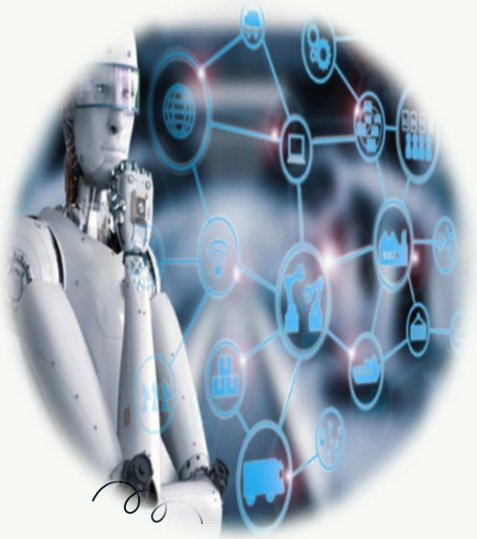
Chapitre 01
Partie 3

Dr: **Hadjadj abdelhalim**

Structure générale d'un algorithme de recherche (Exploration)



Structure générale d'un algorithme de recherche (Exploration)

Les algorithmes d'exploration examinent diverses

↓

séquences d'actions possibles à partir de l'état initial

Ont la même structure de base

ils diffèrent

par la stratégie d'exploration

Nœud de recherche

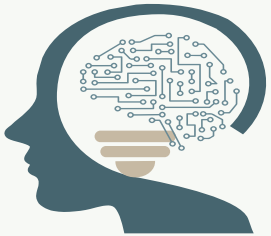
État : L'état de l'espace des états.

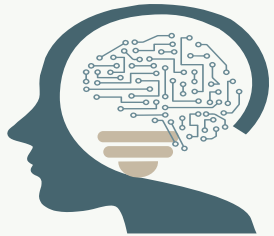
Nœud parent: nœud dans l'arbre d'exploration qui a produit ce nœud.

Action : L'action qui a été appliquée au parent pour générer ce nœud.

Coût du chemin: Coût g(n)du chemin à partir de l'état initial jusqu'à ce nœud.

Frontière : Collection des nœuds (feuilles générées, mais non encore développées)





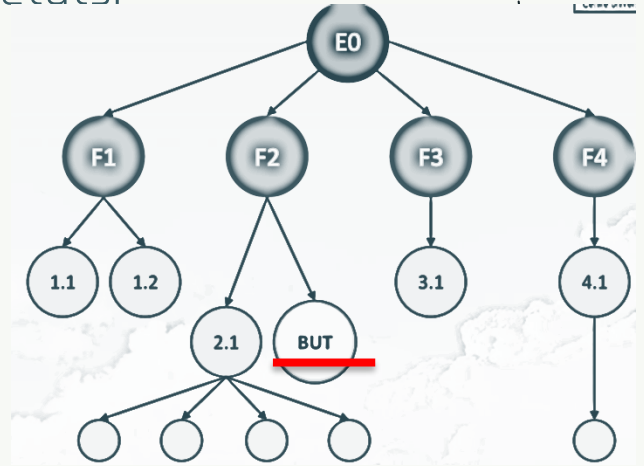
Caractéristiques de l'arbre

b : Facteur de branchement, c'est le nombre maximal de successeurs d'un nœud donné;

d : Profondeur à laquelle se trouve le meilleur nœud solution (moins d'étapes depuis la racine);

m : Longueur maximale d'un chemin dans l'espace d'états.

b ⟶ 4 d ⟶ 2 m ⟶ 3



Algorithme de recherche

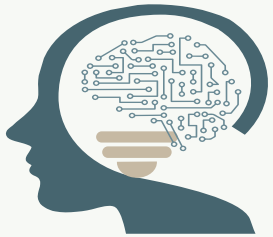
La plupart des algorithmes de recherche suivent à peu près le même schéma,

Nous commençons toujours dans l'état initial et puis nous exécutons les étapes suivantes en boucle jusqu'à terminaison :

- s'il n'y a plus d'états à traiter, renvoyez échec
- sinon, choisir un des états à traiter (X)
- si l'état est un état but, renvoyez la solution correspondante
- sinon, supprimer cet état de l'ensemble des états à traiter, et le remplacer par ses états successeurs

Ce qui va différencier les **différents algorithmes** est la manière dont on effectue le **choix de l'étape (X),**

Algorithme de recherche



Entrées : Problème et Stratégie

Sortie : Solution ou Échec initialiser

l'arbre de recherche avec l'état initial du problème

itérer

si il n'y a pas de nœud à développer alors retourner échec

choisir un nœud à développer en appliquant la stratégie

si le nœud contient un état final alors retourner la solution qui correspond

sinon

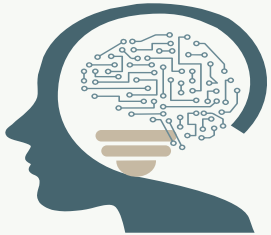
développer le nœud

ajouter les nœuds du résultat dans l'arbre de recherche

Évaluation des algorithmes de recherche

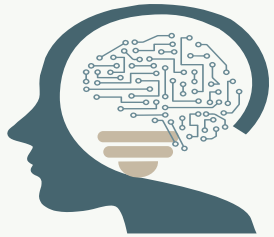
Comment pouvons-nous les comparer ?

- ✓ **Complexité en temps** : Le temps que l'algorithme prend pour trouver la solution ?
- ✓ **Complexité en espace**: Mémoire utilisée lors de la recherche d'une solution ?
- ✓ **Complétude**: SI L'algorithme trouve toujours une solution
- ✓ **Optimalité** : SI L'algorithme renvoie toujours des solutions optimales



Stratégies d'exploration

Non informée

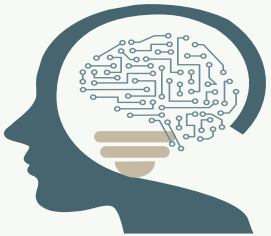


- Des algorithmes ne disposent pas d'informations supplémentaires pour pouvoir distinguer des états prometteurs,
  - Exemple: *joueurs d'échecs qui ne peuvent explorer toutes les possibilités, et se concentrent donc à chaque étape sur un petit nombre de coups qui leur semblent être les meilleurs*
- En l'absence de telles informations, ces algorithmes font une recherche exhaustive de tous les chemins possibles depuis l'état initial.

Stratégies d'exploration

Non-informée

- Largeur d'abord (Breadth first- BFS)
- Profondeur d'abord (first Depth –DFS)
- Coût uniforme (Uniform cost- UFS)
- Profondeur limitée (Depth limited -DLS)
- Itérative en profondeur (Iterative deepening)
- Bidirectionnelle (Bidirectional search)



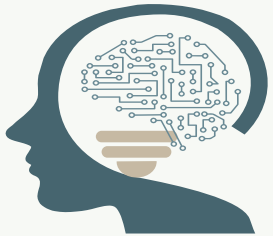
Stratégies d'exploration

Non-informée

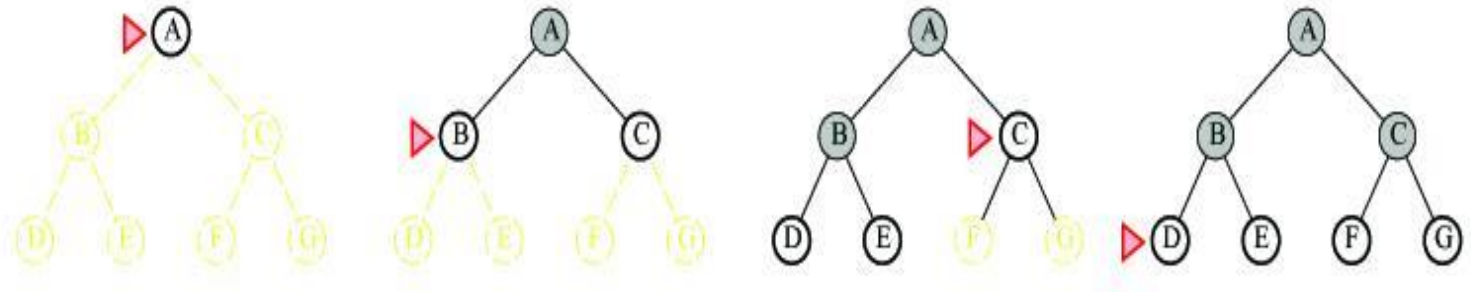
Largeur d'abord

Développer le nœud racine puis tous les nœuds successeurs, puis les successeurs des successeurs ;

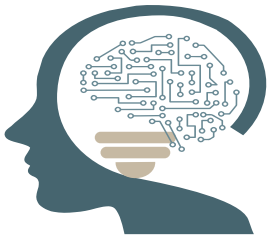
- Implémenté à l'aide d'une file ;
- Développer tous les nœuds au niveau i ;
- Développer par la suite tous les nœuds au niveau i+1 ;



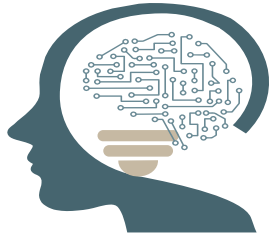
Example de Largeur d'abord (BFS)



b = 2



Largeur d'abord (BFS)

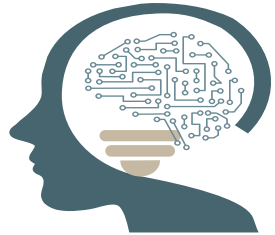


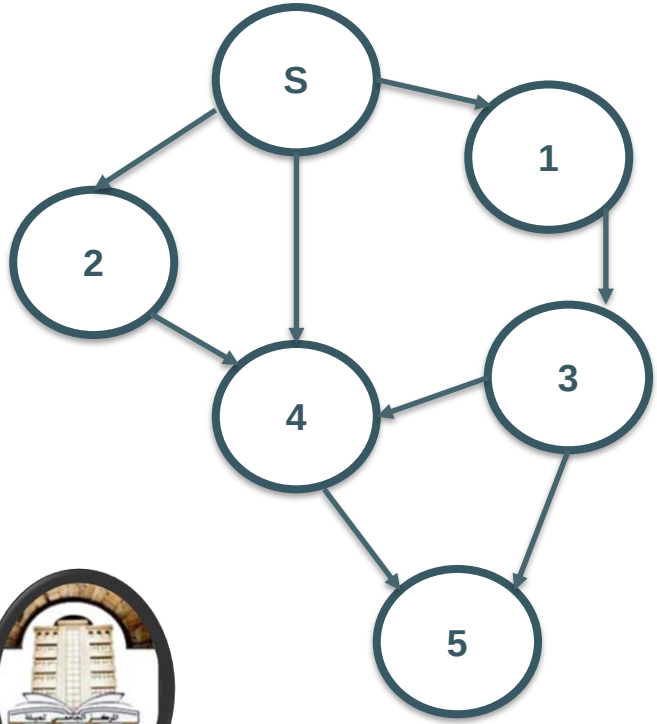
Depth	Nodes	Time	Memory
2	1110	0.11 seconds	1 megabyte
4	111100	11 seconds	106 megabytes
12	10^{13}	35 years	10 petabytes

- **b=10**
- **10,000 nodes/second**
- **1,000 bytes/node**

✓ Complétude : oui (si b est fini) ;
✓ Optimalité : pas nécessairement ;
✓ Complexité : 1+b+b2+b3+…+b d = $O(b^{d+1})$ -(Exponential le temp et escape)

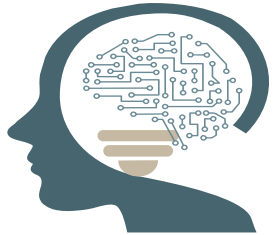
Largeur d'abord (BFS)





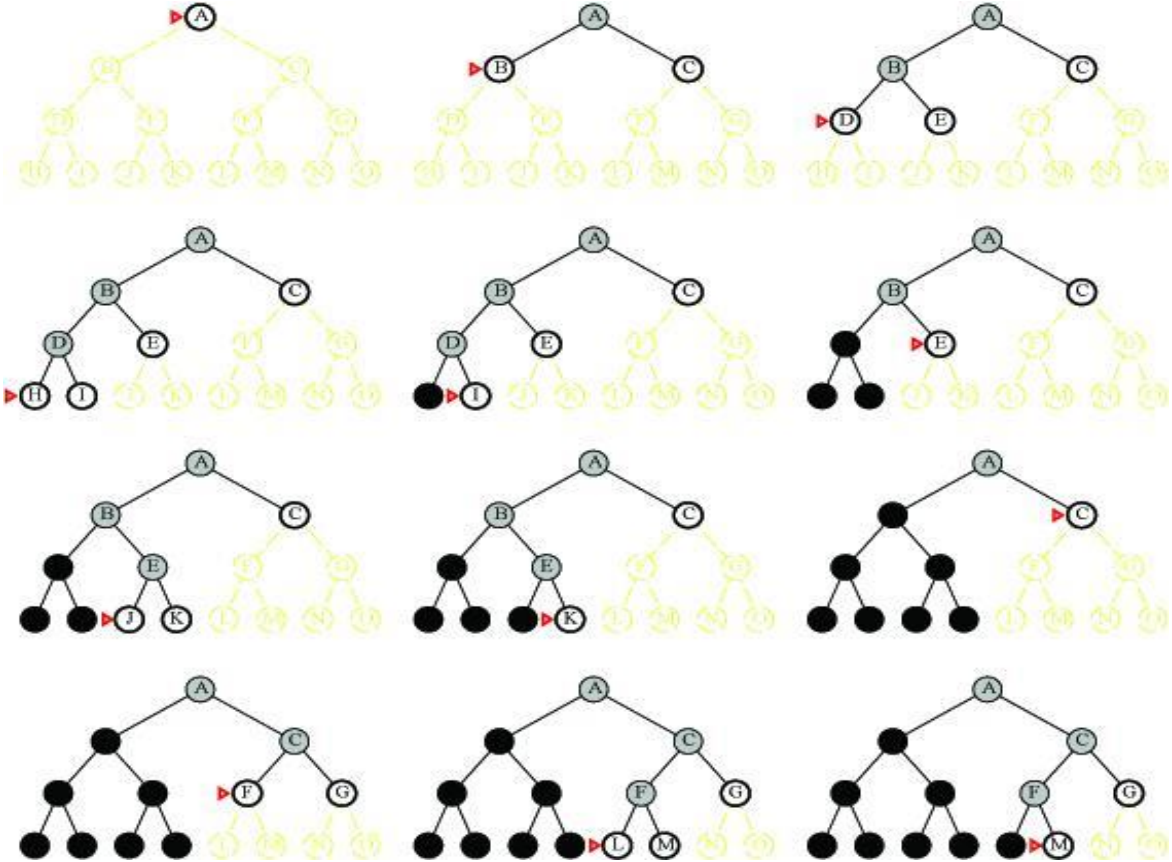
	[S]
S	[S.1,S.2, S.4]
S.1	[S.2, S.4, S.3]
S.2	[S.4, S.1.3, S.2.4]
S.4	[S.1.3, S.2.4, S.4.5]
S.1.3	[S.2.4, S.4.5, S.3.5]
S.4.5	[]

Profondeur d'abord (DFS)

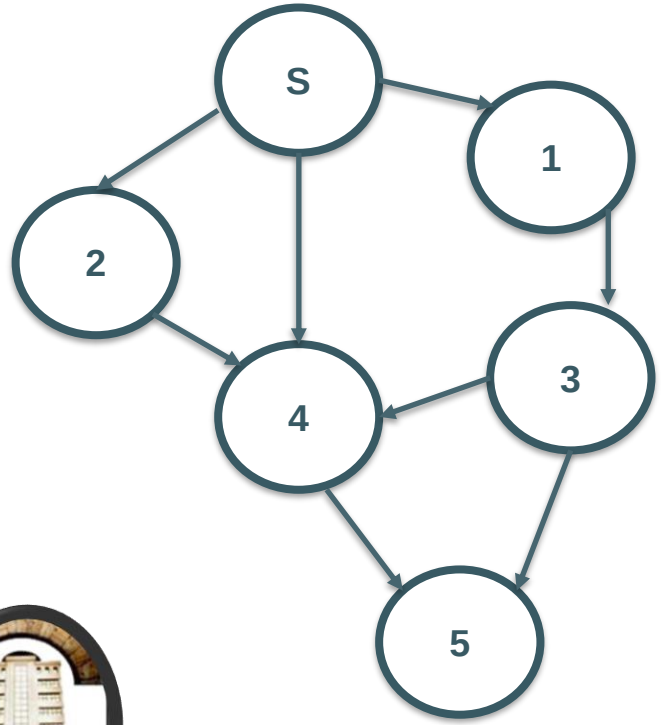


•**Exploration des sommets** :
•Chaque sommet choisi peut être exploré ainsi que tous ses successeurs avant qu'un autre sommet ne soit exploré.
•**Suivi du chemin** :
•Le parcours en profondeur suit le chemin courant aussi longtemps que possible.
•**Implémentation** :
•Pour l'implémentation, il suffit d'ajouter les successeurs du nœud courant au début de la liste des nœuds à traiter.
•**Risque** :
•L'algorithme peut risquer de développer une branche infinie en raison de la présence de cycles dans le graphe.

Example de profondeur d'abord (DFS)

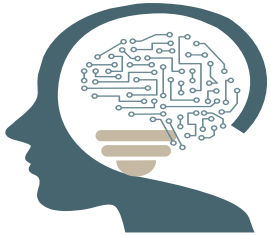


Profondeur d'abord (DFS)



	[S]
S	[S.1, S.2, S.4]
S.4	[S.1.3, S.2, S.4.5]
S.4.5	[]

profondeur d'abord (DFS)



- ✓Complétude : Non (si b est fini) ;
- ✓Optimalité : Non ;
- ✓Complexité :

$1+b+b2+b3+\ldots+bm$ (levels total) = $O(b^m)$,

pour l'espace (bm)

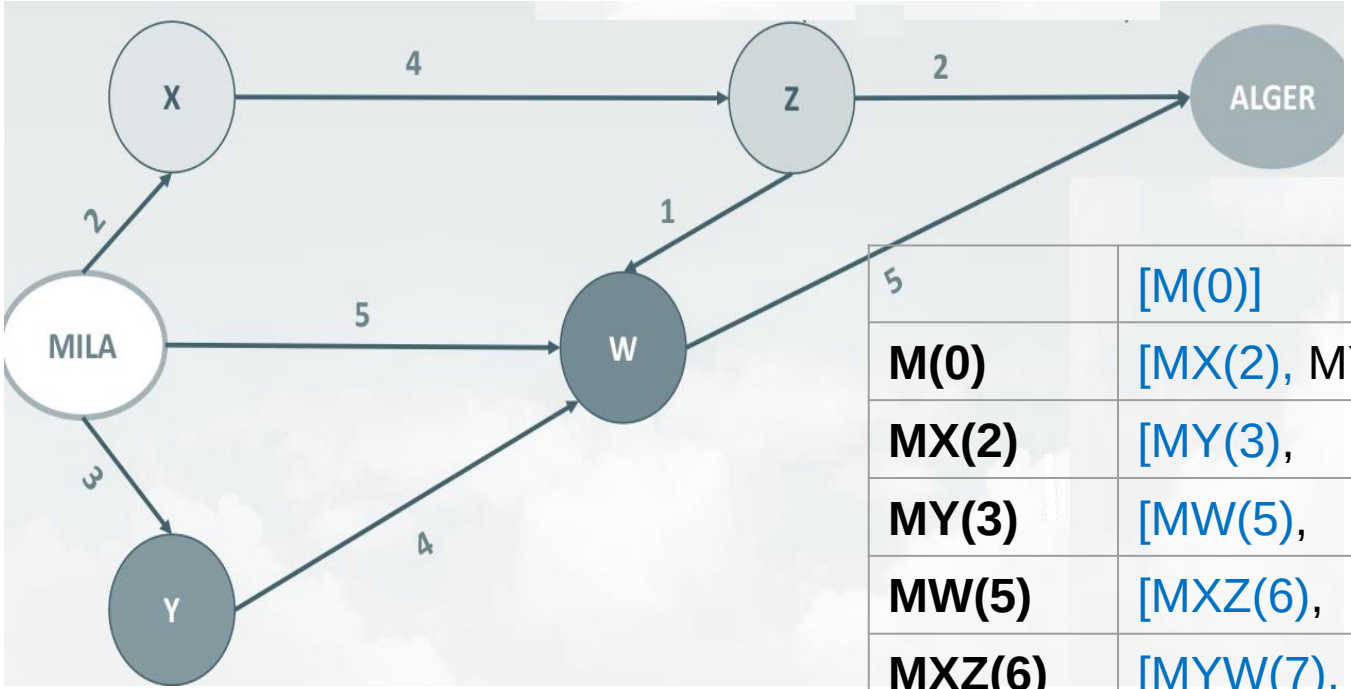
- ✓ Pour la première exemple, DFS demande 118kb par rapport de 10 pétabytes de d=12 (10 moins de billion temps)

Coût uniforme (Uniform-cost - UFS)

- Chaque étapes ayant un cout plus grand à zéro
- Développe le nœud ayant le coût le plus faible ;
- Coût g(n) : Coût depuis le nœud racine jusqu'au nœud actuel n.
- File triée selon le coût ;
- Si le coût des actions est toujours le même ;

Équivalent à largeur d’abord ! Si tous les coûts d'étape sont égaux

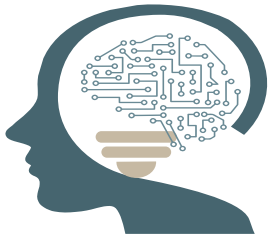
Coût uniforme (Uniform-cost - UFS)



	[M(0)]
M(0)	[MX(2), MY(3), MW(5)]
MX(2)	[MY(3), MW(5), MXZ(6)]
MY(3)	[MW(5), MXZ(6), MYW(7)]
MW(5)	[MXZ(6), MYW(7), MWA(10)]
MXZ(6)	[MYW(7), MXZW(7), MXZA(8), MWA(10)]
MXZA(8)	[]

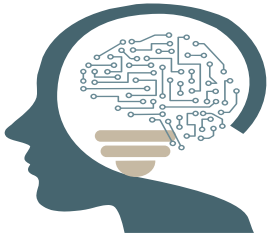
Coût uniforme (Uniform-cost - UFS)

- Complète : oui.
- Complexité en temps :
  - ✓ C* est le coût de la solution optimale
  - ✓ ε le coût minimal de chaque action.
  - ✓ $O(b^{[c*/\varepsilon]})$ ou $O(b^{m})$
- Complexité en espace : même que celle en temps,
- Optimal : oui (les nœuds sont développés dans l'ordre de coût de chemin optimal)



Stratégies d'exploration

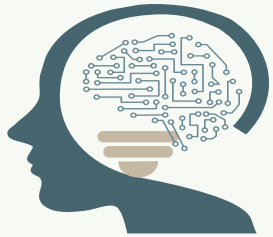
non informées



Non informées sont généralement

- Très peu efficaces
- Ne savent pas si elles approchent du but,
- Trop gourmandes en mémoire et/ou en temps,

Stratégies d'exploration

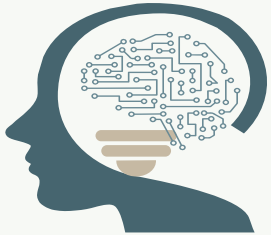


Informé

« Les algorithmes de recherche informée (ou heuristiques) utilisent des informations supplémentaires pour guider la recherche. Ces informations aident à distinguer les états prometteurs des moins prometteurs, ce qui améliore l'efficacité de la recherche. »

Algorithme de recherche heuristique dispose d'une fonction d'évaluation f qui détermine l'ordre dans lequel les nœuds sont traités.
La liste de nœuds à traiter est organisée en fonction des f-valeurs des nœuds, avec les nœuds de plus petite valeur en tête de liste.

Stratégies d'exploration



Informé

- Meilleur d'abord (BFS - Best-first)
  - Meilleur d'abord gloutonne (Greedy best-first)
  - A* (A-Star)
- Algorithmes heuristiques à mémoire limitée
- IDA*, RBFS et SMA*
- Méta-heuristique
  - -Par escalade (Hill-climbing)
  - -Par recuit simulé (Simulated annealing)
  - -Exploration locale en faisceau (Local beam)
  - -Algorithmes génétiques

Informée

heuristiques

Heuristiques--doit guider le choix des états à tester ---les ordonner selon leurs promesses (plus proche d'un but)

- Dépend fortement du problème à traiter ;

  - Une heuristique pauvre basée sur des propriétés trop simples du problème (peu efficace) ;

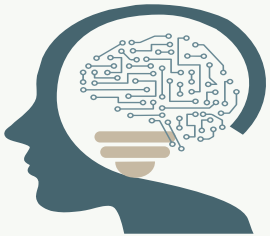
  - Une heuristique riche basée sur des propriétés approfondies du problème (efficace), MAIS difficile à établir

Stratégies d'exploration

Informé

Plus efficaces que l'exploration aveugle (non-informée):

- ✓ Guidage de la Recherche: utilisent des connaissances du problème (fonction heuristique) ;
- ✓ Estimation pour choisir un nœud à visiter (Optimisation du Temps et de l'Espace)
- ✓ Flexibilité : peuvent être adaptées à des problèmes spécifiques

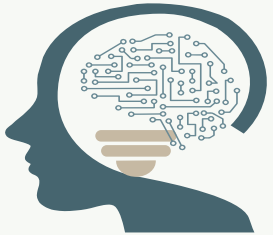


Informé

Exploration A*

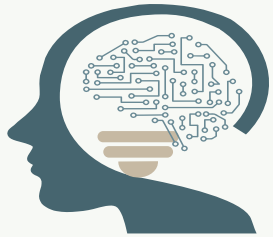
Fonction d’évaluation f :

- f(n)= g(n)+ h(n)
- f(n): coût total estimé de la solution la moins couteuse passant par n
- g(n): coût jusqu’a présent pour attendre nœud n
- h(n): coût estimé du nœud n jusqu’au but
- f(Y)= g(Y) + h(Y)



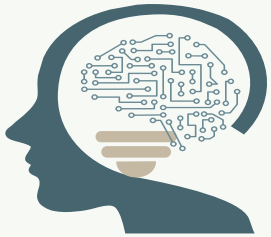
Informé

Exploration A*



h(n) : coût estimé du nœud n jusqu'au but :

- Distance Euclidienne (vol d'oiseau) entre les villes
- N-puzzle tuiles mal placées ou distances des tuiles
- Qualité d'une configuration par rapport à une autre

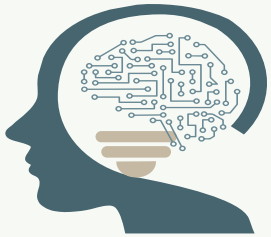


Informé

Exploration A*

Heuristique Admissible

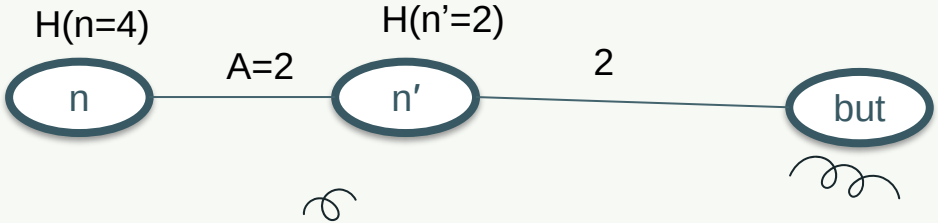
- Une heuristique h(n) est **admissible** si elle **ne surestime jamais** le coût réel pour atteindre le but à partir du nœud n.
- **Formule** :
- h(n)≤ cout réel minimal pour atteindre le but a` partir de
- **Exemple** : Si le coût réel minimal pour atteindre le but à partir du nœud n est de 10, alors h(n) doit être inférieur ou égal à 10.

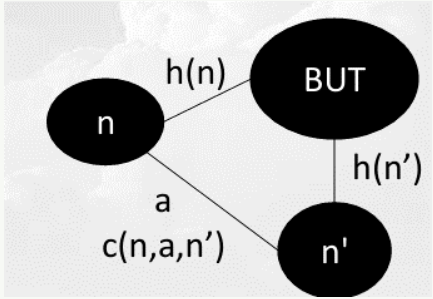


Informé

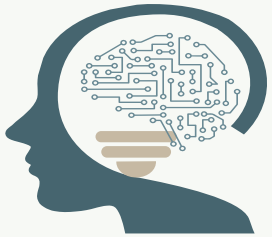
Exploration A*

Une heuristique est **consistante** (ou respecte la monotonie) si, pour tout nœud n et chaque successeur n′ de n obtenu en appliquant une action a, l'inégalité suivante est satisfaite :
$h(n) \leq c(n,a,n') + h(n')$ où $c(n,a,n')$ est le coût pour passer de n à n′ en utilisant l'action a.





Informée

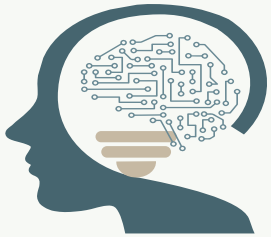


Exploration A*

- Complétude: oui si b est fini
- Optimale:
  - Arbre: oui si h(n) est admissible.
  - Graphe: oui si h(n) est consistante.
- **Complexité temps:** exponentielle, selon la longueur de la solution optimale.
- **Complexité en espace :** exponentielle, selon la longueur de la solution optimale , elle garde tous les nœuds en mémoire.
- *(Habituellement, on manque d'espace bien avant de manquer de temps)*

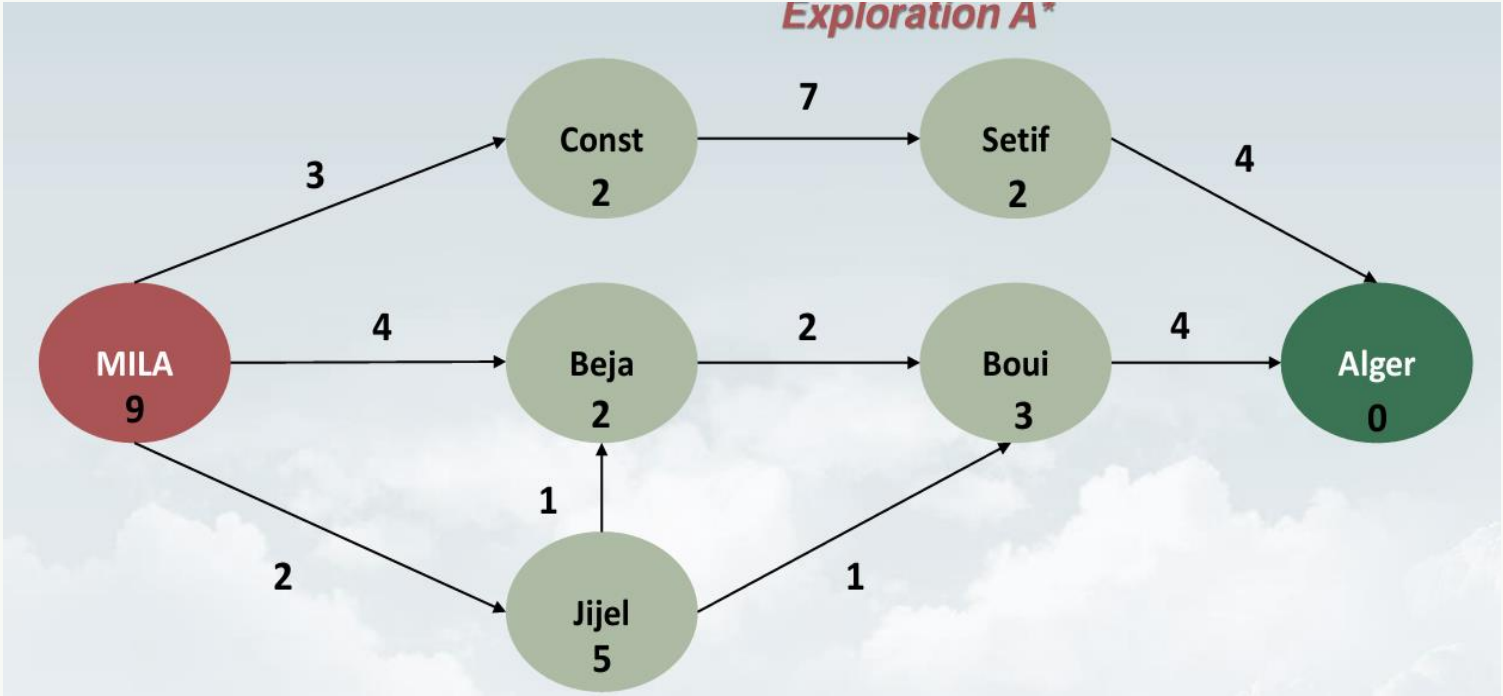
Informé

Exploration A*

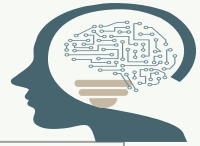


- Les nœuds dans OPEN sont triés selon leur valeur estimée.
- Pour chaque nœud n, f(n) est un nombre réel positif ou nul qui estime le coût du meilleur chemin partant du nœud initial, passant par n, et arrivant au but.
- Dans OPEN, les nœuds sont ordonnés en fonction de leurs valeurs f(n), les nœuds les plus prometteurs étant explorés en premier.
- Dans CLOSED, les nœuds déjà traités sont conservés pour éviter les répétitions et faciliter le chemin optimal.
- Les nœuds dans CLOSED sont également triés de manière croissante selon leurs valeurs f(n).

Exploration A*



Exploration A*



Closed	Open
-	[Mila(9)]
[Mila(9)]	[Mila_const(5), Mila_bejai(6), mila_jijel(7)]
[Mila(9), Mila_const(6)]	[Mila_bejai(6), mila_jijel(7), const_setif(12)]
[Mila(9), Mila_const(6) Mila_bejai(6)]	[mila_jijel(7), beja_Boui (9) ,const_setif(12)]
[Mila(9), Mila_const(6) Mila_bejai(6), mila_jijel(7)]	[~~jijel-bejai(5),~~jijel_bouira(6),bejai_Boui (9) ,const_setif(12)]
[Mila(9), Mila_const(6) Mila_bejai(6), mila_jijel(7), jijel_bouira(6)]	[boui_alg(7), beja_Boui (9), const_setif(12)]
[Mila(9), Mila_const(6) Mila_bejai(6), mila_jijel(7), jijel_bouira(6), boui_alg(7)]	[]

Mila, jijel, bouira, alger

Question?