

Programmation des Systèmes Experts

James L. Crowley

Deuxième Année ENSIMAG

Troisième Bimestre 2000/2001

Séance 6

14 mars 2001

Systèmes de Productions : CLIPS 6.0

Arbres de décisions	2
Classification des animaux	3
GRAPHSEARCH en CLIPS.....	7
Algorithme GRAPHSEARCH (Nilsson).....	13

Arbres de décisions

Les arbres de décision sont utiles pour les problèmes de classification.

Caractéristiques d'un problème de classification :

- 1) L'ensemble des réponses possibles est fini et connu d'avance.
(exemples : problème de diagnoses et de taxonomie.
- 2) L'espace de solutions est réduit par une série de tests.

Les arbres de décisions ne fonctionnent pas bien pour les problèmes de planification, ordonnancement, ou synthèses.

Ils fonctionnent mal s'il faut construire la solution partielle à chaque étape.

Un arbre de décisions est composé de noeuds et de branches.

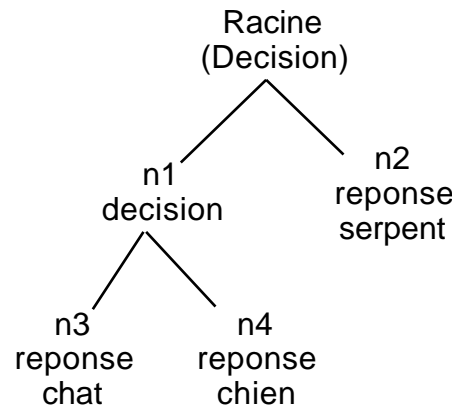
Les noeuds peuvent être les décisions ou les réponses.

Les ensembles de réponses possibles sont les feuilles de l'arbre.

Classification des animaux

Exemple d'un arbre de décisions étant un système pouvant "apprendre" à identifier les animaux :

L'arbre sera représenté par les noeuds du type décision ou réponse :



```

(deffacts arbre
  (noeud racine décision "Est-ce un animal de sang-chaud?" n1
  n2)
  (noeud n1 décision "Est-ce qu'il ronronne ?" n3 n4)
  (noeud n2 reponse serpent)
  (noeud n3 reponse chat)
  (noeud n4 reponse chien)
  )

```

```
;; vielle règle d'init
```

```

(defrule init
  (initial-fact)
=>
  (assert (noeud-actuel racine))
  )

```

:: règle à demander pour le noeud de décision

```
(defrule faire-décision
  ?N <- (noeud-actuel ?nom)
  (noeud ?nom decision ?q ?oui ?non)
=>
  (retract ?N)
  (format t "%s (oui ou non) " ?q)
  (bind ?réponse (read))
  (if (eq ?réponse oui)
      then (assert (noeud-actuel ?oui) )
      else (assert (noeud-actuel ?non))
  )
)
```

:: règle pour traiter les réponses

```
(defrule faire-réponse
  ?N <- (noeud-actuel ?nom)
  (noeud ?nom réponse ?r)
=>
  (printout t "Je pense que c'est un " ?r crlf)
  (printout t "c'est correct ? (oui ou non) ")
  (bind ?rep (read))
  (if (eq ?rep oui)
      then (assert (phase demande-encore))
           (retract ?N)
      else (assert (phase corrige-réponse))
  )
)
```

:: règle pour essayer encore

```
(defrule essaie-encore
  ?phase <- (phase demande-encore)
=>
  (retract ?phase)
  (printout t "essaie encore ? (oui ou non))
  (bind ?rep (read))
  (if (eq ?rep oui)
      then (assert (noeud-actuel racine))
      else (save-facts "animal.dat")
  )
)
```

```
;; règles pour apprendre une nouvelle réponse
```

```
(defrule corrige-réponse
  ?P <- (phase corrige-réponse)
  ?N <- (noeud-actuel ?nom)
  ?D <- (noeud ?nom réponse ?r)
=>
  (retract ?P ?N ?D) ;; demander la bonne réponse
  (printout t "quel est l'animal? ")
  (bind ?new (read))
  (printout t
    "quelle question poser pour distinguer " crlf)
  (printout t "un " ?new " d'un " ?r "? ")
  (bind ?question (readline))
  (bind ?newnode1 (gensym*))
  (bind ?newnode2 (gensym*))
  (assert (noeud ?newnode1 reponse ?new))
  (assert (noeud ?newnode2 reponse ?r))
  (assert
    (noeud ?nom decision ?question ?newnode1 ?newnode2))
  (assert (phase demande-encore))
)
```

```
;; Règles d'ouverture et de lecture du fichier animal.dat
```

```
(defrule init
  (initial-fact)
=>
  (assert (file =(open "animal.dat" data "r")))
)
```

```
;; ouverture et fermeture du fichier animal.dat
```

```
(defrule no-file
  ?f <- (file FALSE)
=>
  (retract ?f)
  (assert (noeud-actuel racine))
)
```

```
;; lecture du fichier
```

```
(defrule init-file
  ?f <- (file TRUE)
=>
  (bind ?in (readline data))
  (printout t ?in crlf)
  (if (eq ?in EOF) then (assert (eof))
      else
        (assert-string ?in)
        (retract ?f)
        (assert (file TRUE))
      )
  )
```

```
(defrule eof
  (declare (salience 30))
  ?f <- (file TRUE)
  ?eof <- (eof)
=>
  (retract ?f ?eof)
  (close data)
  (assert (noeud-actuel racine))
)
```

GRAPHSEARCH en CLIPS

Notre TD de Vendredi 23 mars en salle E106 - E108 a 11h15

L'algorithme GRAPHSEARCH pour un réseau d'emplacements peut être codé en CLIPS.

Dans cet exercice, nous allons représenter le point de départ et le but par les "structures" définies par des "templates" avec les attributs nom, x et y. L'arbre de recherche sera composé d'objets de type Noeud dont les attributs sont :

- nom : Le nom de l'emplacement représenté par ce noeud.
- status : l'état du noeud {Open, Closed, Active, etc}
- père : Le noeud qui est au-dessus dans l'arbre de recherche
- f : Le coût d'un chemin qui passe par cet emplacement.
- g : Le coût dans le graphe entre le départ et cet emplacement.
- h : Le coût estimé entre cet emplacement et le but.
- x, y : Les coordonnées cartésien de l'emplacement.

Nous aurons besoins déclarer une fonction externe "distance" avec l'expression :

```
(deffunction distance (?x1 ?y1 ?x2 ?y2)
  (bind ?dx (- ?x1 ?x2))
  (bind ?dy (- ?y1 ?y2))
  (sqrt (+ (* ?dx ?dx) (* ?dy ?dy)))
)
```

La fonction distance peut être appelée dans une règle comme suit:

```
=>
  (modify ?E (h =( distance ?x1 ?y1 ?x2 ?y2)))
```

Pour chacune des règles que vous écrivez, donnez un nom à la règle de forme Rn où n est soit un entier, soit le nom d'une paire d'emplacements. Exemples, R1, R2

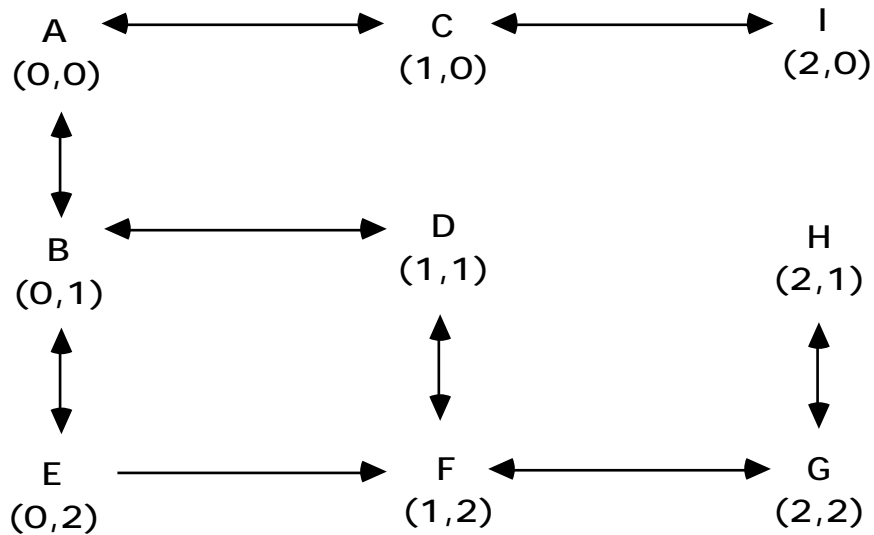
.

a) Ecrire les expressions en CLIPS pour déclarer les templates de type Départ, But et Noeud.

- b) Le noeud choisi pour exploration pendant chaque cycle est étiqueté avec un status "active". Au début, un noeud "active" est créé pour le point de départ. Ecrire une règle (R1) qui crée un noeud pour le point de départ avec un status "active".
- c) Ecrire une règle (R2) pour détecter si le noeud "active" est le but. Quand le système a trouvé le but, faire imprimer "but trouvé" et s'arrêter. Pour assurer que cette règle a priorité sur les règles R1 à R6, donnez lui une salience de 40.
- d) Il faut tester si les noeuds avec status "new" correspondents aux chemins déjà explorés (les noeuds de status "closed"). Ecrire la règle (R3) qui détruit les noeuds "new" avec le même nom qu'un noeud "closed". Pour assurer que cette règle a priorité, donnez lui une salience de 40.
- e) Il faut calculer le coût de chemin passant par chaque noeud de status "new". Ecrire la règle R4 qui calcule le coût h pour un noeud "new" en utilisant la fonction "distance". Vérifier qu'h est > 0 et qu'il n'y a pas un noeud closed avec le même nom.
- f) Ecrire la règle, R5, qui calcule le coût estimé f à partir de h et g. Cette règle doit changer le status à open (ouvert a l'exploration).
- g) Apres le premier cycle, le noeud avec un status "open" et un coût estimé (f) minimal est rendu "Active". Ecrire une règle (R6) pour choisir un noeud avec un coût minimal et le rendre "active". Vérifier qu'il n'existe pas un noeud avec status "new" ou "active" (voir b, c).

Pour tester votre solution :

Soit que vous avez un robot mobile avec le réseau d'emplacements ci-dessus. Votre robot est à l'emplacement A. Il a besoin d'aller à l'emplacement H.



Le réseau est composé d'emplacement défini par le "template" :

```

;;
;; Regles d'affecter les locations des emplacements
;;

(deftemplate emplacement
  (slot nom (type SYMBOL) (default NIL))
  (slot x (type NUMBER))
  (slot y (type NUMBER))
  (multislot voisins (default NIL))
)

(deffacts reseaux-d-emplacements
  (emplacement (nom A) (x 0) (y 0) (voisins B C))
  (emplacement (nom B) (x 0) (y 1) (voisins A E D))
  (emplacement (nom C) (x 1) (y 0) (voisins A I))
  (emplacement (nom D) (x 1) (y 1) (voisins B F))
  (emplacement (nom E) (x 0) (y 2) (voisins B F))
  (emplacement (nom F) (x 1) (y 2) (voisins E D G))
  (emplacement (nom G) (x 2) (y 2) (voisins F H))
  (emplacement (nom H) (x 2) (y 1) (voisins G))
  (emplacement (nom I) (x 2) (y 0) (voisins C))
)

```

```

;;
;; Regles pour le connectivite des emplacements
;; Ces regles genere les nouvelles noueds dans l'arbre de
recherche
;;

(defrule Genere-voisins
  "genere les noeud pour les voisins de ?X"
  (declare (salience 20))
  (noeud (nom ?p) (status active) (g ?g))
  (emplacement (nom ?p) (voisins $? ?v $?))
=>
  (assert (noeud (nom ?v) (status new) (pere ?p) (g =(+ ?g
1))))))
)

;;
;; regle pour fermer un noeud active
;;

(defrule fermer-noeud
  "passer aux status closed apres les voisins sont cree"
  (declare (salience 10))
  ?N <- (noeud (status active))
=>
  (modify ?N (status closed))
)

;;
;; affecter les locations
;;

(defrule Affecter-position-emplacement
  "affecter le location de chaque emplacement"
  (declare (salience 30))
  ?N <- (noeud (nom ?n) (x -1) (y -1))
  (emplacement (nom ?n) (x ?x) (y ?y))
=>
  (modify ?N (x ?x) (y ?y))
)

(defrule Affecter-position-des-buts
  "affecter le location de chaque emplacement"
  (declare (salience 10))
  ?B <- (but (nom ?n) (x -1) (y -1))
  (emplacement (nom ?n) (x ?x) (y ?y))
=>
  (modify ?B (x ?x) (y ?y))
)

```

```
(defrule Affecter-Location-des-depart
  "affecter le location de chaque emplacement"
  (declare (salience 10))
  ?D <- (depart (nom ?n) (x nil) (y nil))
  (emplacement (nom ?n) (x ?x) (y ?y))
=>
  (modify ?D (x ?x) (y ?y))
)

;;
;; Regle d'initialiser le recherche
;;

(defrule init "Initialise avec le depart et le but"
  (initial-fact)
  (not (depart))
=>
  (printout t "Le nom de depart? ")
  (bind ?n (read))
  (assert (depart (nom ?n) ))
  (printout t "Le nom du but? ")
  (bind ?n (read))
  (assert (but (nom ?n)))
)
```

```

;;
;; regle d'areter la recherche
;;

(defrule termine-recherche "close all open nodes"
  (chemin $?)
  ?N <- (noeud (status open))
=>
  (modify ?N (status closed))
  (halt)
)

;;
;; regle de composer le chemin
;;

(defrule compose-chemin "compose un liste du chemin"
  ?C <- (chemin ?n $?q)
  (noeud (nom ?n) (pere ?p&:(neq ?p nil)))
  (but (nom ?b))
  (test (neq ?p ?b))
=>
  (retract ?C)
  (assert (chemin ?p ?n $?q))
)

;;
;; regle d'imprimer le chemin
;;

(defrule imprimer-chemin "compose un liste du chemin"
  (chemin ?d $?q ?b)
  (depart (nom ?d))
=>
  (printout t "Le chemin de " ?d " a " ?b " passe par "
  $?q crlf)
  (halt)
)

```

Algorithme GRAPHSEARCH (Nilsson)

Symboles:

T : Arbre de Recherche

B : Ensemble des buts

s : Noeud de départ (Start)

M : Liste des successeurs

Open : Ensemble des noeuds "ouverts"

(liste des noeuds à explorer)

Closed : Ensemble des noeuds "fermés"

(liste des noeuds déjà explorés)

n, e : noeuds

1) Créer T, Open et Closed, (tous initialisés à l'ensemble vide).

2) Mettre s dans Open et comme racine de T.

3) Loop : si OPEN est vide, Alors Exit avec échec.

4) Extraire n de Open. Open <- Open - n,
Closed <- Closed + n

5) Si n est un élément de B alors : Succès!!

Construire une pile avec les noeuds de T constituant le chemin entre n et s.
Exit en rendant la pile.

FinSi

6) Expansion de n : Créer M (les successeurs de n)

7) Pour chaque e de M,

si e est dans Closed alors M <- M - e.

sinon

a) optionel : Calculer le coût estimé f(e) pour e.

b) ajouter e dans Open (LIFO, FIFO ou Triée)

c) ajouter e à T comme successeur de n.

8) Aller à l'étape 3