
بسم الله الرحمن الرحيم
السلام عليكم ورحمة الله وبركاته

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Abdelhafid Boussouf University Center of Mila
1ère année Master

Cours Intelligence Artificielle et ses Applications

Chapitre 01
Partie 3

Responsable module

Dr. MEGUEHOUT Hamza

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 : Le 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)

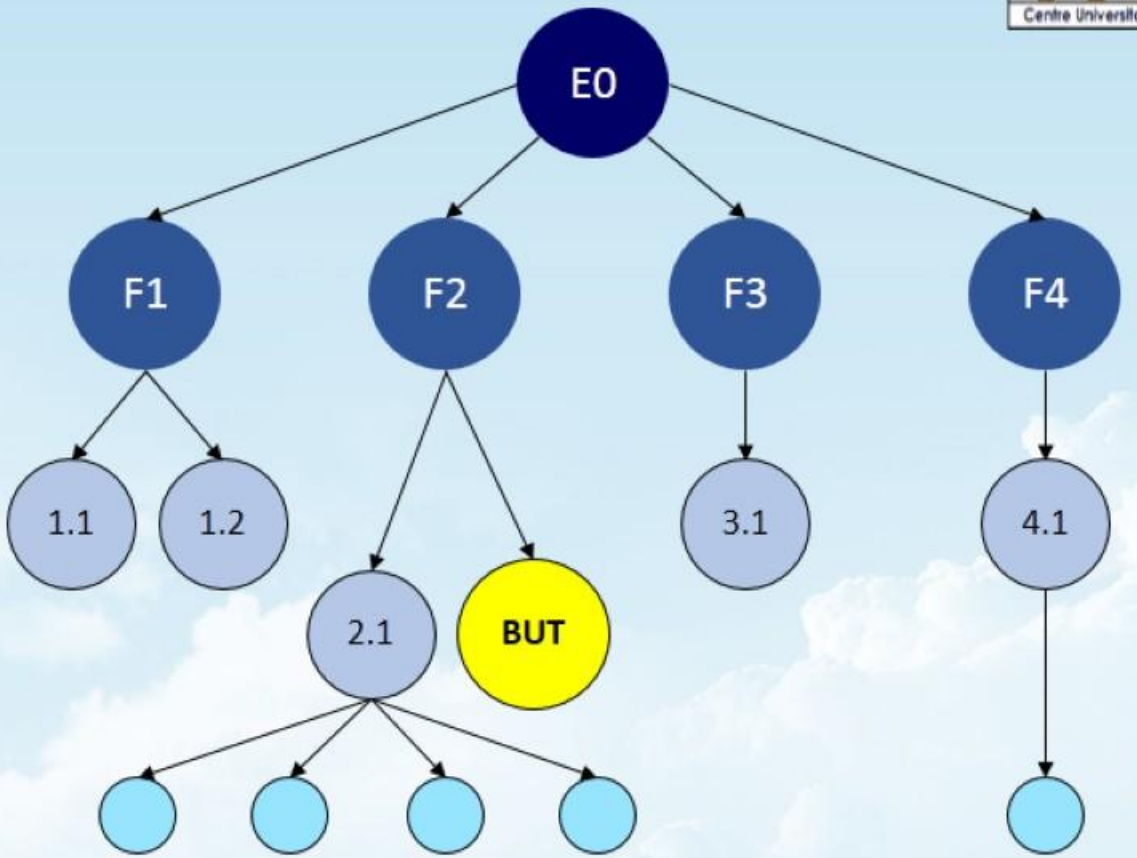
Caractéristiques de l'arbre

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



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 s'il y en a une
- Optimalité : SI L'algorithme renvoie toujours des solutions optimales

Algorithme de recherche

Existe plusieurs formulation

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

Stratégies d'exploration

Types algorithmes de recherche

Non informée

Ils ne disposent pas d'informations supplémentaires pour pouvoir distinguer des états prometteurs *ce n'est pas le cas par exemple des programmes 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.

Informée

Utilisent des informations supplémentaires pour pouvoir mieux guider la recherche.

Tout 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.

Non-informée

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

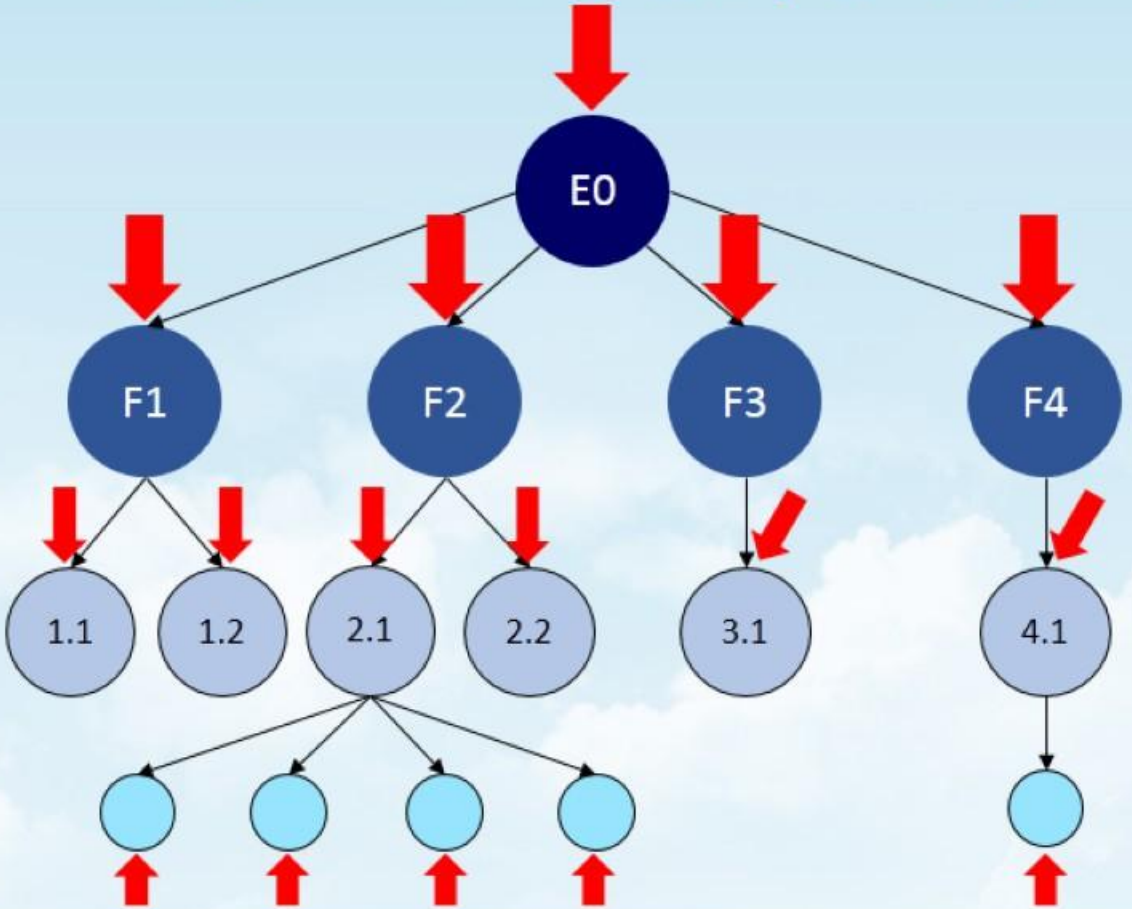
Non-informée

Largeur d'abord (*Breadth-first - DFS*)

- ✓ 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 ;
- ✓ Graphe : on rechercherait tous les points dans un rayon circulaire fixe, augmentant ce cercle pour rechercher des intersections de plus en plus loin du nœud initial ;
- ✓ Complétude : oui (si **b** est fini) ;
- ✓ Optimalité : pas nécessairement ;
- ✓ Complexité : $1+b+b^2+b^3+...+b^d = O(b^d)$.

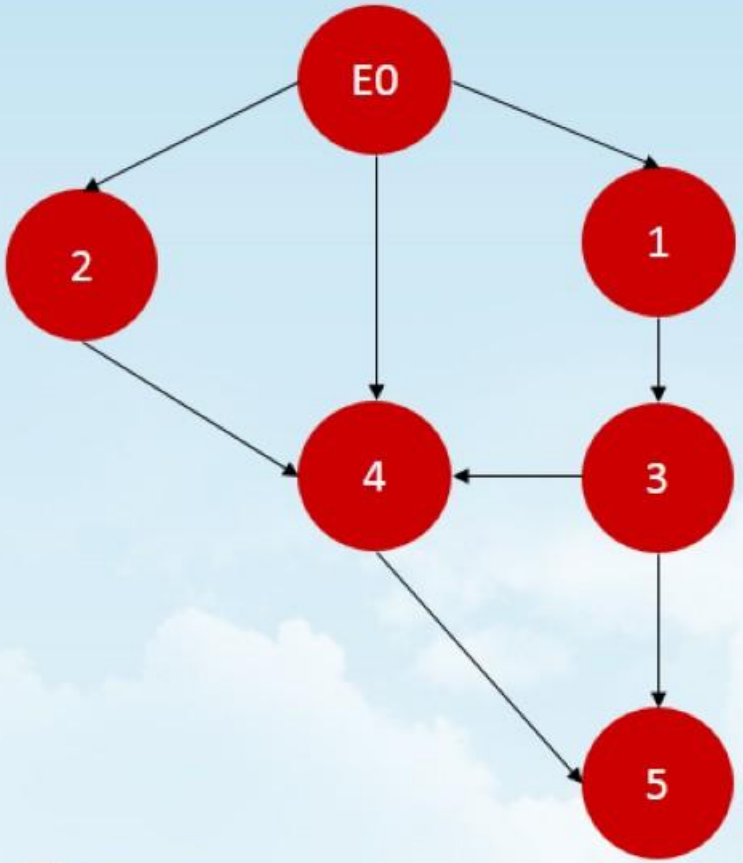
Non-informée

Largeur d'abord (*Breath-first - DFS*)



Non-informée

Largeur d'abord (*Breath-first - DFS*)



	E0
E0	E0.1 – E0.2 – E0.4
E0.1	E0.2 – E0.4 – E0.1.3
E0.2	E0.4 – E0.1.3 – E0.2.4
E0.4	E0.1.3 – E0.2.4 – E0.4.5
E0.1.3	E0.2.4 – E0.4.5 – E0.1.3.4 – E0.1.3.5
E0.4.5	

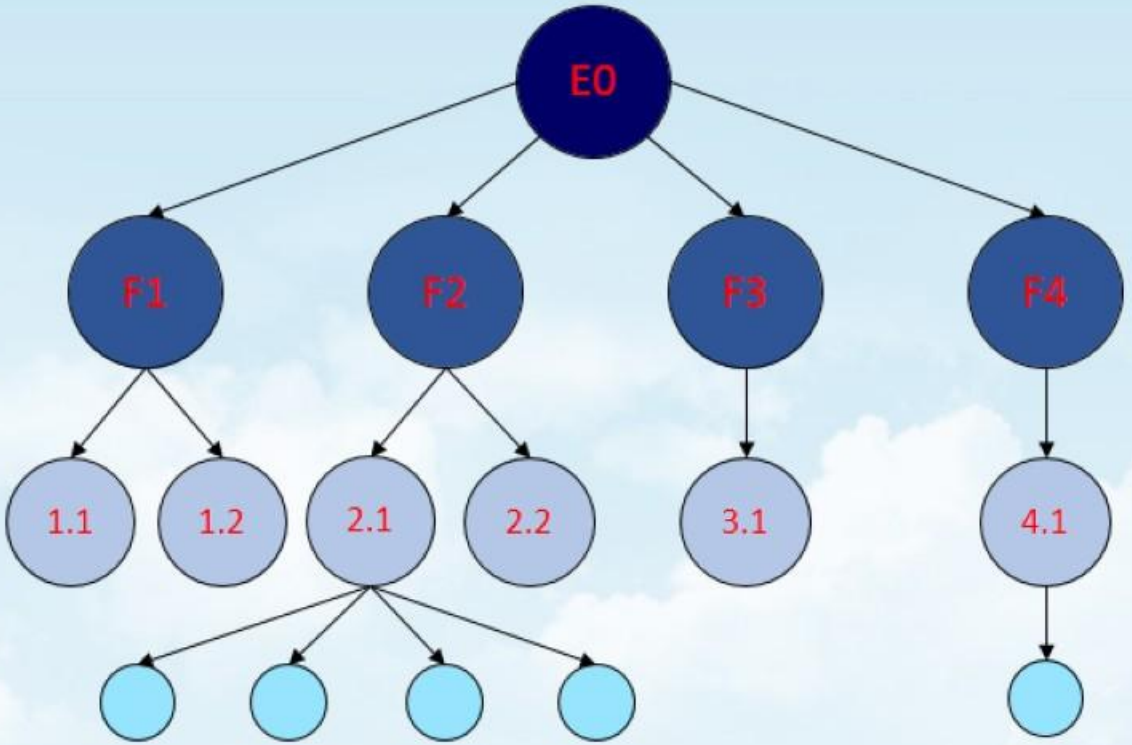
Non-informée

Profondeur d'abord (*Depth-first - DFS*)

- ✓ Développe le nœud le plus profond ;
- ✓ Implémenté à l'aide d'un pile (last-in-first-out) ;
- ✓ Complétude :
  - Non si la profondeur est infinie, s'il y a des cycles.
  - Oui, si on évite les états répétés ou si l'espace de recherche est fini.
- ✓ Complexité en temps : $O(b^m)$
  - Très mauvais si m est plus grand que d.
  - Mais si les solutions sont denses, il peut être beaucoup plus rapide que largeur d'abord.
- ✓ Complexité en espace : O(bm), linéaire
- ✓ Optimal : Non

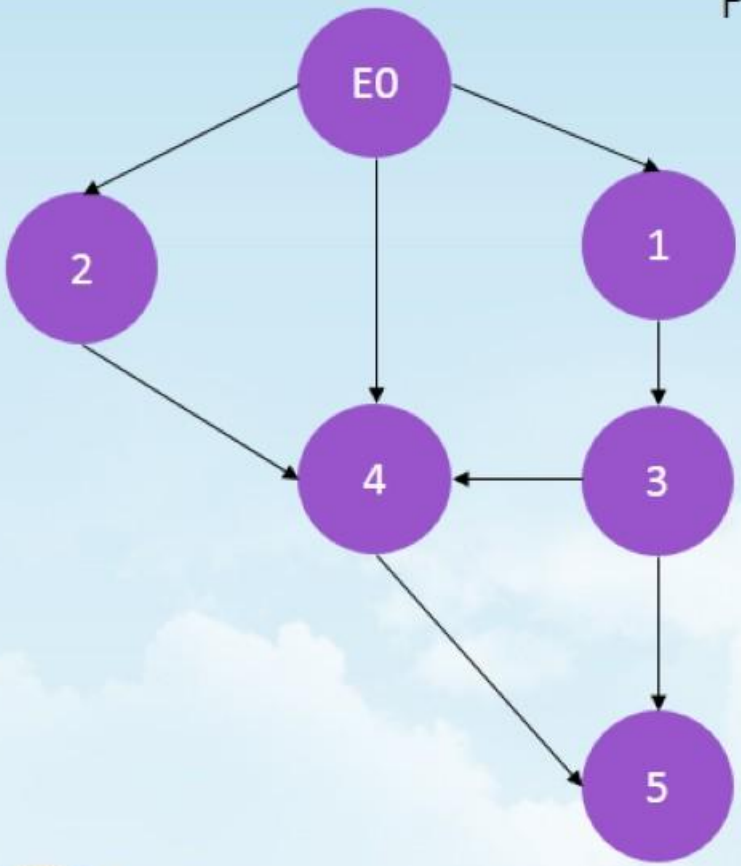
Non-informée

Profondeur d'abord (*Depth-first - DFS*)



Non-informée

Profondeur d'abord (*Breath-first - DFS*)



	E0
E0	E0.1 – E0.2 – E0.4
E0.4	E0.2 – E0.4 – E0.4.5
E0.4.5	

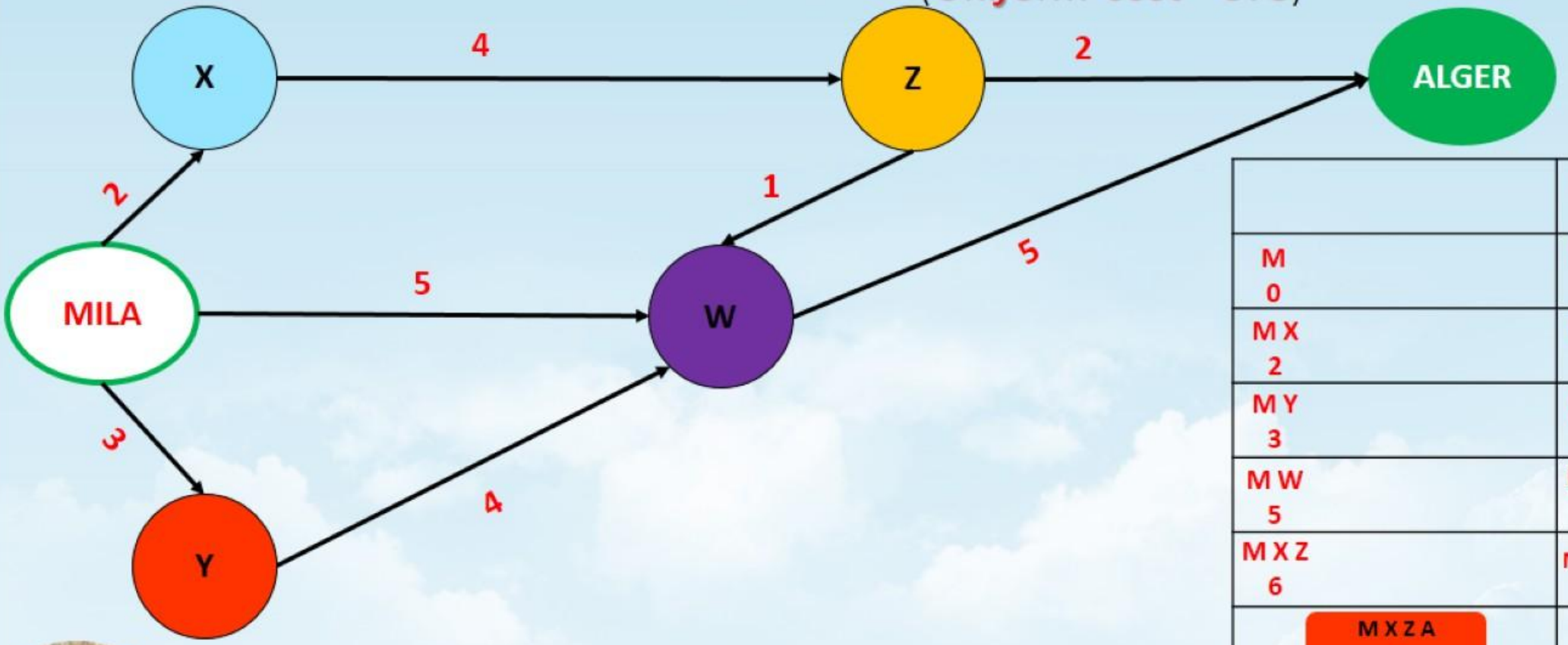
Non-informée

Coût uniforme (*Uniform-cost - UFS*)

- ✓ Développe le nœud ayant le coût le plus faible ;
  - **g(n)** = coût du nœud initial au nœud développé.
- ✓ 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
- ✓ Complète : **oui**, si le coût > ε (éviter les chemins infinis d'actions à coût nul).
- ✓ Complexité en temps :
  - C^* est le coût de la solution optimale
  - ε le coût minimal de chaque action.
  - $O\left(b^{\lfloor c^*/\varepsilon \rfloor}\right)$
- ✓ Complexité en espace : même que celle en temps
- ✓ Optimal : **oui** → les nœuds sont développés dans l'ordre de coût de chemin optimal

Non-informée

Coût uniforme (*Uniform-cost - UFS*)



	M 0
M 0	M X 2 M Y 3 M W 5
M X 2	M Y 3 M W 5 M X Z 6
M Y 3	M W 5 M X Z 6 M Y W 7
M W 5	M X Z 6 M Y W 7 M W A 10
M X Z 6	M Y W 7 M X Z W 7 M X Z A 8 M W A 10
M X Z A 8	

Non-informée

Stratégies d'exploration **non informées** sont généralement

Très peu efficaces

Ne savent pas si elles approchent du but

Trop gourmandes en mémoire et/ou en temps

Informée

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

Informée
heuristiques

Heuristique → 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**.

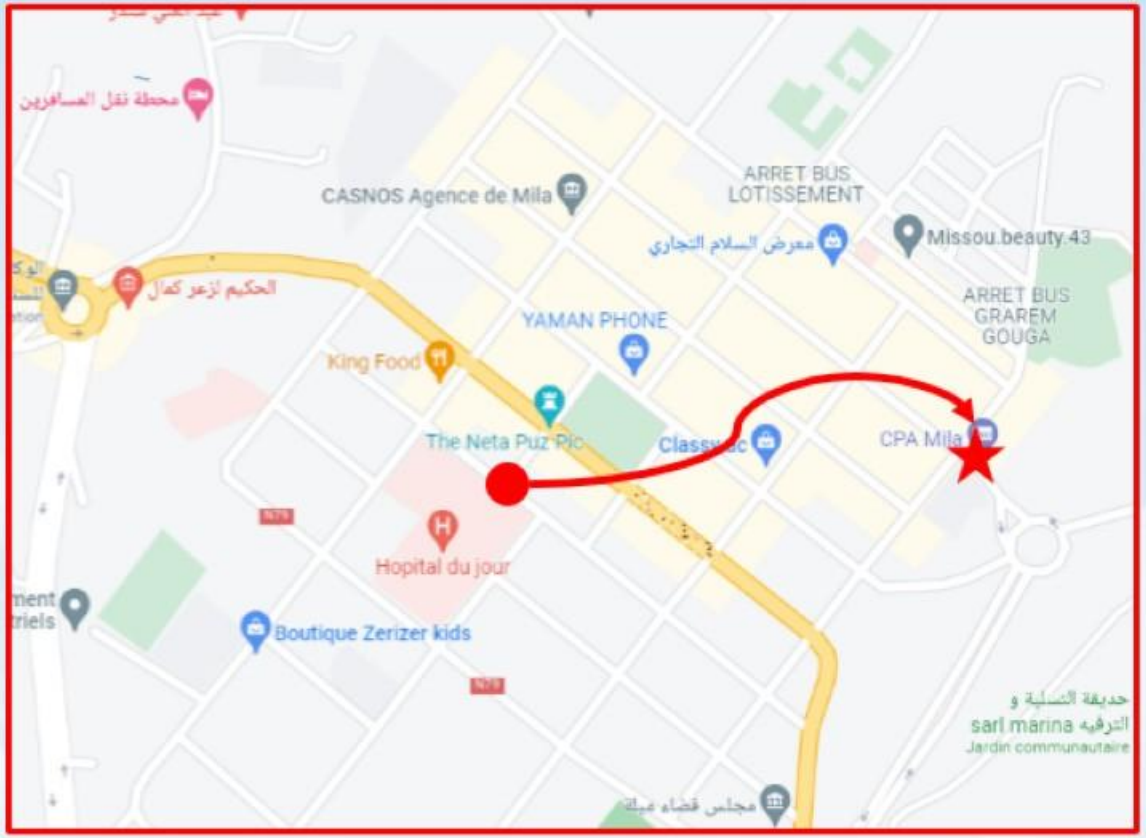
Informée
heuristiques

Stratégies d'exploration **informées** :

- Utilisent des **connaissances du problème** : une fonction heuristique ;
- **Estimation** pour choisir un nœud à visiter ;
- Plus **efficaces** que l'exploration aveugle (non-informée).

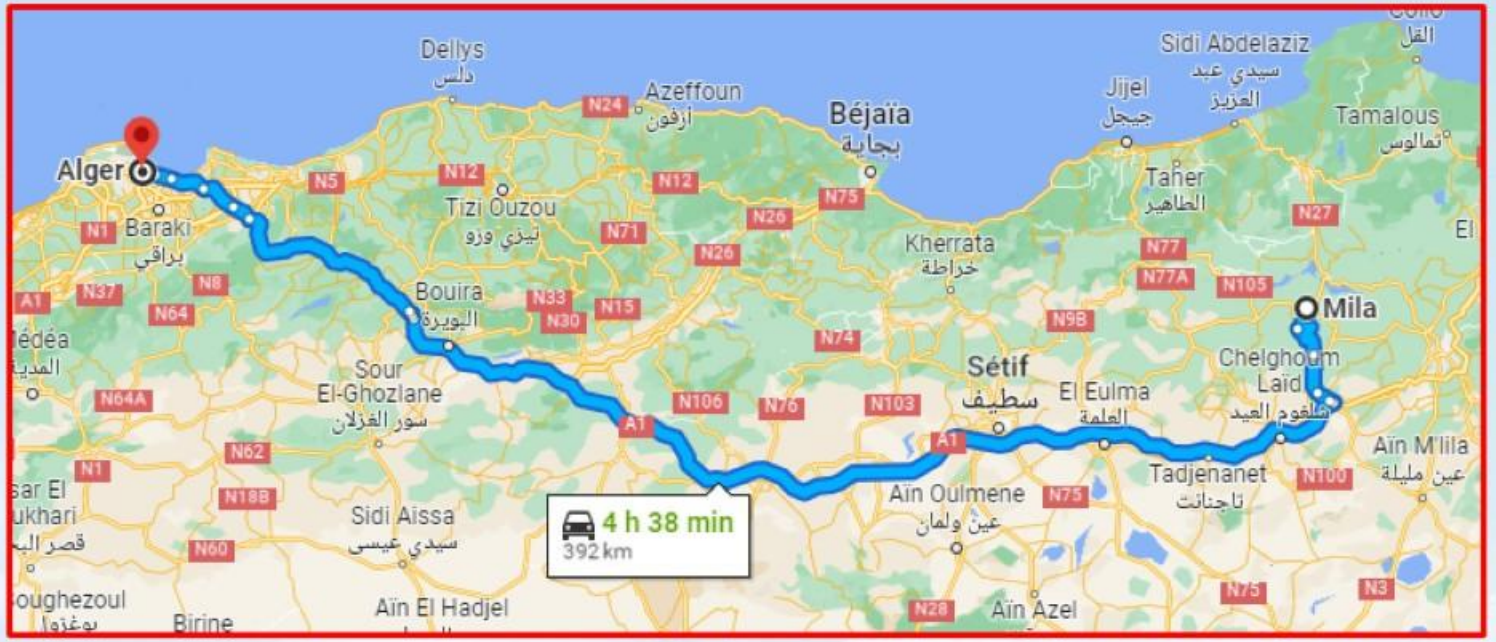
Exemples

Chemin dans la ville



Exemples

Google Maps

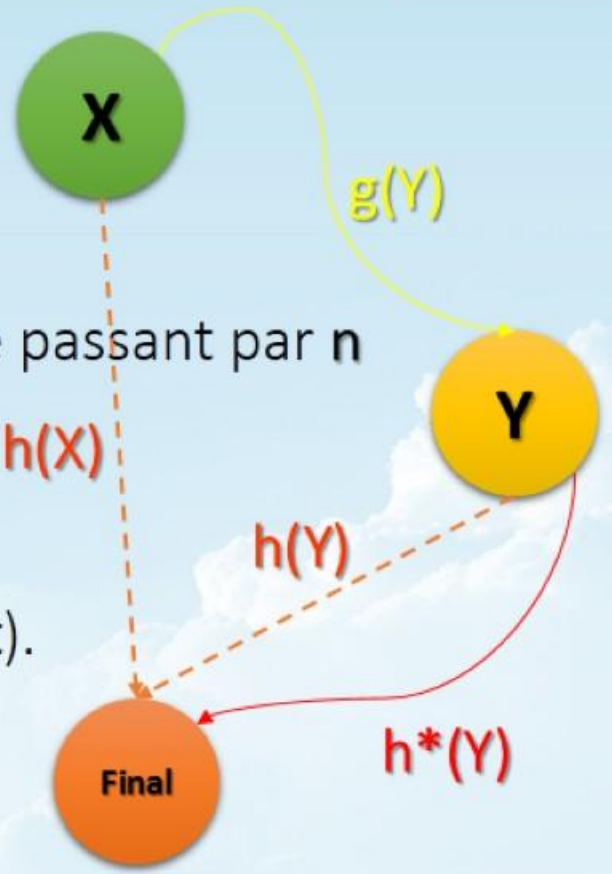


Informée

*Exploration A**

- Fonction d'évaluation **f** :
  - ✓ **f(n) = g(n) + h(n)**
  - ✓ **f(n)** est le coût total estimé de la solution la moins couteuse passant par **n**
  - ✓ **g(n)** : coût du nœud de **départ** jusqu'au nœud **n**
  - ✓ **h(n)** : coût **estimé** du **nœud n** jusqu'au **but**
  - ✓ **h*(n)** : coût réel (**somme** des coûts du **nœud n** au **nœud but**).

$$f(Y) = g(Y) + h(Y)$$



Informée

*Exploration A**

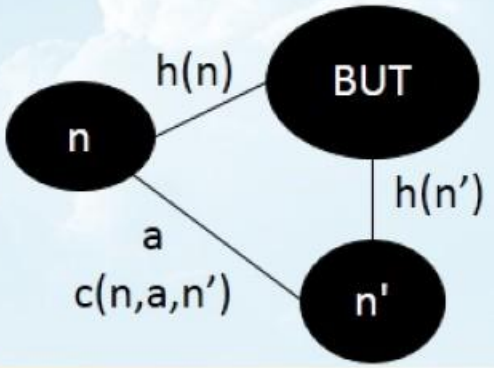
h(n) : coût **estimé** du **nœud n** jusqu'au **but :**

- 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

Informée

*Exploration A**

- Doit utiliser une heuristique **h(n)** **Admissible**.
  - ✓ Une heuristique **h** est admissible si elle **ne surestime jamais le coût réel (véritable coût)** pour atteindre le but : **$h(n) \leq h^*(n)$**
- **Consistance** (monotonie) : pour l'exploration de graphes :
  - ✓ **h(n)** est consistante si pour nœud **n** et chaque successeur **n'** de **n**, produit par une action **a**, **$h(n) \leq c(n,a,n') + h(n')$**



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 de 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.

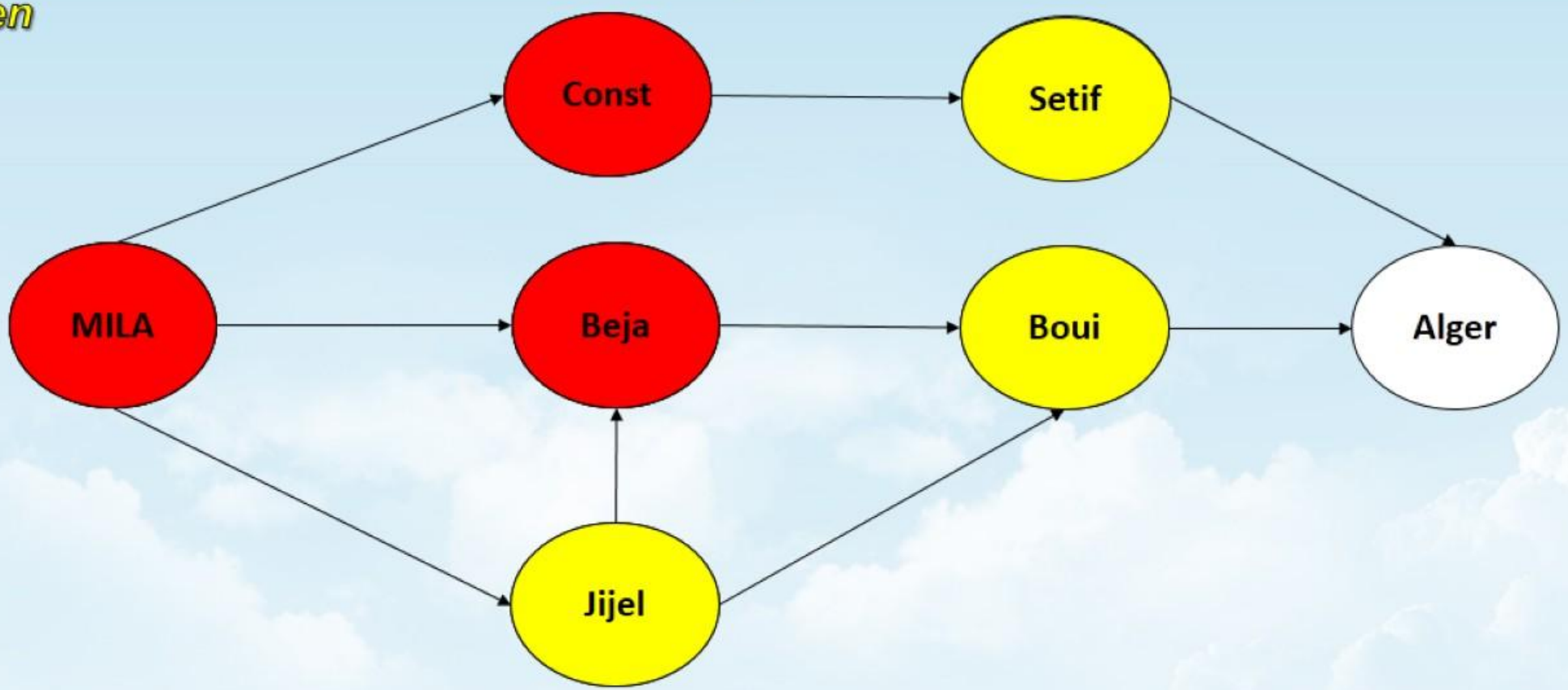
(*Habituellement, on manque d'espace bien avant de manquer de temps*)

Informée

*Exploration A**

Closed

Open



Informée

*Exploration A**

- ✓ Les nœuds n dans OPEN sont triés selon l'estimé f(n) de leur « valeur »
- ✓ Pour chaque nœud **n**, **f(n)** est un nombre réel positif ou nul, estimant le coût du meilleur chemin partant du nœud initial, passant par **n**, et arrivant au **but**
- ✓ Dans OPEN, les nœuds se suivent en **ordre croissant** selon les **valeurs f(n)** On explore les nœuds les plus « **prometteurs** » en premier
- ✓ Dans CLOSED, les nœuds déjà traités,

Informée

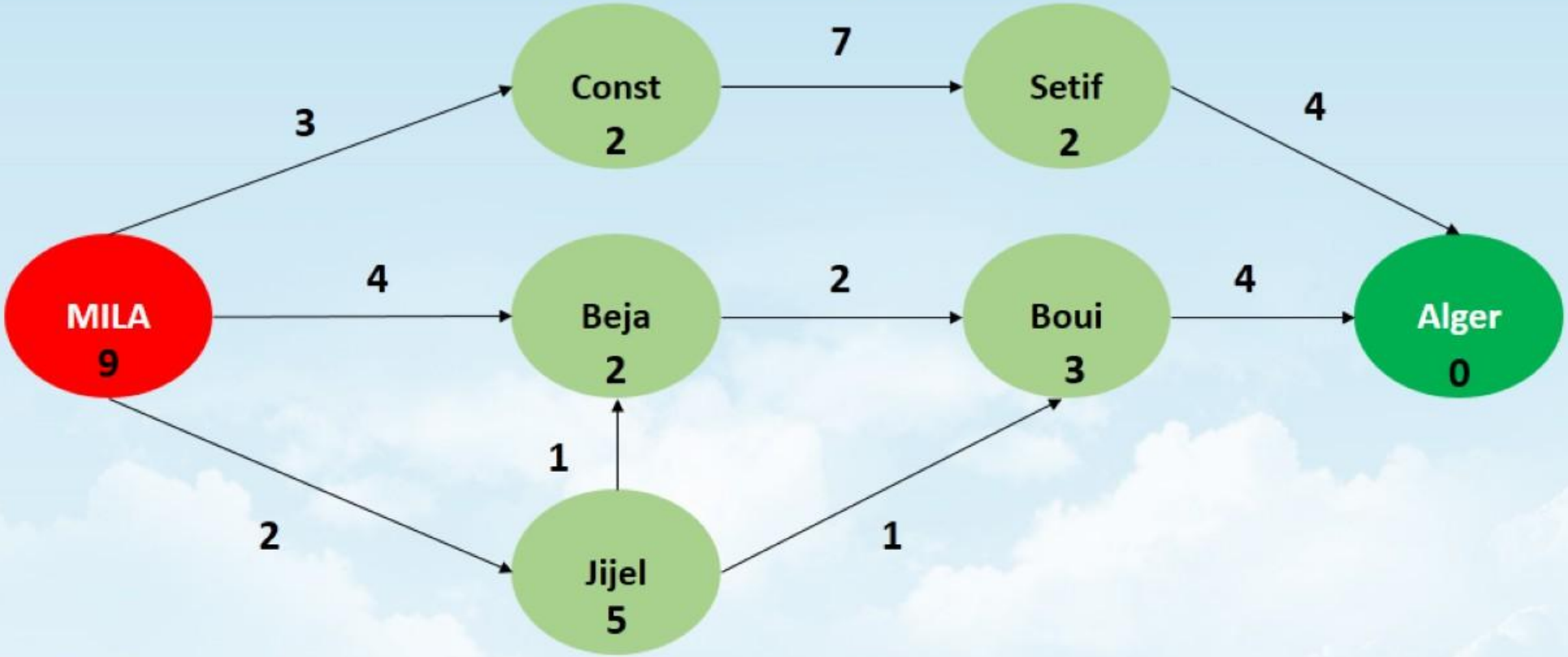
*Exploration A**

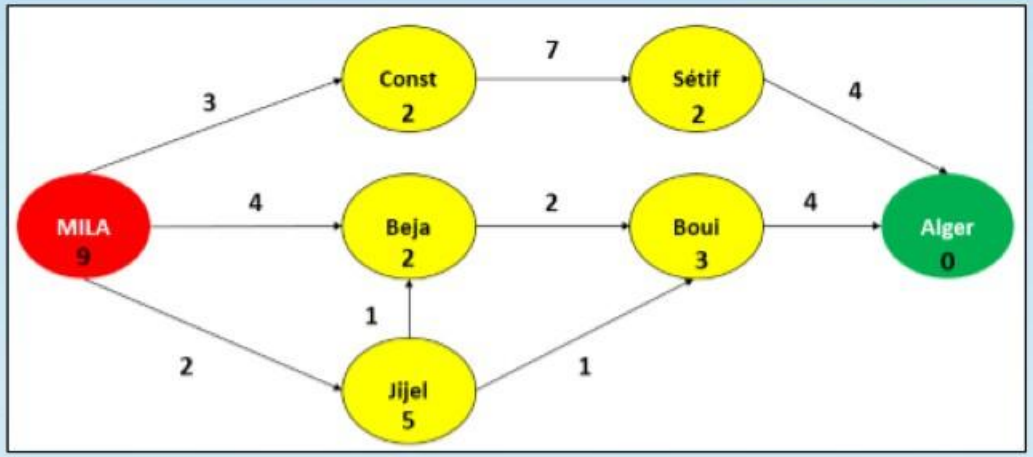
Algorithme RECHERCHE-DANS-GRAPHE(*noeudInitial*)

1. déclarer deux nœuds : *n*, *n'*
2. déclarer deux listes : *open*, *closed* // *toutes les deux sont vides au départ*
3. insérer *noeudInitial* dans *open*
4. tant que (1) // *la condition de sortie (exit) est déterminée dans la boucle*
   5. si *open* est vide, sortir de la boucle avec échec
   6. *n* = nœud au début de open;
   7. enlever *n* de *open* et l'ajouter dans *closed*
   8. si *n* est le but (*goal*(*n*) est *true*), sortir de la boucle avec succès en retournant le chemin;
   9. pour chaque successeur *n'* de n (chaque *n'* appartenant à *transitions*(*n*))
      10. initialiser la valeur $g(n')$ à $g(n) + c(n,n')$
      11. mettre le parent de *n'* à *n*
      12. si *closed* ou *open* contient un nœud *n''* égal à *n'* avec $f(n') \leq f(n'')$
          13. enlever *n''* de *closed* ou *open* et insérer *n'* dans *open* (ordre croissant selon $f(n)$)
      11. si *n'* n'est ni dans *open* ni dans *closed*
          15. insérer *n'* dans *open* (ordre croissant selon $f(n)$)

Informée

*Exploration A**





Informée

*Exploration A**

Closed	Open
-	[-_Mila_9]
[-_Mila_9]	[Mila_Const_5] ; [Mila_Beja_6] ; [Mila_Jijel_7]
[-_Mila_9] ; [Mila_Const_5]	[Mila_Beja_6] ; [Mila_Jijel_7] ; [Const_Sétif_12]
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Beja_6]	[Mila_Jijel_7] ; [Const_Sétif_12] ; [Beja_Boui_9]
[-_Mila_9] ; [Mila_Const_5] ; **[Mila_Beja_6]** ; [Mila_Jijel_7]	[Const_Sétif_12] ; **[Jijel**_Boui_6] ; **[Jijel_Beja_5]**
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Jijel_7] ; [Jijel_Beja_5]	[Const_Sétif_12] ; [Jijel_Boui_6] ; ~~[Beja_Boui_8]~~ ;
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Jijel_7] ; [Jijel_Beja_5] ; [Jijel_Boui_6]	[Const_Sétif_12] ; **[Boui_Alger_7]**

[Mila_Jijel_Boui_Alger]

Closed	Open
	[-_Mila_9]
[-_Mila_9]	[Mila_Const_5] ; [Mila_Beja_6] ; [Mila_Jijel_7]
[-_Mila_9] ; [Mila_Const_5]	[Mila_Beja_6] ; [Mila_Jijel_7] ; [Const_Sétif_12]
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Beja_6]	[Mila_Jijel_7] ; [Const_Sétif_12] ; [Beja_Boui_9]
[-_Mila_9] ; [Mila_Const_5] ; **[Mila_Beja_6]** ; [Mila_Jijel_7]	[Const_Sétif_12] ; [**Jijel**_Boui_**6**] ; **[Jijel_Beja_5]**
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Jijel_7] ; [Jijel_Beja_5]	[Const_Sétif_12] ; [Jijel_Boui_6] ; ~~[Beja_Boui_8]~~ ;
[-_Mila_9] ; [Mila_Const_5] ; [Mila_Jijel_7] ; [Jijel_Beja_5] ; [Jijel_Boui_6]	[Const_Sétif_12] ; **[Boui_Alger_7]**

[Mila_Jijel_Boui_Alger]


Ceux qui ne font rien ne se trompent jamais