Dimitri Charneux
Rémi Kroll

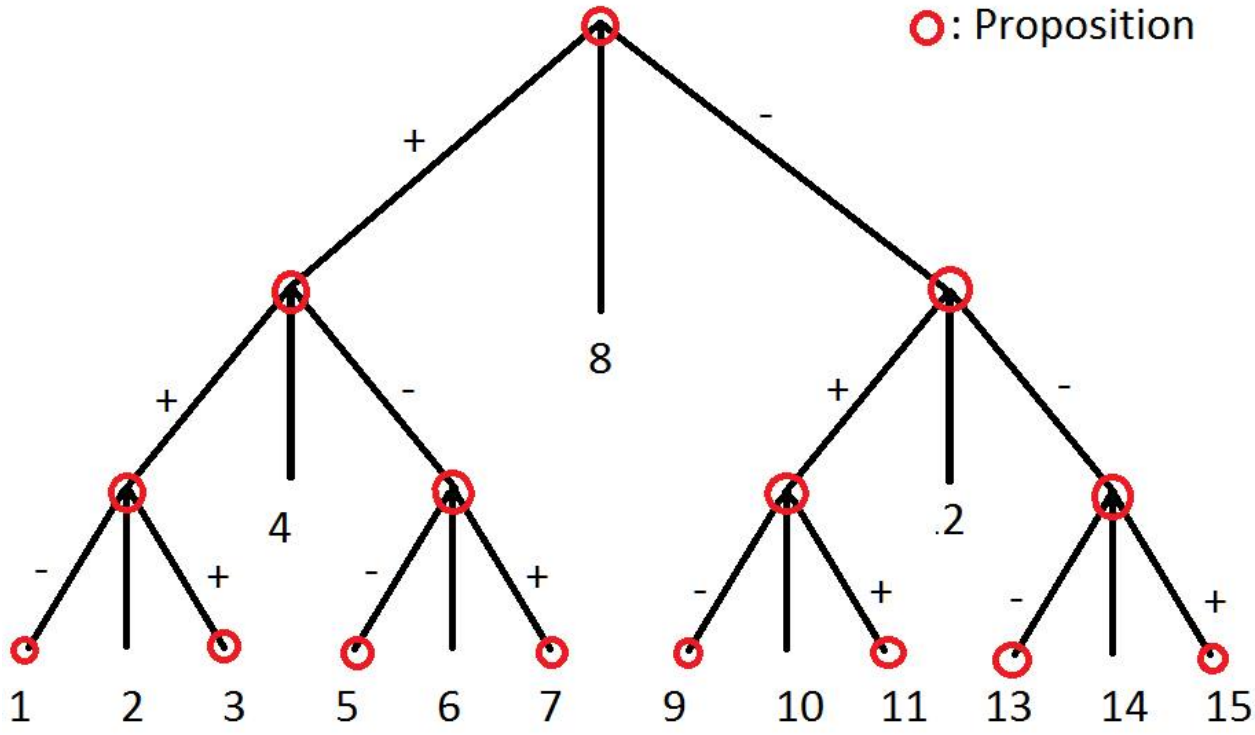
Rendu tp: Arbres de décision

Introduction

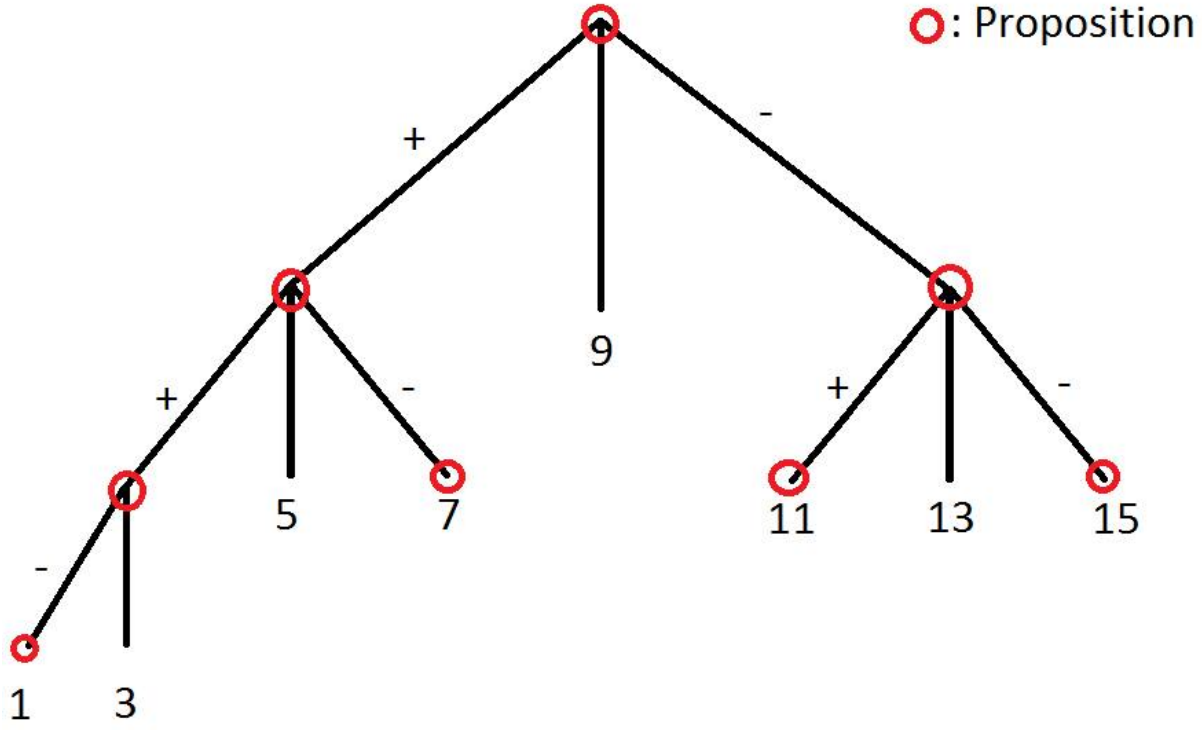
Durant ce TP, nous allons voir comment utiliser efficacement les arbres de décisions. Pour ce faire, nous allons prendre une variante du jeu du pendu où l'ordinateur va devoir deviner un mot dans un dictionnaire en nous demandant si une lettre appartient ou non au mot choisi.

1. Question de bon sens

Pour la première question, nous trouvons que N vaut 15. En effet, comme on peut le voir sur cette arbre, à chaque proposition, le nombre proposé peut être soit plus grand, soit plus petit, soit égal. Au bout de quatre propositions, le joueur peut deviner à coup sûr le nombre exacte parmi 15 nombres.



En prenant en compte A qui dit qu'il faudra exactement quatre propositions, B peut conclure que le nombre qu'il cherche est parmi 1, 3, 5, 7, 9, 11, 13, 15 car ce sont les seuls nombres qui demande exactement 4 coups pour être trouvé. Grâce à cela, on se retrouve avec le graphe suivant où on peut gagner en moins de quatre coups dans 7 cas sur 8. B est donc presque sûr de gagner avant les quatre coups.



2. Variante du jeu du pendu

Dans cette partie, nous allons voir les étapes qui nous ont conduite à la réalisation de cette variante du pendu.

Si le programme est sûr de trouver en au plus p questions, nous pouvons déduire la relation suivante entre p et n : log2(n) >= p, n étant le nombre de mots dans le dictionnaire.

Pour commencer à réaliser ce jeu, nous avons utilisé la fonction ci-dessous qui nous permet de transformer une lettre de l'alphabet en un entier compris entre 1 et 26.

```
maFonction ← function(x) { strtoi(charToRaw(x),16L)-96 }
```

Ensuite, nous avons créé une matrice de taille (26, n) initialisée à 0 où pour chaque mots, nous mettons la valeur 1 dans les cases correspondant au lettres que contient le mot. Par exemple, pour le mot 'oiseau', la colonne serait la suivante : '10001000100000100010100000'. Le code nous permettant de créer cette matrice est le suivant :

```
mat = matrix(rep(0,26*n),nrow=26, ncol=n);
for (i in 1:n)
{
  c = str2int(noms[i]);
  mat[c,i] <- 1;
}
```

Ensuite, nous avons calculer l'entropie de chaque lettre pour déterminer laquelle nous

permettra de séparer au mieux la liste des mots. Une fois l'entropie trouver, l'ordinateur va choisir la lettre ayant l'entropie la plus élevée pour jouer.

Une fois ces fonctions faites, nous avons codé la fonction qui se trouve en annexe servant à faire tourner le jeu. La fonction jouer se déroule de la manière suivantes :

Tant que la liste de mots contient plus d'un mot, nous allons calculer l'entropie des lettres. Nous allons prendre la lettre qui a l'entropie la plus grande puis séparer la liste en deux sous-listes, une avec les mots contenant la lettre, l'autre avec ceux ne la contenant pas. Nous allons demander au joueur si la lettre est dans le mot, si elle l'est, nous allons recommencer l'opération avec la liste des mots contenant la lettre, sinon, nous allons recommencer avec l'autre liste.

Il ne faut pas oublier dans ce code de stocker les lettres proposées pour ne pas les utiliser plusieurs fois et rester bloqué.

Quand la liste ne contient qu'un mot, nous pouvons retourner ce mot au joueur.

3. Arbre de décision de la variante du jeu du pendu

Nous avons ensuite réalisé l'arbre de décision de ce jeu. Pour ce faire, comme vous pouvez le voir en annexe, nous avons fait un algorithme similaire à celui du jeu. La différence étant qu'au lieu de continuer uniquement sur une moitié de la liste, nous continuons sur les deux moitiés pour calculer tout les cas possibles.

Voici un exemple de branche de l'arbre de décision :

```
avec E  ,avec A  ,avec O  ,avec R  ,avec I  ,sans N  ,sans U  ,sans C  ,avec L lamproie
```

Le mot « lamproie » est trouvé en prenant la liste des mots contenant un 'e', puis celle contenant un 'a', etc jusqu'à ce qu'il ne reste qu'un mot.

Conclusion

Durant ce TP, nous avons appris à utiliser des arbres de décisions pour résoudre des problèmes assez facilement. Ces arbres permettent également de voir les étapes qui nous ont permises de résoudre le problème posé.

Annexe

Code de la fonction jouer :

```
jouer <- function(){
  matrice = mat
  lettreDemande = c()
  cpt = 1
  while(length(matrice[1,]) > 1){
    ent = entropie(mat)
    for(var in lettreDemande){
      ent[var] = 0
    }

    lettre = which.max(ent)
    lettreDemande[cpt] = lettre
    cat("Le mot contient-il la lettre", int2str(lettre), "?")
    rep = readline()
    if(rep == "oui"){
      matrice = sousEnsembleAvec(matrice, lettre)
    } else {
      matrice = sousEnsembleSans(matrice, lettre)
    }
    cpt = cpt +1
  }
  if(length(matrice[1,]) == 1){
    mot = trouveLeMot(matrice)
    cat("le mot est", mot)
  } else {
    cat("mot non trouvé")
  }
}
```

Code pour créer l'arbre de décision :

```
lancementArbre <- function(){
  arbreDeDecision(mat, c(), 1, "")
}

arbreDeDecision <- function(matrice, lettreDemandeDecision, cptArbreDecision,
texte){
  if(length(matrice[1,]) == 1){
    mot <-  trouveLeMot(matrice)
    cat(texte, mot, "\n")
  } else if(length(matrice[1,]) > 1){
    ent <-  entropie(mat)
    for(var in lettreDemandeDecision){
      ent[var] = 0
    }

    lettre <-  which.max(ent)
    lettreDemandeDecision[cptArbreDecision] = lettre
    tmp <- cptArbreDecision
    cptArbreDecision <-  cptArbreDecision +1
    texteTmp <- paste(texte, " ,avec", int2str(lettre))
    arbreDeDecision(sousEnsembleAvec(matrice, lettre), lettreDemandeDecision,
cptArbreDecision, texteTmp)
    texteTmp <- paste(texte, " ,sans", int2str(lettre))
    arbreDeDecision(sousEnsembleSans(matrice, lettre), lettreDemandeDecision,
cptArbreDecision, texteTmp)
```

```
    }
}
```