CHAPITRE 1

PARCOURS D'ARBRES ET DE GRAPHES

Ce premier chapitre résume un peu les premières interrogations des débuts de l'intelligence artificielle. Comment planifier une action, comment explorer les actions possibles à partir d'un état du problème considéré ? Beaucoup de problèmes se ramènent à chercher un chemin dans un graphe, mais avant cela, il aura fallu définir les nœuds de ce graphe. C'est tout le problème de la représentation des états d'un problème et plus généralement de la représentation des connaissances.

Il n'est jamais facile a priori de dire si une modélisation est meilleure qu'une autre. L'espace de recherche est plus ou moins vaste et les considérations informatiques de programmation ont leur influence sur cette représentation souvent plus encore que les considérations algorithmiques de recherche d'une solution. Poser formellement un problème, c'est déjà la moitié du travail en vue de sa résolution.

1.1 SPECIFICATION ET PLANIFICATION

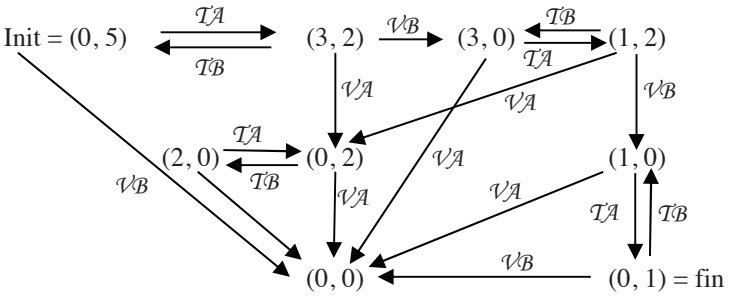
Un premier exemple montrera, malgré sa simplicité, combien l'esprit humain se précipite sur la résolution d'un problème avant de le spécifier calmement et de songer à une représentation simple des états que peut prendre un système simple soumis à des opérations simples.

1.1.1 Définition d'un espace d'états : le problème des cruches

Etant donnés deux cruches, A de 5 litres initialement pleine et B de 2 litres initialement vide, on cherche à obtenir A vide et exactement un litre dans B. Les seules opérations possibles sont : vider une cruche (dans l'évier), transvaser le contenu d'une cruche dans l'autre complètement si c'est possible sans faire déborder ou jusqu'à ce que la cruche réceptrice soit pleine *[Foutelet]*.

Chercher une représentation des états du problème puis définir les opérateurs, enfin dessiner le graphe complet de tous les états possibles.

> La représentation la plus simple semble être le couple (contenu de A, contenu de B)
> Les opérations Vider et Transvaser peuvent s'écrire ainsi ou bien avec 3 arguments en donnant le nom de la cruche.
> VA (x, y) → (0, y) TA (x, y) → (max(0, x – 2 + y), min(2, x + y)
> VB (x, y) → (x, 0) TB (x, y) → (min(5, x + y), max(0, y – 5 + x))



1.1.2 Exemple de la pizza

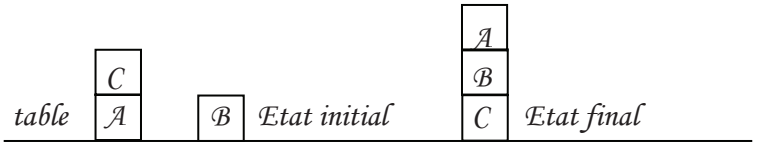
Décrire l'achat d'une pizza par téléphone.

> On peut simplement décrire les faits T = avoir un téléphone qui fonctionne, A = avoir de l'argent, C = passer commande, M = être à la maison, R = être au restaurant, P = avoir la pizza et les actions appeler, aller, commander, acheter. Les états initial et final sont alors Init = {T, A, M} et Fin = {M, P}.

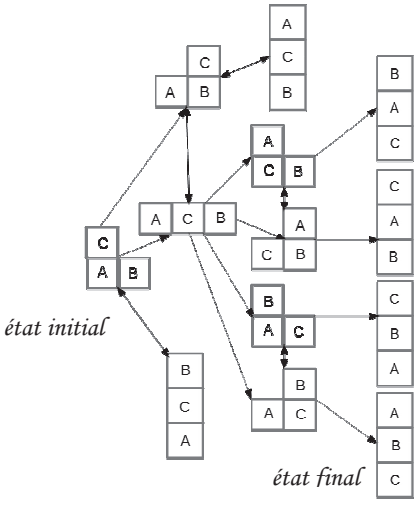
1.1.3 Exemple des cubes

Trouver les prédicats nécessaires à l'écriture des conditions pour passer de l'état initial (à gauche) à l'état final (à droite).

Les seuls opérateurs sont « prendre un cube », le « poser sur la table » ou « sur un autre cube ».



> Les prédicats peuvent être libre(X), surtable(X), enmain(X), mainvide, sur(X, Y)
> Mais on peut toujours trouver autre chose.
> Ces états sont Init = {libre(c), libre(b), surtable(a), surtable(b), sur(a, c), mainvide}
> Final = {libre(a), sur(a, b), sur(b, c), mainvide, surtable(c)}
>
> Les opérateurs (en notant les présupposés, suppressions et ajouts) :
> poser(X) pré et suppr : enmain(X), ajout : surtable(X), libre(X), mainvide
> soulever(X) pré et suppr : mainvide, surtable(X), ajout : enmain(X)
> empiler(X, Y)
> pré et suppr : enmain(X), libre(Y),
> ajout : mainvide, sur(X, Y)
> depiler(X, Y) pré et suppr : mainvide, libre(X), ajout : enmain(X), libre(Y)
> Ici la solution est simple :
> depiler(c, a), poser(c), empiler(b, c), soulever(a), empiler(a, b)
> Graphe complet ci-après (il n'y a que 3 emplacements sur la table) :



Bien sûr, les règles peuvent être multiples, ordonnées dans le temps, avoir des pondérations, on pourrait décrire des plans conditionnels etc.

ALGORITHME DE PLANIFICATION

Les premiers algorithmes de planification sont *GPS* (*General Problem Solver, E. Newell et H. Simon 1961)*, STRIPS (*STanford Research Institute Problem Solver, Richard Fikes et Nils Nilsson en 1971),* GraphPlan…

Après la définition des prédicats et des opérateurs (à chaque opérateur est associé une liste de préconditions, une liste de suppression et une liste d'ajouts correspondant à son action), les états du problème sont empilés et on peut décrire de façon informelle :

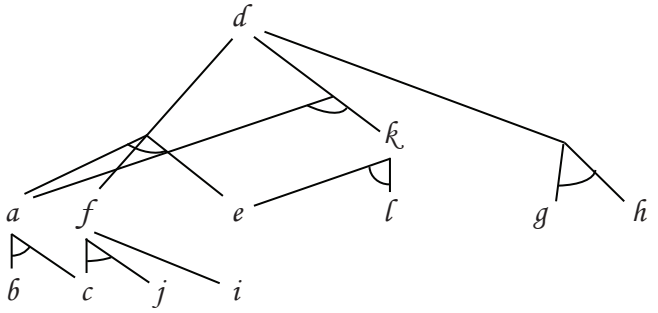
satisfaction (Buts, e, pile) (* e état courant *)
pour chaque b ∈ Buts faire
    NouvelEtat e' ← realisation (b, e, pile)
    si e' = échec alors échec
    sinon si tous les b ∈ Buts sont satisfaits dans l'état e' alors e'
    sinon échec

réalisation (b, e, Pile)
    si b satisfait dans l'état e alors e
    sinon si b ∈ pile alors échec
    sinon pour chaque operateur op pouvant satisfaire le but faire
        e' ← appliquer (op, e, b :: pile) (* on rajoute le but en question *)
        si e' ≠ échec alors b satisfait dans l'etat e et retourner e'
        sinon échec

appliquer (op, e, Pile)
e' ← satisfaction (préconditions de op, e, pile)
si e' <> échec alors faire l'action de op,
e' ← e' – liste de suppressions de op
et retourner e' :: liste des ajouts de op
sinon échec

1.1.4 Exemple d'hypergraphe

Il s'agit simplement d'est un graphe qualifié de *et* / *ou* dans lequel les branches reliées par un arc signalent une conjonction *et*. En partant du but d, les règles sont :
b, c → a; a, e, f → d; a, k → d; i → f; c, j → f; g, h → d; e, l → k.



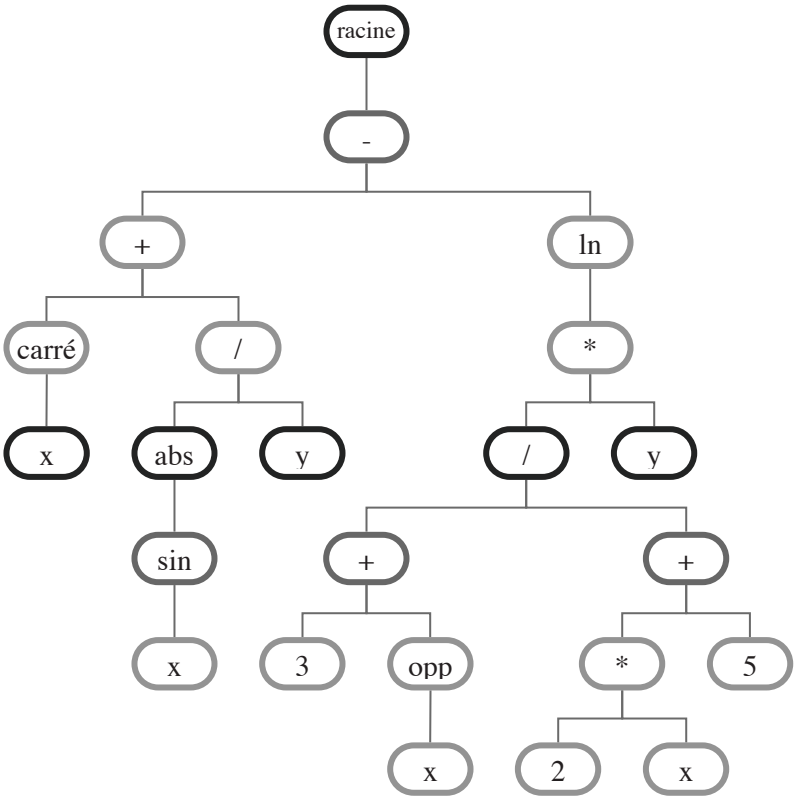
STRATÉGIES DE PARCOURS D'ARBRE

Le but final de tout problème, se traduisant par un graphe d'états, est d'obtenir une solution, mais aussi la suite des actions pour y parvenir, voire l'ensemble des solutions, d'où l'importance des méthodes de parcours de graphes en recherche opérationnelle et d'abord pour les arbres. Rappelons les stratégies de lecture d'arbres. Si un arbre est (récursivement) la donnée (r, f) d'une racine et de la liste de ses sous-arbres fils, alors sa lecture peut être réalisée par des parcours différents.

Le plus simple est de prendre par exemple une expression mathématique *E* ici écrite suivant la façon traditionnelle, ou notation infixe, ce qui signifie par exemple que le signe + se trouve entre les deux termes de l'addition.

$$E = \sqrt{x^2 + \frac{|\sin(x)|}{y} - \ln\frac{3+(-x)}{2x+5}}$$

En fait, une telle expression correspond à l'arborescence suivante, et la notation infixe résulte d'une lecture gauche - racine - droite, de l'arbre.



La notation suffixée ou polonaise, (utilisée dans le langage de programmation Forth), résulte d'une lecture gauche - droite - racine, ce qui donne par exemple ici :

$$E_S = x\ carré\ x\ sin\ abs\ y\ /\ +\ 3\ x\ opp\ +\ 2\ x\ *\ 5\ +\ /\ y\ *\ Ln\ -\ \sqrt{}$$

On a noté – la soustraction de deux arguments et *opp* la fonction *opposé* à un seul argument. Si le même symbole n'est jamais utilisé pour des opérateurs d'arités différentes, alors les parenthèses deviennent inutiles.

Cette notation est, de loin, la plus rationnelle, car elle établit une correspondance entre l'ordre d'écriture de gauche à droite et l'ordre opératoire, ceci permettant de débuter les calculs avant même que l'expression soit terminé d'être donnée.

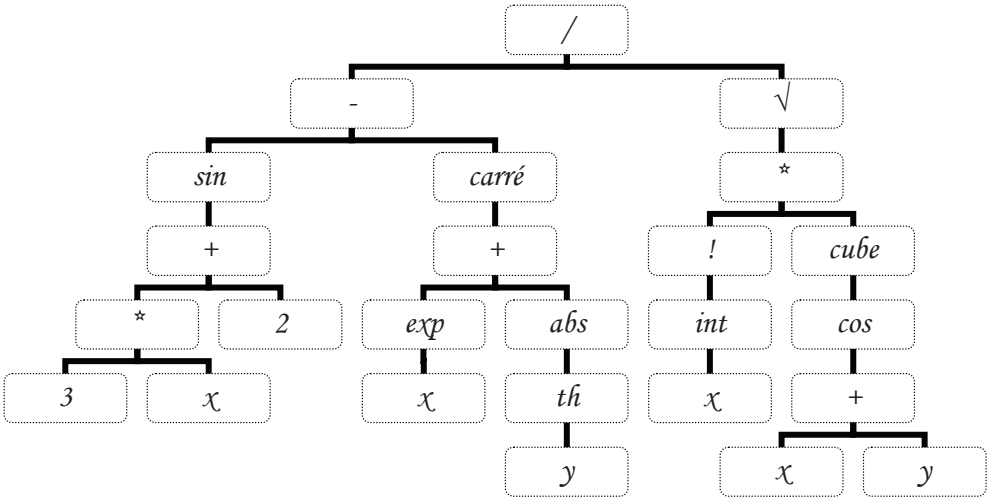
Par contre, c'est la notation préfixée (correspondant à la lecture racine-gauche-droite de l'arbre) du Lisp, qui consiste à écrire au contraire, la fonction avant ses opérandes, ce qui donne ici entièrement parenthésée :

$$E_P = (\sqrt{}\ (-\ (+\ (carré\ x)\ (/\ (abs\ (sin\ x))\ y))\ (Ln\ (*\ (/\ (+\ 3\ (opp\ x))\ (+\ (*\ 2\ x)\ 5))\ y))))$$

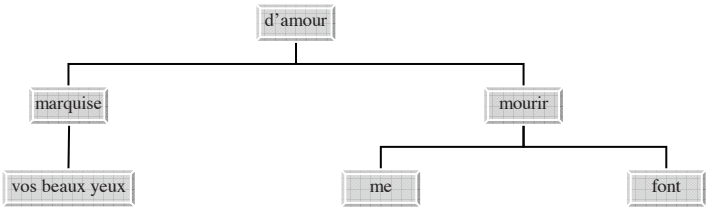
1.1.5 Tracer l'arbre de l'expression algébrique

$$E = \frac{\sin(3x+2) - (e^{x} + |\text{th y}|)^{2}}{\sqrt{(\text{Int } x)! \cdot \cos^{3}(x+y)}}$$

En déduire par les parcours gauche-droite-racine (profondeur d'abord, suffixée ou polonaise) et racine-gauche-droite, (expression préfixée).



Es = 3 x * 2 + x exp y th abs + carré + x int fac x y + cos cube * rac /
Ep = / – sin + * 3 x 2 carré + exp x abs th y rac * fac int x cube cos + x y
Lire l'arbre ci-contre en rgd, grd et gdr.



L^{suffixe} = vos beaux yeux, marquise, me font mourir d'amour
$L^{\text{préfixe}}$ = d'amour, marquise, vos beaux yeux, mourir, me font
L^{infixe} = marquise, vos beaux yeux, d'amour, me mourir, font

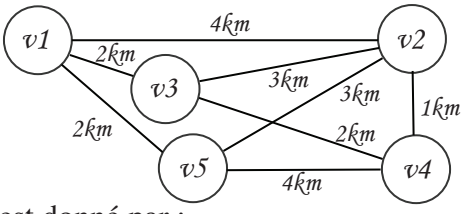
1.1.6 Transformation en postfixe d'une expression parenthésée usuelle

Par exemple, pour *E = sin(3x + 2) – exp(5*x / 3)* il faudra obtenir l'expression postfixée *3 x * 2 + sin 5 x * 3 / exp –*.

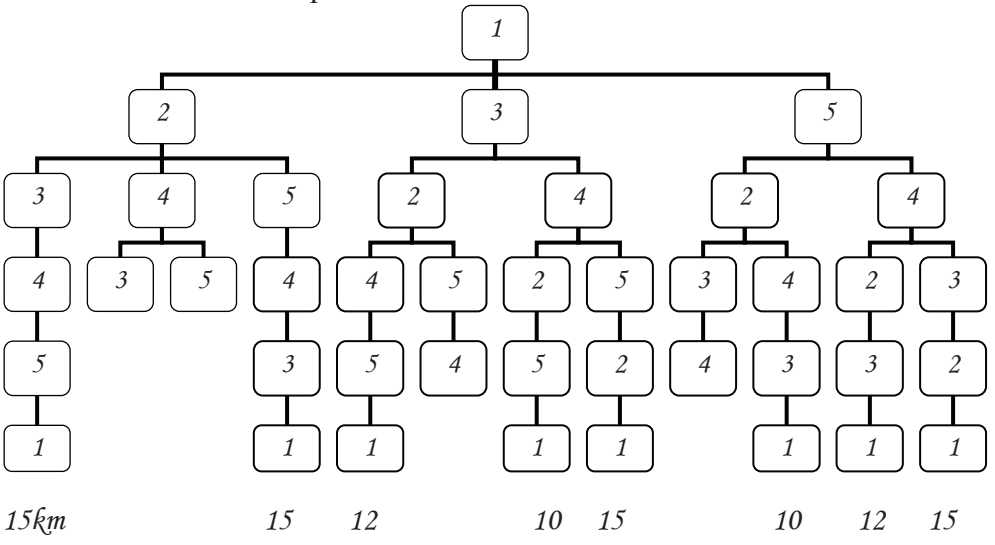
```
Solution en Lisp, d'abord tout parenthèser :
(defun postfix (E) (cond ((atom E) E)
        ((null (cddr E)) (list (postfix (cadr E)) (car E)))
        ((list (postfix (car E)) (postfix (caddr E)) (cadr E)))))
Exemple (postfix '((sin ((3 * x) + 2)) – (exp ((5 * x) / 3))))
→ ((((3 X *) 2 +) SIN) (((5 X *) 3 /) EXP) –)
```

1.1.7 Exemple de parcours systématique pour le voyageur de commerce

En partant de la ville *v1*, construire l'arbre de toutes les possibilités de réalisation d'un parcours complet avec retour en *v1*, sans passer deux fois par la même ville (chemin hamiltonien).



L'arbre est donné par :



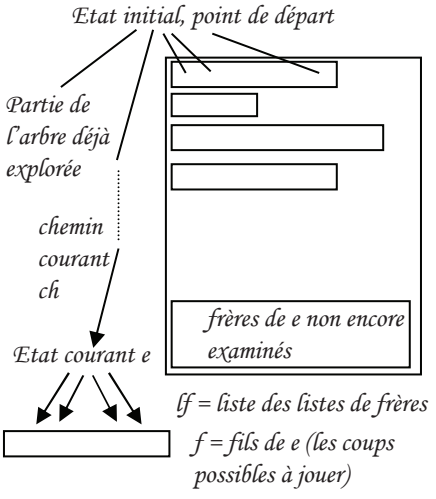
Arbre complet des boucles possibles de la ville 1 et retour en 1 avec totaux, on voit donc qu'il y a deux plus courts chemins de 12km.

1.2 L'ALGORITHME DU « BACKTRACKING »

Cet algorithme, appelé en français « essais avec retour en arrière », consiste à lire un arbre suivant la stratégie « racine - gauche - droite ». Ainsi, on retient en mémoire à chaque instant, une branche de l'arbre avec à chacun de ses étages, la liste des fils non encore explorés. De cette façon, on ira beaucoup plus loin dans la résolution informatique d'un problème qu'en développant entièrement toutes les branches jusqu'à une profondeur donnée. Cette dernière méthode, dite de lecture en largeur, ne tient pas longtemps avant d'arriver au « stack overflow ».

Chercher à résoudre un problème en construisant pas à pas l'éventuelle solution, quitte à revenir en arrière dans la suite des choix à faire, est connu sous le nom d'algorithme à retour ou « backtracking ». Il s'agit en fait d'explorer une arborescence en suivant l'ordre racine-gauche-droite encore appelé « parcours en profondeur d'abord ». L'exemple le mieux connu d'un tel problème est celui des reines de Gauss : placer 8 dames sans qu'aucune prise ne soit possible, sur un damier 8*8.

Dans le schéma ci-contre, on considère qu'à chaque état *e* du problème, on a une suite d'états *ch* qui y a amené et que *f* est la liste des états suivants.
Le cas où la liste *f* est vide, se décompose en deux, si rien n'a pu être choisi car *ch* est vide, c'est l'échec, sinon on retire le dernier choix de *ch*, et on remonte à l'examen des frères de l'état précédent, c'est le retour en arrière : une remontée dans l'arbre. On tiendra donc en réserve une liste *lf* des listes de frères non encore explorés.



Le troisième cas indique que si le premier des choix possibles arrive au but du problème, alors on renvoie cette solution, qui est la première rencontrée. Le cas suivant exprime une descente en ajoutant l'état suivant à la liste *ch*. Sinon, on va voir les frères, c'est un déplacement horizontal à droite dans l'arbre.