Résolution de problèmes

Modélisation, algorithmes A*, min-max et alpha-beta

Rappel

Les algorithmes de recherche sont un mécanisme général de résolution de problèmes très utilisé en intelligence artificielle.

- Le problème est décrit par un espace d'états contenant :
  - Un état initial
  - Un ou plusieurs états finaux (évalués par une fonction test-but)
  - Un ensemble d'actions possibles permettant de transiter d'un état à un autre pour un coût donné.
- Une résolution correspond à une suite d'actions (appelée chemin) qui mène de l'état initial à l'état final.
- La meilleure solution est la résolution la moins coûteuse.

1 La traversée du pont

Quatre personnes doivent traverser un pont en 17 minutes. Chacune d'entre elles marche à une vitesse maximale donnée. Une personne peut traverser le pont en 1 minute, une autre en 2 minutes, une autre en 5 minutes et la dernière en 10 minutes.
Il est impossible de traverser le pont sans torche, et ces quatre personnes n'ont en tout qu'une torche. Le pont ne peut supporter que le poids de 2 personnes.

On cherche à déterminer l'ordre dans lequel ces quatre personnes doivent traverser.
Q. 1 : Choisir un formalisme permettant de représenter les états possibles.
Q. 2 : Identifier, dans ce formalisme, l'état initial et la fonction test-but.
Q. 3 : Définir les opérateurs avec leur coût.
Q. 4 : Quel algorithme trouvera la solution optimale.

2 Problème du taquin

Ce jeu est composé de 8 pièces numérotées et d'une case vide que l'on peut déplacer. Le but est partant d'une configuration mélangée de retrouver la configuration *but* illustrée par la figure 1.

2.1 Formalisation

Q. 1 : Choisir un formalisme permettant de représenter les états possibles.
Q. 2 : Combien y a-t-il d'états possibles ?
Q. 3 : Identifier, dans ce formalisme, l'état initial et la fonction test-but.

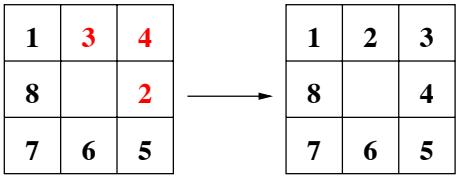

FIGURE 1 – Un état initial (à gauche) et l'**état-but** (à droite) du jeu du taquin

2.2 Application de l'algorithme A*

A* est un algorithme de recherche heuristique (voir Algorithme 1), il doit donc être guidé par une fonction heuristique qui estime la qualité d'un nœud. Pour un nœud n, on utilise une fonction de la forme $f(n) = g(n) + h(n)$ où g est le coût du chemin depuis l'état initial jusqu'au nœud et h est une estimation du coût du chemin entre le nœud et l'état final.

Q. 1 : Proposer au moins trois heuristiques pour résoudre le jeu de taquin avec un algorithme de type A*.

Q. 2 : Appliquer l'algorithme A* avec la meilleure heuristique que vous ayez trouvée sur le taquin.

Algorithme 1 A (A* si h est admissible)

s_0 est l'état initial
OUVERTS est la liste des nœuds à étudier
FERMES est la liste des nœuds étudiés
la fonction tête(OUVERTS) rend le premier élément de la liste OUVERTS
la fonction chemin(s) appelle chemin(pere(s)) si $s \neq s_0$ puis rend s

OUVERTS $\leftarrow \{s_0\}$
FERMES $\leftarrow \emptyset$
$g(s_0) \leftarrow 0$
s $\leftarrow s_0$
tant que OUVERTS $\neq \emptyset$ et tête(OUVERTS) non terminal **faire**
  Supprimer s de OUVERTS et le mettre dans FERMES
  pour tout nœud s' successeur de s **faire**
    si $s' \notin (OUVERTS \cup FERMES)$ *ou* $g(s') > g(s) + cout(s, s')$ **alors**
      $g(s') \leftarrow g(s) + cout(s, s')$
      $f(s') \leftarrow g(s') + h(s')$
      $pere(s') \leftarrow s$
      Ranger s' dans OUVERTS dans l'ordre f croissant
      si s' $\in$ FERMES **alors**
        FERMES $\leftarrow$ FERMES $\setminus$ {s'}
      fin si
    fin si
  fin pour
  si $OUVERTS \neq \emptyset$ **alors**
    $s \leftarrow tete(OUVERTS)$
  fin si
fin tant que
si $OUVERTS = \emptyset$ **alors**
  le problème n'admet pas de solution
sinon
  le chemin(s) est solution
fin si

3 Algorithme de jeux

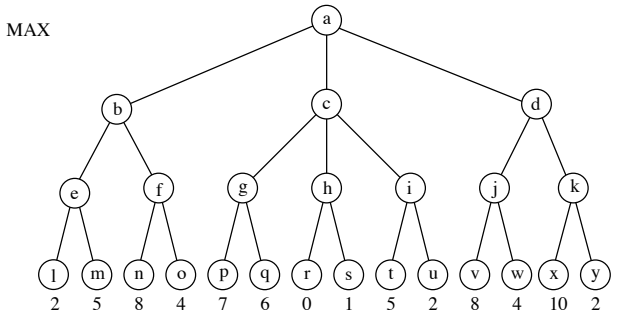

FIGURE 2 – Arbre de jeu. La racine est un nœud maximisant. Chaque lettre identifie un nœud. Chaque chiffre correspond à l'évaluation d'une feuille.

Q. 1 : Appliquer l'algorithme α-β à l'arbre de jeu donné par la figure 2.
a) Supposer d'abord que les fils des nœuds sont parcourus de gauche à droite et indiquer tous les nœuds qui ne sont pas considérés par l'algorithme.
b) Faire la même chose en supposant que les fils sont parcourus de droite à gauche.
Q. 2 : Commenter

Algorithme 2 Min-Max

```
int minmax (Node s, int depth, int limit)
Vector v = new Vector() ;
si isTerminal(s) || depth == limit alors
  // en fin de partie ou à la profondeur max, renvoyer l'évaluation de l'état
  return GainEvaluation(s) ;
sinon
  // appliquer minmax sur les fils de s et mémoriser leurs valeurs
  tant que s.hasMoreSuccessors() faire
    v.addElement(minmax(s.getNextSuccessor(),depth+1,limit)) ;
  fin tant que
  si isComputerTurn(s) alors
    // tour du programme (AMI) : renvoyer le max des éval. des successeurs
    return maxOf(v) ;
  sinon
    // tour de l'adversaire (ENNEMI) : renvoyer le min des éval. des successeurs
    return minOf(v) ;
  fin si
fin si
```

limit correspond à la profondeur limite de recherche.
Elle réduit l'explosion combinatoire en diminuant l'espace de recherche.

Algorithme 3 Alpha-Beta

```
alphabeta(Node s, int alpha, int beta, int depth, int limit)
si isTerminal(s) || depth == limit alors
  // en fin de partie ou à la profondeur max, renvoyer l'évaluation de l'état
  return GainEvaluation(s) ;
sinon
  // appliquer alpha-beta sur les fils de s et calculer leurs valeurs
  si isComputerTurn(s) alors
    // tour du programme (AMI) : m-à-j de alpha (max)
    pour tout successeur s' de s faire
      val = alphabeta(s', alpha, beta, depth+1, limit) ;
      alpha = max(alpha, val) ;
      si alpha ≥ beta alors
        break ; // coupe
      fin si
    fin pour
    return alpha ;
  sinon
    // tour de l'adversaire (ENNEMI) : m-à-j de beta (min)
    pour tout successeur s' de s faire faire
      val = alphabeta(s', alpha, beta, depth+1, limit) ;
      beta = min(beta, val) ;
      si alpha ≥ beta alors
        break ; // coupe
      fin si
    fin pour
    return beta ;
  fin si
fin si
```