DM1 : IA SOLVE

Auguste Gardette

20 Octobre 2022

1 Recherche d'un chemin optimal par l'algorithme A*

Soit le graphe correspondant à la figure 1 dans lequel les nombres sur les arcs indiquent le coût associé à cet arc, et les nombres donnés pour chaque nœud indiquent la valeur retourné par l'heuristique pour ce nœud et le but « G ».

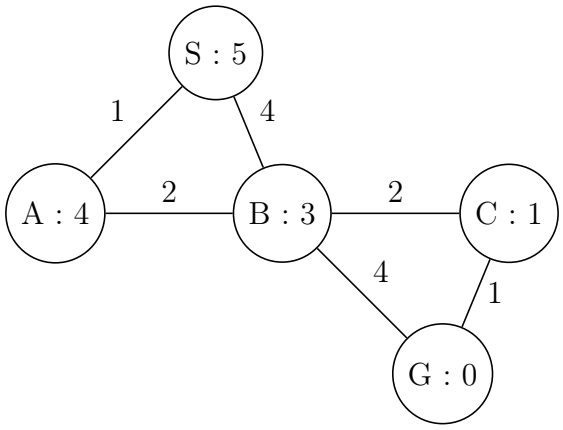

Figure 1 – Graphe.

Donnez l'ordre d'exploration des nœuds développés par une recherche en meilleur d'abord commençant par le nœud « S ». Montrez l'évolution des listes OUVERT et FERMÉ.

Tour	Ouvert	Fermé
0	S(0 + 5)	
1	A(1 + 4) ; B(4 + 3)	S(5)
2	B(3 + 3) par A ; B(7) par S	S(5) ; A(5)
3	C(5 + 1) ; G(7 + 0)	S(5) ; A(5) ; B(6) par A
4	G(6 + 0) par C G(7) par B	S(5) ; A(5) ; B(6) par A ; C(6)
4		S(5) ; A(5) ; B(6) par A ; C(6) ; G(6) par C

Le Chemin optimal retourné par l'algorithme du meilleur d'abord est :

$$S \rightarrow A \rightarrow B \rightarrow C \rightarrow G$$

2 Optimalité de l'algorithme A*

Prouvez que l'algorithme A* retourne un chemin de coût optimal si la fonction heuristique h(n) est admissible.

Supposons un chemin optimal C^* mais que notre algorithme retourne $C > C^*$. Quelque soit C, on est nécessairement passé par un nœud optimal.

On aurait donc :	$f(n) > C^*$
Par définition :	$f(n) = g(n) + h(n)$
n sur le chemin optimal :	$f(n) = g^*(n) + h(n)$
h(n) admissible :	$f(n) \leq g^*(n) + h^*(n)$
Contradiction :	$f(n) \leq C^*(n)$

Donc si f(n) est plus faible en coût que $f^*(n)$, $f^*(n)$ ne sera jamais exploré car non-optimal. h(n) admissible retourne donc nécessairement le chemin optimal.

3 Transformation de A* pour trouver un chemin le plus long

L'algorithme A* permet de trouver un plus court chemin dans un graphe. On va chercher ici comment le transformer pour qu'il permette de trouver un plus long chemin.

1. Que se passe-t-il si on n'utilise pas de fonction heuristique ?

Si on n'utilise pas de fonction heuristique, on se retrouve sur une recherche non informée. On peut néanmoins trouver le meilleur chemin avec un algorithme de coût uniforme, largeur d'abord ou profondeur itérative mais la compléxité en temps et en espace sera plus lourd. (Nombre de nœuds dévellopés et en mémoire augmente).

2. Quelle doit être la propriété d'une fonction heuristique pour que le chemin retourné soit un plus long chemin ?

On peut adapter la propriété $h(n) \leq h^*(n)$ pour sélectionner le chemin le plus long, tel que $h^*(n)$

$$\forall\ n_1, n_2, h(n_1) \geq h(n_2)$$

On obtient donc :

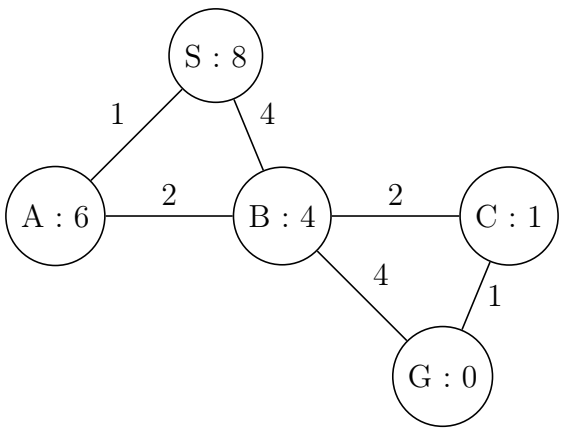

Figure 2 – Graphe adapté

3. Donner l'algorithme retournant un plus long chemin.

Pour chaque tour, on choisit dans notre liste ouverte le nœud n donc la fonction f(n) est la plus coûteuse et dont il existe au moins un nœuds voisins pouvant encore être développés jusqu'à atteindre l'objectif.

Algorithme 1 : algorithme d'un chemin le plus long

```
OUVERT ← état initial ; FERME ← ∅ ; Succès ← Faux;
while OUVERT ≠ ∅ et Succès = Faux do
    Etape 1 : Calculer les f(n) des noeuds ouverts;
    Etape 2 : Sélection noeud n avec f(n) max parmis les noeuds ouverts ;
    si noeud choisi n = objectif alors
            Succès = vrai ;
            ajouter noeud n à FERMER ;
            retourner FERMER ;
    sinon
            ajouter noeud n à FERMER ;
            noeud parent = noeud n ;
            ajouter noeuds enfants ayant encore un noeud voisin ouvert de noeud n à
        OUVERT ;
    fin
end
```

4. Le tester sur le petit graphe de la figure 1 et montrer l'ordre d'exploration des nœuds.

Tour	Ouvert	Fermé
0	S(0 + 8)	
1	A(1 + 6) ; B(4 + 4)	S(8)
2	G(8 + 0) ; C(6 + 1) ; A(1 + 6) par S ; A(6 + 6) par B*	S(8) ; B(8)
3		A(8) ; B(8) ; G(8)

* Au tour 2, le nœud A n'est pas admissible par il n'y a pas de noeud pouvant être développé après et que ce n'est pas un noeud objectif.

Le Chemin retourné par l'algorithme du plus long d'abord est :

$$S \rightarrow B \rightarrow G$$

4 Jeu à trois joueurs

Comment pourrait-on modifier l'algorithme minimax pour qu'il soit utilisable dans le cas d'un jeu à information parfaite à trois joueurs ?
Par exemple, considérons un jeu à trois joueurs nommés 0, 1 et 2 (voir figure 2). On va supposer que la fonction d'évaluation retourne un triplet de valeurs indiquant l'évaluation de la position pour les joueurs 0, 1 et 2. Par exemple, le triplet (6, 2, 4) indique que cette position est très

désirable pour le joueur 0, nettement moins pour le joueur 1 et moyennement pour le joueur 2.

1. Compléter l'arbre de jeu de la figure 2 en remplissant les triplets jusqu'à la racine.

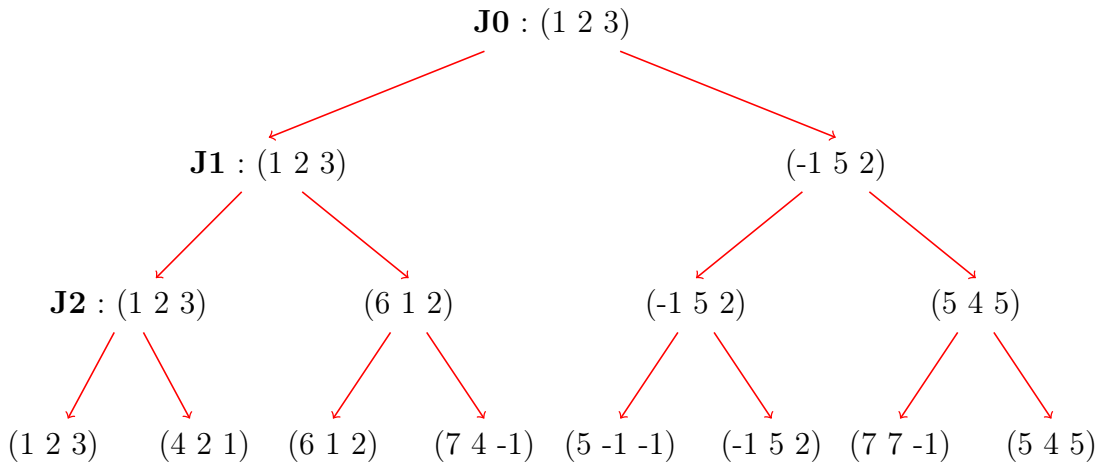

Figure 2 – Les trois premiers coups d'un jeu à trois joueurs.

2. Ré-écrire le programme MinMax pour qu'il joue correctement sur ce type d'arbre.

Algorithmes de remontée des étiquettes numériques :
- Si J0 : Max des valeurs pour la position 0 dans les triplets des successeurs.
- Si J1 : Max des valeurs pour la position 1 dans les triplets des successeurs.
- Si J2 : Max des valeurs pour la position 2 dans les triplets des successeurs.

Généralisation du programme MinMax pour ce type de jeu :
- Soit n joueurs : Pour Ji : Max des valeurs pour la position i dans les n-uplets des successeurs.

Dans le cas où les valeurs à la position i dans les n-uplets des succeurs pour un joueur i sont identiques, on pourrait rajouter dans notre algortihme la règle suvante :
- Ji cherche à minimiser la somme total du gain de ses adversaires dans le cas d'une égalités des valeurs à la position i du n-uplet.

Mais l'objectif d'un jeu ne doit pas nécessairement de faire perdre le plus possible ses adversaires... surtout s'il y a plusieur parties.