

Path Finding (PF)

Table of Contents

Path Finding (PF)

- 1. Contexte
 - 1.1 Introduction au Path Finding
 - 1.2 Des applications du Path Finding (PF)
 - 1.3 Cas d'étude : Autonomous Mobile Robot (AMR)
- 2. Etude algorithmique
 - 2.1 Formalisation d'un problème de recherche
 - 2.2 Algorithmes de recherche
 - 2.3 Principe de l'algorithme BFS
 - 2.4 Principe de l'algorithme de Dijkstra
 - 2.5 Principe de l'algorithme A*
 - 2.6 Principe de l'algorithme glouton
- 3. Solution logicielle
- 4. Travail à conduire (partie 1)
 - 4.1. Instances numériques à disposition
 - 4.2 Production

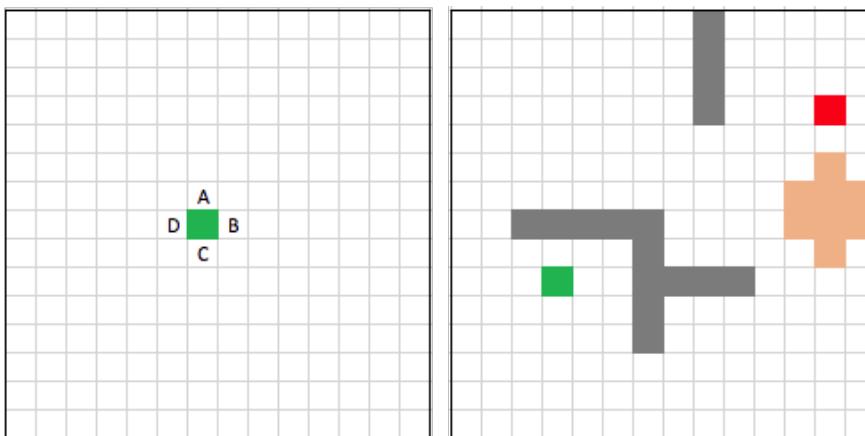


1. Contexte

1.1 Introduction au Path Finding

Problème...

- ... de recherche d'un chemin dans un environnement contraint



- ... central tant en recherche opérationnelle qu'en intelligence artificielle

1.2 Des applications du Path Finding (PF)

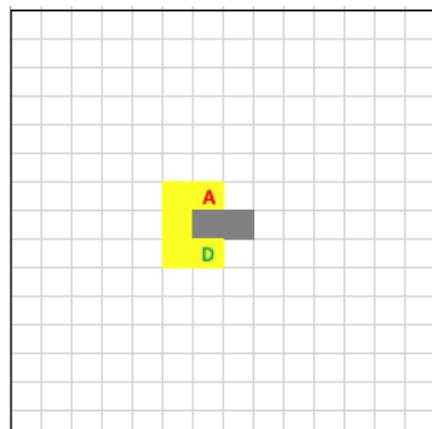
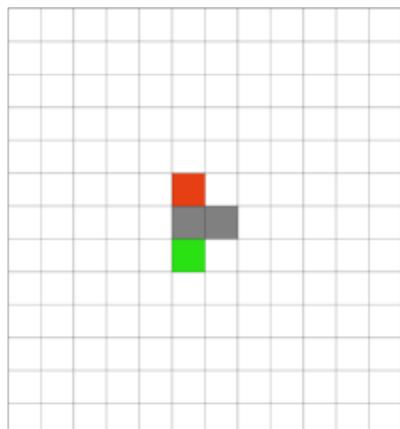
- **Les jeux vidéos**
 - comment éviter des obstacles habilement, comment trouver le chemin le plus efficace sur différents terrains...
- **Les véhicules autonomes**
 - comment se déplacer dans un espace contenant des obstacles connus et inconnus en minimisant les collisions et le temps moyen pour accomplir le chemin...
- **La logistique d'entrepôt** (MAPF : variante du PF dite *Multi-agent path finding*)
 - comment déplacer un ensemble d'agents depuis leur point de départ individuel à leur point d'arrivée sans collision
- **La circulation de trains** (MTPF : variante du MAPF dite *Multi-Train Path Finding*)
 - comment établir un plan de circulation de trains sans collision.

1.3 Cas d'étude : Autonomous Mobile Robot (AMR)

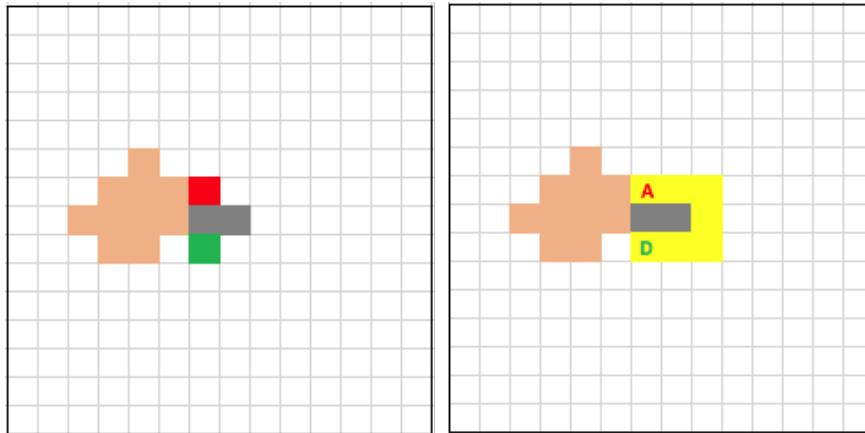
Un *Autonomous Mobile Robot* (AMR) est un robot intelligent, au déplacement autonome, conçus pour réaliser différentes tâches sans intervention humaine. Les AMRs utilisent des capteurs sophistiqués, l'intelligence artificielle et l'optimisation pour comprendre et se déplacer dans leurs environnements dynamiques. Ils sont ainsi capables de s'adapter aux changements d'environnement, éviter des obstacles et optimiser leurs routes dynamiquement.



Situation 1



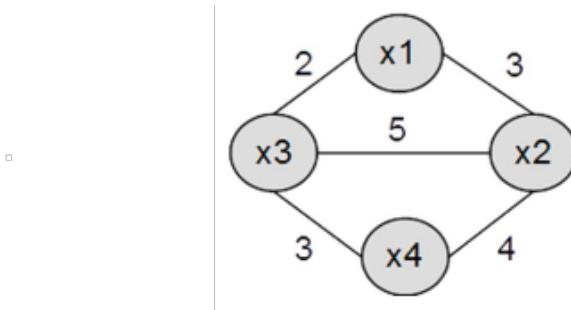
Situation 2



2. Etude algorithmique

2.1 Formalisation d'un problème de recherche

Notions de graphes



Soit $G(V, E, w)$, un graphe pondéré non-orienté, composé

- de sommets (noeuds) $v \in V$ avec $n = |V|$,
- d'arêtes $e \in E$ avec $m = |E|$, une arête étant $e = (i, j)$ pour $i, j \in V$,
- de poids $w(e)$ sur les arêtes $e \in E$, où $w : E \rightarrow \mathbb{R} : e \mapsto w(e)$, une fonction affectant à chaque arête un poids réel.
- Sommet successeur : v est un sommet successeur de u si $(u, v) \in E$
- Ensemble des sommets successeurs de u : $\Gamma^+(u) = \{v \mid (u, v) \in E\}$
- Chaine : séquence de sommets $< v_1, v_2, \dots, v_k >$ telle que $(v_{i-1}, v_i) \in E$
- Chaine élémentaire : chaîne où tous les sommets sont distincts

Entrées

- Un graphe $G(V, E, w)$
- Un sommet initial (départ) v_D
- Un sommet final (arrivée) v_A

Sorties

- Solution = chemin dans un graphe (chaîne élémentaire)
- Le coût d'une solution est la somme des poids des arêtes dans la solution.
- Il peut y avoir plusieurs solutions équivalentes.

2.2 Algorithmes de recherche

Recherche non informée :

- Algorithme BFS: Breadth-First-Search (recherche en largeur; Flood Fill)
- Algorithme de Dijkstra

Recherche informée :

- Algorithme glouton: Greedy Best-First-Search
- Algorithme A*

2.3 Principe de l'algorithme BFS

Principe :

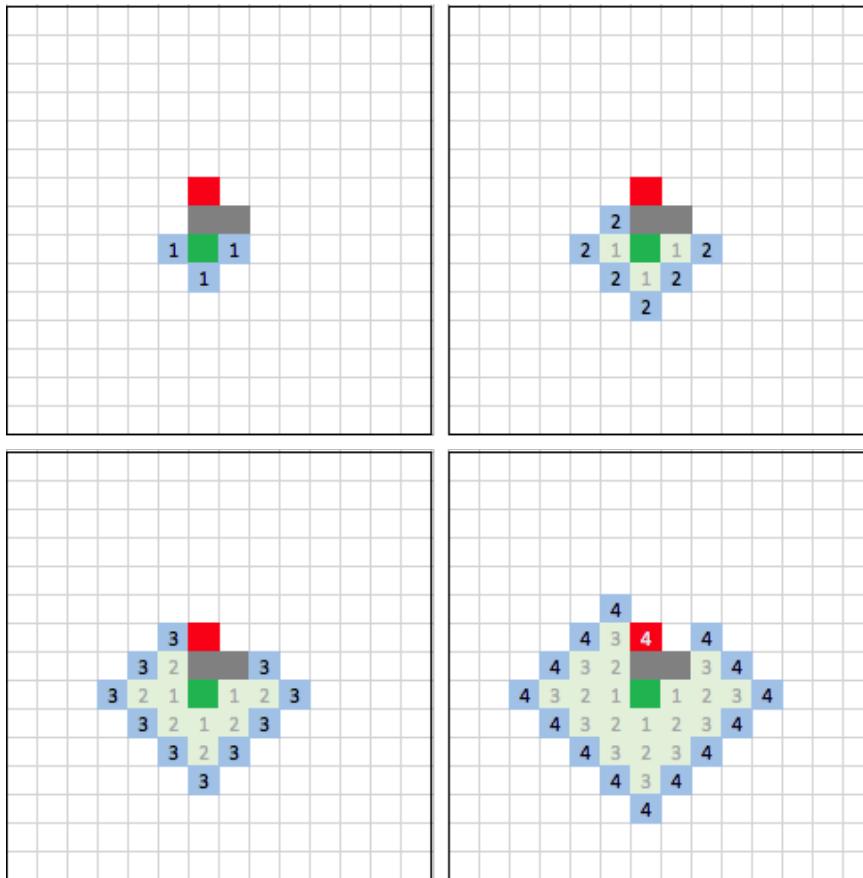
```
function BFS(G,vD,va)
#   Créer une file F
#   Enfiler vD dans F
#
#   tant F ≠ ø faire
#
#       u = défiler F
#       marquer u comme visité
#       S = successeurs(u)
#
#       pour s ∈ S faire
#           si s n'est pas visité alors
#               ajouter s dans F
#           fsi
#       fpour
#
#   ftant
#
end
```

Quid de :

- but atteint ?
- but non-atteint ?

Illustration :

- sur situation 1



2.4 Principe de l'algorithme de Dijkstra

But :

- Plus court chemins optimaux d'un sommet de départ vers tous les autres sommets atteignables

Conditions d'application :

- Graphes orientés et graphes non orientés
- Valuations non-négatives

Principe :

```
function Dijkstra(G,vD,vA)
    #   créer une liste L
    #   pour_tout sommet v ∈ V faire
    #       distance(v) ← +∞
    #       precedent(v) ← Λ
    #       permanent(v) ← Faux
    #       ajouter v dans L
    #   fpour
    #   distance(vD) ← 0
    #   tantQue il existe des sommets non permanents dans L faire
    #       u ← sommet non permanent dans L de valeur distance(u) minimale
    #       permanent(u) ← Vrai
    #
```

```

#           S = successeurs non permanents de u
#           pour v ∈ S faire
#               distance(v) ← Min( distance(v), distance(u) + segment(u,v) )
#               si distance(v) a été mise à jour alors
#                   precedent(v) ← u
#   fsi
#   fpour
#
#   ftant
end

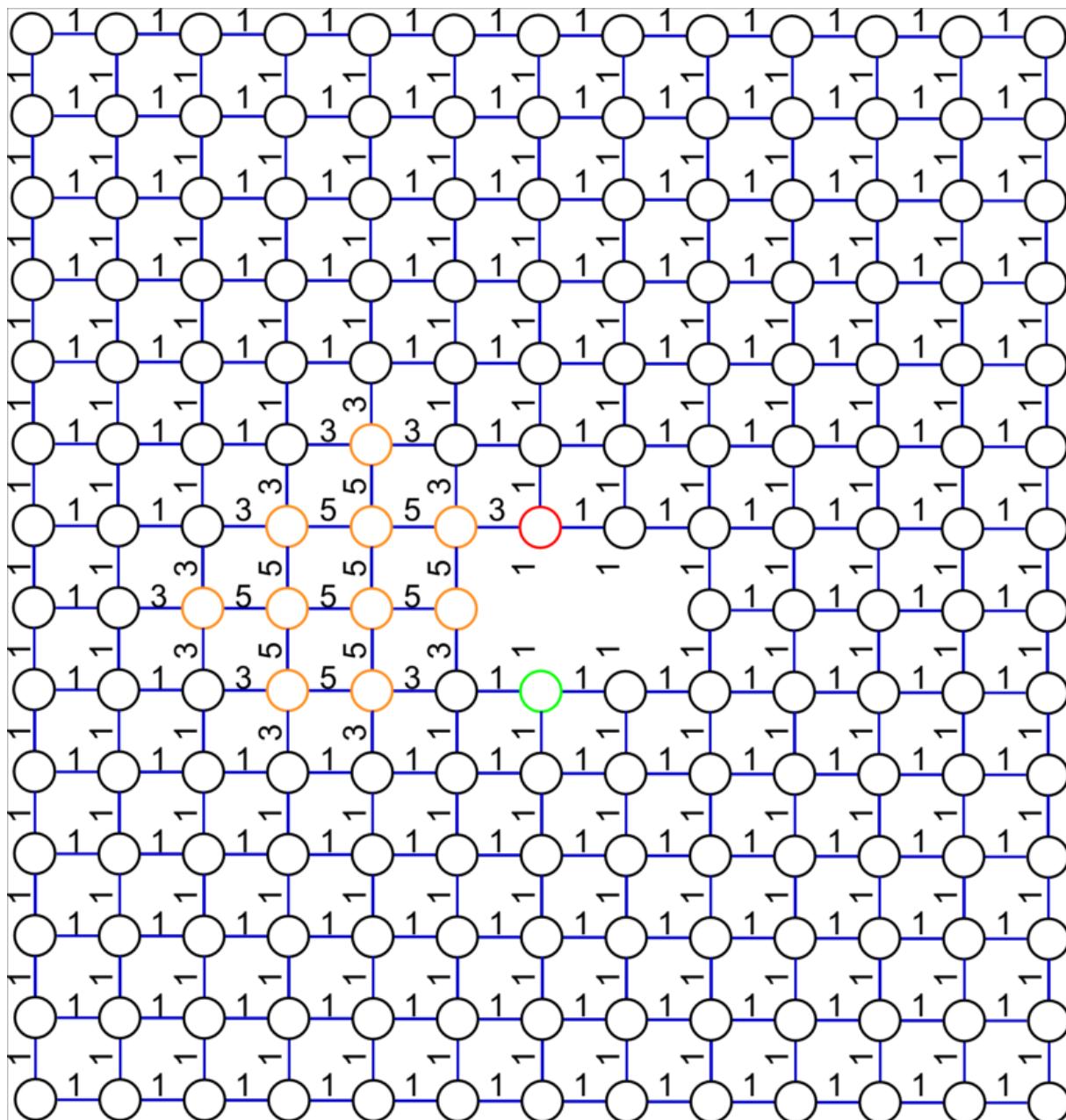
```

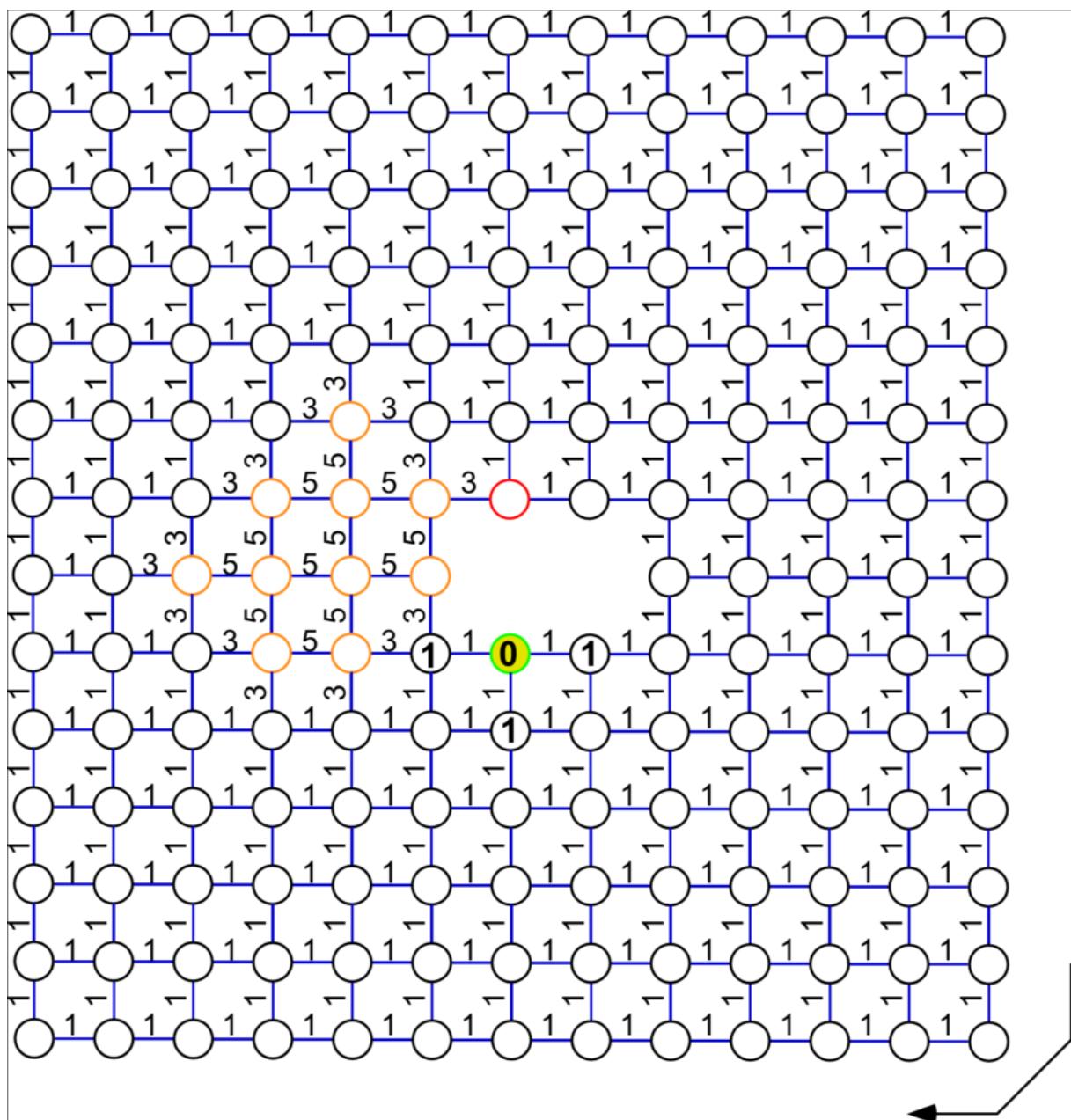
Quid de :

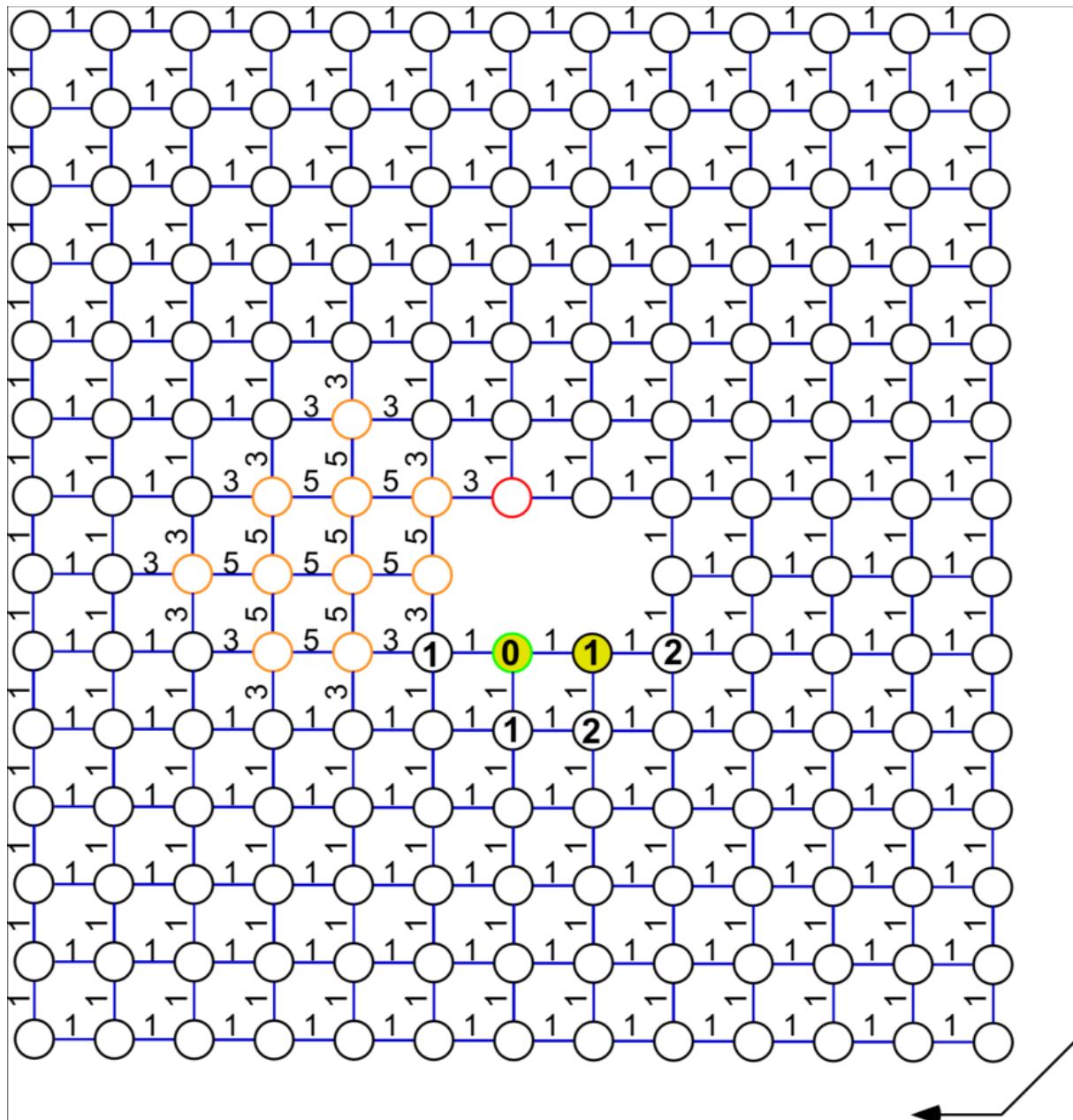
- but atteint ?
- but non-atteint ?

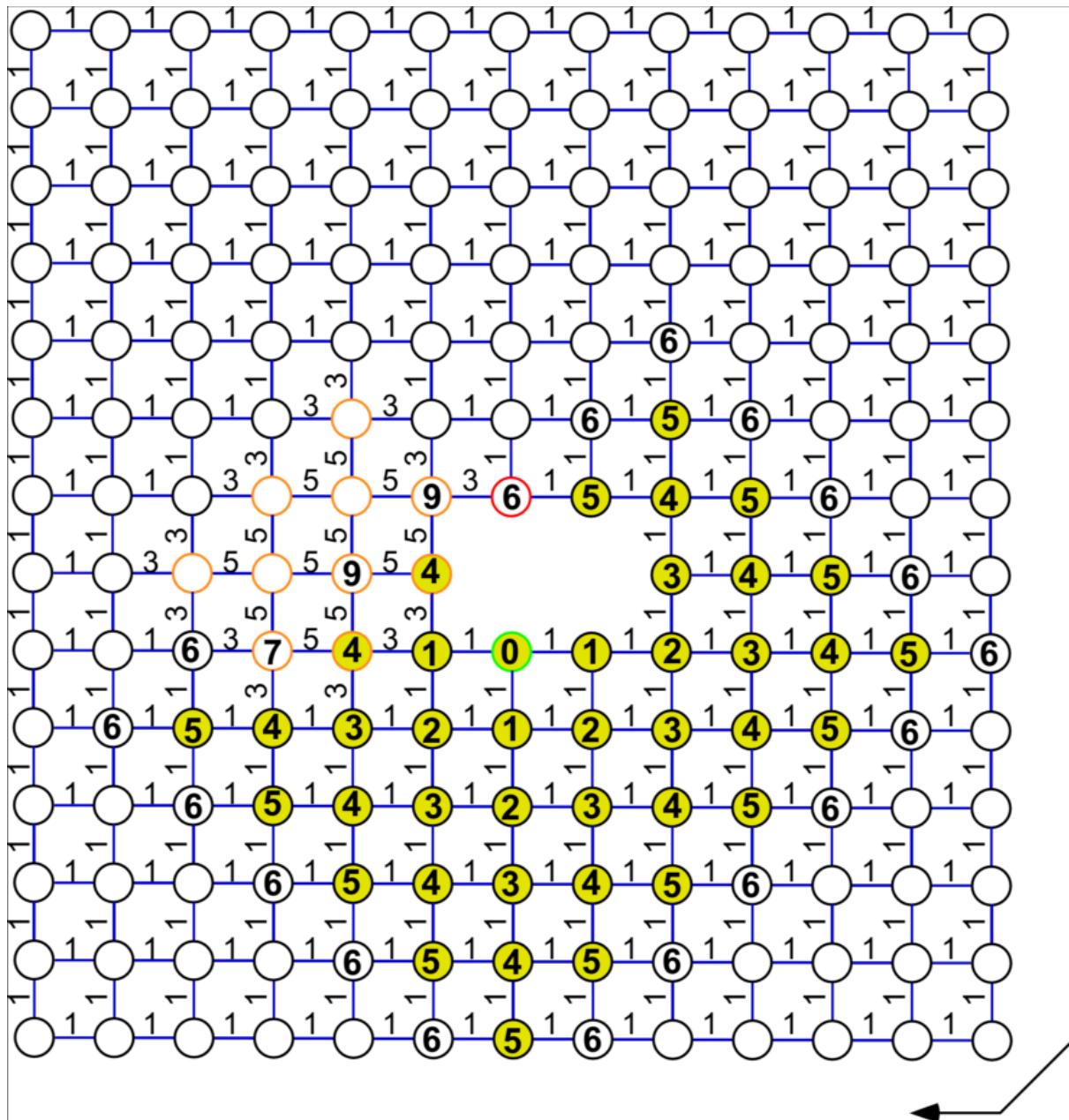
Illustration :

- sur situation 2









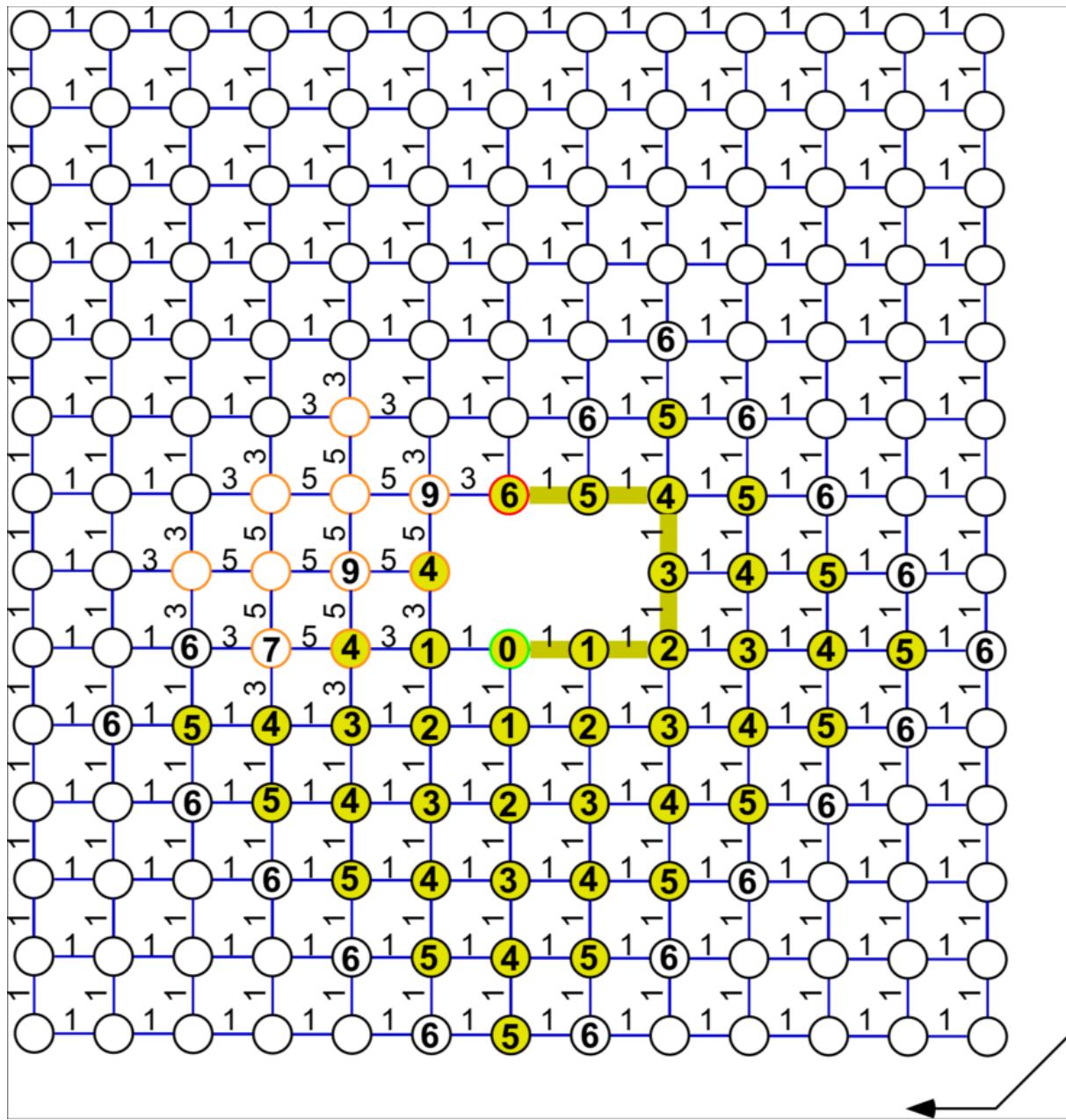
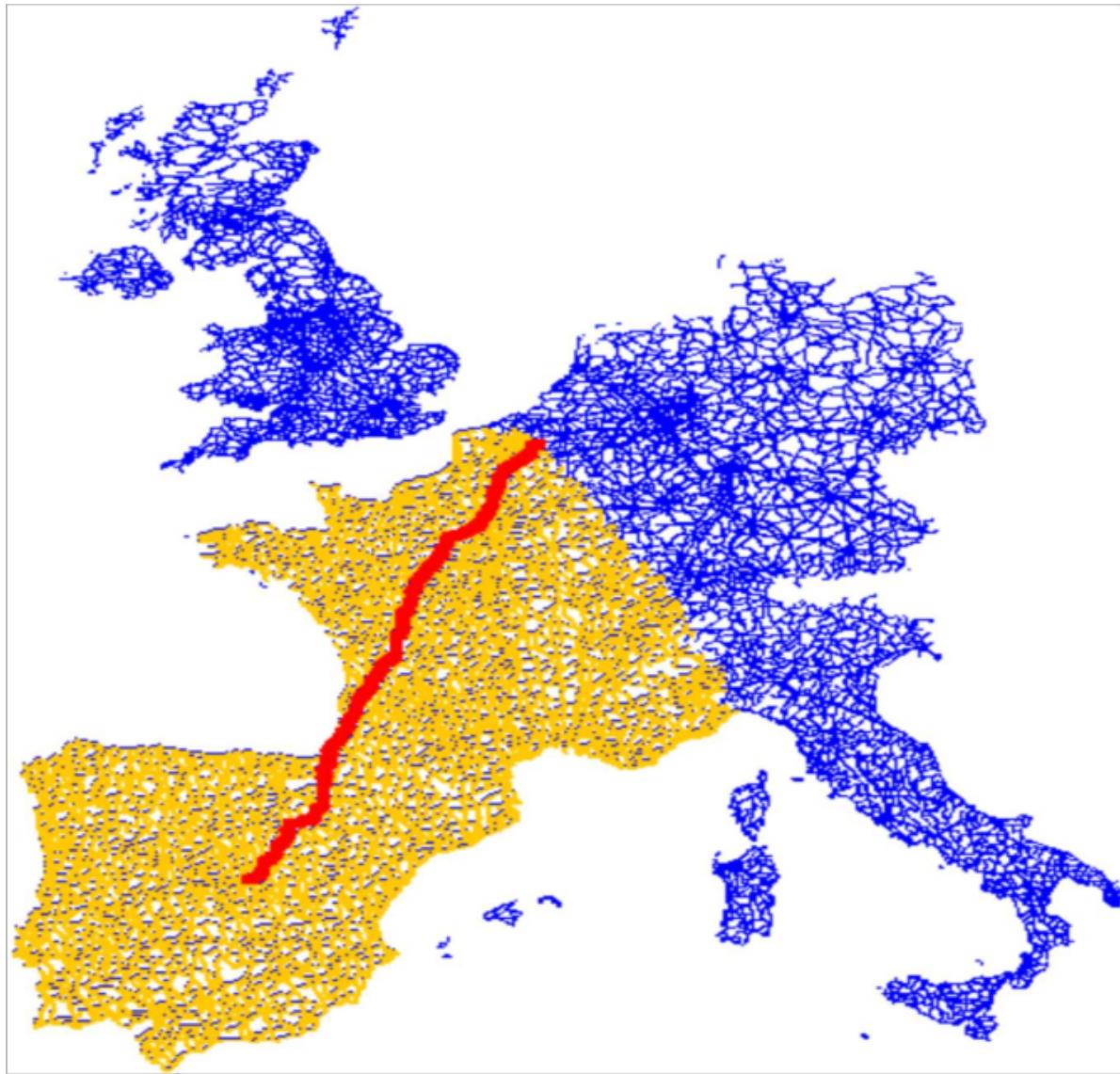


Illustration :

- sur chemin entre Madrid et Bruxelles



2.5 Principe de l'algorithme A*

Introduction :

Soit f , la fonction qui désigne la distance entre le sommet de départ v_D et le sommet d'arrivée v_A . En pratique on ne connaît pas cette distance, c'est ce qu'on cherche !

Par contre on peut connaître la distance optimale dans la partie explorée entre le sommet initial v_D et un sommet v déjà exploré. On peut donc séparer $f(v)$ en deux parties :

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

avec

- $g(v)$, le coût réel du chemin optimal partant du sommet v_D à v dans la partie déjà explorée.
- $h(v)$, le coût estimé du reste du chemin partant de v jusqu'au sommet v_A . Ici, $h(v)$ est une fonction heuristique.

Quid de :

- but non atteint ?

Illustration :

- sur situation 2

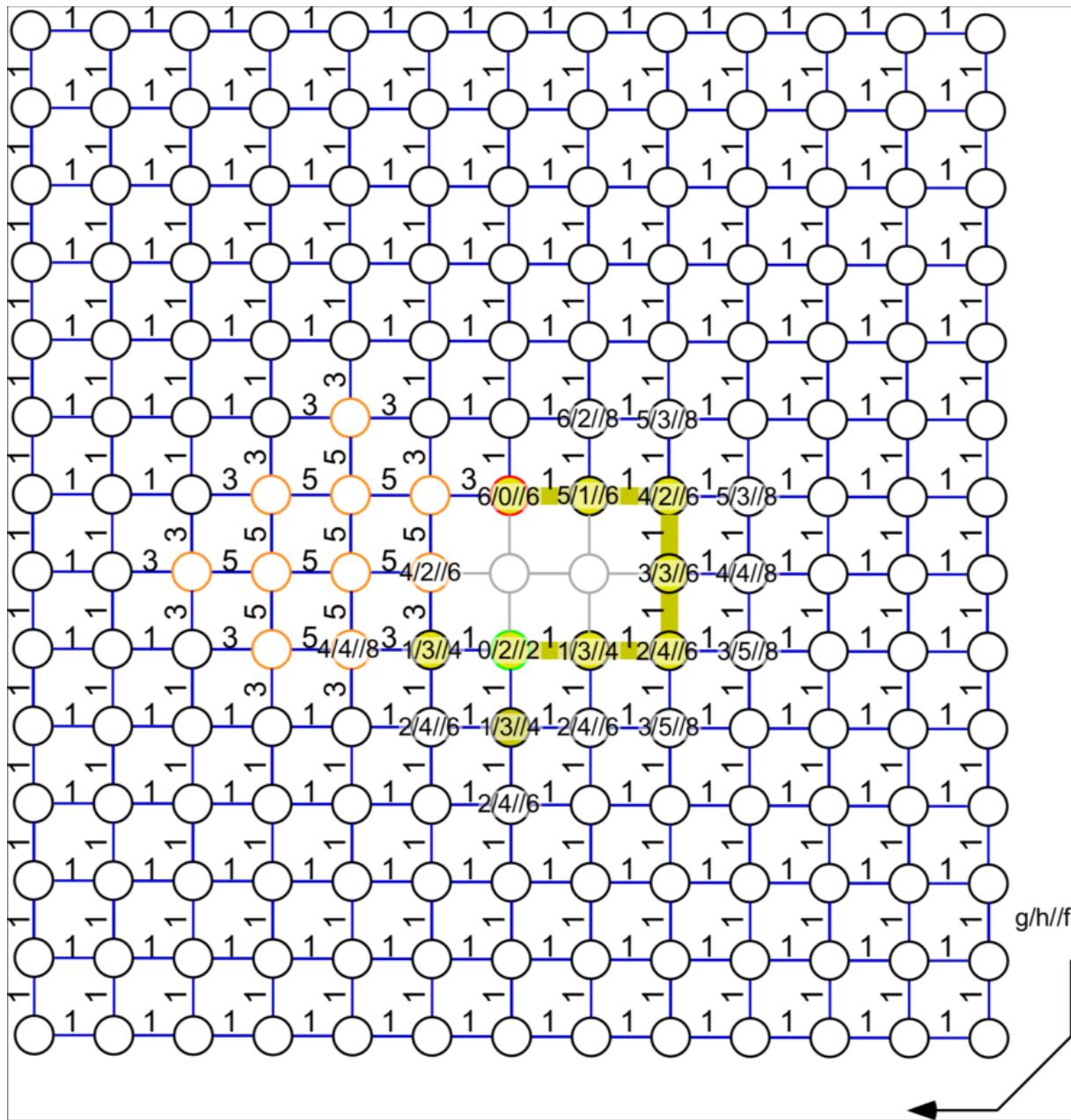
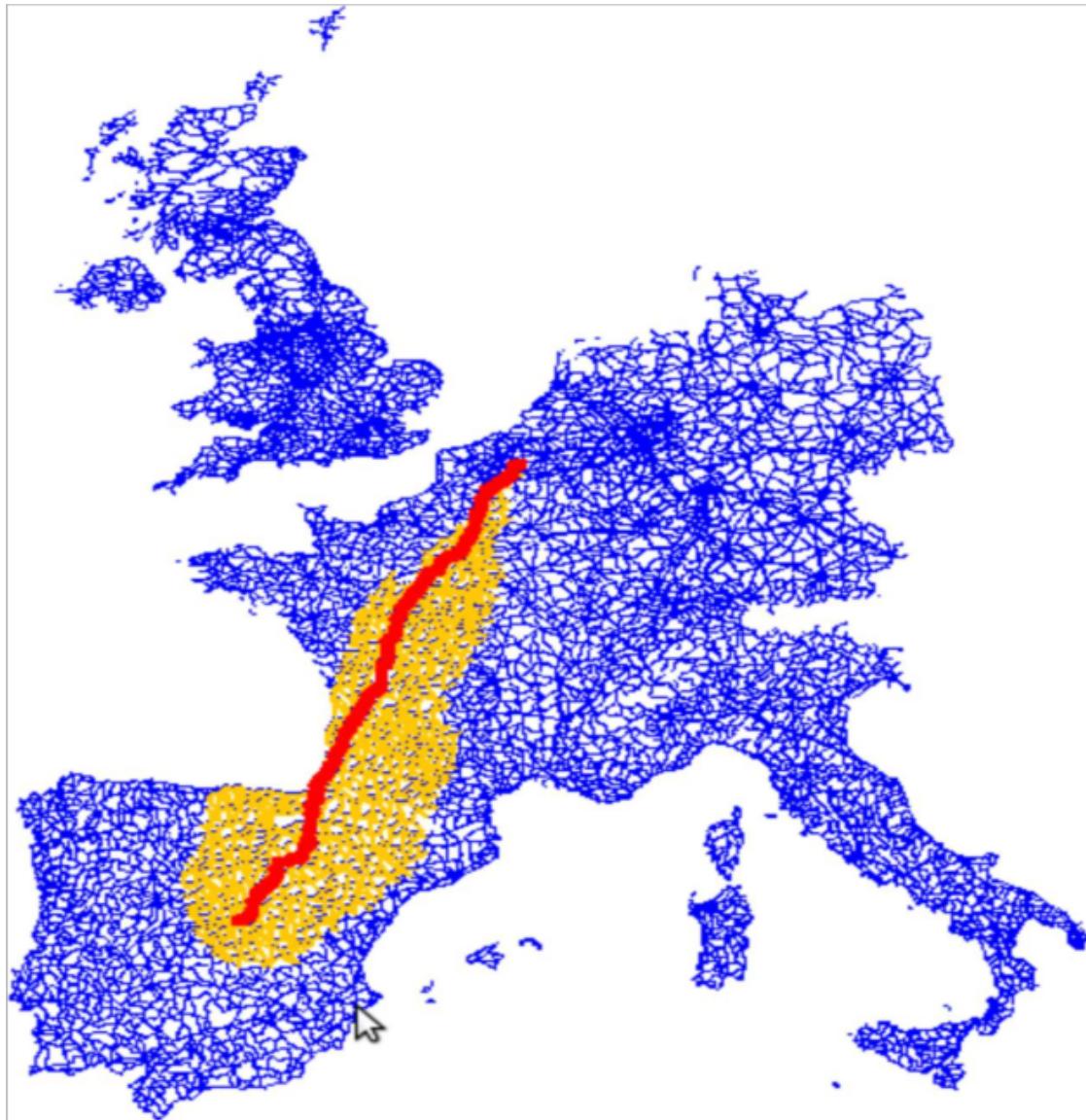


Illustration :

- sur chemin entre Madrid et Bruxelles



2.6 Principe de l'algorithme glouton

Algorithme de Dijkstra

$$f(v) \leftarrow g(v)$$

Algorithme A*

$$f(v) \leftarrow g(v) + h(v)$$

Algorithme Glouton

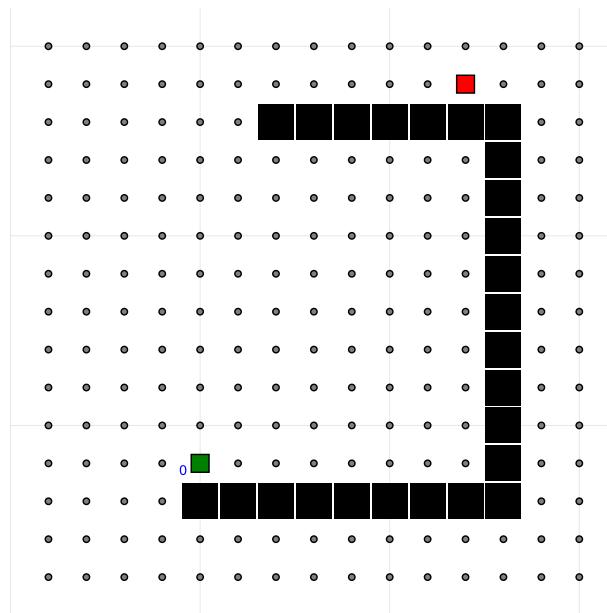
$$f(v) \leftarrow h(v)$$

3. Solution logicielle

Implémentation :



didactic0



Breadth-First-Search



Solution :

CPUTime (s)	:	8.7e-5
Distance D → A	:	17
Number of states evaluated	:	178
Path D → A	:	

(12, 5)→(11, 5)→(10, 5)→(9, 5)→(8, 5)→(7, 5)→(6, 5)→(5, 5)→(4, 5)→(3, 5)→(2, 5)→(2, 6)→(2, 7)→(2, 8)→(2, 9)→(2, 10)→(2, 11)→(2, 12)

4. Travail à conduire (partie 1)

4.1. Instances numériques à disposition

Collection 2D Pathfinding Benchmarks :

- <https://movingai.com/benchmarks/grids.html>

héberge des données et résultats pour des instances de :

- jeux vidéos : Dragon Age 2, Warcraft III, Starcraft, etc.
- villes réelles : Paris, NewYork, Milan, etc.

- situations créées artificiellement : labyrinthe, terrain, etc.

exemple :

- un quartier de Paris (Paris_2_256.map) :



inspection d'un fichier :

```
afficheMapTerminal (generic function with 1 method)
```

didactic0 ▾

```
Instance: didactic0
Map of size:
height: 15
width : 15

123456789012345
1..... .
2..... .
3..... @@@@ @@@@ ..
4..... @..
5..... @..
6..... @..
7..... @..
8..... @..
9..... @..
0..... @..
1..... @..
2..... @..
3..... @@@@ @@@@ ..
4..... .
5..... .
```

évaluation du coût d'un déplacement sur une case admissible de la carte :

```
Selon que la case à visiter a pour valeur
'S' alors coutMvmt ← 5
'W' alors coutMvmt ← 8
autrement coutMvmt ← 1
fselon
```

Il n'y a pas de coût transitoire à envisager.

Donc si

- la valeur du chemin en $i=(1,1)$ est de 20
- la case en $j=(1,2)$ est 'W'

alors la valeur du chemin après le déplacement de i en j est de 28

4.2 Production

Etapes

1. Travail préparatoire (lecture, étude, application des quatre algorithmes).
2. Préparer et implémenter en Julia les quatre algorithmes en prenant soin à la qualité de l'implémentation (performances).
3. Valider numériquement les quatre implémentations à l'aide d'au moins trois instances choisies dans la collection indiquée.
4. Rapporter une étude comparative succincte des résultats récoltés avec les quatre algorithmes sur les instances retenues.
5. Livrable d'étape : une **version fonctionnelle mais pas nécessairement finale**.

Indications

Mesure de l'activité de l'algorithme

Elle se mesure en comptant le nombre de cases de la carte visitées par l'algorithme. Si une même case est visitée plusieurs fois, on adopte comme principe de ne pas tenir compte de cette multiplicité.

Expérimentation numérique

Pour être qualifiée, votre application doit être rendue fonctionnelle sur les 4 algorithmes. Elle sera testée sur (minimum trois) instances (.map) tirées de la collection d'instances indiquée.

Protocole de test

Votre algorithme doit pouvoir être invoqué de la manière suivante :

```
algoBFS(fname, D, A)
algoDijkstra(fname, D, A)
algoGlouton(fname, D, A)
algoAstar(fname, D, A)
```

avec comme paramètres :

- fname | type : String | exemple : "didactic.map"
- D | type : Tuple{Int64, Int64} | exemple : (12, 14)
- A | type : Tuple{Int64, Int64} | exemple : (4, 5)

Exemple :

```
fname = "didactic.map"; D = (12, 14); A = (4, 5)
algoDijkstra(fname, D, A)
```

Résultats attendus

at minima affichage de :

- la distance de D à A
- la mesure de l'activité de l'algorithme
- le chemin de D à A parcouru par l'algorithme

exemple :

- Distance D → A : 4
- Number of states evaluated : 10
- Path D → A : (2, 11)→(2, 10)→(2, 9)→(1, 9)→(1, 8)

Facultativement :

- temps de calcul
- empreinte mémoire
- visualisation du résultat (textuellement ou graphiquement)

Consignes

- Libre de s'inspirer des packages relatifs aux graphes mais ne pas les utiliser pour mettre en place vos algorithmes ⇒ les structures et algorithmes de PF sont à implémenter par vous-même.
- Libre d'utiliser n'importe quel autre package disponible dans la librairie standard Julia, mais sans en abuser.
- Remise du livrable :
 - déposer les productions (codes, données, documents) sur un dépôt Github
 - communiquer par email à la date d'échéance l'URL du dépôt Github
 - organiser les fichiers dans les répertoires `src`, `dat`, `doc`, ...
 - fournir les indications à l'aide du `readme.md` du dépôt Github
- Procédure d'évaluation :
 - téléchargement en local depuis votre dépôt github
 - exécution sous linux (ordinateur CIE) depuis le REPL de Julia 1.11
 - vérification à l'aide d'une instance mystère tirée de la collection
 - vérification du document rapportant l'expérimentation numérique

TEMPS ESTIMÉ : 20h