École Centrale Lyon

UE ELC

Algorithmes collaboratifs et applications

Rapport

Rapport BE

Élèves :
Yuhang YAN
Hanxuan Wei

Enseignant :
Prénom Nom

20 mars 2022

Table des matières

1 Introduction

West Jatarka est un des villes qui ont des plus mauvaises conditions de circulation dans le monde entier dû à la croissance rapide du nombre de véhicules privées. D'après un enquête réalisé par le gouvernement de West Jatarka, seulement 2.07% de la population choisir de prendre le bus à cause des arrêts de bus localisés de manière inappropriée. Pendant ce BE, nous cherchons à trouver une solution qui donnera des localisations appropriées des arrêts de bus à partir des coordonnées des résidences et puis calculera un plus court chemin parcourant ces arrêts. Nous utiliserons le méthode de décalage moyen pour la première partie et l'algorithme de colonie fourmi pour la deuxième. Finalement, nous visualiserons le processus.

2 Localisation des arrêts par décalage moyen

Le clutering à décalage moyen permet de calculer les centres des clusters d'un ensemble de points. Avec un bandwidth fixé, pour chaque point de l'ensemble, nous souhaitons pouvoir trouver un centre dont la distance avec ce point est inférieure au bandwidth. En utilisant ce méthode sur l'ensemble des résidences, nous pouvons obtenir une allocation d'arrêts de bus qui permet à tous les citoyens de trouver un arrêt aux alentours de chez eux. Cela résoudre bien le problème.

Avant de commencer à expliquer le principe, nous donnons d'abord deux formules :

— Noyau de Gaussian : $K(s-x) = e^{-c\|s-x\|^2}$

— Centre de cluster :$m(x) = \frac{\sum_{s\in S} K(s-x)s}{\sum_{s\in S} K(s-x)}$, avec x le latitude ou le longitude du centre ancien du cluster et s les points appartenant au cluster

Étant donné un bandwidth, nous pouvons calculer des centres par itération. Prenons l'ensemble des points donnés comme la liste initiale des centres. Dans chaque itération, nous répétons les étapes suivantes :

1. Nous avons une liste de centres anciens. Pour un centre, des points appartenant à son cluter sont ceux ayant une distance avec lui inférieur au bandwidth donné.
2. Les deux nouvelles coordonnées d'un centre sont données par la formule de centre du cluster ci-dessus en prenant x et s des longitudes et puis des latitudes.
3. Nous simplifions la nouvelle liste de centres selon le principe suivant : si deux centres ont une distance inférieure au bandwidth entre eux, ce ayant moins de points dans son cluster sera supprimé.
4. Nous sortirons l'itération jusqu'à ce que la nouvelle liste de centres soit équivalente à l'ancienne. Sinon nous recommençons une nouvelle itération avec la nouvelle liste de centres.

```
# If the distance between two kernels is less than the bandwidth
# then we have to remove one because it is a duplicate. Remove the one with fewer points.
unique = np.ones(len(centers_num), dtype=bool)# to indicate which kernel will be removed
centers = np.array([np.array(item[0]) for item in centers_num])
for index, center in enumerate(centers_num):
    if unique[index]:
        # computer kernels(cen) in the area of center i
        for cen in range(len(centers)):
            if not unique[cen]:
                pass
            if np.linalg.norm(centers[cen] - centers[index]) < self.bandwidth_ and cen != index:
                if center[1] < centers_num[cen][1]:
                    unique[index] = 0
                    break
                else:
                    unique[cen] = 0
centers_final = centers[unique]
self.centers_ = centers_final
```

FIGURE 1 – Supprimer un centre

Tous les démarches ci-dessus sont groupées dans la classe *MeanShift* ayant comme variables internes *self.bandwidth_* et les centres calculés selon ce bandwidth *self.centers_* avec la fonction *self.fit(self,data)*. En créant une variable de classe MeanShift et introduisant les coordonnées des résidences comme le paramètre data dans le méthode *fit*, nous obtenons les centres des clusters comme les coordonnées des arrêts de bus.

Remarquons que le résultat sera fortement influencé par la valeur de bandwidth. Pour évaluer la performance du résultat, nous introduisons une autre variable interne *self.err_* dans la classe *MeanShift*. La valeur de cette variable est le nombre de points non recouvru par aucun centre. Dans ce cas, plus petit est cette valeur, plus performant est le résultat.

Finalement, quand nous appliquons ce méthode sur un groupe de résidences, nous allons créer une série de variables de classe MeanShift avec des bandwidths différents. Après avoir calculé les groupes de centres correspondants, nous allons choisir ce qui a le plus grand bandwidth parmi les groupes qui ont le *self.err_* le plus bas. Cela permet de économiser des ressources en créant le moins d'arrêts possibles tout en restant le plus performant.

```
# create a serie of bandwidth then choose
#the one whose error is the smallest
band_serie = np.linspace(150, 75, 20)
```

FIGURE 2 – Créer une série de bandwidth

3 Plus court chemin par colonie de fourmi

L'algorithme de colonie de fourmis est un algorithme bionique inspiré du comportement de recherche de nourriture des fourmis dans la nature. Dans la nature, pendant le processus de recherche de nourriture des fourmis, la colonie de fourmis peut toujours trouver un chemin optimal entre le nid de fourmis et la source de nourriture.

Dans l'algorithme conçu par nous, le nombre de toutes les fourmis dans la colonie de fourmis est 50 dans chaque itération. La phéromone entre toutes les villes est représentée par une matrice. La matrice de phéromone est renouvelée par l'équation

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}$$

où $\tau_{ij} = \dfrac{Q}{L}$, L est la distance que la fourmi a parcouru

Q est la phéromone apportée par une fourmi.

La fourmi dont le chemin est le plus court est *best_ ant* et va être rouvelée par chaque itération. Chaque fourmi a sa propre mémoire, et une table *open_ table_ city* est utilisée pour stocker les villes que la fourmi a visitées, indiquant qu'elle ne pourra pas visiter ces villes lors de recherches futures. Il y a aussi une matrice *select_ citys_ prob* pour enregistrer la probabilité qu'une fourmi se déplace d'un arrêt à un autre arrêt. Cette probabilité est calculée par l'équation :

$$P_{ij} = \frac{(\tau_{ij}^{\alpha})(\eta_{ij}^{\alpha})}{\sum(\tau_{ij}^{\alpha})(\eta_{ij}^{\alpha})}$$

L'algorithme s'exécute au maximum 400 fois pour trouver le chemin le plus court.

Le processus de calcul de l'algorithme de colonie de fourmis est le suivant :

1. Initialisation. 50 fourmis sont créées dans les arrêts au hasard.
2. Selon la matrice de phéromones actuelle, chaque fourmi parcourt les arrêts. La fourmi dont le chemin est le plus courte est enregistré.
3. Renouveler la matrice de phéromones selon les chemins parcourus par les 50 fourmis.
4. 50 nouvelles fourmis sont créés, on recommence à partir de l'étape 2 jusqu'à l'itération maximale.
5. Le chemin de la meilleure fourmi actuelle est le chemin le plus court.

4 Visualisation

Dans cette partie finale, nous avons visualisé le programme. Le fenêtre est comme ci-dessous :

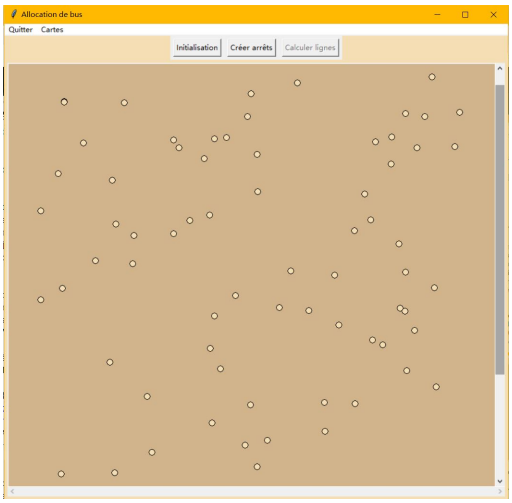

FIGURE 3 – Fenêtre principale

La fenêtre est composée d'un menu, d'une barre d'outils et un canvas. Initialement, des résidences dans une carte sont présentées présentées sur le canvas par des points beiges. L'option **'Quitter'** dans le menu permet de rompre le programme et fermer la fenêtre. L'option **'Cartes'** fait appel un radiobox qui permet de changer de nouvelle carte préexistant dans le programme et la carte actuelle sera marquée. En ajoutant des éléments dans la variable carte du code source, le programme rajoutera directement des nouvelles cartes dessus.

Sur la barre d'outils, le bouton **'Initialisation'** sert à arrêter les calculs en cours et à effacer tous les arrêts et les chemins tracés dans le canvas. Ce bouton va aussi activer le bouton 'Créer arrêts' et paralyser le bouton 'Calculer lignes' car il est impossible de calculer le plus court chemin avant de créer les arrêts. Le bouton 'Créer arrêts' trace les centres de cluters (donc les arrets de bus) calculés par décalage moyen en croix verts.

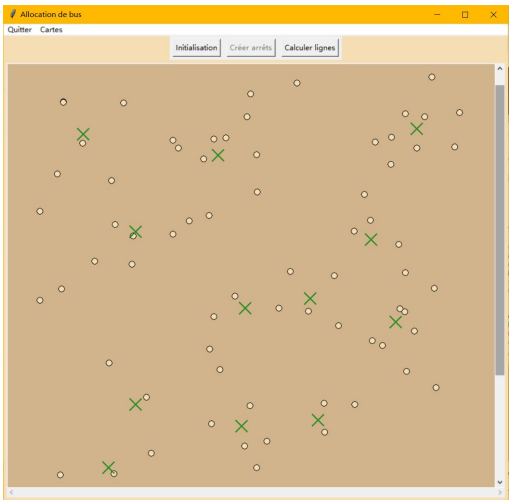

FIGURE 4 – Arrêts créés

Après que ce calcul sera fini, le bouton **'Créer arrêts'** sera paralysé et le bouton **'Calculer lignes'** sera activé. Après une clique sur ce dernier, l'algorithme de colonie fourmi sera lancé. Le chemin le plus court pendant le calcul se tracera en temps réel en ligne noire fine et le résultat final se tracera en ligne verte grosse.

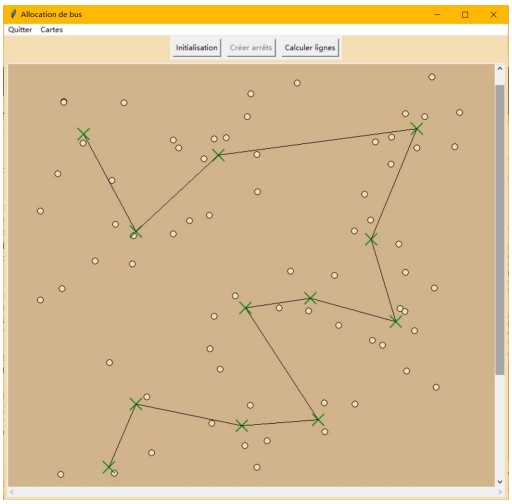

FIGURE 5 – Chemin plus court en temps réel

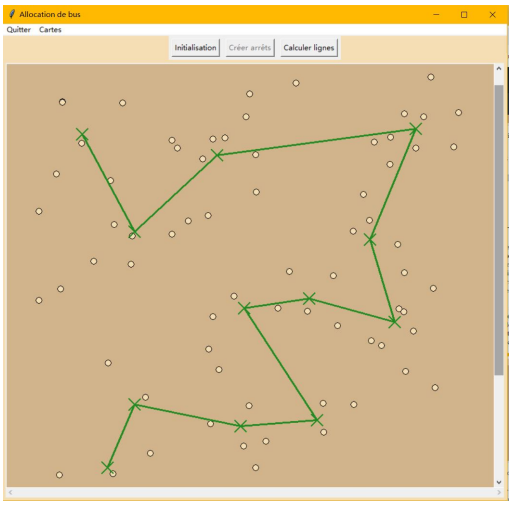

FIGURE 6 – Résultat final