TD - Introduction à l'Intelligence Artificielle n° 2
(Correction)

Algorithmes pour les jeux (2)

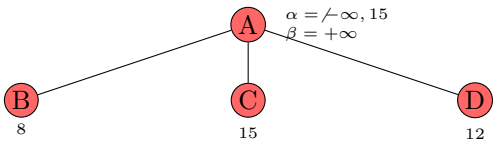
Exercice 1 Iterative Deepening $\alpha\beta$

On considère l'algorithme $\alpha\beta$, dans sa version où il exécute une recherche à profondeur incrémentale. On suppose que lors de la première exploration de chaque nouveau niveau de l'arbre, les différents fils d'un même nœud sont explorés suivant l'ordre lexicographique.

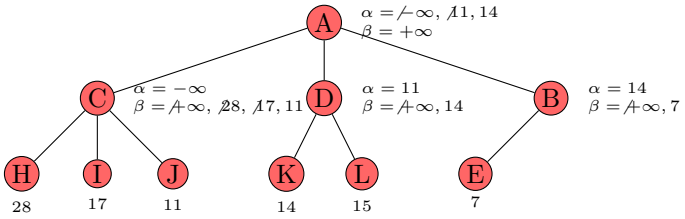
1. Détailler le fonctionnement de cet algorithme sur le graphe décrit implicitement dans le tableau suivant :

noeud n	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
$h(n)$	13	8	15	12	7	55	19	28	17	11	14	15	10	11	
$succ(n)$	B C D	E F G	H I J	K L	M N	O P	Q R S	T U	V W	X Y	Z A1	A2 A3	a b	c d	
noeud n	O	P	Q	R	S	T	U	V	W	X	Y	Z	A1	A2	A3
$h(n)$	20	13	10	11	6	14	30	18	19	12	10	14	12	15	8
$succ(n)$	e	f	g	h i	j k	l	m n	o	p	q r s	t u	v w	x1 x2	y1 y2	z1 z 2
noeud n	a	b	c	d	e	f	g	h	i	j	k	l	m	n	
$h(n)$	30	12	8	15	19	3	7	17	12	10	6	14	12	8	
noeud n	o	p	q	r	s	t	u	v	w	x1	x2	y1	y2	z1	z2
$h(n)$	10	8	12	8	15	6	7	14	19	6	14	15	16	12	10

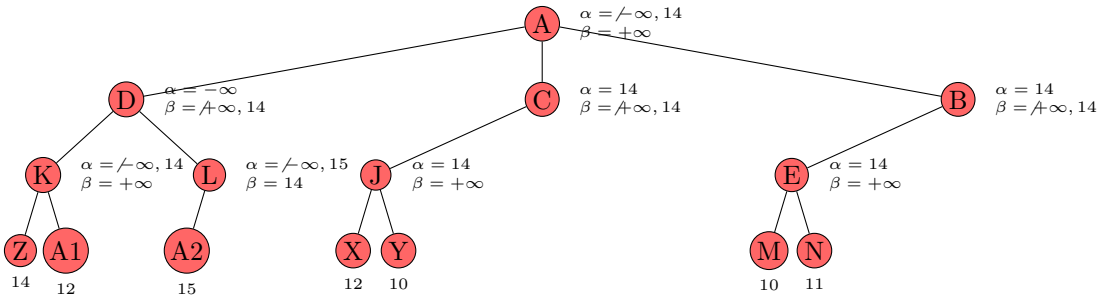
Correction :
ProfMax = 1 :



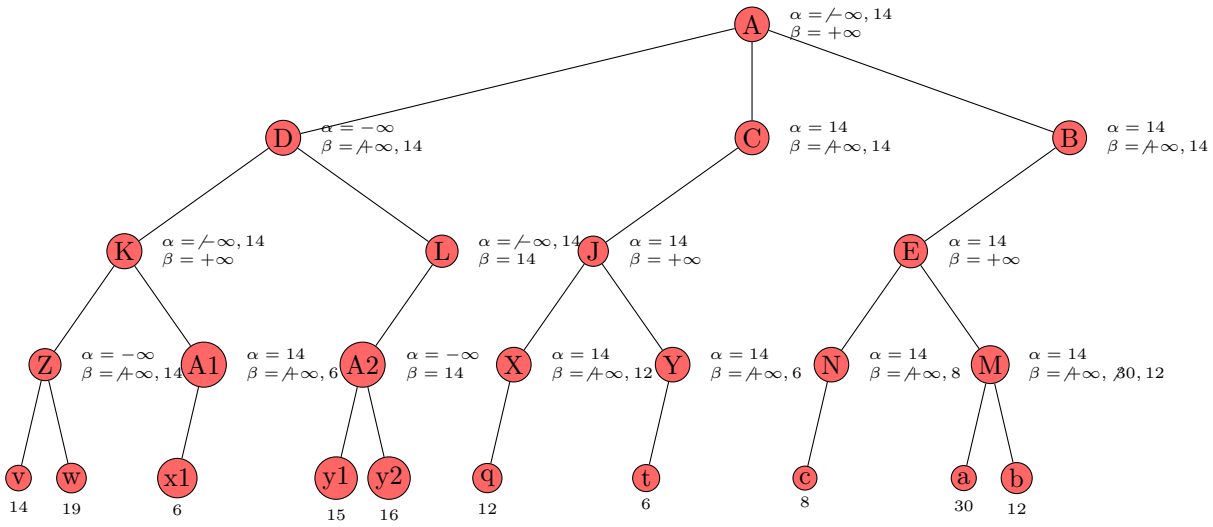
ProfMax = 2 :



ProfMax = 3 :



ProfMax = 4 :



2. Comment gérer en pratique le réordonnancement des fils tout en limitant la consommation mémoire ?

Correction : [Laurent S.]

Pour gérer en pratique le réordonnancement des fils, il ne faut pas garder tout l'arbre en mémoire sinon cela prendrait trop de temps à trier à chaque fois, et surtout le traitement en mémoire de tout l'arbre prendrait au final trop de place...

On peut utiliser une table de hashage avec oubli en cas de collision pour stocker les noeuds déjà visités et leur valeur heuristique associée. Cela permet de maitriser la mémoire allouée.

Mieux encore, on peut ne stocker que la valeur de hashage de l'état considéré, sans garder l'état lui-même. Ainsi on gagne énormément de place (les plateaux ne sont pas en mémoire, seul un entier long l'est). D'accord, on peut se planter en pensant revoir un noeud déjà vu, mais ce n'est pas grave dans la mesure où la valeur heuristique du noeud ne sert qu'à ordonner les fils. Cela ne remet pas en cause la valeur minimax finale, qui reste, elle toujours bonne.

Autre optimisation possible : ne pas garder la valeur heuristique de chaque noeud (ce qui oblige à ordonner tous les fils à chaque fois), mais garder les 2 ou 3 meilleurs coups pour chaque noeud... Ainsi on n'a plus rien à faire pour ordonner les fils... Et les fils 4, 5, ... ben on les explore au hasard :).

Tout le problème vient maintenant de pouvoir associer une valeur de hashage facilement à un état. Ce sera vu en cours.

En gros : A chaque position de chaque pièce sur le plateau on associe un entier aléatoire, mais stocké en mémoire. La valeur de hashage du plateau est le xor logique de toutes les valeurs associées aux positions de chaque pièces. Ainsi, un mouvement sur le plateau (l'exemple est pris sur les échecs) se fait en faisant très peu d'opération (un xor pour enlever la piece et un nouveau xor pour la mettre à son arrivée).