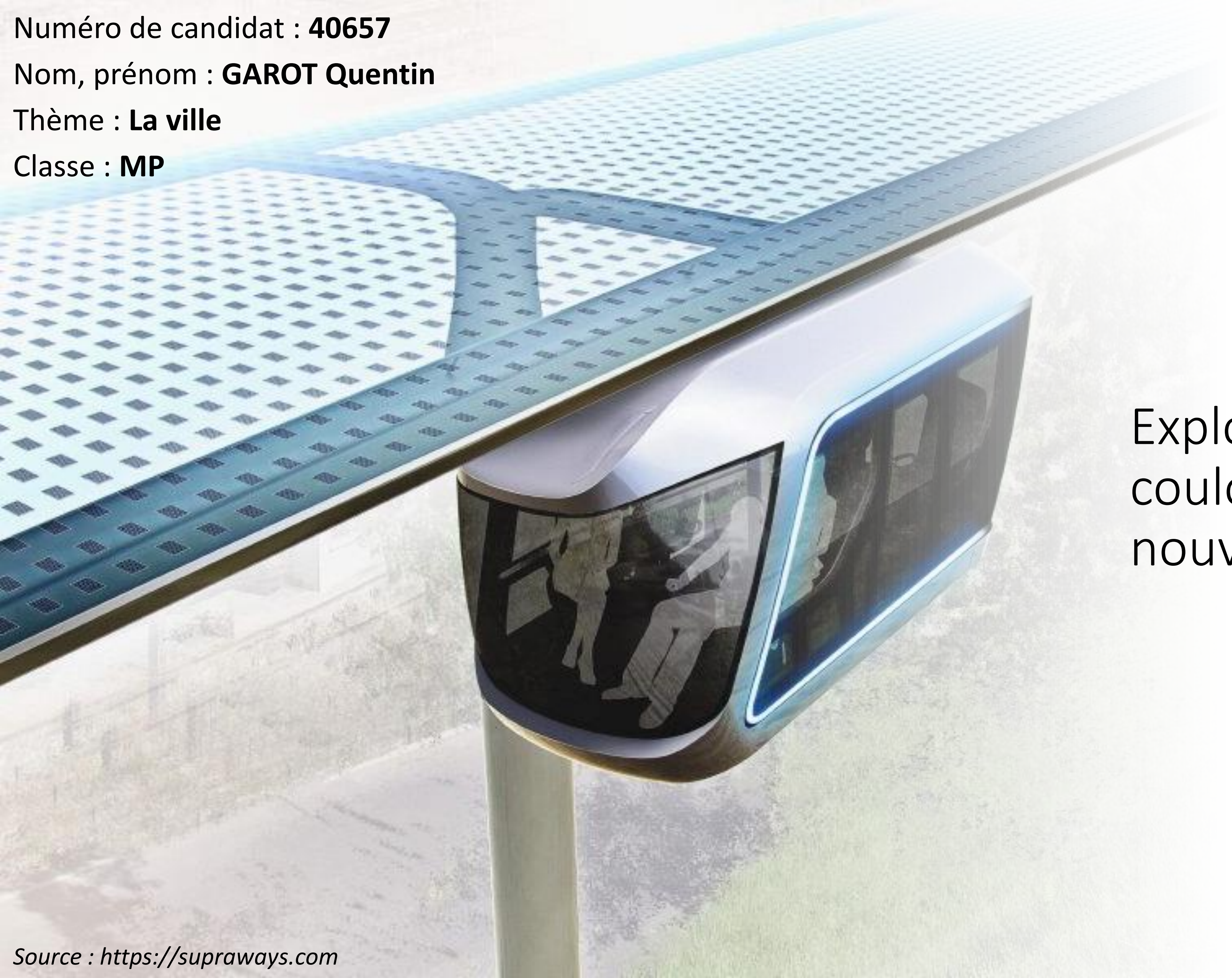
Numéro de candidat : **40657**

Nom, prénom : **GAROT Quentin**

Thème : **La ville**

Classe : **MP**



Source : https://supraways.com

Exploitation optimale des couloirs aériens pour une nouvelle mobilité urbaine

Problématique retenue

Comment peut-on implémenter des algorithmes permettant d'une part de représenter le fonctionnement d'un nouveau moyen de transport urbain et d'autre part de s'approcher d'une fluidité optimale du trafic ?

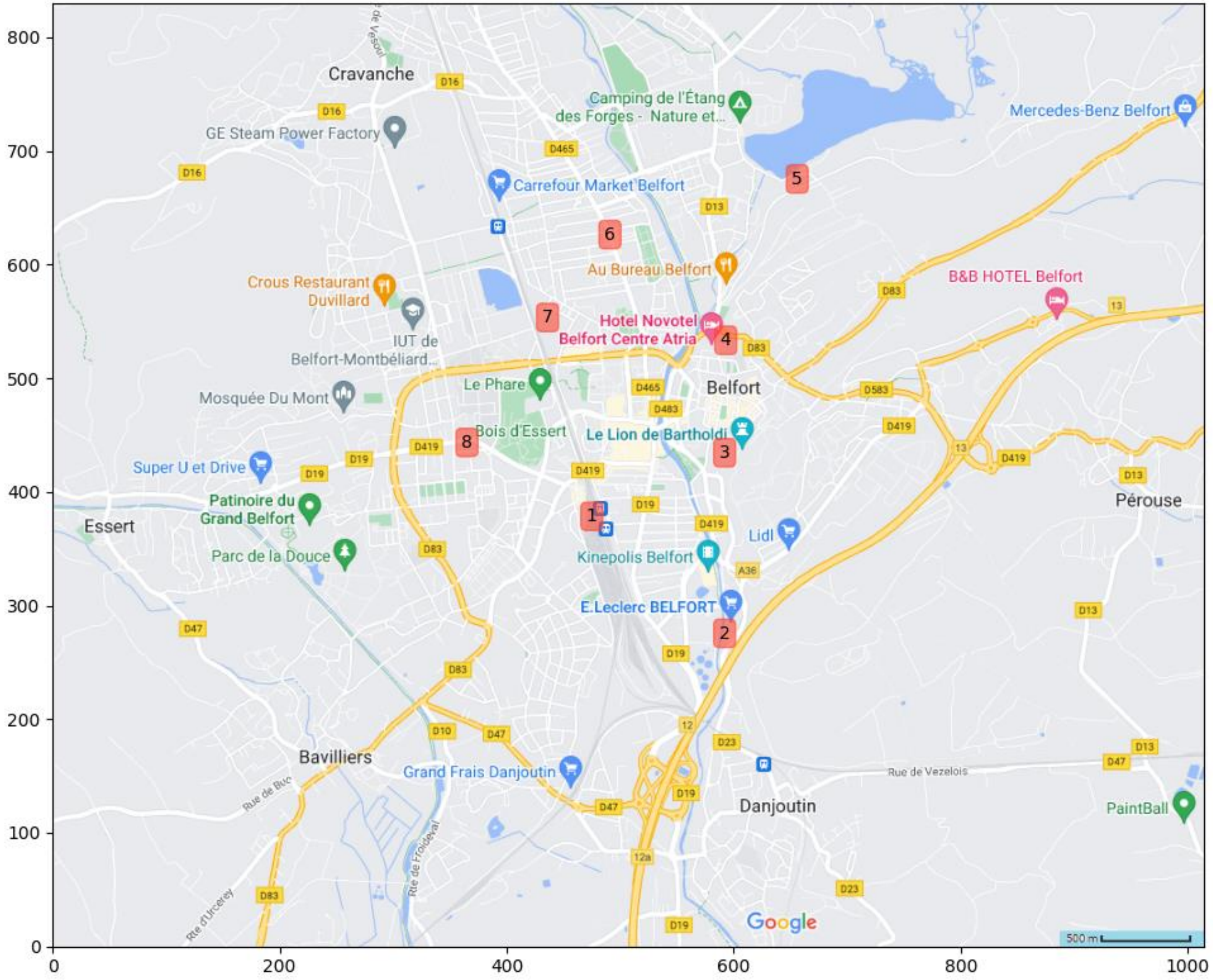
Sommaire

1. Principe du nouveau moyen de transport *Supraways* et implémentation informatique possible
2. Algorithme indispensable pour un trafic optimal : l'algorithme A*
3. Répartition des cabines en fonction des demandes
4. Annexe (démonstration complète et code)

I. Principe et implémentation possible

I.1. Généralités

- Installation de stations
- Rails aériens
- Représentation : graphe
- Programme : choix de l'emplacement des stations

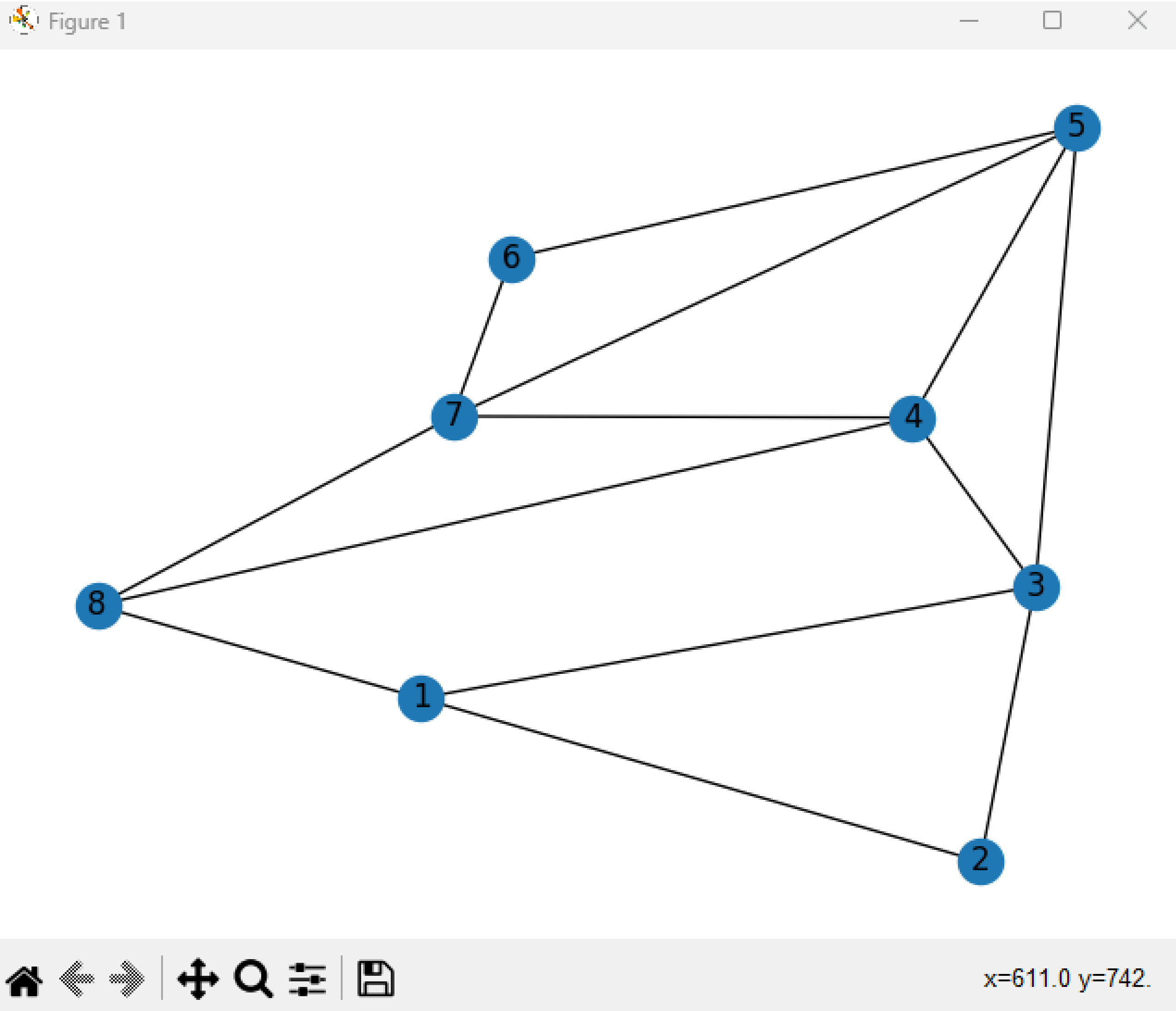


I.1. Généralités

Représentation du graphe
Bibliothèque Python utilisée : *NetworkX*

NetworkX
Network Analysis in Python

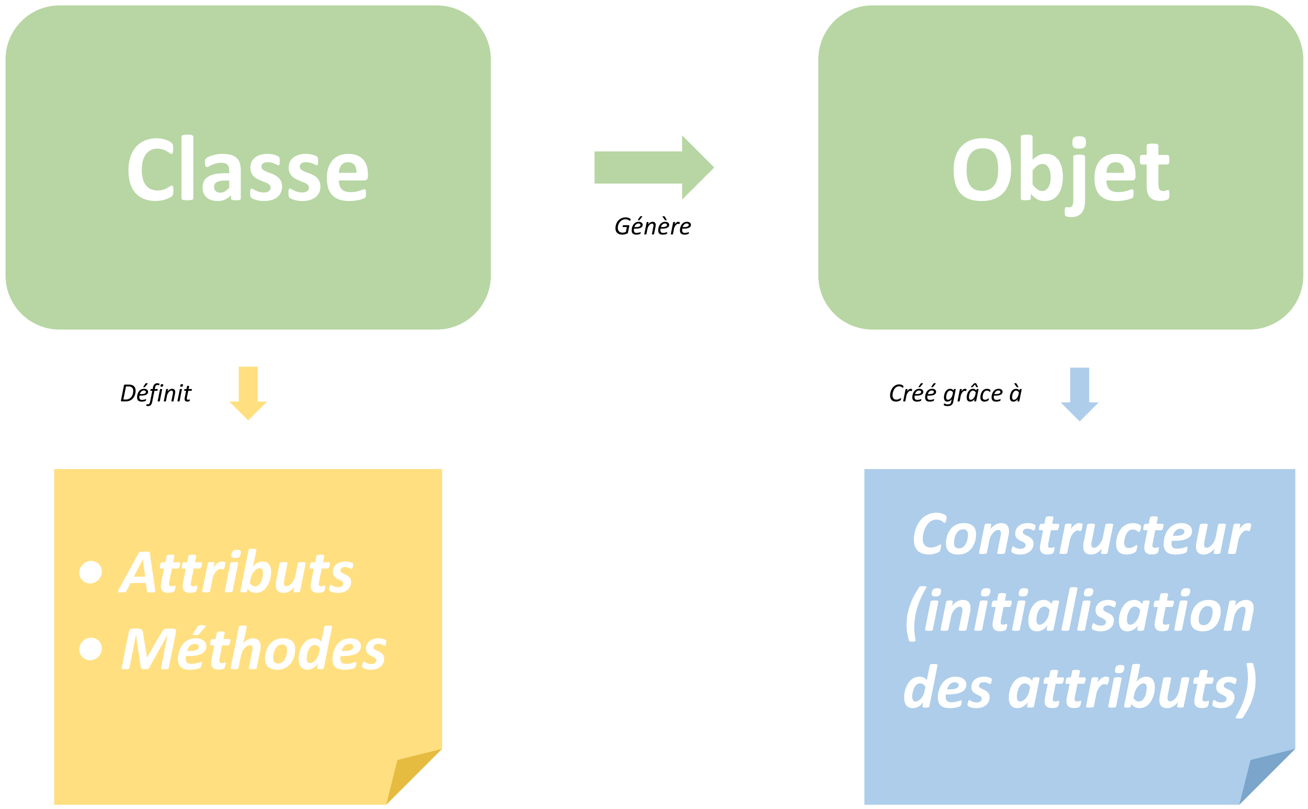
Source : https://networkx.org



Comment suis-je parvenu à réaliser ceci ?

I.2. Programmation orientée objet (POO)

I.2.a. De quoi s’agit-il ?



I.2. Programmation orientée objet (POO)

I.2.a. De quoi s’agit-il ?

I.2.b. Quelques exemples simples

```python
class Point:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def get_x(self) -> float:
        return self.x

    def get_y(self) -> float:
        return self.y
```

```python
class Node(Point):
    def __init__(self, id: int, x: int, y: int):
        super().__init__(x, y)
        # Identifiant
        self.id = id

    def get_id(self) -> int:
        return self.id
```

I.2. Programmation orientée objet (POO)

I.2.a. De quoi s’agit-il ?

I.2.b. Quelques exemples simples

I.2.c. POO et base de données (BDD)

Structure de la table *stations*

```sql
CREATE TABLE stations (
  id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  name VARCHAR(255),
  localisation_x FLOAT,
  localisation_y FLOAT,
  current_gondola INT NOT NULL DEFAULT 0
);
```

Structure de la classe *Database*

Database

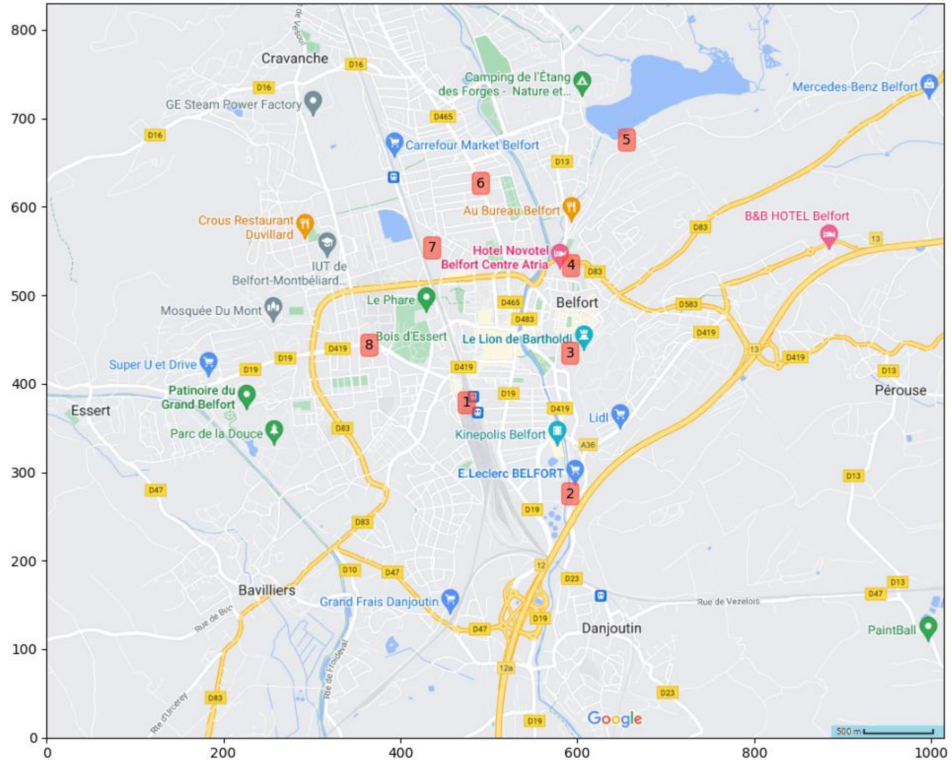
```python
set(self, sql: str) -> None:

get(self, sql: str) -> list[tuple]:
```

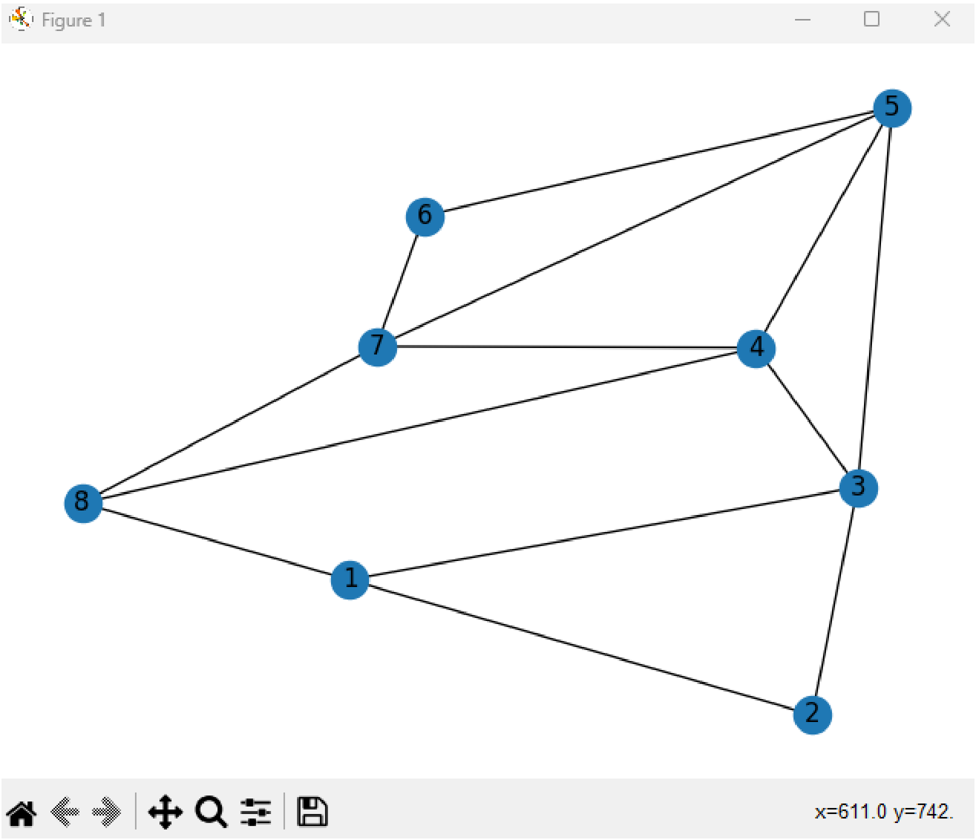
I.3. Réalisation de la carte interactive

Démarche

1. Recherche du plan de la ville de Belfort
2. Documentation de la bibliothèque *Matplotlib*
3. Création d'une classe *Map*, comportant 3 fonctionnalités principales :
    - Affichage du plan (interactif)
    - Positionnement des stations sur le plan & insertion dans la BDD
    - Affichage du graphe

Intérêt : Etude de l'optimalité du trafic

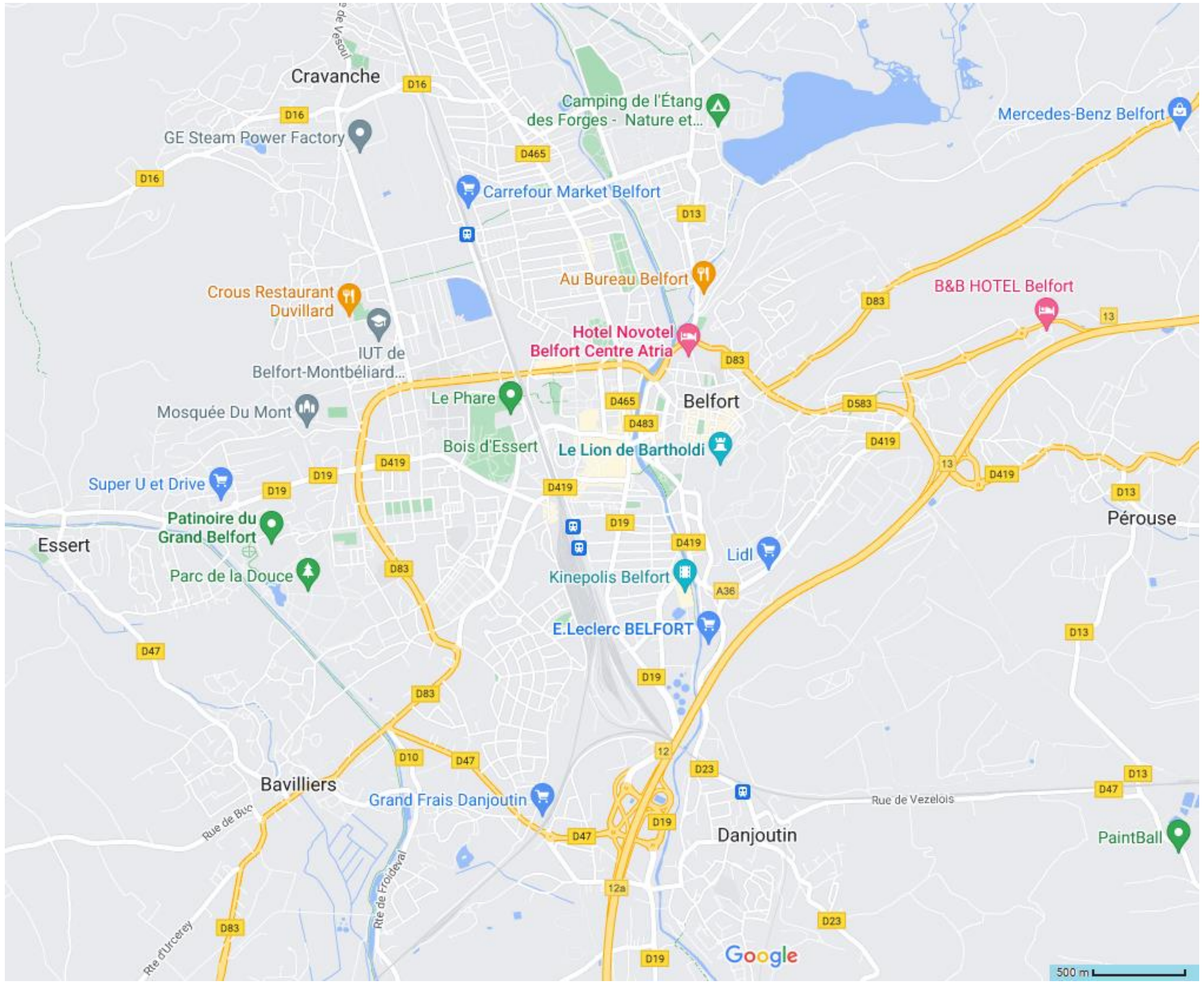




II. Algorithme important : algorithme A*

II.1. Mise en situation – Personal Rapide Transit (PRT)

- Définition PRT
- Principe



Source des icons : https://flaticon.com

II.2. Algorithme A*

II.2.a. Définitions – notations

$G = (S, A, w)$

Définition : heuristique pour la recherche d'un sommet t

$$h : S \to \mathbb{R}_+ \mid h(t) = 0$$

Notation : poids d'un plus court chemin entre deux sommets a et b

$$d(a,b) = min(\{w(p) \text{ avec } p \text{ un chemin de } a \text{ à } b\})$$

Définition : heuristique admissible pour la recherche d'un sommet t

h est *admissible* lorsque $\forall s \in S, h(s) \leq d(s,t)$

II.2.b. Propriétés des nœuds

- **g** : coût de déplacement
- **h** : heuristique
- **f** : g + h (priorité d'un nœud)
- **Nœud parent**

Getters

```
get_cost(self) -> int | None:
get_heuristic(self) -> float:
get_f(self) -> float:
get_parent_node(self) -> Self:
```

Setters

```
set_cost(self, cost: float | None) -> None:
set_heuristic(self, heuristic: float | None) -> None:
set_parent_node(self, node: Self | None) -> None:
```

II.2. Algorithme A*

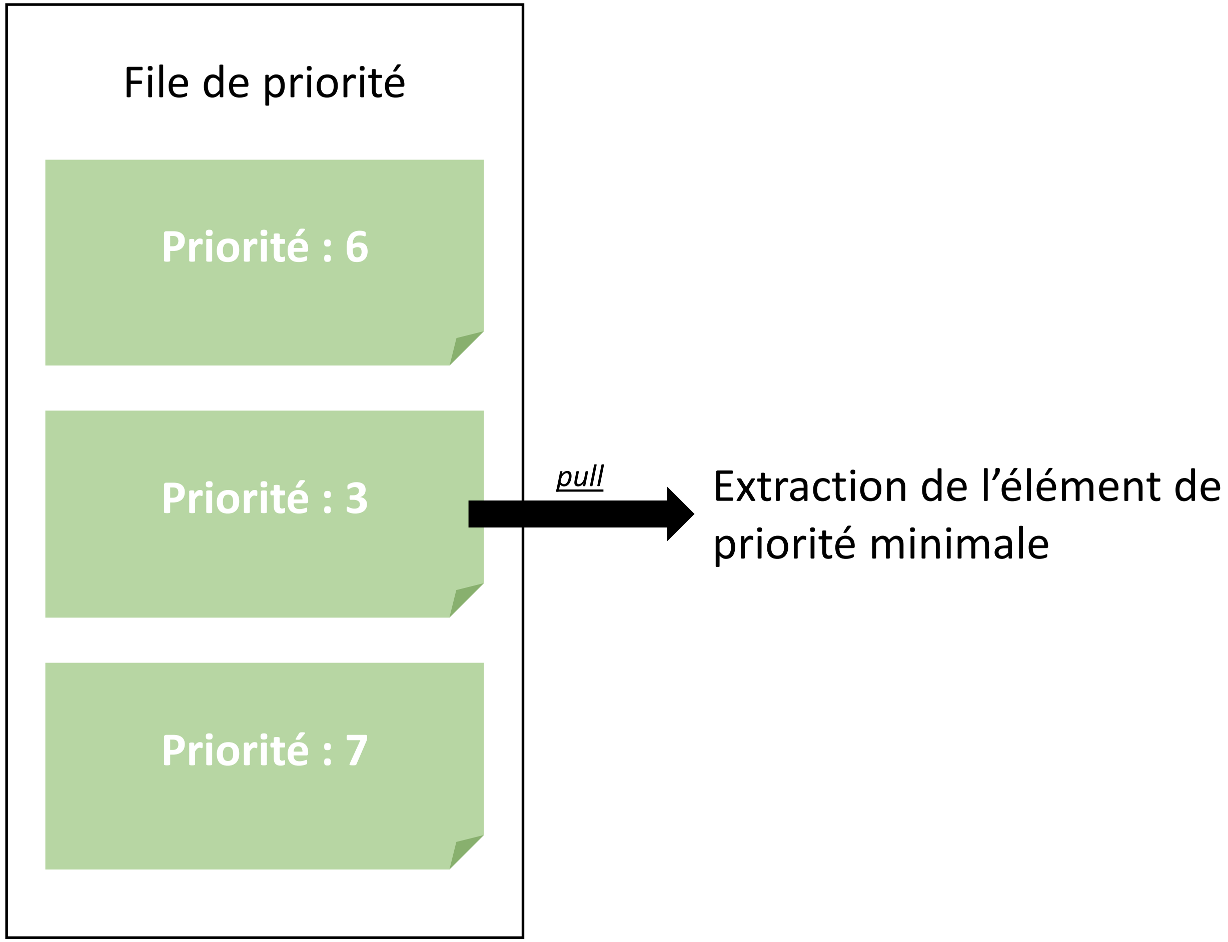
II.2.a. Définitions – notations
II.2.b. Propriétés des nœuds
II.2.c. File de priorité

Interface

```
add(self, x: Node) -> None:
pull(self) -> Node:
is_empty(self) -> bool:
has(self, x: Node) -> bool:
```

Priorité

Valeur de f



II.2. Algorithme A*

II.2.a. Définitions – notations
II.2.b. Propriétés des nœuds
II.2.c. File de priorité
II.2.d. L'algorithme sur un exemple simple

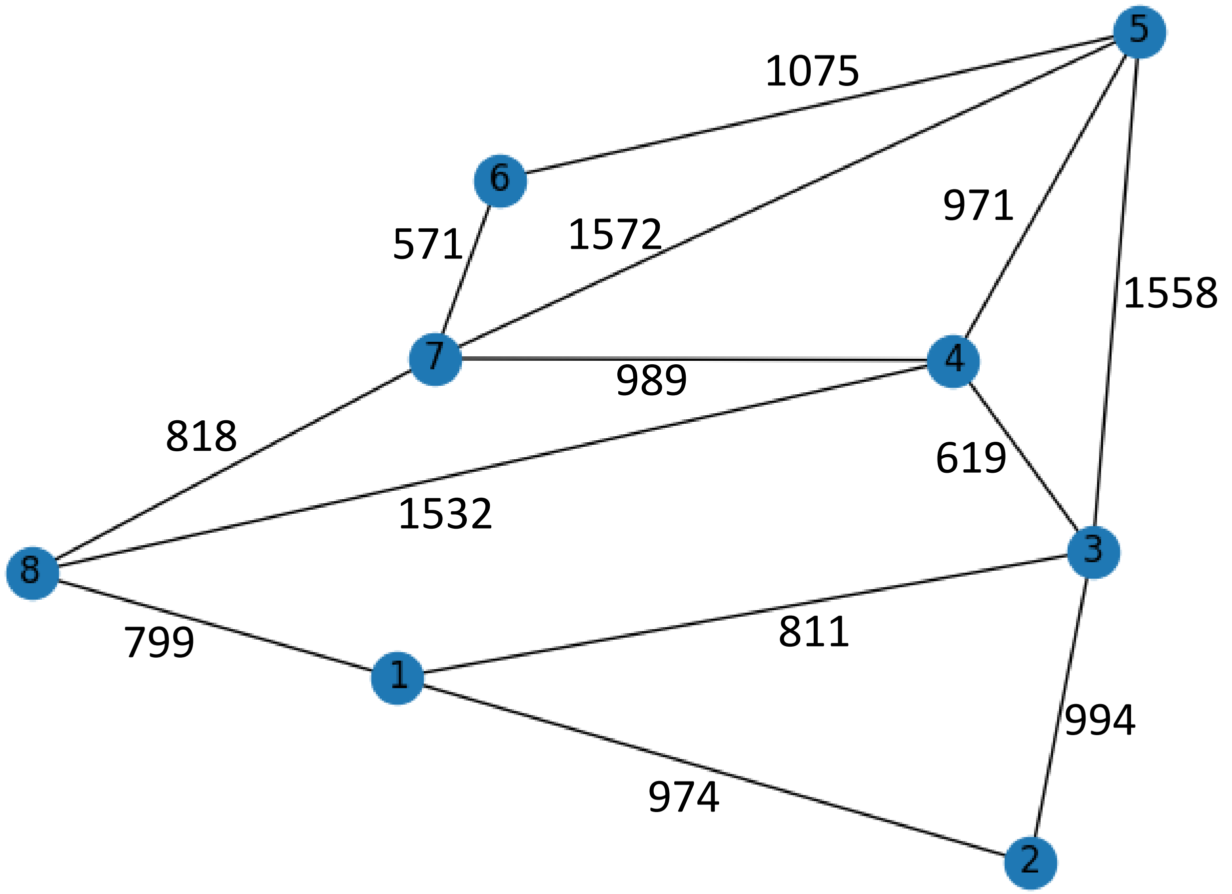
Sommet de départ : **8**
Sommet d'arrivée : **5**

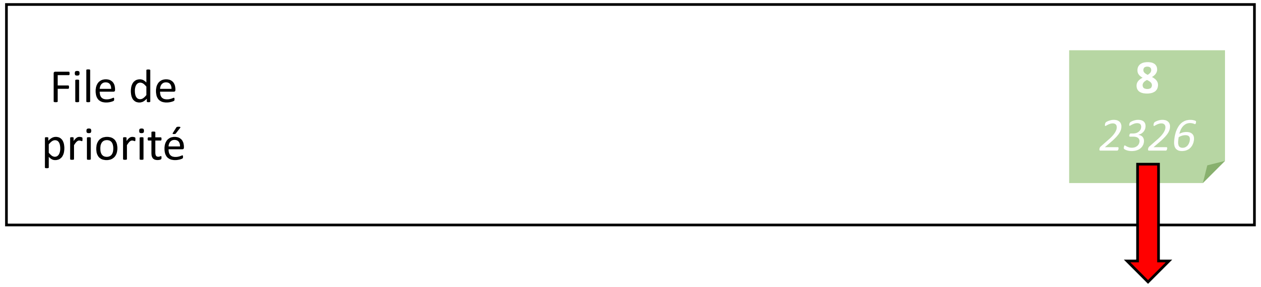
Initialisation du coût de départ & de l'heuristique pour **8**

Extraction de l'élément de priorité minimale : **8**

Voisins de 8 : **[1, 4, 7]**

Pour chaque voisin **v** de **8** :
- Déf. du coût de déplacement : $g(v) = g(8) + w(8, v)$
- Calcul de l'heuristique : $h(v)$
- Détermination du nœud parent : **8**
- Ajout de **v** dans la **file de priorité**





II.2. Algorithme A*

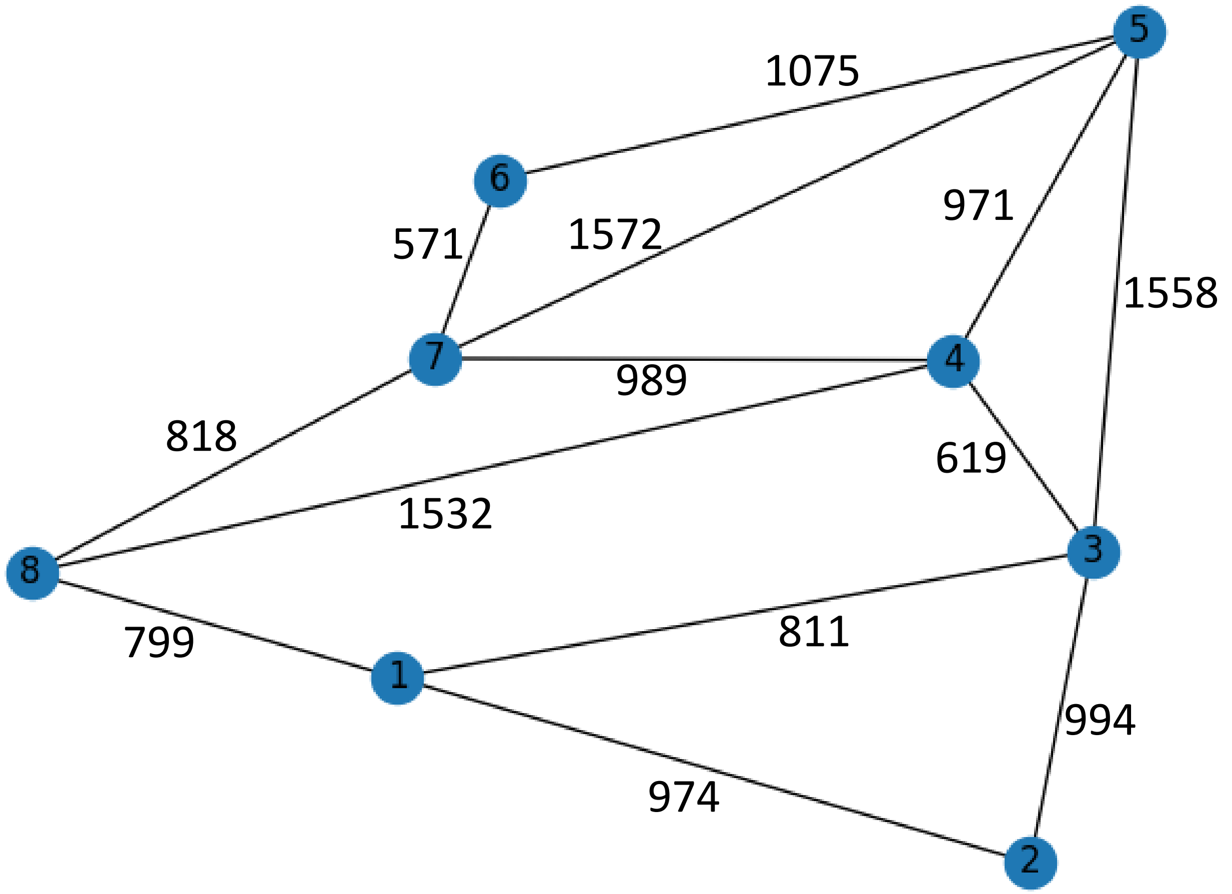
II.2.a. Définitions – notations
II.2.b. Propriétés des nœuds
II.2.c. File de priorité
II.2.d. L'algorithme sur un exemple simple

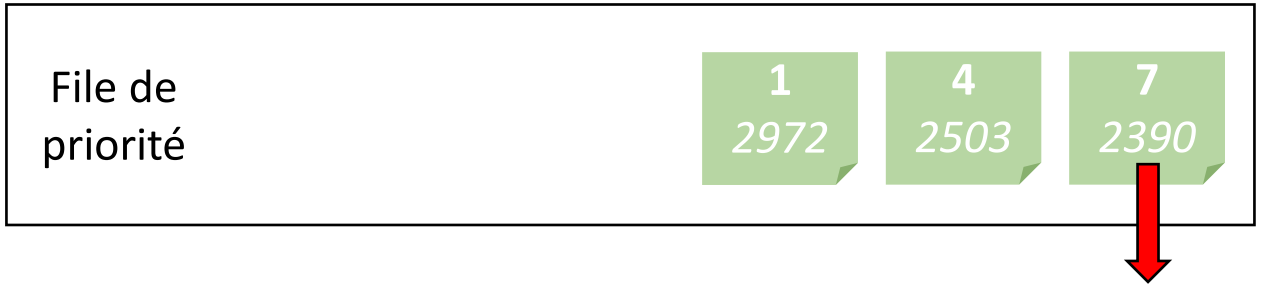
Extraction de l'élément de priorité minimale : **7**

Voisins de 7 : **[5, 6, 4, 8]**

Pour chaque voisin **v** de **7** :

- Nouveau coût g' de déplacement : $g(7) + w(7, v)$
- Si $g(v)$ non déf. ou si $g' < g(v)$
  - Mise à jour de $g(v)$
  - Calcul de l'heuristique : $h(v)$
  - Détermination du nœud parent : **7**
  - Ajout de **v** dans la **file de priorité**

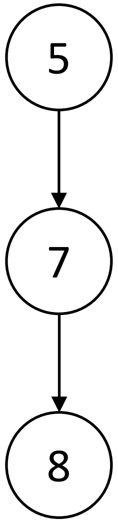


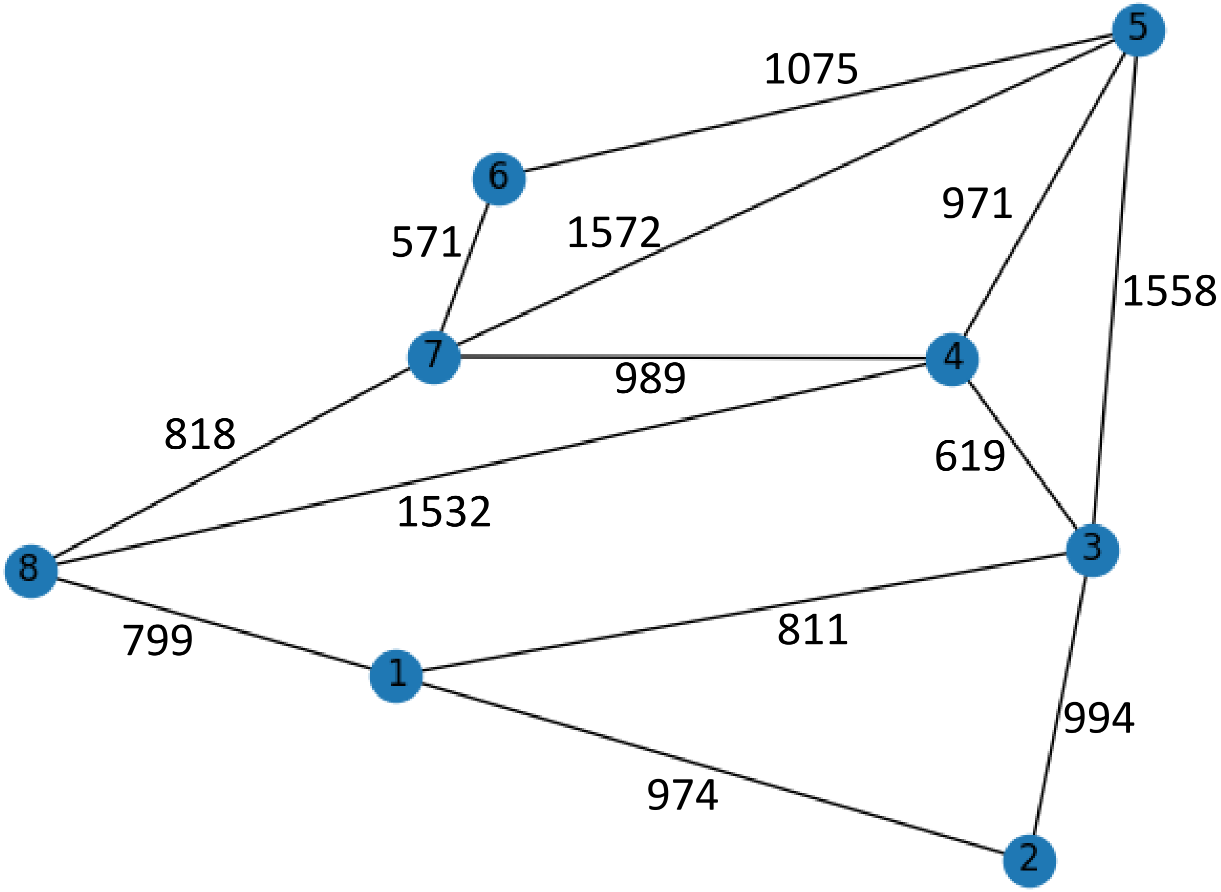


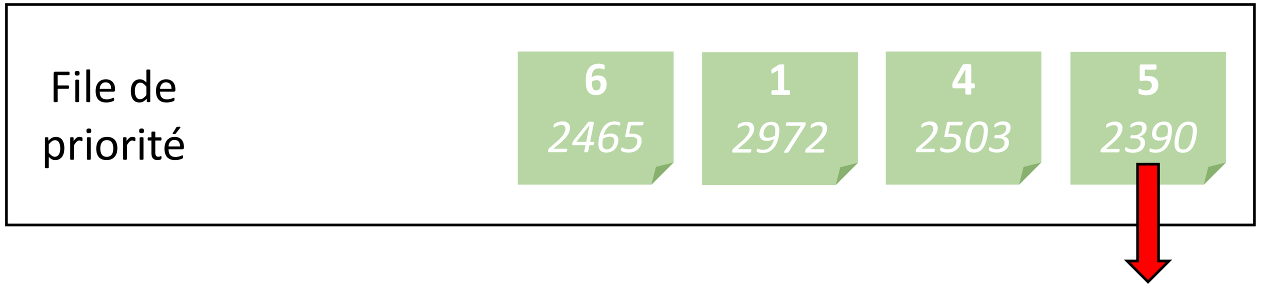
II.2. Algorithme A*

II.2.a. Définitions – notations
II.2.b. Propriétés des nœuds
II.2.c. File de priorité
II.2.d. L'algorithme sur un exemple simple

Extraction de l'élément de priorité minimale : **5**
Construction du chemin grâce aux nœuds parents :



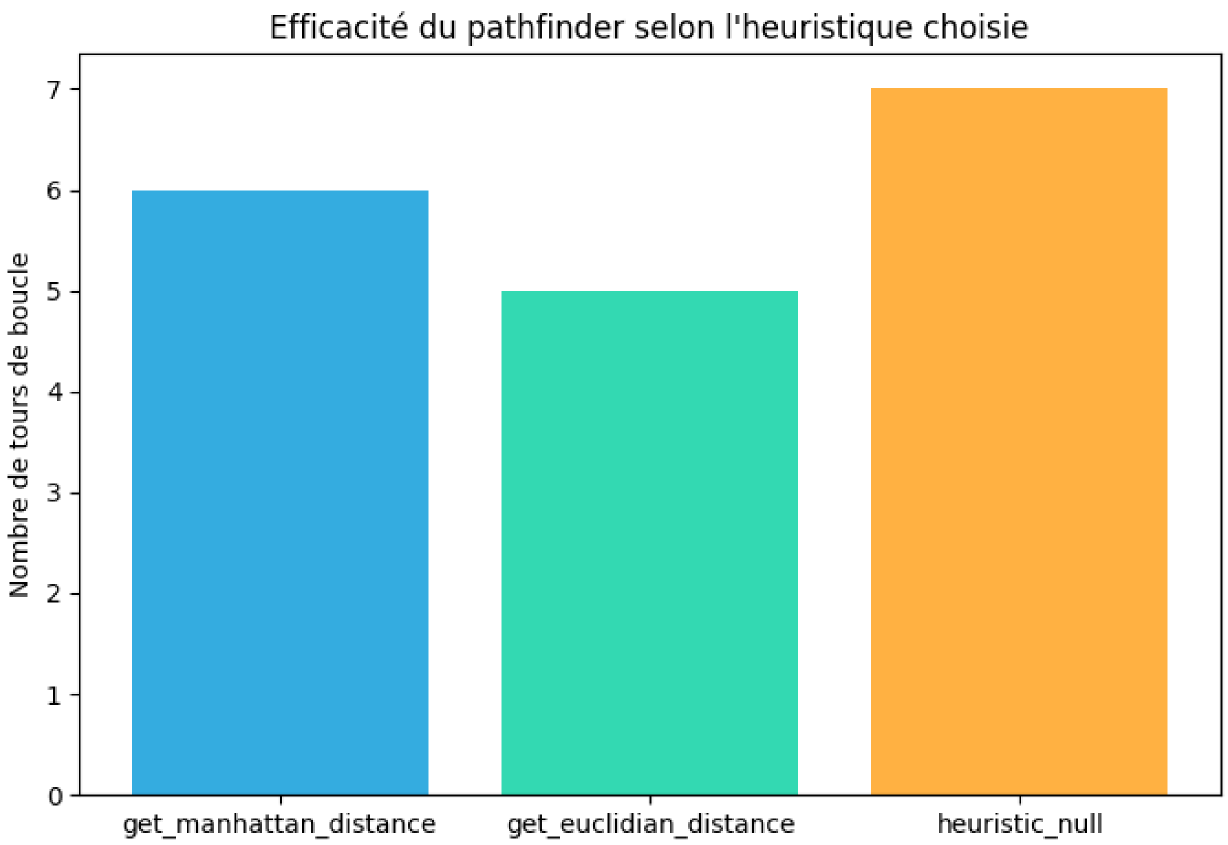




II.3. Preuve de l'optimalité de l'algorithme A*

Importance du choix de l'heuristique

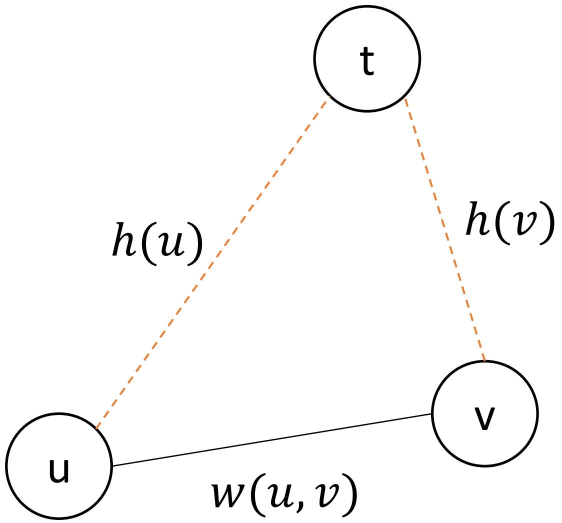
- Temps d'exécution



- Heuristique admissible

Définition : heuristique monotone (pour la recherche de t)

$$\forall (u,v) \in A,\ h(u) \leqslant w(u,v) + h(v)$$



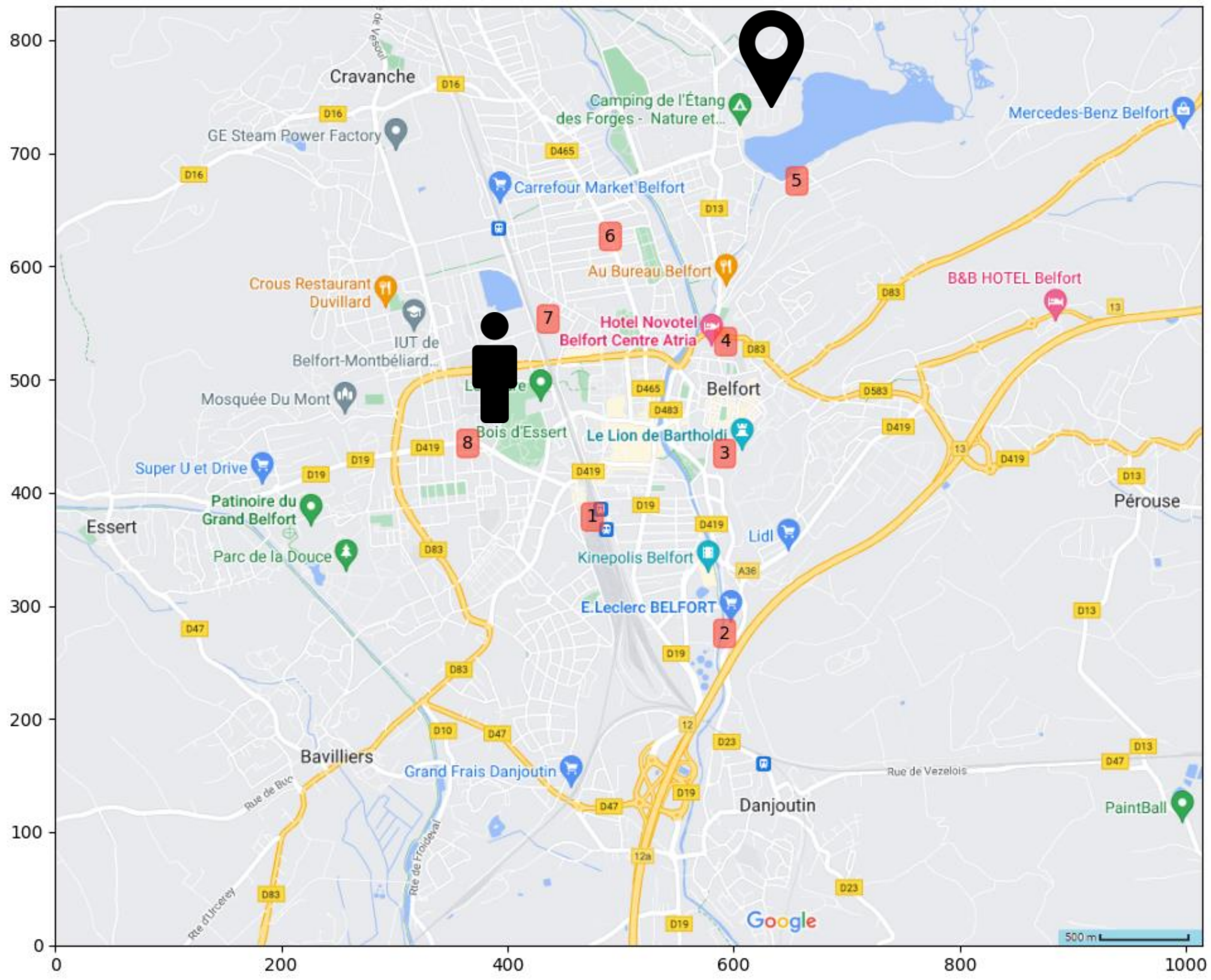
Proposition : h est monotone ⇒ h est admissible

Proposition : h est admissible ⇒ plus court chemin

(Démonstrations en annexe)

II.4. Exemple

- Recherche de la station la plus proche du lieu de départ
- Recherche de la station la plus proche de la destination
- Calcul du plus court chemin entre ces deux stations

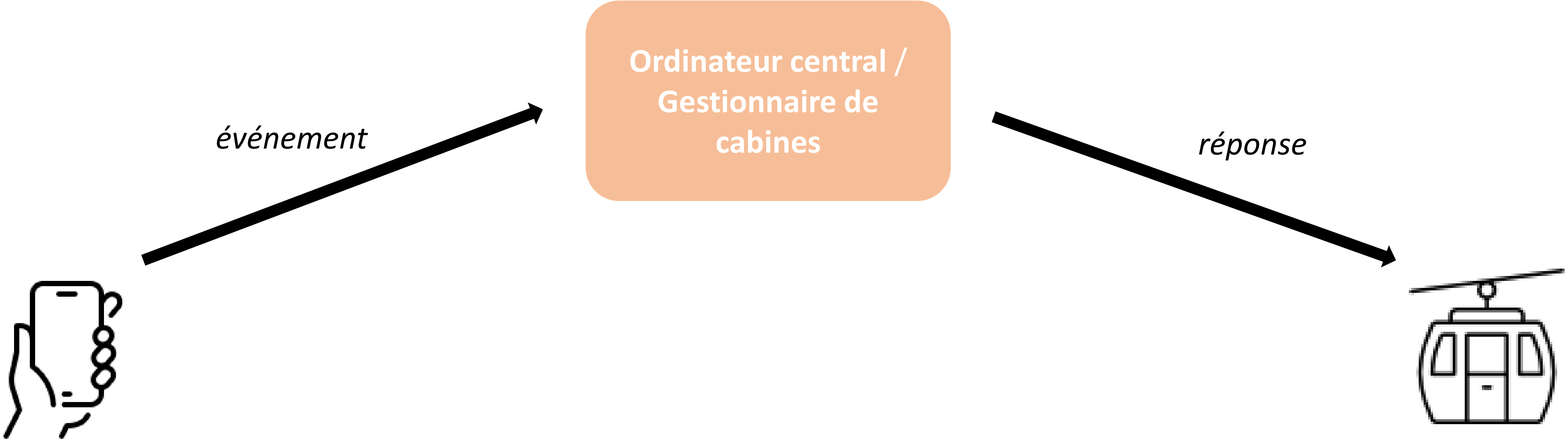


Résultat :

```
Chemin : [8, 7, 5]
Temps de trajet en cabine : ~3min
```

Comment gérer l'appel d'une cabine ?

III. Répartition des cabines

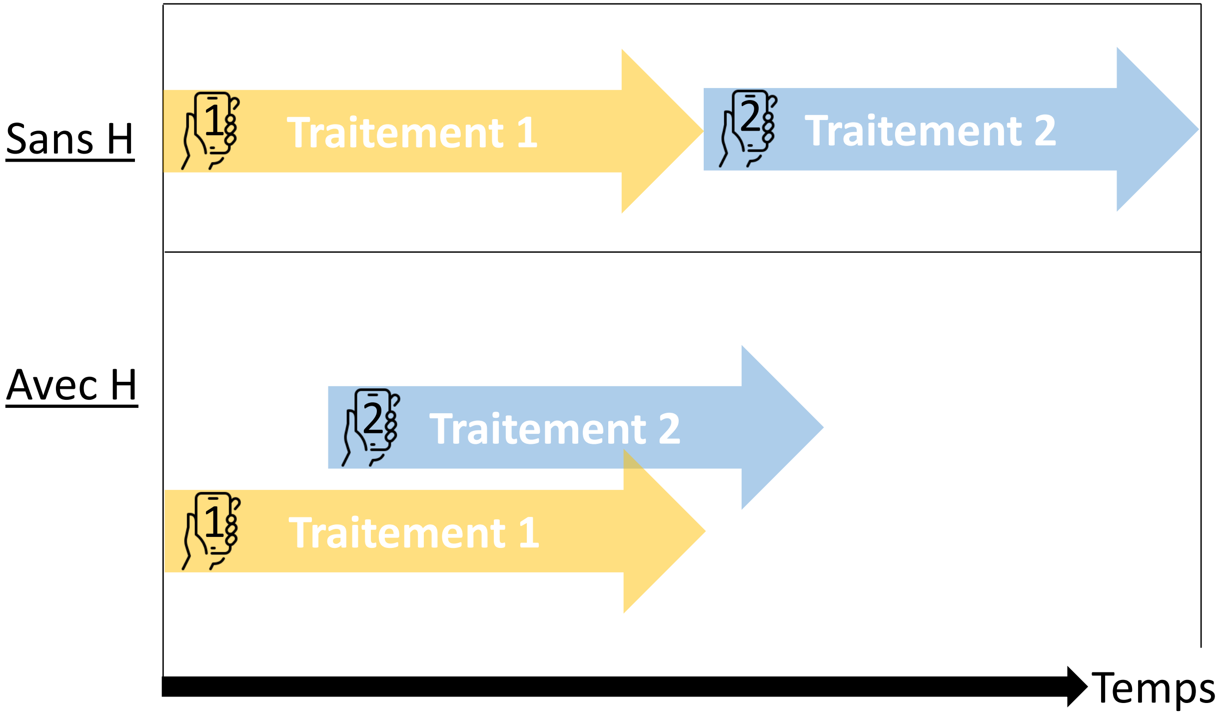


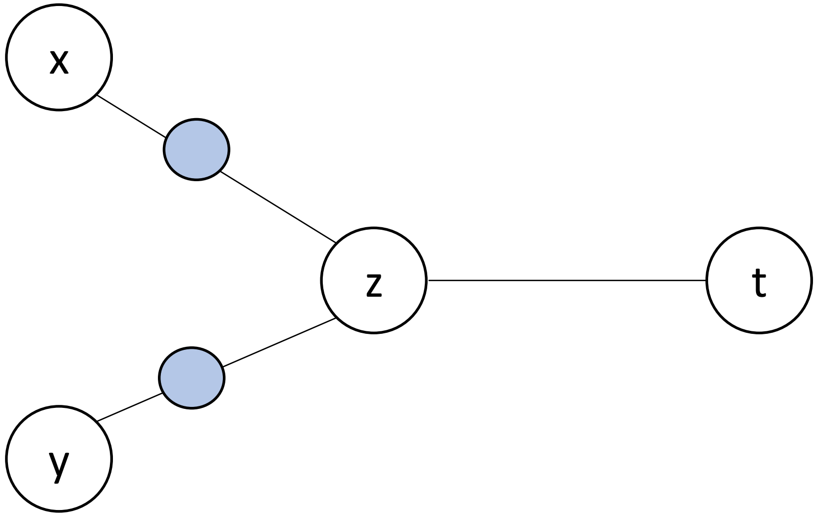
III. Répartition des cabines

III.1. Hypothèses de travail

Temps réel &
gestion de plusieurs tâches en même temps

Gestion des collisions

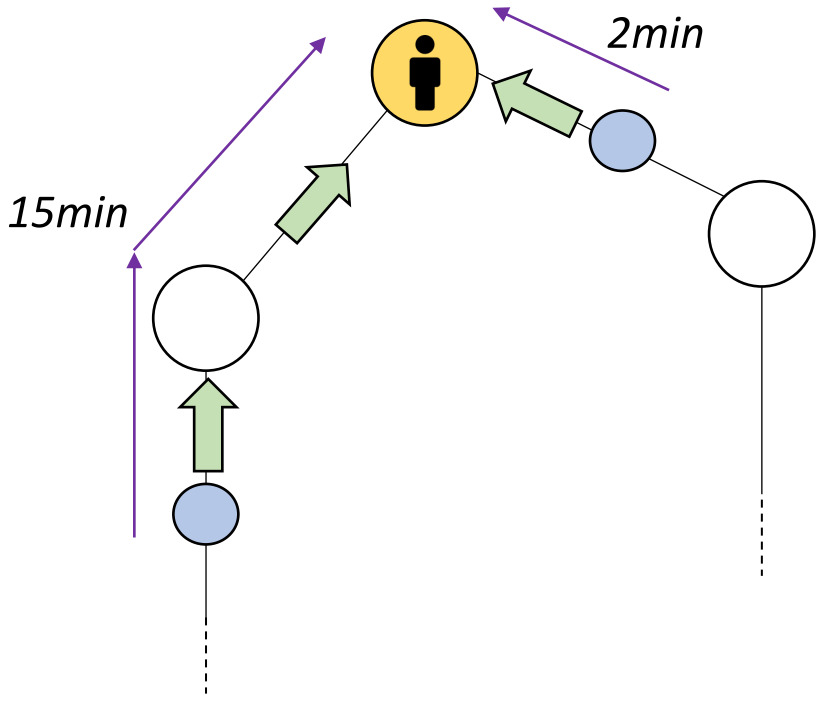




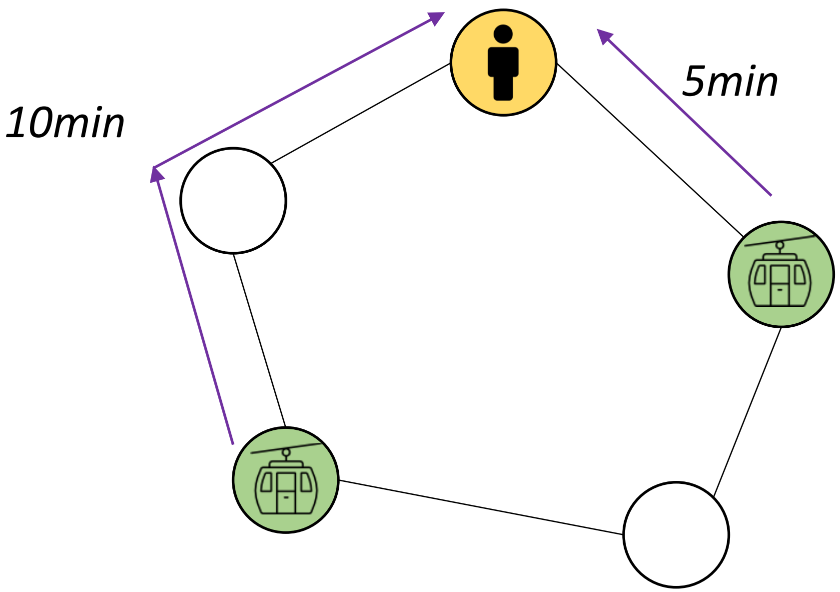
III. Répartition des cabines

III.2. Réponse (appel d'une cabine)

- Attente d'une cabine en cours de route



- Appel d'une cabine depuis une station

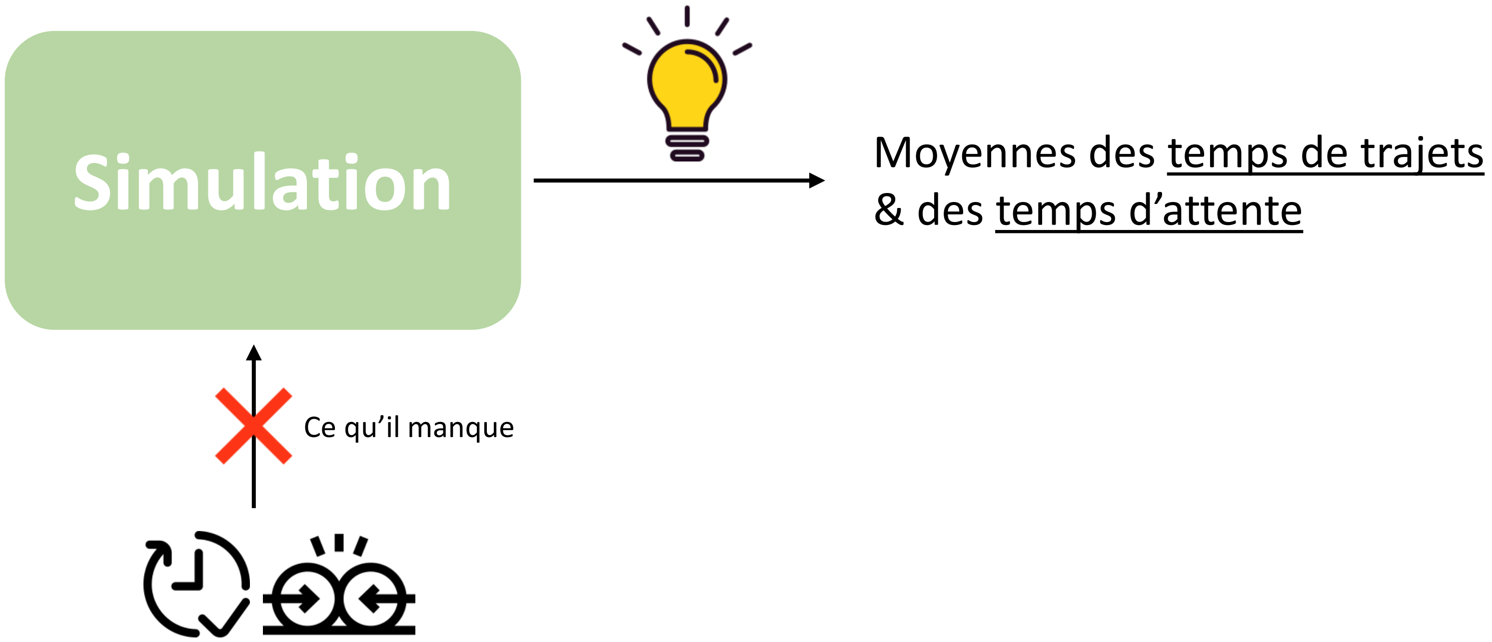


Recherche de ces informations à l'aide de la <u>base de données</u>

III. Répartition des cabines

III.2. Problème rencontré

Optimalité du trafic ?



Conclusion

Validation des objectifs :

- Représentation visuelle d’un réseau (POO, BDD) ✓
- Etude de l’algorithme A* ✓
- Etude de la répartition des cabines en fonction des demandes ✓
- Comparaison avec un moyen de transport actuel ✗

Je vous remercie de votre attention.

Annexe

Démonstrations et code

Proposition : h monotone => h admissible

Démonstration (par récurrence) :
Soient s un sommet, un plus cours chemin de s à t noté $s_0 \cdots s_n$. Récurrence :

$$\forall i \in [\![0\ ;\ n]\!]\ ,\ h(s_i) \leqslant \sum_{j=i}^{n-1} w(s_j, s_{j+1})$$

- Initialisation : $h(s_n) = h(t) = 0 = \sum_{j=n}^{n-1} w(s_j, s_{j+1})$

- Hérédité : $h(s_{i-1}) \leqslant w(s_{i-1}, s_i) + h(s_i)$ (Monotonie)

$$\leqslant w(s_{i-1}, s_i) + \sum_{j=i}^{n-1} w(s_j, s_{j+1}) \quad \text{(HR)}$$

$$= \sum_{j=i-1}^{n-1} w(s_j, s_{j+1}) \quad \blacksquare$$

<u>Proposition</u> : heuristique admissible => plus court chemin

Démonstration (par l'absurde) :

$\mathcal{C}_{\mathcal{A}^*} = s_0 \ldots s_m$: chemin de poids *d* retourné par A*

$f(t) = g(t) = d \quad \star$

$\mathcal{C} = a_0 \ldots a_n$: chemin de poids <u>minimal</u> *d'*.

Récurrence : $\forall i \in [\![0, n]\!], g(a_i) = d(s, s_i)$ et $f(a_i) \leq d'$.

- <u>Initialisation</u> : $g(a_0) = d(s, a_0) \qquad f(a_0) \leq d'$
- <u>Hérédité</u> : soit $i \in [\![0, n]\!] \mid g(a_i) = d(s, a_i)$ et $f(a_i) \leq d'$.
  Montrons que $g(a_{i+1}) = d(s, a_{i+1})$ et $f(a_{i+1}) \leq d'$.

(HR) $f(a_i) \leq d' < d$.

1. Si $g(a_{i+1})$ n'est pas défini ou si $g(a_{i+1})$ est défini
   et que $g(a_i) + w(a_i, a_{i+1}) < g(a_{i+1})$;

$g(a_{i+1}) = g(a_i) + w(a_i, a_{i+1})$

$g(a_{i+1}) = d(s, a_i) + w(a_i, a_{i+1}) = d(s, a_{i+1})$ (HR)

$f(a_{i+1}) \leq d(s, a_{i+1}) + d(a_{i+1}, t) = d'$ (Admissibilité)

2. Sinon, $g(a_{i+1})$ est déjà défini
   et $g(a_{i+1}) \leq g(a_i) + w(a_i, a_{i+1})$;

$g(a_{i+1}) \leq d(s, a_i) + w(a_i, a_{i+1})$. (HR)

$g(a_{i+1}) = d(s, a_i) + w(a_i, a_{i+1})$

$f(a_{i+1}) \leq d'$

- <u>Conclusion</u> :

$f(t) = f(a_n) \leq d' < d$, absurde.

■