Méthodes et Outils pour l'Intelligence Artificielle

F. Bouquet

Master S&T - Mention Informatique

Première année

L'intelligence Artificielle

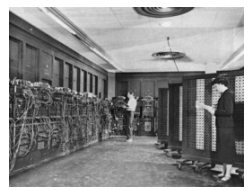
Philosophie :	Logique, méthode de raisonnement, esprit comme système physique, apprentissage, langage
Mathématiques :	Représentations formelles et preuves, algorithmes (in)décidabilité, probabilité
Psychologie :	Adaptation, phénomène de perception et de contrôle moteur, techniques expérimentales
Linguistique :	Représentation des connaissances, grammaire
Neurosciences :	Substrat physique de l'activité mentale
Théorie du contrôle :	Systèmes asservis, stabilité, concept d'agent

Plan

1. Historique
2. Structuration de la connaissance
   - Langue naturelle, Semi-formelle, Formelle, Logique
3. Méthodes de résolution générales
   - Méthodes de base
   - Évolutions et Heuristiques
   - Satisfaction de contraintes
4. Domaine d'application
   - Système expert / Apprentissage
   - Jeux
   - Planification / Ordonnancement
   - Traitement du langage naturel

Historique 1/3

- -3000 : Papyrus premières règles de production (médecine)
- -350 : Aristote invente le syllogisme
- 1679 : Leibnitz invente l'arithmétique binaire
- 1854 : Boole et son algèbre
- 1938 : Shannon **BI**nary digi**T**
- 1941 : I. Asimov écrit Robbie et les trois lois de la robotique
- 1943 : McCulloch & Pitts modélisation neurones et cerveau
- 1944 : **ENIAC**
- 1945 : J. Von Neumann et O. Morgenstein :
  - Solution théorique : jeux de stratégie / démo. automatique
  - "Intelligence Artificielle" $\neq$ "Bêtise naturelle"



Historique 2/3

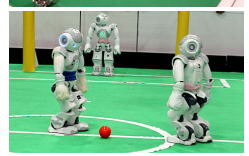
- 1950 : "Computing Machinery and Intelligence" de Turing
- 1957 : Thème de l'IA : **G**eneral **P**roblem **S**olver
  N. Chomsky (MIT, linguiste) sémantique $\neq$ syntaxe
- J. Mac Carthy : Créé le premier laboratoire IA (MIT, 1958)
  Lisp (1958), Algorithme **Alpha–Béta** (1961)
- 1965 : Feigenbaum, Buchaman et Lederberg **DENDRAL** :
  - Mécanismes de raisonnement inductifs / empiriques
  - Problème consistant

  J.A Robinson invente la procédure de résolution
  ELIZA (MIT) simule le dialogue avec un psychologue
- 1969 : Shakey 1^{er} robot déplacer, percevoir, résoudre problèmes

Historique 3/3

- 1972-74 : A. Colmerauer et P. Roussel écrivent Prolog
- 1980 : Création de **A**merican **A**ssociation **A**rtificial **I**ntelligence
- 1985 : Linguistique : Vocabulaire, reconnaissance du discourt, multi-utilisateur.
- 1990 : Recherche linguistique / Jeux de stratégie
- 1997 : Deep Blue bat G. Kasparov – Premier match de football avec des robots
- 2000 :
  - Animaux de compagnie robot (AIBO)
  - Carneggie Mellon : les robots nomades (Antarctique)
  - Kinect : Xbox interface avec mouvement du corps 3D
- 2010 : Siri / Google Now, Google Car, *Deep Learning*





En résumé

- Qu'est-ce les systèmes à base d'IA savent faire ?
  - Jouer décemment au ping-pong
  - Conduire un véhicule sur une route de montagne / dans la circulation
  - Jouer aux cartes
  - Découvrir et prouver un nouveau théorème mathématique
  - Inventer une histoire drôle (de façon intentionnelle)
  - Donner des conseils avisés dans des domaines pointus
  - Traduire des langues, parler une langue, parler en temps réel.

Principe

- **Donner les moyens à l'ordinateur de :**
  - Acquérir de l'information (connaissances)
  - Raisonner sur les informations
  - Donner les résultats
- **But :**
  - Transcrire le français pour l'ordinateur
  - Codage mémoire de l'information
  - Traitement
  - Retranscrire

Français $\Longleftrightarrow$ Machine

- Logique Classique :
  - Logique propositionnelle
  - Calcul des prédicats
- Logique non classique :
  - Défaut de Reiter
  - Logique Multi-valuée (logique floue)
- Règle de production
- Objets structurés :
  - Réseaux bayéseins
  - Chaînes de Markov
  - Réseaux de neurones

Logique propositionnelle

[Aristote à Boole]

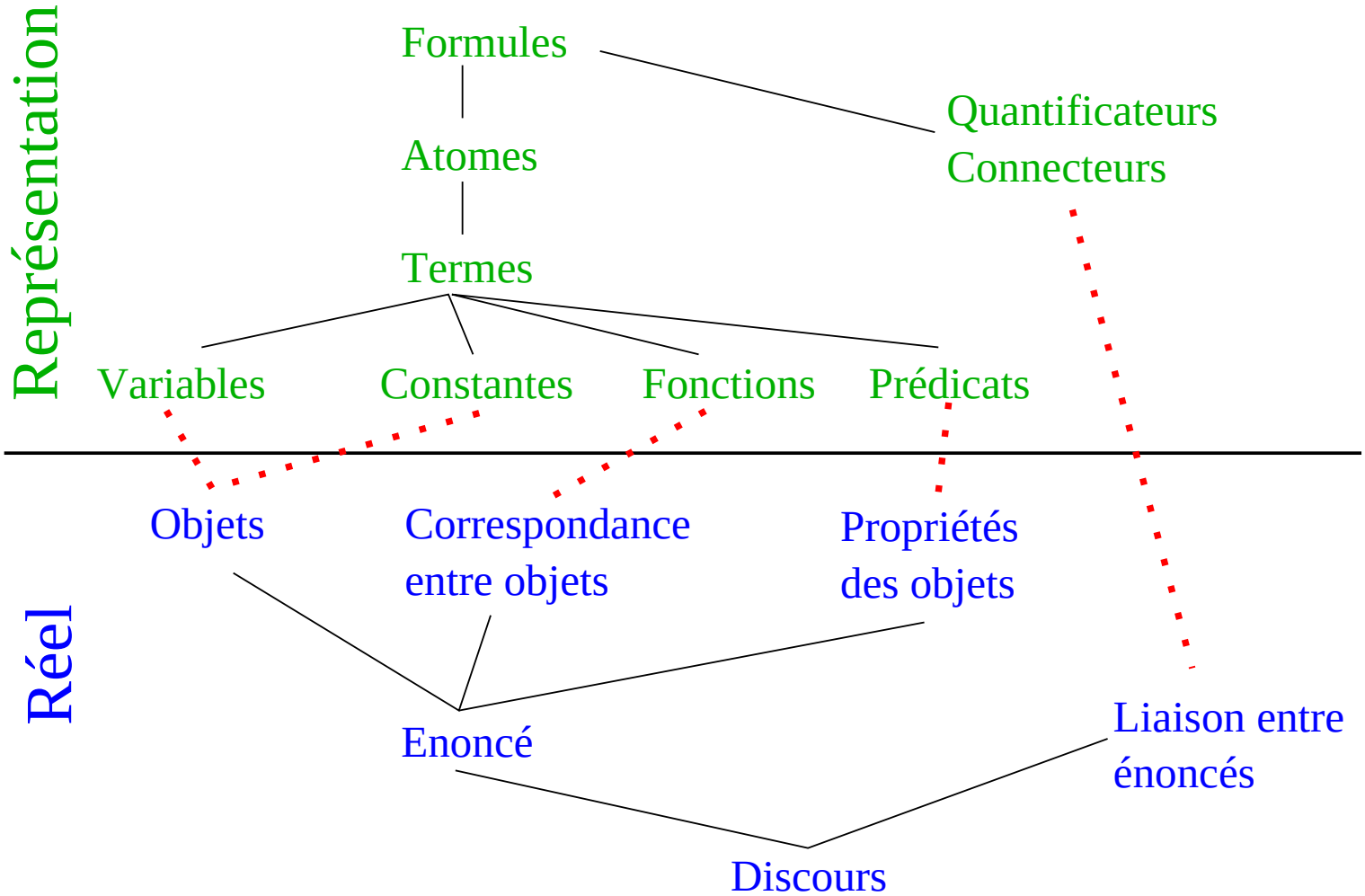
- **Ensemble de formules :**
  - *Propositions* : lettres majuscules
  - *Connecteurs* : $\neg$, $\rightarrow$ ($\wedge$, $\vee$, $\oplus$...)
  - *Parenthèses* : ( )
- **Axiomes :**
  1. $F \rightarrow (G \rightarrow F)$
  2. $(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$
  3. $((\neg G) \rightarrow (\neg F)) \rightarrow (F \rightarrow G)$
- **Règle(s) d'inférence (Modus ponens/ Modus Tollens) :**
  - Si (F et $F \rightarrow G$) alors G
  - Si ($\neg G$ et $F \rightarrow G$) alors $\neg F$

Logique des prédicats

Extension de la Logique propositionnelle [Kleene 72]

- **Extension des Formules :**
  - Constantes, Variables
  - Fonctions, Prédicats
- **Extension des Axiomes :**
  4. Occurrence de x dans F liée, $\forall \mathbf{x}(\mathbf{F} \rightarrow \mathbf{G}) \rightarrow (\mathbf{F} \rightarrow \forall \mathbf{x}\mathbf{G})$
  5. Occurrence de x liée, $\forall \mathbf{x}\mathbf{F}(\mathbf{x}) \rightarrow \mathbf{F}(\mathbf{t})$
- **Quantificateur(s) :**
  - Universel : $\forall$
  - Existentiel : $\exists$

Formalisation du discours



Défaut [Reiter 79]

Extension de la Logique des prédicats

- **Théorie des défauts, $\Delta = (D, W)$:**
  - W : ensemble de faits, formules closes du 1^{er} ordre
  - D : ensemble de défauts, règles d'inférence à contenu spécifique

- **Un défaut :**
  Soient les formules du 1^{er} ordre avec x comme variable libre :
  - u(x), prérequis
  - v(x), la justification
  - w(x), la conséquence

$$\frac{\mathbf{u(x)} : \mathbf{v(x)}}{\mathbf{w(x)}}$$

Multi-valuée [Kleene 38]

p	q	$\neg p$	$p \rightarrow q$	$p \vee q$	$p \wedge q$	$p \equiv q$
0	0	1	1	0	0	1
0	$\frac{1}{2}$	1	1	$\frac{1}{2}$	0	0
0	1	1	1	1	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
$\frac{1}{2}$	1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	0
1	0	0	0	1	0	0
1	$\frac{1}{2}$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	0
1	1	0	1	1	1	1

$$\begin{cases} I(\neg p) = 1 - I(p) \\ I(p \vee q) = Max(I(p), I(q)) \\ I(p \wedge q) = Min(I(p), I(q)) \\ I(p \rightarrow q) = Max(1 - I(p), I(q)) \\ I(p \equiv q) = (I(q) = I(p)) \end{cases}$$

- Tiers exclu : $p \vee \neg p$
- Principe de non-contradiction $\neg(p \wedge \neg p)$

$\hookrightarrow$ ne sont pas des tautologies

Règles de production

Si A **alors** B $(0{,}8)$

- Une règle se décompose en **3 parties :**
  - Condition d'application : A
  - Déduction obtenue : B
  - Facteur de certitude : $0{,}8$
- **Calcul d'une transition** $A \xrightarrow{FC} B$:

$$\begin{cases} FC(source) = 1 \\ FC(B) = Max(0, FC(A)) \times FC \end{cases}$$

Objets structurés

- **Basé sur les réseaux sémantiques (graphe) :**
  - Nœud représente des concepts
  - Arc traduit les relations entre concepts

    ↪ Recherche de solutions dans graphe

- **Ajouter pour l'inférence :**
  - Règles de production
  - Opérateurs logiques

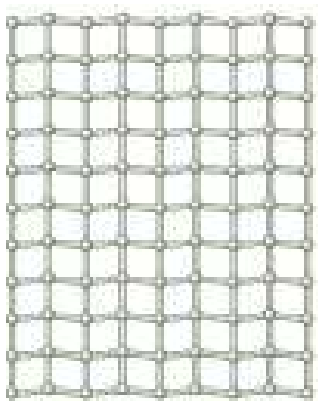
Réseau Bayésein

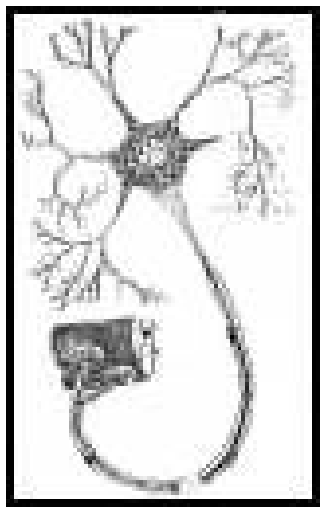
- **Constitution :**
  - Graphe orienté acyclique :
    - Noeud représente une variable aléatoire
    - Arc les relations entre les variables
  - Série de distributions conditionnelles
- **Théorème de Bayes :** $Prob(A|B) = \frac{Prob(B|A) \times Prob(A)}{Prob(B)}$
- *Rapport de vraisemblance :* $\lambda(B|A) = \frac{Prob(B|A)}{Prob(B|\bar{A})}$
- *Vraisemblance a priori :* $V(A) = \frac{Prob(A)}{Prob(\bar{A}|B)}$
- *Vraisemblance a posteriori :*
  $V(A|B) = \frac{Prob(A|B)}{Prob(\bar{A}|B)} = \lambda(B|A) \times V(A)$

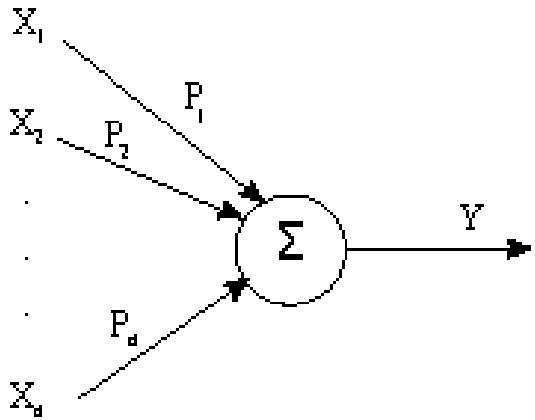
Chaîne de Markov (Monte carlo)

- **Codage d'évolution/transition :**
  - Probabilité déplacement de $x \rightarrow y$: $p(x, y)$
  - Distribution uniforme $\pi(x) = \int_{\mathbb{P}} \pi(y) \times p(y, x) \delta y$
- **Définir p(x, y) :**
  - Echantillonneur de Gibbs
  - Algorithme de Metropolis-Hastings
- **Satisfait :**
  - Réversibilité
  - Irréductible
  - Apériodique
  - Récurrence

Réseau de neurones







Signal reçu :

$$S_r = \Sigma_{i=1}^{n}(p_i \times x_i)$$

Signal émis :

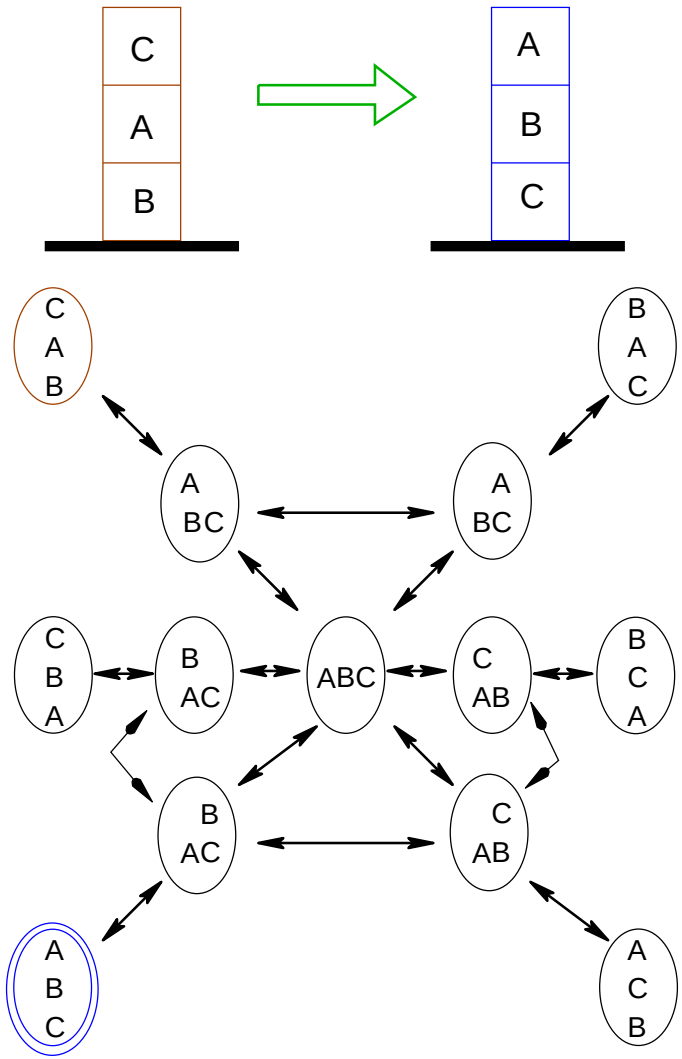
$$\begin{cases} 1 & \text{si} \quad S_r \geq \theta \\ 0 & \text{si} \quad S_r < \theta \end{cases}$$

Problème

- **Définitions :**
  - Tout problème est relatif à un certain ensemble d'objets que l'on peut décrire sous forme de **variables d'état** du problème
  - Un **état du problème** est l'ensemble des valeurs que prennent ses variables d'états à un instant donné.
  - L'**espace d'états** d'un problème est l'ensemble des états possibles pour le problème considéré.

Problème

- **Décrit par**
  - État initial (Initialisation)
  - Transition (Règles)
  - Espace de recherche/états (atteignables)
  - État final (condition d'arrêt)
- **Résolution**
  - Parcours de l'espace de recherche
  - Stratégies (Largeur, Profondeur ...)
  - Heuristiques (Best-First, Taboo, Recuit-simulé, Sac à dos ...)



Parcours en profondeur

Algorithme 1 : Algorithme : Profondeur(n,Sol)

Input : un nœud (n)

Output : renvoie Vrai si chemin solution Sol et faux sinon.

Sol ← Sol ∪ {n} ;

if *n est solution* **then** fin ← Vrai;

else

    fin ← Faux ; n_1 ← successeur(n) ;

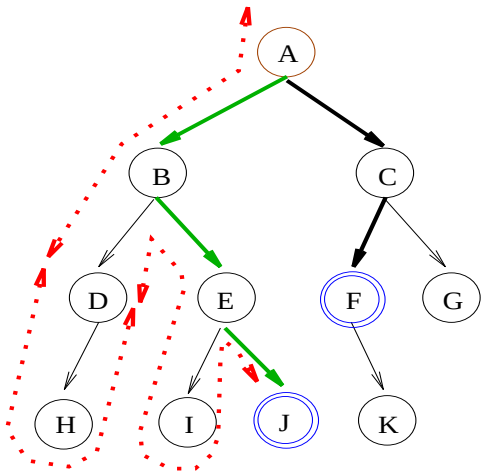
    while ***non****(fin)* **et** n_1 *est valide* **do**

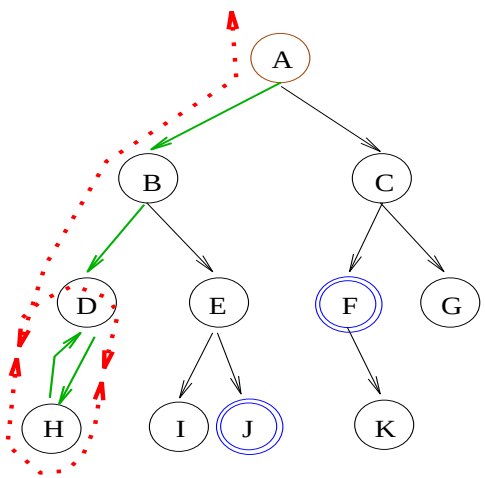
        fin ← Profondeur(N_1, Sol) ;

        n_1 ← successeur(n) ;

    if ***non****(fin)* **then** Sol ← Sol \ {n};

return fin;





- **Problème :** Les cycles dans les graphes

Parcours en profondeur bornée

Algorithme 2 : ProfondeurB(n, p, Sol)

Input : un nœud (N), une profondeur p

Output : renvoie Vrai si chemin solution Sol et faux sinon.

Sol $\leftarrow$ Sol $\cup$ {n} ;

if *n est solution* **then** fin $\leftarrow$ Vrai;

else

  fin $\leftarrow$ Faux ; $n_1 \leftarrow$ successeur(n) ;

  p $\leftarrow$ p - 1;

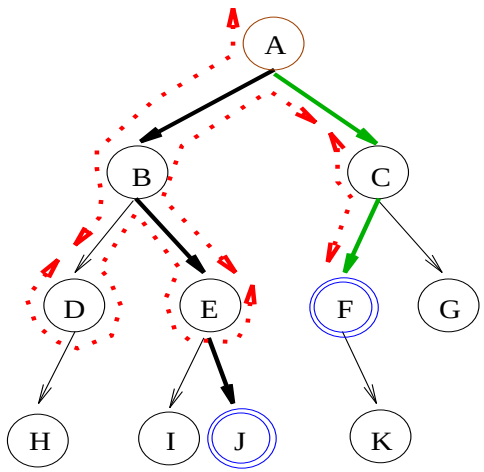
  while ***non****(fin)* **et** n_1 *est valide* **do**

    fin $\leftarrow$ Profondeur(N_1, p, Sol) ;

    $n_1 \leftarrow$ successeur(n) ;

  if ***non****(fin)* **then** Sol $\leftarrow$ Sol $\setminus$ {n};

return fin;



Parcours en largeur

Algorithme 3 : Largeur(Liste, Sol)

Input : une liste de chemin

Output : renvoie Vrai si un chemin solution Sol, faux sinon.

Ch ← Premier(Liste) ; fin ← Faux;

while *Ch* ≠ *{}* ***et non****(fin)* ***et*** *L* > *0* **do**

  Liste ← Liste \ {Ch} ; n ← dernierNoeud(Ch);

  n_1 ← successeur(n);

  while ***non****(fin)* ***et*** n_1 *est valide* **do**

    if n_1 *est solution* **then**

      Sol ← Ch $\cup_f$ {n_1} /* $\cup_f$ ajoute à la fin */;

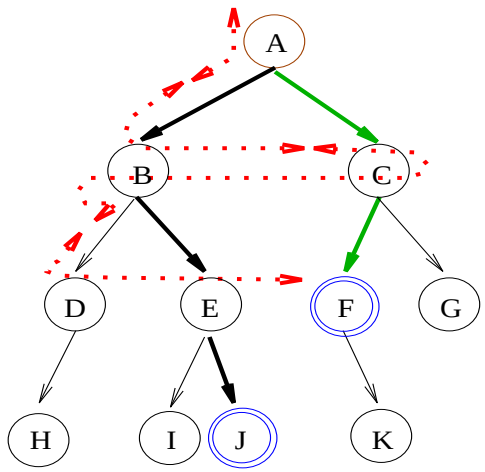
      fin ← Vrai;

    else Liste ← Liste $\cup_f$ {(Ch $\cup_f$ {n_1})};

    n_1 ← successeur(n);

  Ch ← Premier(Liste);

return fin;



Parcours en largeur itérative

Algorithme 4 : LargeurIt(Liste, L, Sol)

Input : une liste de chemin, L la borne pour la profondeur

Output : renvoie Vrai si un chemin solution Sol, faux sinon.

Ch ← Premier(Liste) ; fin ← Faux ; $Liste_1$ ← {} ;

while *Ch* ≠ *{}* ***et non****(fin)* ***et*** *L* > *0* **do**

  Liste ← Liste \ {Ch} ; n ← dernierNoeud(Ch);

  n_1 ← successeur(n);

  while ***non****(fin)* ***et*** n_1 *est valide* **do**

    if n_1 *est solution* **then**

      Sol ← Ch $\cup_f$ $\{n_1\}$ `/* ∪f ajoute à la fin */`;

      fin ← Vrai;

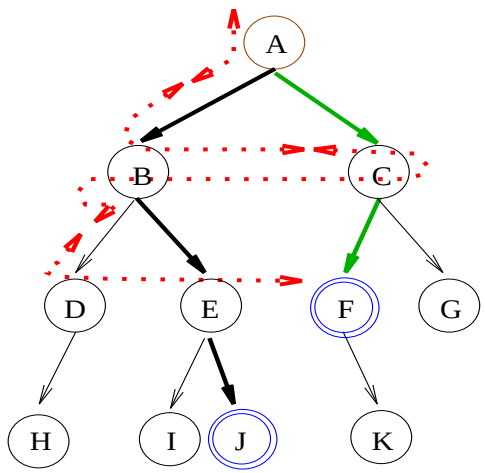
    else $Liste_1$ ← $Liste_1 \cup_f$ {(Ch $\cup_f$ $\{n_1\}$)};

    n_1 ← successeur(n);

  Ch ← Premier(Liste);

if ***non****(fin)* ***et*** *(L > 0)* **then** fin ← Largeur($Liste_1$, L-1, Sol);

return fin;



Critères de comparaison

- **Complétude** : est-ce que la méthode garantit de trouver une solution si elle existe ?
- **Complexité en temps** : combien de temps faut-il pour trouver la solution ?
- **Complexité en espace** : quel espace mémoire faut-il pour effectuer la recherche ?
- **Optimalité** : est-ce que la méthode trouve la meilleure solution s'il en existe plusieurs ?

Comparaison

Méthode	Complexité		Optimal	Complet
	Temps	Espace		
Largeur	$O(b^m)$	$O(b^m)$	oui	oui
Largeur itératif	$O(b^m)$	$O(b \times m)$	oui	oui
Profondeur	$O(b^m)$	$O(m)$	non	oui
Profondeur bornée	$O(b^l)$	$O(l)$	non	oui si $l \geq p$
Bi-directionnal	$O(b^{\frac{m}{2}})$	$O(b^{\frac{m}{2}})$	oui	oui

- b : facteur de branchement
- m : profondeur maximal de l'arbre
- p : profondeur de la solution
- l : limite de la profondeur de recherche

Bilan sur les Parcours "aveugle"

- 3 méthodes explorent l'espace de recherche
- Profondeur plus facile mais cycle (limiter recherche, marquage)
- Largeur plus difficile à implanter et gestion liste
- Problème d'optimisation : associer coût aux arcs (transitions)
- Explosion combinatoire plus important pour la largeur
  $\Longrightarrow$ Utilisation Heuristique

Heuristique

$$h : \mathbb{U} \longrightarrow \mathbb{R}^+$$

- **But :** Améliorer la convergence vs Exploration aveugle
- **Comment :** prendre en compte des informations sur le graphe
- **Type d'heuristique :**
  - **Parfaite** : $\forall u, v \; h(u) = h(v) \Leftrightarrow h^*(u) = h^*(v)$
  - **Presque parfaite** : $\forall u, v \; h(u) < h(v) \Rightarrow h^*(u) < h^*(v)$
  - **Monotone/consistante** : $\forall u, \forall v, \; h(u) - h(v) \leq k(u, v)$, v descendant de u
  - **Minorante/admissible** : $\forall u, h(u) \leq h^*(u)$
- $h(u)$: heuristique, coût entre u est état final ; $k(u, v)$: poids des arcs

Heuristique et propriété

Une heurisitique est spécifique au problème posé :

- Elle peut définir une relation ordre **partiel** ou ordre **complet**
- Si l'heuristique est **parfaite**, alors l'algorithme converge immédiatement vers le but
- Si l'heuristique est **minorante** alors l'algorithme A^* trouvera toujours le chemin optimal
- Si l'heuristique **monotone** et **admissible** garantie algorithme A^* pour tout nœud développé calcule chemin optimal.

Stratégies de recherche :

- Deux types de **développement d'états** :
  - Complètement développé
  - Partiellement développé
- Trois types d'**organisation des alternatives** :
  - LIFO
  - FIFO
  - Recherche ordonnée

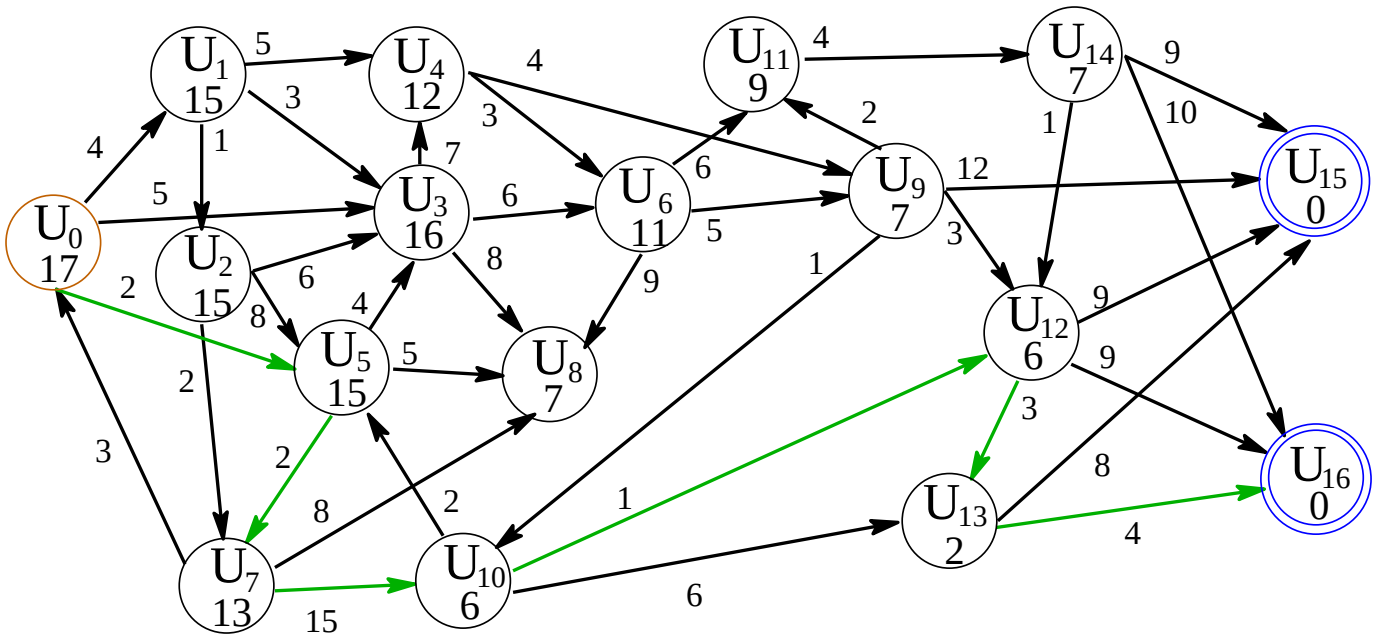
Les méthodes du gradient

- **Gradient** : variation - progression vers la solution. On essaie d'atteindre un **optimum global** par une succession d'**optimums locaux**.
- *Exemple* : pour traverser une ville du nord au sud, on essaiera de minimiser ou maximiser l'écart par rapport à la direction.
  - Algorithme du simplex en programmation linéaire
  - Méthode par approximations successives en analyse numérique
  - Min-Max en programmation des jeux
  - Algorithme A* ...

Best-Fisrt

- **But :** heuristique pour converger le plus rapidement
- **Idée :** continuer la recherche avec le meilleur candidat
- **Algorithme associé :**
  - Gourmand (Greedy) ou Escalade (hill-climbing)
  - Algorithme A*
  - Variantes de A* : IDA*, SMA*, AO*...

Exemple - Gourmand



- Chemin solution : $\{U_0, U_5, U_7, U_{10}, U_{12}, U_{13}, U_{16}\}$,
- Coût : $f(U_{16}) = 26$

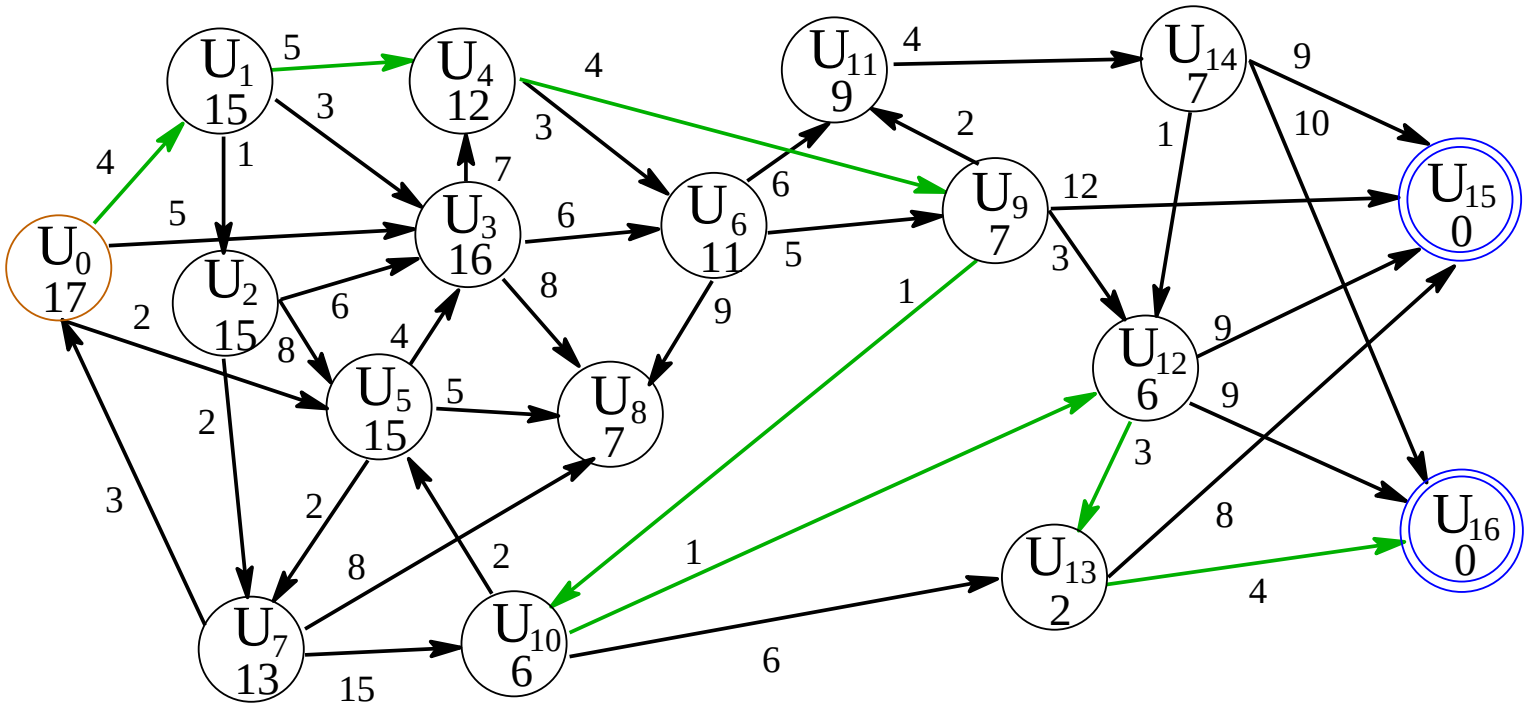
Algorithme A^*

Algorithme 5 : Algorithme : A*(Liste, Sol)

Input : le graphe
Output : g(u_t) et on utilise père(u_t) pour reconstruire le chemin.

```
G ← {u0};
u ← premier(G);
D ← ∅ ;
g(u0) ← 0, f(u0) ← 0 ;
while (G ≠ ∅) et (u ∉ T) do
    G ← G \ {u} ;
    D ← D ∪ {u} ;
    if (u ∉ T) then
        v ← suivant(u) ;
        while (v ≠ ∅) do
            if (v ∉ D ∩ G) ou (g(v) > g(u) + k(u,v)) then
                g(v) ← g(u) + k(u,v);
                f(v) ← g(v) + h(v);
                père(v) ← u;
                insérer-selon-f(G,v);
            v ← suivant(u);
    u ← premier(G);
```

Exemple - A*



- Chemin solution : $\{U_0, U_1, U_4, U_9, U_{10}, U_{12}, U_{13}, U_{16}\}$,
- Coût : $f(U_{16}) = 22$

Terminaison de l'algorithme A*

Pour toute h, application de $\mathbb{U} \longrightarrow \mathbb{R}^+$, et pour tout G (graphe **fini** à valuations positives ou nulles) **A*** **s'arrête**. Soit sur un état terminal, soit faute d'état ($G \neq \emptyset$) s'il n'existe pas de chemin de u_0 à T.

Si G et h vérifient les hypothèses suivantes :

- G est un δ-graphe fini ou infini à valuations positives ou nulles dans lequel il existe au moins un chemin de longueur finie entre u_0 et un état terminal.
- h est une application de $\mathbb{U} \longrightarrow \mathbb{R}^+$ telle qu'il existe un majorant ν avec $h(u_i) \leq \nu$ pour tout état u_i sur un chemin **optimal** de u_0 à un état terminal.

alors A* s'arrête après un nombre fini d'étapes et fournit un chemin de u_0 à T.

Complexité

- Soit n le nombre d'états de l'espace d'états du problème.
  - Pire des cas, la complexité est en 2^n
  - Si h est une heuristique **minorante**, complexité n^2 .
  - Si h est une heuristique **monotone**, complexité n.
- **Remarque** : Même une complexité linéaire sur le nombre d'états est souvent inutilisable dans la plupart des cas pratiques
- *Exemple* : Voyageur de commerce $n = v!$ (v = nombre de villes).

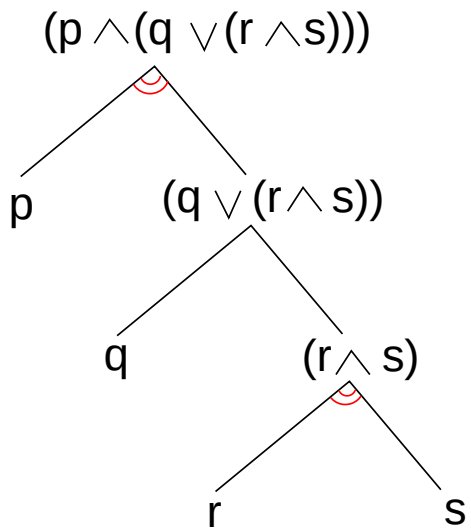
Variante A*

- Interative deepening A* : IDA*
- Recursif Best-First Search : RNFS
- Algorithme AO* :
  - Développement
  - Mise à jour ascendante
  - Définition de la nouvelle meilleure solution

Graphe ET-OU

Problèmes qui peuvent se décomposer en éléments *mutuellement indépendants*

- **Heuristiques :**
  - Lorsque u est un nœud **ET**
    $h(u) = \Sigma_{v \in \text{Successeur}(u)}(k(u,v) + h(v))$
  - Lorsque u est un nœud **OU**
    $h(u) = min_{v \in \text{Successeur}(u)}(k(u,v) + h(v))$



Algorithme AO*

Algorithme 6 : Algorithme : OA^*(Liste, Sol)

Input : le graphe
Output : Si $f(u_0) = \infty$ pas solution sinon solution ds $G(u_0)$ & coût $f(u_0)$.
P $\leftarrow$ Premier(Liste), Q $\leftarrow \emptyset$, f(u_0) $\leftarrow$ h(u_0), u $\leftarrow u_0$;
while *(u $\neq$ null) et (f(u_0) $\neq \infty$)* **do**

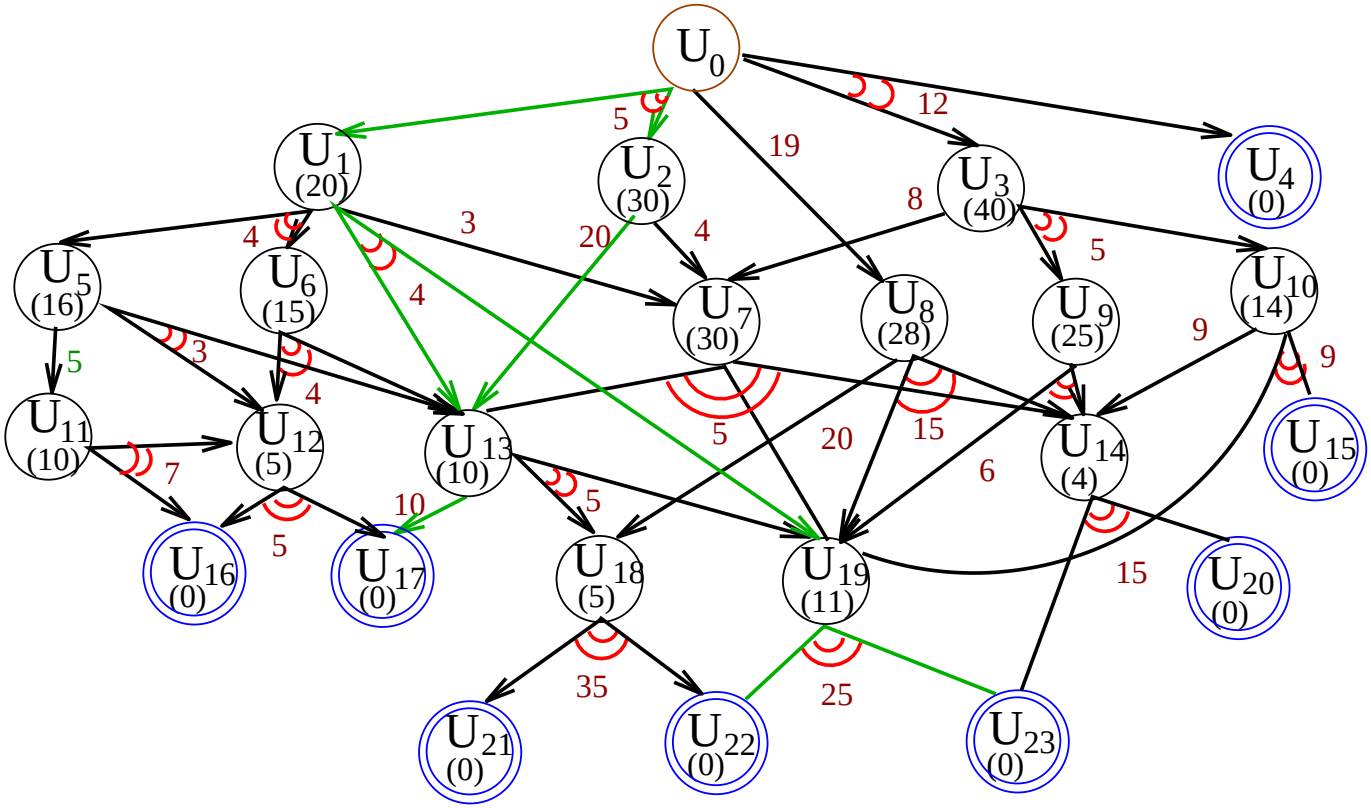
- P $\leftarrow$ P $\setminus$ {u}, Q $\leftarrow$ {u};
- **for** $i \in I(u)$ **do**
  - v $\leftarrow S_i$(u) ;
  - **while** $v \neq \emptyset$ **do**
    - **if** *non*$(v \in P \cup Q)$ **then** P $\leftarrow$ P $\cup$ {v};
    - f(v) $\leftarrow$ h(v) ;
    - v $\leftarrow S_i$(u);
  - fi(u) $\leftarrow$ k(u,i) + $\Sigma_{v \in S_i(u)}$f(v);
- **if** $I(u) = \emptyset$ **then** f(u) $\leftarrow \infty$;
- **else** f(u) $\leftarrow$ min$\{f_i(u) | i \in I(u)\}$;
- I_{min}(u) $\leftarrow$ indice_du_minimum ;
- A $\leftarrow S^{-1}$(u);
- **for** $w \in S^{-1}(u)$ **do**
  - I_0(w) $\leftarrow$ { i $\in$ I(w) | u $\in S_i$(w)};

while $A \neq \emptyset$ **do**

- Choisir v $\in$ A tel que A $\cap$ S(v) $= \emptyset$;
- A $\leftarrow$ A $\setminus$ {v};
- **for** $i \in I_0(v)$ **do**
  - f_i(v) $\leftarrow$ k(v,i) + $\Sigma_{w \in S_i(u)}$f(w);
- **if** $f_{I_{min}}(v) \neq min\{f_i(v) \mid i \in I(v)\}$ **then**
  - I_{min}(v) $\leftarrow$ indice_du_minimum ;
- **if** $f(v) \neq f_{I_{min}}(v)$ **then**
  - f(v) $\leftarrow f_{I_{min}}$(v), A $\leftarrow$ A $\cup S^{-1}$(v) ;
  - **for** $w \in S^{-1}(v)$ **do**
    - I_0(w) $\leftarrow I_0$(w) $\cup$ {i $\in$ I(w) | v $\in S_i$(w)}
- I_0(v) $\leftarrow \emptyset$;

G(u_0) $\leftarrow$ Définir_Solution(u_0),
u $\leftarrow$ Choisir_Etat(G(u_0));

Exemple - AO*

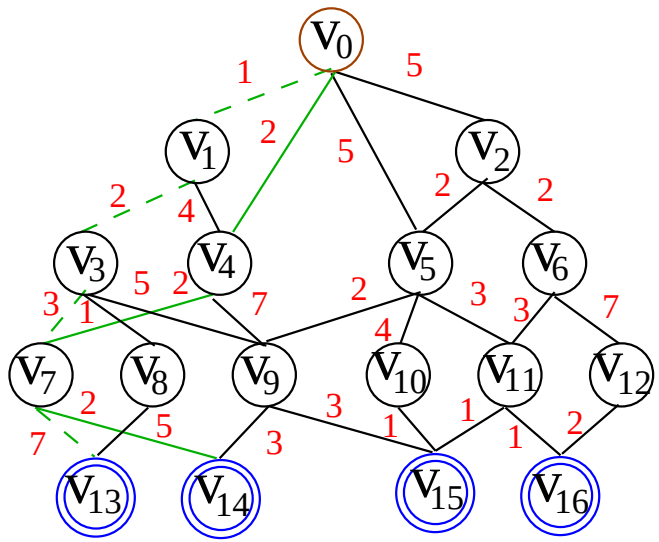


- Chemin solution : $\{U_0, U_1, U_2, U_{13}, U_{17}, U_{19}, U_{22}, U_{23}\}$,
- Coût : $f(U_0) = 74$

Branch & Bound

Algorithme 7 : Algorithme : Branch & Bound(Graphe, Cout, Sol)

```
Cout ← +∞, Sol ← {}, V ← {n}, Val(n) ← 0, Pere(n) ← ⊥;
while V ≠ {} do
    n ← prendre un élément de V ; V ← V - {n} ; if val(n)
    < Cout then
        for Chaque fils f de n do
            if val(f) > val(n) + k(f,n) then
                Val(f) ← Val(n) + k(f,n) ; Pere(f) ←
                n;
                if val(f) < Cout then
                    if f est un nœud terminal then
                        Cout ← val(f) ; Sol ←
                        chemin menant à f;
                    else V ← V ∪ {f};
                    ;
```



- Limite initiale : 13
- Limite finale : 6

Principe :
- Backtracking
- Élagage

Problème :
- Évaluation de la limite
- Complexité

Avantage :
- Complet
- Optimal

Recuit simulé

- Pourquoi : Eviter minimum local, effet plateau
- Comment : concept de température
- Approche : Récuit de la thermodynamique
- Algorithme : basé sur méthode Métropolis et distribution de Bolzmann

Algorithme :

s $\leftarrow$ s_0, t $\leftarrow$ t_0, e $\leftarrow$ E(s)
tant que t $>$ Tmin **et** e $>$ Emax
  sn $\leftarrow$ voisin(s)
  De $\leftarrow$ E(s) - E(sn)
  si Accept(De,t) **alors**
    s $\leftarrow$ sn
  t $\leftarrow$ Decroissance(t)

Algorithme Accept :

Si De $<$ 0 **alors** vraie
A $\leftarrow$ $e^{\frac{-De}{T}}$
Si Alea(0,1) $<$ A
  alors vraie
  sinon faux