément (d) dans le (Ligne),
  Donc, il donnera les coordonnées (**L,C**) de l'élément **d** dans le initial state **Ini**.

```
final_state(Fin), nth1(3,Fin,Ligne), nth1(2,Ligne,P)
```

- **final_state(Fin):** Laisse le variable Fin comme la Situation final (une matrice 3x3),
- **nth1(3,Fin,Ligne):** l'indice (3) de tous les Lignes (Ligne) dans l'état final (Fin),
- **nth1(2,Ligne,P):** l'indice (2) de l'élément (P) dans le (Ligne),
  Donc, il donnerait l'élément **P** dans les coordonnées 3,2 de l'état final **Fin**.

c) Quelle requête Prolog permettrait de savoir si une pièce donnée P (ex : a) est bien placée dans U0 (par rapport à F) ?

```
final_state(Fin), nth1(1, Fin, L), nth1(1, L, a).
```

d) Quelle requête permet de trouver une situation suivante de l'état initial du Taquin 3×3 (3 sont possibles) ?
S2 est la situation suivante de l'état initial Ini après faire le rule R.

```
initial_state(Ini), rule(R, 1, Ini, S2).
```

e) Quelle requête permet d'avoir ces 3 réponses regroupées dans une liste ?
L, c'est la liste ou se trouve les réponses de les situation suivantes de l'état initial Ini.

```
initial_state(Ini), findall(S2,rule(R, 1, Ini, S2),L).
```

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 ?
L, c'est la liste ou se trouve tous les couples [A, S] tels que S est la situation qui résulte de l'action A en l'état initial Ini.

```
initial_state(Ini), findall([A,S], rule(A, 1, Ini, S),L).
```

2) Développement des 2 heuristiques

a) L'heuristique du nombre de pièces mal placées
le prédicat malplace(P,U,F) qui est vrai si les coordonnées de P dans U et dans F sont différentes.

```
malplace(P,U,F) :-
      coordonnees1(P, U, X, Y),
      coordonnees1(P1, F, X, Y),
      P \= vide,
      P1 \= P.
```

Détermine toutes les pièces mal placées et les compte

```
heuristique1(U, H) :-
      final_state(Fin),
      findall(P,malplace(P,U, Fin),L),
      length(L, H).
```

On peut tester avec les requêtes suivants

```
initial_state(Ini), heuristique1(Ini, H).

final_state(Fin), heuristique1(Fin, H).
```

b) L'heuristique basée sur la distance de Manhattan.
distance de Manhattan (DT) entre sa position courante (U) et sa position dans l'état final (F).

```
dm([],_,_, 0).

dm([P|R],U, F, DT) :-
      coordonnees([L1,C1], U, P),
      coordonnees([L2,C2], F, P),
      dm(R, U, F, D),
      DT is (abs(L1-L2) + abs(C1-C2)) + D.
```

Cette heuristique évalue pour chaque pièce la distance minimale à parcourir pour rejoindre sa position finale.

```
heuristique2(U, H) :-
      final_state(Fin),
      findall(P,malplace(P,U, Fin),L),
      dm(L,U, Fin, H).
```

On peut tester avec les requêtes suivants

```
initial_state(Ini), heuristique2(Ini, H).

final_state(Fin), heuristique2(Fin, H).
```

3) Implémentation de A*

a) expand(+Current_State, ?[F, H, G], ?List)

les successeurs S de l'etat U en forme de List L

```
expand(U, [_,_,G], L) :-
      findall([S, [Fs, Hs, Gs], U, Action],
      (rule(Action, Cost, U, S),
      heuristique(S, Hs),
      Gs is G + Cost,
      Fs is Hs + Gs),
      L).
```

Pour tester

```
testSucceseurs.
```

b) loop_successors(+List, +Pf, ?PfResultat, +Pu, ?PuResultat, +Q).

si S est connu dans Q alors oublier cet état (S a déjà été développé).

```
loop_successors([[S,_,_,_]|Tail], Pf, Pfn, Pu, Pun, Q) :-
      belongs([S,_,_,_], Q),
      loop_successors(Tail, Pf, Pfn, Pu, Pun, Q).
```

Pour tester.

```
testSdansQ.
```

si S est connu dans Pu alors garder le terme associé à la meilleure évaluation

```
loop_successors([[S,Val,_,_]|Tail], Pf, Pfn, Pu, Pun, Q) :-
    \+belongs([S,_,_,_], Q),
    belongs([S,Val1,_,_], Pu),
    ( compare_values(Val, Val1) ->
       suppress_min([Val1, S], Pf, Pf1),
       suppress([S, Val1, _, _], Pu, Pu1),
       insert([Val, S], Pf1, Pf2),
       insert([S, Val, _, _], Pu1, Pu2),
       loop_successor(Tail, Pf2, Pfn, Pu2, Pun, Q)
    ;
       loop_successor(Tail, Pf, Pfn, Pu, Pun, Q)
    ).
```

S est une situation nouvelle

```
loop_successors([[S, [F, H, G], Pere, Action]|Tail], Pf,
Pfn, Pu, Pun, Q) :-
      \+belongs([S,_,_,_], Q),
      \+belongs([S,_,_,_], Pu),
      insert([[F,H,G], S], Pf, Pf1),
      insert([S, [F, H, G], Pere, Action], Pu, Pu1),
      loop_successors(Tail, Pf1, Pfn, Pu1, Pun, Q).
```

Pour tester ce deux cas on peut voir le premier itération d'aétoile

```
testFirstIt.
```

c) Analyse expérimentale

i) quelle taille de séquences optimales (entre 2 et 30 actions) peut-on générer avec chaque heuristique (H1, H2) ?

Taille	Temps H1	Temps H2
2	0.01942 ms	0.00723 ms.
5	0.002908 ms	0.00370 ms.
10	0.015903 ms	0.00874 ms.
20	stack overflow	0.04486 ms
30	stack overflow	stack overflow

ii) Quelle longueur de séquence peut-on envisager de résoudre pour le Taquin 4x4 ?
En tenant compte du temps de calcul pour résoudre le taquin 3x3 et du fait que l'ajout d'une autre ligne, d'une autre colonne consommerait plus de temps de développement pour chaque branche, cela pourrait être entre 10 et 15 coups.

iii) A* Trouve-t-il la solution pour la situation initiale suivante ?
A* ne trouve pas de solution pour la situation initiale donnée, l'algorithme développerait toutes les branches de chaque état possible, ce qui consommerait la mémoire de l'ordinateur.

iv) Quelle représentation de l'état du Rubik's Cube et quel type d'action proposeriez-vous si vous vouliez appliquer A*?
Il pourrait être représenté sous la forme d'une matrice 6x9, où chaque ligne contiendrait la représentation d'une face de le cube avec les 9 configurations de couleurs respectives. Les actions seraient les tours possibles d'un des axes.

INSA TOULOUSE

ALGO MINMAX - APPLICATION AU TICTACTOE

1. Familiarisation avec le problème du Tic Tac Toe 3×3

a. Quelle interprétation donnez-vous aux requêtes suivantes :

```
situation_initiale(S), joueur_initial(J).
```

S est la situation initiale une grille 3x3 empty
J c'est le joueur initial soit x soit o qui commence le partie.

```
situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o)
```

C'est la situation initiale **S** après le premier coup du joueur "**o**" en le coordonné (3,2) de la grille.

b. alignement(Ali, Matrice)
On peut tester la requête suivante.

```
M = [[a,b,c], [d,e,f], [g,h,i]], alignement(Ali, M).
```

c. possible(Ali, Joueur)
On peut tester les requêtes suivantes.

```
A=[_,_,_], possible(A,x).
A=[x,_,x], possible(A,x)
A=[_,o,x], possible(A,x).
```

Soit le partie1 le suivante

```
partie1([ [x,o,o],
          [o,x,o],
          [o,o,x] ]).
```

On peut tester alignement_gagnant(A, J) et alignement_perdant(A, J)

d. alignement_gagnant(A, J)

```
test1 :- partie1(M), alignement(Ali, M),
alignement_gagnant(Ali, x).
test2 :- partie1(M), alignement(Ali, M),
alignement_gagnant(Ali, o).
```

e. alignement_perdant(A, J)

```
test3 :- partie1(M), alignement(Ali, M),
alignement_perdant(Ali, o).
test4 :- partie1(M), alignement(Ali, M),
alignement_perdant(Ali, x).
```

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

a. heuristique(Joueur,Sit,H) qui retourne la valeur de l'heuristique pour le joueur J dans une situation donnée. Tests unitaires
Soit le partie2 le suivante

```
partie2([ [x,_,_],
          [o,_,_],
          [x,_,_] ]).
```

on peut tester l'heuristique de la manière suivante

```
test5 :- situation_initiale(M), heuristique(x,M,H),
writeln(H).
test6 :- partie2(M), heuristique(x,M,H), writeln(H).
```

3. Développement de l'algorithme Negamax

a. 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.

```
situation_initiale(S), joueur_initial(J),
findall([C, S],successeur(J,S,C),L).
```

b. Test unitaires
successeurs(+J,+Etat, ?Succ)
retourne la liste des couples [Coup, Etat_Suivant] pour un joueur donne dans une situation donnée.

```
situation_initiale(S), joueur_initial(J),
successeurs(J,S,Succ).

partie1(S), joueur_initial(J), successeurs(J,S,Succ).

partie2(S), joueur_initial(J), successeurs(J,S,Succ).
```

meilleur(+Liste_de_Couples, ?Meilleur_Couple)
Le couple de L. Entre X et Y on garde celui qui a la petite valeur de V.

```
meilleur([[Coup1, 1]], BestC).

meilleur([[Coup1, 5], [Coup2, 2], [Coup3, 8]], BestC).
```

4. Expérimentation et extensions

a. 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

```
testNegamax1.
[[2,2],4]

testNegamax2.
[[2,2],1]

testNegamax3.
[[2,2],3]

testNegamax4.
[[2,2],1]

testNegamax5.
[[2,2],3]

testNegamax6.
[[2,2],1]

testNegamax7.
[[2,2],2]

testNegamax8.
[[1,1],10000]

testNegamax9.
ERROR: Stack limit (1.0Gb) exceeded
```

b. Comment ne pas développer inutilement des situations symétriques de situations déjà développées ?
Si les états déjà développés sont mémorisés, il n'y aurait pas besoin de les recalculer.

c. Que faut-il reprendre pour passer au jeu du puissance 4 ?
Les fonctions doivent être modifiées pour accepter unjeu du puissance 4 comme l'utilisation d'une matrice 4x4, le calcul de l'heuristique, des successeurs, des alignements possibles ainsi que des alignements gagnants et perdants.

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

L'algorithme peut être amélioré en utilisant l'élagage alpha-βeta. Cette méthode consiste à utiliser deux variables alpha et bêta qui sont transmises par les nœuds pour connaître les états qui ne doivent pas être développés. L'exploration des branches Min (resp. Max) d'un nœud Max (resp. Min)
u s'arrête dès que la valeur α (resp. β) qu'il remonte d'un de ses fils dépasse la valeur β (resp. α) qu'il a reçue de son père. Dans ce cas, on sait qu'il n'existe pas de meilleure évaluation que celle trouvée précédemment, il n'est donc pas nécessaire d'explorer cette branche.