TIPE : Proposition d'un modèle de système de navigation pouvant répondre au problème de migration pendulaire.

Sommaire :

Quel est le problème ?

Le transport est un des problèmes majeurs des villes.

Parmi les problèmes récurrents des grandes villes : la **migration pendulaire** = déplacement de la population le matin et le soir, entre les lieux de domiciles et les lieux de travail ou de scolarité.

Depuis quelques années, les **systèmes de navigation** se sont généralisés et sont utilisés par les usagers pour trouver leur meilleur chemin dans un réseau routier surchargé.

On peut parfois se demander si c'est vraiment efficace. *[Faire appel à l'expérience des examinateurs]*

Problème : Dans **quelles conditions** les systèmes de navigation pourraient-ils améliorer la circulation urbaine pendant les migrations pendulaires ?

Choix d'une modélisation

On se propose d'étudier le problème dans le cadre d'un modèle simplifié :

- On va assimiler l'environnement à un réseau de flots : une source, un puits, un graphe orienté et pondéré reliant la source au puits. On considère ainsi que la migration pendulaire s'effectue entre une zone résidentielle (la source) vers une zone de travail (le puits) en utilisant le réseau routier (le graphe).

- Au début, le réseau est vide. On introduit alors N véhicules au niveau de la source. Chaque véhicule partant avec un écart de Tau secondes après celui qui le précède (pour une durée totale de N x Tau secondes).

- Pour tenir compte de la possible saturation des routes, la pondération des arêtes n'est pas fixe, mais elle est fonction du nombre de véhicules présents sur l'arête.

- Enfin, on considère que toutes les voitures sur le réseau utilisent le même navigateur et qu'elles suivent scrupuleusement les consignes données par celui-ci.

Cette dernière hypothèse peut paraître un peu forte. Si c'est le cas, on pourra imaginer qu'il s'est écoulé quelques années et qu'il s'agit d'un parc de voitures autonomes.

Comment marche les navigateurs GPS ?

Les systèmes de navigation par satellite ont été conçus pour atteindre un point quelconque d'un territoire par le chemin le plus court en distance.

Mais comment trouver le chemin le plus rapide ? La plus méthode la plus simple, la plus naïve, consiste à considérer que le réseau est vide et que seul compte la longueur de chaque arête. Par exemple, le mathématicien néerlandais Dijkstra a élaboré un algorithme qui donne la solution à ce problème, c'est cet algorithme que nous allons utiliser aujourd'hui.

Évidemment, cette méthode naïve donnera la même réponse pour chaque nouvelle voiture qui arrive sur le réseau. Ce trajet optimal à vide va se trouver saturé, laissant le reste du réseau non utilisé. C'est manifestement non optimal.

Une première amélioration consiste à utiliser l'état réel du réseau au moment où le véhicule arrive au niveau de la source. C'est ce que font les navigateurs GPS actuels : ils utilisent les informations transmises par la communauté de leurs utilisateurs pour connaître cet état du réseau et déterminer le chemin le plus court sur cette base.

Mais est-ce vraiment la meilleure façon de faire ? Au fur et à mesure que la voiture va avancer sur sa trajectoire, l'état du réseau entre elle et le puits aura évolué et le meilleur chemin ne sera peut-être plus le même. C'est cette idée que j'ai voulu explorer. Peut-on encore aller plus vite ?

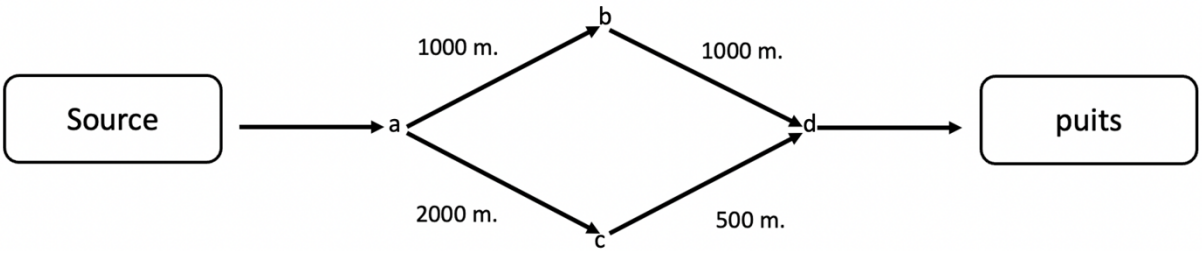
Il faut évidemment souligner que le modèle que nous allons prendre, oublie de nombreuses optimisations qui ne sont pas pris en compte, et qui rendent les navigateurs bien

plus efficaces. Nous allons en fait étudier plus concrètement, la façon avec laquelle les navigateurs distribuent initialement les voitures sur un réseau routier

Le nouveau modèle proposé :

Nous allons donc proposer une solution différente. Une des suppositions du problème que nous avons posé et que l'ensemble des véhicules sont liés à ce navigateur. Nous pouvons donc prévoir où seront toutes les voitures à tout moment sur le graphe. Nous allons donc utiliser cette astuce pour modéliser le flux de voitures sur toutes les arêtes à chaque instant. Cela nous permettra de palier le problème mis en évidence. Et donc de pouvoir distribuer l'ensemble des véhicules sur le réseau routier de manière optimal, tel que l'écart type soit le plus petit possible et que le temps de trajet moyen le soit aussi.

Mais concrètement, comment faire ?



Le tableau des états du graphe :

Ce model marche sur le fait que nous pouvons prévoir où seront l'ensemble des véhicules. Nous devons donc sauvegarder la position de chaque voiture à chaque instant sur le réseau routier.

Pour cela nous allons définir l'objet suivant : **Le tableau des états du graphe**.

Chaque ligne représentera un intervalle d'une seconde et chaque colonne représentera une arête du graphe.

Le nombre de lignes se définissent par le nombre de secondes que vont prendre toutes les voitures

1) On définit le tableau :

ab	ac	bd	cd

2) On introduit une voiture qui va faire le trajet suivant :
   `V1 : a - (3s) -> b - (3s) -> d`
   Le tableau mis à jour sera alors :

	ab	ac	bd	cd

[0 ;1[	1	0	0	0
[1 ;2[	1	0	0	0
[2 ;3[	1	0	0	0
[3 ;4[	0	0	1	0
[4 ;5[	0	0	1	0
[5 ;6[	0	0	1	0

3) On introduit la voiture suivante :
`V2: a - (3s) -> b - (3s) -> d`
Le tableau mis à jour sera alors :

	ab	ac	bd	cd
[0 ;1[	2	0	0	0
[1 ;2[	2	0	0	0
[2 ;3[	2	0	0	0
[3 ;4[	0	0	2	0
[4 ;5[	0	0	2	0
[5 ;6[	0	0	2	0

4) On introduit la voiture suivante :
`V3 : a - (5s) -> c - (2s) -> d`
Le tableau mis à jour sera alors :

	ab	ac	bd	cd
[0 ;1[	2	1	0	0
[1 ;2[	2	1	0	0
[2 ;3[	2	1	0	0
[3 ;4[	0	1	2	0
[4 ;5[	0	1	2	0
[5 ;6[	0	0	2	1
[6 ;7[	0	0	0	1

Le modèle de vitesse :

Le tableau d'état nous permet donc de savoir le nombre de voitures à chaque instant sur une arête.
Pour pouvoir prévoir la vitesse sur une arête, on va donc créer un modèle de vitesse.

On va définir la pondération des arêtes du graphe comme étant :
- Son sommet de départ et d'arrivé
- Sa vitesse limite (par exemple 50km/h en agglomération)
- La longueur de l'arête

On peut aussi considérer que l'arête va aussi se définir par sa colonne dans la table des états, car cela permet de savoir le nombre d'arêtes qu'il y a sur cette route à chaque instant.

On définit donc un modèle de vitesse dépendant des conditions limites, et du nombre de personnes qui seront sur l'arête à chaque instant.

`vitesse` est la fonction donnant la vitesse sur une route selon ces conditions initiales, et le nombre de voitures sur l'arête.

Le poids des arêtes :

Pour pouvoir calculer le plus court chemin entre la source et le puits, il est nécessaire de définir le poids des arêtes sachant que celui-ci va varier durant le temps (la densité de trafic n'étant pas le même durant l'ensemble de l'expérience, la vitesse des voitures ne va donc pas être constante.

Nous allons donc définir le poids des arêtes comme étant la fonction `temps` suivante :

Soit t la durée d'arrivée à l'arête, `temps(t)` donnera le temps qu'il faut pour traverser l'arrête si une voiture si introduite à partir de t.

Pseudo-code de la fonction `temps` :

```
ln, liste des voitures sur l'arête ab, une case est une seconde
t, la date à laquelle arrive la voiture (int)
longueur_ab, la longueur de ab

temps (t, longueur_ab) :
     duree = 0
     longueur = 0
     while longueur < longueur_ab :
          duree += 1
          vitesse_debit = vitesse(ln[duree], ab)
          longueur += vitesse_debit
     return duree
```

`duree` sera donc le temps qu'une voiture mettra pour traverser l'arête, on définit donc cette durée comme le poids de l'arête à l'instant t.

L'algorithme du plus court chemin :

Comme cela est dit précédemment, nous allons nous baser sur l'algorithme de plus court chemin conçu par Dijkstra. La seule différence est que nous allons considérer le poids des arêtes comme étant la fonction définit juste au-dessus.

Mesure de la performance de l'algorithme

Pour mesurer la performance de ce modèle, nous allons le comparer à deux autres modèles.

Le modèle naïf :

Ce modèle va appliquer l'algorithme de Dijkstra de manière naïve, et va donner le même trajet à chaque voiture : celui qui serait optimal si le graphe était vide.
On peut considérer ce navigateur comme le pire, car dans la situation où le graphe va recevoir un nombre important de voitures, celui-ci va être complétement saturé, puisque tout le monde suivra exactement le même itinéraire.

Le modèle d'aujourd'hui :

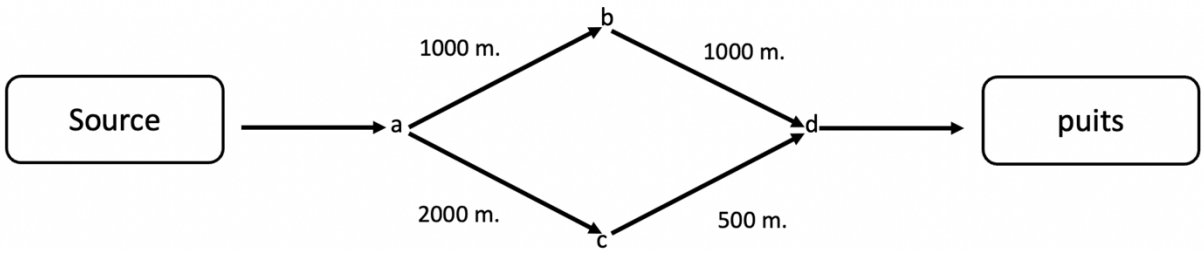
Le modèle présenté au début, c'est celui que la généralité des gens utilisent. Il permettra de savoir si le nouveau modèle proposé est plus performant.
Bien sur celui-ci n'est pas comparable à un navigateur comme Waze, car il est évidemment impossible à mon échelle de reprogrammer l'ensemble des optimisations de Waze.

Pour le programmer, nous avons aussi utiliser un algorithme de type Dijkstra, mais pour définir le poids nous n'avons utiliser que la ligne contenant la période durant laquelle une nouvelle voiture a demandé son trajet.

Les résultats obtenus :

	Le meilleur résultat
	Le résultat moyen
	Le pire résultat

Exemple 1 :



Trajet le + court : 144

Nombre de voitures : 1000
Temps d'attente entre les voitures : 0.1

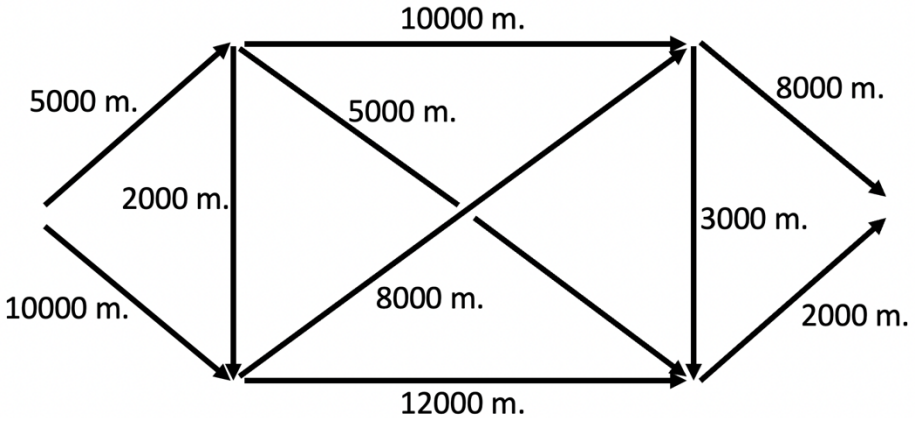
	Naïf	Initial	Dynamique
Durée total	1435	895	802
Trajet le + long	1336	796	703
Médiane	750	433	432
Moyenne	744	437	430

Écart - type	346	171	159

Nombre de voitures : 4000
Temps d'attente entre les voitures : 0.1

	Naïf	Initial	Dynamique
Durée total	5191	2754	2686
Trajet le + long	4792	2355	2287
Médiane	2493	1234	1234
Moyenne	2486	1230	1229
Écart - type	1336	615	613

Exemple 2 :



Trajet le + court : 1080

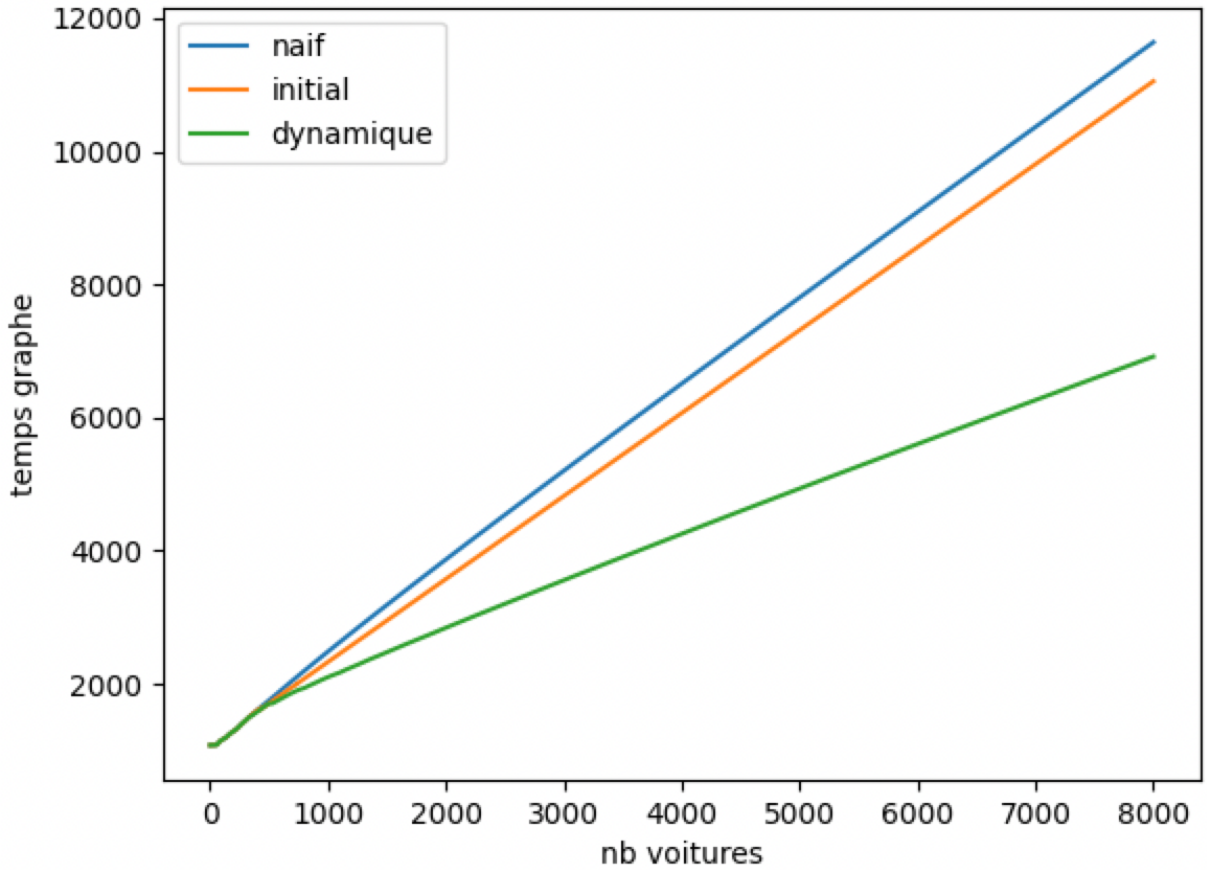
Nombre de voitures : 1000
Temps d'attente entre les voitures : 0.1

	Naïf	Initial	Dynamique
Durée total	2488	2332	2104
Trajet le + long	2389	2233	2005
Médiane	1707	1658	1656
Moyenne	1704	1648	1588
Écart - type	398	346	283

Nombre de voitures : 4000
Temps d'attente entre les voitures : 0.1

	Naïf	Initial	Dynamique
Durée total	6513	6071	4255
Trajet le + long	6114	5672	3856

Médiane	3673	3379	2645
Moyenne	3636	3376	2607
Écart - type	1464	1328	761



Nous pouvons donc clairement voir que l'algorithme dynamique est de plus en plus efficace, comparé au modèle initial, lorsque le nombre de voitures augmente. Cela montre que dans le cas de cet exemple le but est rempli.

Des résultats plus globaux :

Pour s'assurer de la fiabilité accrue, de ce nouveau modèle.
Nous avons testé et comparé les deux différents modèles sur des graphes aléatoires et plus représentatifs des réseaux routiers.

Sur 50 villes d'environ 30 sommets qui sont en moyenne de degré 3.
Nous avons eu les résultats suivant pour 3000 voitures :

En moyenne	Initial	Dynamique
Durée totale (en s)	4149	3450
Nombre de trajets différents	3.64	5.34
Trajet le + long (en s)	3852	3152
Médiane (en s)	2520	2400
Moyenne (en s)	2506	2318
Ecart-type (en s)	730	562

Les points à développer

Plusieurs points pourraient être à développer, pour que ce modèle soit plus performant :

- Le réseau d'une ville étant bien plus vaste que ceux qui ont été pris en exemple, il serait intéressant de développer un algorithme A*, qui permettrait de limiter la complexité du programme dû à l'algorithme Dijkstra.
- Il serait aussi intéressant de faire un pré-traitement du graphe. Dans le cadre de la théorie algorithmique des jeux, il a été prouvé qu'une distribution pouvant paraitre intuitive ne l'est finalement pas (paradoxe de Braess).
    - Je n'ai pas introduit ce pré-traitement dans le navigateur, car je ne pouvais pas programmer les optimisations liées au navigateur Waze, il n'y aurait alors plus eu de sens à comparer les deux modèles.
- Optimisation du modèle de vitesse selon les villes. Ce modèle ayant comme objectif de régler le problème de migration pendulaire. Il serait intéressant d'optimiser le modèle de vitesse selon la ville et ces contraintes, cela permettrait une optimisation efficace de la distribution de voitures.