



# Détermination de l'emplacement des marchés en fonction de l'algorithme de clustering

Université de Montréal

Auteurs : Mohsen Feizabadi  
Professeurs : Jean Daoust, Antoine Prince

# Table des matières

|   |           |
|---|-----------|
| <b>1 Avant-propos</b>   | <b>4</b>  |
| <b>2 Arcpy en Tant qu'Interface Entre Python et ArcGIS</b>                      | <b>5</b>  |
| <b>3 Le Projet</b>  | <b>6</b>  |
| <b>4 Prétraitement</b>  | <b>7</b>  |
| 4.1 Sources de Données . . . . .  | 7         |
| 4.2 Données Requises . . . . .  | 9         |
| 4.3 Comment Créer Les Données Requises . . . . .                                | 9         |
| 4.4 Géoréférencement et Numérisation . . . . .                                  | 11        |
| <b>5 Analyse des Données</b>  | <b>12</b> |
| 5.1 Algorithme de Mise en Grappe (Clustering) . . . . .                         | 13        |
| 5.1.1 Définition de Mise en Grappe . . . . .                                    | 13        |
| 5.1.2 Mise en Grappe Dans Notre Projet . . . . .                                | 14        |
| 5.2 Groupement des Marchés afin de Déterminer le Dépôt le Plus Proche . . . . . | 18        |
| 5.3 Déterminer l'Emplacement des Dépôts . . . . .                               | 20        |
| <b>6 Résultats</b>  | <b>22</b> |
| <b>7 Conclusion</b>   | <b>24</b> |
| <b>8 Prochaine Étude</b>  | <b>25</b> |
| <b>9 Annexe</b>   | <b>26</b> |
| 9.1 Annexe 1 . . . . .  | 26        |
| 9.2 Annexe 2 . . . . .  | 27        |
| 9.3 Annexe 3 . . . . .  | 31        |
| <b>Références</b>   | <b>34</b> |

# Table des figures

|    |   |    |
|----|---|----|
| 1  | (GIS, 2018) . . . . .   | 5  |
| 1  | Ajouter des attributs géométriques et créant une coordonnée pour chaque polygone . . . . .  | 10 |
| 2  | Les étapes de la procédure de Georeferencing . . . . .  | 11 |
| 3  | Procedure de mise en grappe (Weston.pace, 2007) . . . . .   | 13 |
| 4  | Importer des données et les préparer pour le calcul . . . . .   | 14 |
| 5  | Scatterplot de données avec une couleur différente . . . . .  | 14 |
| 6  | Première boucle du programme . . . . .  | 14 |
| 7  | Deuxième boucle du programme . . . . .  | 15 |
| 8  | Trouver des lieux commerciaux à proximité des centres qui ont été sélectionnés à l'étape précédente. . . . .  | 16 |
| 9  | Les tracés (plot) de résultats . . . . .  | 17 |
| 10 | Sauvegarder les résultats . . . . .   | 17 |
| 11 | Exportation du résultat des marchés sélectionnés dans ArcGIS . . . . .  | 18 |
| 12 | Importation des données requises dans la deuxième partie du traitement . . . . .  | 18 |
| 13 | Calcul de la distance entre les points et la valeur de la combinaison . . . . .   | 19 |
| 14 | Groupement les points en supprimant chaque groupe après le calcul . . . . .   | 20 |
| 15 | Tracer (plot) les groupes . . . . .   | 20 |
| 16 | Détermination de la place du dépôt en fonction de chaque groupe de marchés . . . . .  | 21 |
| 17 | Exportation du résultat des dépôts sélectionnés dans ArcGIS . . . . .   | 21 |
| 18 | Certaines parties des codes d'utilisation des terres. (Retour) . . . . .  | 26 |
| 19 | Ville de Montréal (lignes bleues) avec ses quartiers (lignes rouges)(Retour) . . . . .  | 27 |
| 20 | Ville de Montréal après suppression des lignes et des polygones supplémentaires (Retour) . . . . .  | 28 |
| 21 | Les résultats de la procédure de Georeferencing avec 5 points (Retour) . . . . .  | 29 |
| 22 | Différence entre le polygone disponible des quartiers de Montréal et celui numérisé (ligne bleue) (Retour) . . . . .  | 30 |
| 23 | Distribution d'emplacements résidentiels, commerciaux, institutionnels, industriels et de bureaux à Montréal. Différentes couleurs montrent différents types de données. (Retour) . . . . . | 31 |
| 24 | Les résultats de différents mise en grappe de 5 grappes à 20 grappes (Retour) . . . . .   | 32 |
| 25 | Emplacement des dépôts pour chaque groupe de marchés (3 points) (Retour)  | 33 |

## Liste des tableaux

|   |  |    |
|---|--|----|
| 1 | Codes de chaque quartier de Montréal . . . . .   | 8  |
| 2 | Différence entre le polygone disponible des quartiers de Montréal et celui numérisé. . . . . | 12 |
| 3 | Valeur de poids attribuée pour chaque utile sol . . . . .                                    | 15 |
| 4 | Coordonnées des marchés et des dépôts sélectionnés . . . . .                                 | 22 |
| 5 | Nombre de données pour chaque type de utile sol (Retour) . . . . .                           | 26 |

# 1 Avant-propos

Ce projet a été réalisé à partir de données disponibles et sous licence qui sont distribuées par le gouvernement ou des entreprises du gouvernement. Le but de ce projet est l'étude des fonctions spatiales pour résoudre un problème avec une combinaison de données spatiales et non-spatiales. Les sources de toutes les données utilisées sont mentionnées dans la section pertinente.

Pour écrire le projet, la licence de logiciel libre **LAT<sub>E</sub>X** a été appliquée. Pour l'analyse et la visualisation des données, **ArcGIS** et paquet **Spyder** (python) du logiciel **Anaconda** ont été utilisés.



## 2 Arcpy en Tant qu'Interface Entre Python et ArcGIS

Il existe différentes bibliothèques qui sont utilisées dans un environnement python et chacune d'elles définit ou crée des conditions pour résoudre des problèmes ou des situations spécifiques. Par exemple, pour calculer certaines fonctions mathématiques spécifiques, la bibliothèque "**math**" doit être chargée en python ou pour représenter les cartes géoréférencées ou globales, les "**matplotlib.pyplot**", "**pandas**", "**mpl\_toolkits.basemap**", ... bibliothèques devraient être utilisé.

Par la même mentalité, nous devons utiliser la bibliothèque particulière pour faire un pont entre Python et ArcGIS. Cette bibliothèque s'appelle "**Arcpy**".

Les principaux points d'utilisation de l'arcpy sont :

- Les données d'importation et d'exportation d'ArcGIS (c.-à-d. principalement des shapefiles) ont des caractéristiques spatiales. Cela signifie qu'en plus de la valeur d'existence pour chaque objet (c.-à-d. point, ligne et polygone), leurs emplacements sur l'espace doivent également être définis.
- L'application de certaines fonctions spatiales (par exemple clip, buffer, dissolve, ...) nécessite de définir une bibliothèque différente car ces processus ne sont pas des calculs de routine comme l'ajout de deux chiffres ou leur soustraction.

Cependant, cette question pourrait augmenter. Pourquoi n'utilisons-nous pas l'environnement python qui se trouve dans ArcGIS ? Il doit être mentionné que, il peut être utilisé mais pas pour une grande procédure. En d'autres termes, le python d'ArcGIS est très utile pour exécuter une instruction au cours d'une procédure.

D'un autre coté, en utilisant individuellement l'environnement python, nous pouvons en tirer parti pour résoudre les questions ArcGIS.

Il y a différentes fonctions python pour l'analyse spatiale et leur connaissance est difficile, mais "**Model Builder**" (langage de programmation visuel pour construire des workflows de géotraitement ([ESRI, 2017](#))) d'ArcGIS nous aide à importer la fonction spatiale requise et à exporter les structures en code python.

Dans cette étude, le projet sera défini dans la section [Le Projet](#). La préparation des données nécessaires et leur traitement préalable seront décrites dans la section [Prétraitement](#). Le traitement de l'analyse des données sera expliqué dans la section [Analyse des Données](#) et les résultats du projet seront discutés dans la section [Résultats](#) et la section [Conclusion](#).

Afin de préserver la continuité du contenu de la procédure, quelques figures sont ajoutés dans la section [Annexe](#).



(GIS, 2018)

### 3 Le Projet

Une chaîne d'hypermarchés souhaite évaluer la situation de l'établissement de plusieurs marchés dans la ville de Montréal. Cette décision doit être basée sur la population des régions. Ces marchés doivent avoir une superficie comprise entre 5000 et 20000  $m^2$  et disposer de moins de 35% de terrain pour le stationnement. En outre, cette compagnie décide de vendre des marchandises en ligne. Par conséquent, ils veulent établir des dépôts qui répondent aux besoins des marchés et des ventes en ligne.

Dans la première phase de ce projet, les directeurs de cette entreprise souhaite utiliser le système d'information géographique (SIG). Cette entreprise décide d'établir les marchés sur des lieux avec une population suffisante dans cette région pour garantir leurs ventes.

Dans ce projet, on essaie d'utiliser différentes techniques spatiales pour obtenir des résultats. La zone d'étude est Montréal avec tous les quartiers. Afin de fournir des données appropriées et requises, un prétraitement doit être effectué sur les données source qui seront expliquées dans la section [Prétraitement](#).

C'est supposé ça ; certaines données requises n'existent pas et nous avons des données brutes qui doivent être préparées pour l'application. En d'autres termes, un projet avec un minimum de données sera exécuté. Bien sûr, il est possible d'obtenir un résultat différent lorsque les données réelles sont utilisées mais ce point clé peut être appliqué pour comparer les résultats.

En plus d'utiliser ArcGIS, pour analyser les données et visualiser les résultats, le langage de programmation Python est utilisé pour exécuter les algorithmes.

## 4 Prétraitement

Pour implémenter de chaque projet qui est exécuté dans un système, la sélection de données correctes - en entrée - est importante. Ce point essentiel donnera un résultat avec une erreur minimum ou une fiabilité maximale. Dans ce projet, certaines données sont directement entrées dans le processus de calcul, mais d'autres données doivent être prétraitées.

Dans cette section après la présentation de la source de données (section 4.1), les données requises (section 4.2) et le traitement pour créer les données requises (section 4.3), l'étape primaire pour analyser et vérifier les données sera terminée.

### 4.1 Sources de Données

Les données spatiales requises ont été téléchargées à partir de plusieurs sources ; Données de types de points, de polygones et de lignes avec les spécifications d'utilisation des terres (les détails de chaque type de données sont expliqués dans la section [Données Requises](#))

[Ressources UdeM couvrant Tout le Canada](#) est la plateforme de recherche, de visualisation et de téléchargement des données géospatiales de la collection de la Cartothèque de l'Université de Montréal.

[Communauté Métropolitaine de Montréal](#) c'est une autre source de données spatiales. Ces couches géographiques portent sur différentes thématiques en lien avec les compétences de la CMM et ont été réalisées, pour certaines d'entre elles, afin d'appuyer les outils de planification de la CMM, en particulier le Plan métropolitain d'aménagement et de développement (PMAD).

À l'intérieur de chaque source de données, il y a des fichiers .pdf qui expliquent les attributs et l'abréviation des données. Les données appliquées peuvent être répertoriées comme ci-dessous ;

- /DMTI (UdeM)/CanMap Streetfile/CanMap Streetfiles/QC/streets
  - QCmaf (Quebec Municipal Amalgamations File)
  - QCmun (Quebec Municipality Boundaries)
- /DMTI (UdeM)/CanMap Streetfile/CanMap Streetfiles/QC/Topo
  - QCbfr (Quebec Building Footprints)
  - QClur (Quebec Land Use)
- Dossier d'agglomération de Montréal qui comprend 16 quartiers de Montréal au format 66AAA-US-2016 (les AAA sont les codes de chaque quartier). Selon le fichier Municipal.csv, ces codes se transforment au nom des quartiers (Tableau 1) ;

TABLE 1 – Codes de chaque quartier de Montréal

|       |                         |
|-------|-------------------------|
| 66007 | Montréal-Est            |
| 66023 | Montréal                |
| 66032 | Westmount               |
| 66047 | Montréal-Ouest          |
| 66058 | Côte-Saint-Luc          |
| 66062 | Hampstead               |
| 66072 | Mont-Royal              |
| 66087 | Dorval                  |
| 66092 | L'Île-Dorval            |
| 66097 | Pointe-Claire           |
| 66102 | Kirkland                |
| 66107 | Beaconsfield            |
| 66112 | Baie-D'Urfé             |
| 66117 | Sainte-Anne-de-Bellevue |
| 66127 | Senneville              |
| 66142 | Dollard-Des Ormeaux     |

Les attributs de cette couche de polygone impliquent ;

- util\_sol  
Une des 25 classifications de l'utilisation du sol (vous pouvez voir certaines de ces utilisations des terres dans la section [Annexe 1](#))
- etage\_min, max, moyen  
Nombre étage minimum, maximum ou moyenne arrondi au plus proche
- log\_min, max, moyen  
Nombre de logement minimum, maximum ou moyenne arrondi au plus proche
- terrain  
Valeur du terrain
- terrm2  
Valeur minimum du terrain au mètre<sup>2</sup>
- bat\_min, max, moyen  
Valeur minimum, maximum ou moyenne du bâtiment

En utilisant les outils de "Merge", tous ces fichiers de formes ont été fusionnés ensemble.

## 4.2 Données Requises

En fonction du problème défini, les données suivantes doivent être utilisées ;

**Quartiers de Montréal**

Ce shapefile polygone a été extrait des données de shapefile de "♣ QCmun" données source. La frontière de chaque quartier est déterminée dans ces données. Le nom de cette couche est "♠ Montreal\_poly".

**Image de Montréal avec Tous les Quartiers**

Cette donnée raster (format .jpg) a été créée à partir des données de shapefile de Montréal.

**Les parcelles requises en fonction de leur attribut d'utilisation du sol**

Ce shapefile polygone décrit l'état et les spécifications des constructions avec leur emplacement. Ces données ont été extraites de shapefile de "♣ QCbfr" et "♣ QClur" données source. Les noms de ces couches sont "♠ Montreal\_Residential", "♠ Montreal\_Commercial", "♠ Montreal\_Bureau", "♠ Montreal\_Indutrie" et "♠ Montreal\_Institution".

**Marchés et dépôts Valides à Montréal**

Ces données sont également des couche de shapefile de type point qui ont été extraite des données de shapefile de "♣ QClur" et "♣ QCbfr" données source. L'emplacement de chaque marché et dépôt en fonction de ses coordonnées est indiqué dans ces données. Les noms de ces couches sont "♠ Markets" et "♠ Depot".

## 4.3 Comment Créer Les Données Requises

Pour préparer les données requises, les procédures suivantes ont été effectuées ;

I Parce que toutes ces données appartiennent à la province du Québec, nous devons les découper. Ces procédures ont été effectuées à partir d'un shapefile disponible de Montréal qui montre les limites de la ville de Montréal. Ce fichier a été créé en combinant les quartiers de Montréal (voir le Tableau 1 et la Figure 19 dans la [Annexe 2](#)).

II La limite de la ville de Montréal est un peu plus grande que la ville qu'il faut vérifier. Pour cette raison, des outils "Editor" ont été appliqués et des lignes et des polygones supplémentaires ont été supprimés (la Figure 20 dans la [Annexe 2](#)).

III Dans cette étape, un fichier image (format .jpg) a été pris à la frontière de la ville de Montréal. Ce fichier a été enregistré en résolution 300 dpi.

IV Les parcelles requises en fonction de leur attribut d'utilisation des terres sont extraites. Selon notre objectif, les parcelles seulement avec des attributs résidentiels, commerciaux, institutionnels, industriels et bureau peuvent être utilisées. L'utilisation de ces parcelles est (figure 18) :

- Résident : 100 - 114
- Commerce : 200
- Bureau : 300
- Industrie : 400
- institution : 500 - 520

Ces données ont été extraites en utilisant l'option "**Selection by Attribute**" et pour chacun des différents shapefiles ont été créés.

Les résultats de ces sélections sont représentés dans le tableau 5 dans la [Annexe 1](#).

Afin d'analyser les données, ces shapefiles ont été convertis et combinés les uns aux autres et après cela un fichier au format .csv a été créé pour être utilisé dans le programme python. Le nom de ceci est "**♠ Data.csv**".

- V La couche de marché doit être un shapefile de formes point. Dans cette étape, les données de ce type de point ont été extraites du shapefile polygone de Montréal. Premièrement, en utilisant l'option "**Selection by Attribute**" et en choisissant "Util\_sol = 200", tous les centres de commerce ont été sélectionnés. Après ça ;

"ArcToolBox"

→ Data Management Tools

→ Features

→ Add Geometry Attribute (script)  
et cliquez sur le "Centroid\_inside".

En outre, cette étape a été faite pour tout le sol utile.

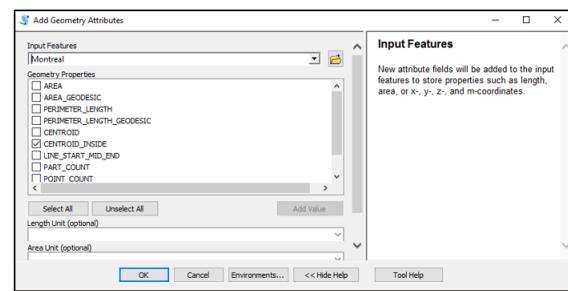


FIGURE 1 – Ajouter des attributs géométriques et créant une coordonnée pour chaque polygone

Nous n'avons pas besoin de tous les points de commerce comme le marché parce que certains d'entre eux sont de petits magasins. Par conséquent, nous devons sélectionner certains d'entre eux comme marchés disponibles. Dans ce cas, certains attributs significatifs ont été sélectionnés dans le shapefile de Montréal. Elles sont ;

**Util\_sol = 200 AND**

**Buildings >= 5000 m<sup>2</sup> AND Buildings <= 10000 m<sup>2</sup> AND**

**Ratio <= 0.35**

Mais dans la couche de Montréal, il n'y a pas de champ pour la superficie de construction (Building) and Ratio (Le rapport du terrain à la superficie totale de la parcelle).

Dans cette situation, quatre champs ont été ajoutées à la couche de Montréal ("Test", "Only\_Terra", "Building" et "Ratio"). En divisant Terrain\_moy (valeur du terrain) par Terrm2\_moy (valeur de chaque m<sup>2</sup>), le champ Only\_terra est créé mais il faut mentionner que certains valeurs de Terrm2\_moy ont une valeur nulle qui crée des erreurs de calcul. Le champ de Test a été créé pour gérer ce problème.

Tout d'abord, la zone commerciale (*Util\_sol = 200*) et *Terrm2\_moy = 0* ont été sélectionnées et en utilisant "Field Calculator" la valeur du champ Test sera égale à 1. Maintenant, nous pouvons calculer les champs "Only\_Terra". Le champ de Building est la soustraction de "Shape\_area" et "Only\_Terra". Enfin, le ratio de "Only\_Terra" et "Shape\_area" (*Only\_Terra/Shape\_area*) est calculé (champ "Ratio").

**140 points** sont extraits pour le marché. Ces points avec le format .csv ont été importés en python. Le nom de ce fichier est "**♠ Commercial.csv**".

VI La couche dépôt a également été sélectionnée comme la couche marchés mais sa spécification est différente. Ces points ont été obtenus sur la base des critères suivants :  
**Util\_sol = 200 AND Bâtiment >= 9000 m<sup>2</sup> AND Bâtiment <= 11000 m<sup>2</sup>.**

22 points sont extraits pour la place de dépôt. Ces points avec le format .csv ont été importés en python. Le nom de ce fichier est " **Depot.csv**".

#### 4.4 Géoréférencement et Numérisation

Cette étape est faite pour unifier le système de coordonnées d'un fichier raster (le fichier jpg) avec d'autres couches. Le nom de l'outil pour ce problème de transformation est "**Georeferencing**".

L'outil de Georeferencing est sélectionné et l'image est entrée sur ArcMap. Maintenant, dans le Georeferencing, nous pouvons voir notre image. En fait, cet outil effectue une transformation entre deux systèmes de coordonnées basés sur au moins 3 points (méthode de transformation "affine"). Par conséquent, les mêmes points doivent être sélectionnés dans deux espaces.

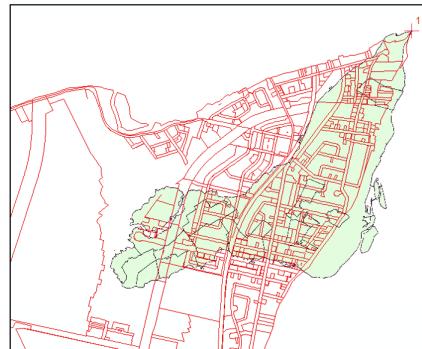


FIGURE 2 – Les étapes de la procédure de Georeferencing

Pour obtenir des résultats plus précis de la transformation, 5 points ont été sélectionnés. Vous pouvez voir les résultats dans la Figure 21 de la Section [Annexe 2](#).

La **Numérisation** d'images signifie la transformation de l'image analogique en image numérique. En d'autres termes, en sélectionnant chaque point sur une image analogique, nous pouvons obtenir sa coordonnée. C'est important pour nous parce que nous devrions avoir des données spatiales (point, polygone ou ligne) et sur ce type de données que nous pouvons traiter.

Il y a un point important. Avant de procéder à la numérisation, une feature classe doit être créée et sur cette classe, nous pouvons la numériser avec l'outil "**Editor**".

Chaque quartier doit être numérisé individuellement. Dans certains cas, quelques polygones doivent être fusionnés entre eux (par exemple, "Côte Saint Luc") ce qui est fait par l'option "Merge" dans l'éditeur. De plus, il est parfois nécessaire de couper un polygone qui se trouve à l'intérieur d'un autre polygone (par exemple, "Westmount" doit être coupé de Montréal) parce que les valeurs de superficie obtenues seront différentes. Ce problème est effectué en utilisant l'option "**Cut Polygon Tool**" dans l'outil Éditeur.

En ouvrant la tableau attributaire de cette couche numérisée, nous pouvons voir les longueurs et superficie de chaque polygone et nous pouvons définir le nom ou l'étiquette de chaque polygone à l'aide d'outils "**Editor**".

Nous avons déjà les frontière des quartiers et il est possible de calculer les différences entre l'image numérisée et le shapefile disponible. La différence de ces shapefiles a été faite en utilisant l'outil "**Erase**". Dans cet outil, deux couches sont sélectionnées et leur différence sera calculée. Le tableau des résultats est présenté dans le tableau 2 et sa figure est représentée à la figure 22 de la Section [Annexe 2](#)

TABLE 2 – Différence entre le polygone disponible des quartiers de Montréal et celui numérisé.

| Quartier                | La Superficie d'image numérisée ( $km^2$ ) | La Superficie de Frontière Disponibles ( $km^2$ ) |
|-------------------------|--|---|
| Baie Durfe              | 6.025                                      | 6.014   |
| Beaconsfield            | 11.061                                     | 11.027  |
| Cote Siant Luc          | 6.998                                      | 6.957   |
| Dollard Des Ormeaux     | 15.147                                     | 15.123  |
| Dorval                  | 21.039                                     | 20.999  |
| Hampstead               | 1.786                                      | 1.790   |
| ILE Dorval              | 0.192                                      | 0.192   |
| Kirkland                | 9.708                                      | 9.690   |
| Mont Royal              | 7.562                                      | 7.574   |
| Montreal                | 362.203                                    | 361.603   |
| Montreal East           | 12.638                                     | 12.592  |
| Montreal Ouest          | 1.408                                      | 1.404   |
| Pointe Claire           | 18.951                                     | 18.931  |
| Sainte Anne de Bellevue | 10.582                                     | 10.565  |
| Senneville              | 7.410                                      | 7.398   |
| Westmount               | 4.036                                      | 4.033   |

La différence entre le shapefile et notre image numérisée est  $2.40\ km^2$ . La raison de ces différences peut être résumée dans :

- (1) la précision de la sélection des mêmes points entre deux systèmes de coordonnées et
- (2) également la précision de dessin des lignes sur le fichier raster.

## 5 Analyse des Données

La procédure d'analyse des données est divisée en trois parties. Premièrement, la méthode de clustering détermine l'emplacement du marché dans toute la ville de Montréal (section [5.1](#)). Après cela, il faut grouper les marchés pour déterminer l'emplacement du dépôt (section [5.2](#))

et enfin pour chaque groupe de marchés, un dépôt sera attribué (section 5.3).

Afin de résoudre le problème, 2 codes python ont été écrits. Le premier est appliqué pour le clustering (il s'appelle "Project") et le second a des solution pour les deux autres sections (il s'appelle "Grouping").

## 5.1 Algorithme de Mise en Grappe (Clustering)

### 5.1.1 Définition de Mise en Grappe

L'analyse de cluster est la tâche de regrouper un ensemble d'objets en tant qu'objets dans le même groupe (appelé un cluster) sont plus semblables les uns aux autres qu'à ceux des autres clusters. C'est une tâche principale de la théorie de l'exploration de données et elle est utilisée dans de nombreux domaines.

L'analyse de cluster elle-même n'est pas un algorithme spécifique, mais ses notions populaires incluent des groupes avec de petites distances entre les membres du cluster.

Dans cette étude, nous utilisons un algorithme "*K-means*" qui est un sous-ensemble de partitionnement et nous utilisons une version modifiée de l'algorithme k-means qui est appelée "*K-medoids*" clustering ([Äyrämö & Kärkkäinen, 2006](#)).

L'algorithme K-means vise à partitionner  $n$  observations en  $k$  clusters dans lesquels chaque observation appartient à un cluster avec la moyenne la plus proche. Par conséquent, la distance entre les points est le paramètre pour résoudre le problème. Afin d'expliquer l'algorithme, les figures suivantes peuvent nous aider (figure 3).

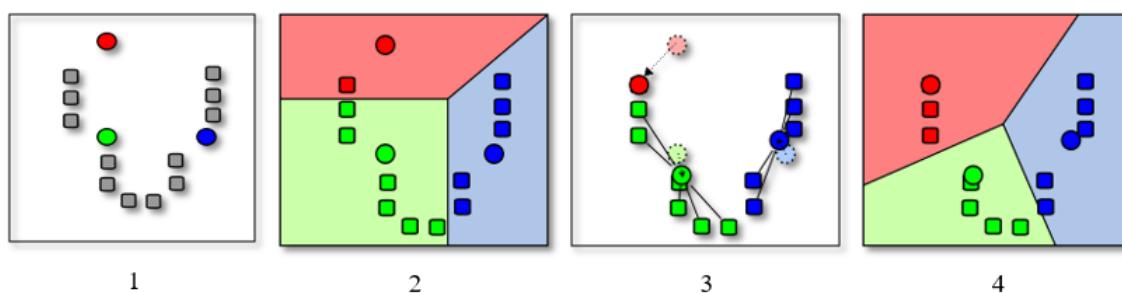


FIGURE 3 – Procédure de mise en grappe ([Weston.pace, 2007](#))

Dans la première partie, trois centres ont été sélectionnés aléatoirement (en couleur). Après cela, les  $K$  clusters sont créés en associant chaque observation à la moyenne la plus proche. Les partitions représentent ici le diagramme de "*Voronoi*" généré par les moyennes. Maintenant, le centre de chaque groupe prend la nouvelle moyenne et leurs centres se déplacent (3<sup>ème</sup> partie). En changeant l'emplacement du centre, le diagramme de voronoi change également (dernière partie) et les étapes 2 et 3 sont répétées jusqu'à ce que la convergence soit atteinte ([MacKay, 2003](#)).

Mais il y a un problème. Quand nous avons les points exacts et que nous voulons les diviser en groupes spécifiques, leur valeur moyenne pourrait ne pas être disponible dans notre point. Par conséquent, nous devons trouver un autre algorithme qui peut soutenir notre idée. Dans ce cas, l'algorithme K-medoids est utilisé. La structure de cet algorithme est la même avec

k-means mais au lieu d'utiliser la valeur moyenne pour déterminer le nouveau centre des clusters, le paramètre médian est appliqué.

### 5.1.2 Mise en Grappe Dans Notre Projet

#### Étape 1

Après avoir importé les fichiers .csv requis dans python ("Data.csv" et "Commercial.csv") (figure 4), tout d'abord le programme dessine un scatterplot de points (figure 5) et nous pouvons voir cette situation dans la figure 23 de section [Annexe 3](#).

```

22 File_path = 'C:\Users\Mohsen\Documents\ArcGIS\GEO 6352\Project\Data.csv'
23 Data = genfromtxt(File_path, delimiter=',')
24 Commercial_path = 'C:\Users\Mohsen\Documents\ArcGIS\GEO 6352\Project\Commercial.csv'
25 Commercial = genfromtxt(Commercial_path, delimiter=',')
26 Size_Data = Data.shape[0]
27 Dim_Data = Data.shape[1]
28 Size_Commercial = Commercial.shape[0]
29 Index = numpy.zeros(shape=(1,Size_Data))
30 old_Index = numpy.zeros(shape=(1,Size_Data))
31 Weight_values = np.unique(Data[:,2])
32 colors = cm.rainbow(np.linspace(0, 1, Weight_values.shape[0]))

```

FIGURE 4 – Importer des données et les préparer pour le calcul

```

35 for i in Weight_values:
36     B = np.where(Data[:,2] == i)
37     plt.scatter(Data[B[:,0], Data[B[:,1], s=0.1, c=colors[v]]
38     v+=1
39     plt.axis('equal')
40     plt.savefig("C:\Users\Mohsen\Documents\ArcGIS\GEO 6352\Project\Image\Cluster.png")
41 plt.show()

```

FIGURE 5 – Scatterplot de données avec une couleur différente

Avec "**plt.savefig**" (la ligne 40 de la figure 5), cette figure est sauvegardée dans chemin du fichier.

#### Étape 2

Le noyau du programme comprend deux boucles "**while**" dont l'une est dans l'autre. La première boucle contrôle à la fois les itérations (cela sera expliqué plus tard) et inclut un avertissement pour choisir le bon nombre de cluster (figure 6). Cela signifie que, quand on détermine le nombre de clusters (variable "**k**" en 51<sup>ème</sup> ligne de la figure 6), si le nombre de clusters est supérieur au nombre de données, le programme affiche le nombre de données et avertit l'utilisateur.

```

47 iteration = 0
48 request2 = 1
49 while iteration < request2:
50     start = timeit.default_timer()
51     k = int(input('Number of Clusters: '))
52     CENTERS = numpy.zeros(shape=(k,3))
53     if max(MostObject[:,1]) < k:
54         print "WARNING: Your number of clusters are more than the points with biggest weighted value."
55         print "Number of points with biggest weighted values = ", max(MostObject[:,1])
56         request1 = raw_input('Is it OK? [Y] or [N] :')
57         if request1 == 'y' or request1 == 'Y':
58             Centers = random.sample(Data, k)
59             print Centers
60         else:
61             continue
62     else:
63         Centers = random.sample(data, k)
64     Centers = np.asarray(Centers)

```

FIGURE 6 – Première boucle du programme

**NOTE :** Parce que la population est plus distribuée dans les régions résidentielles, nous voulons déterminer l'emplacement des marchés à proximité d'eux. En d'autres termes, il est essentiel pour nous que les marchés s'établissent à proximité des régions surpeuplées. Par conséquent, nous déterminons des valeurs pondérées pour chaque type d'utilisation des terres afin de contrôler les calculs et les situations conçues. Ces valeurs de poids sont indiquées dans le tableau 3.

TABLE 3 – Valeur de poids attribuée pour chaque utile sol

|             | Poids |
|-------------|-------|
| Résidentiel | 10    |
| Commercial  | 4     |
| Institution | 3     |
| Industrie   | 2     |
| bureau      | 1     |

La variable "**weight\_values**" (qui a été définie dans la première partie du programme, la ligne 31 de la figure 4) a été créée pour cette raison.

Après cela, en définissant la variable "**Centres**", le centre de chaque cluster est sélectionné, aléatoirement (fonction "**random.sample**") et ils sont placés dans un array (fonction "**np.asarray**").

### Étape 3

Dans la deuxième boucle, les distances minimales entre les centres sont calculées. Afin de calculer la distance entre les points "**np.linalg.norm**" fonction est utilisée (ligne 73 de la figure 7). En fait, cette fonction de norme calcule la seconde norme qui est la distance euclidienne. Après cela, les distances entre les centres de cluster et tous les points sont calculées et cette procédure est répétée jusqu'à ce que la convergence soit atteinte.

```

65 f = 0
66 while True:
67     old_Index = numpy.zeros(shape=(1,Size_Data))
68     old_Index = old_Index + Index
69     for num3 in range(Size_Data):
70         x = Data[num3][:]
71         d = numpy.zeros(shape=(1,k))
72         for num4 in range(k):
73             d[0][num4] = np.linalg.norm(Centers[num4]-x)
74             Index[0][num3] = np.argmin(d.min(0))
75     A = numpy.zeros(shape=(2,Size_Data))
76     for num5 in range(k):
77         A = np.where(Index == num5)
78         Centers[num5][:] = numpy.mean(Data[A[:,1],:],axis=0)
79         dcc = numpy.zeros(shape=(1,len(A[:,1])))
80         for num6 in range(len(A[:,1])):
81             T = A[1][num6]
82             ddc[0][num6] = np.linalg.norm(Centers[num5]-Data[T][:])*(1/Data[T,2]**2)
83             n = np.argmin(ddc.min(0))
84             Centers[num5][:] = Data[A[1][n],:]
85     f+=1
86     if np.all(Index==old_Index):
87         break

```

FIGURE 7 – Deuxième boucle du programme

Dans cette boucle, d'abord, les coordonnées de tous les points importent à la variable "**x**" (ligne 70 de la figure 7) et les distances entre les centres et les autres points sont mises en

"d" variable (ligne 71 de la figure 7). En utilisant une boucle "for", "d" prend les valeurs de distance et l'indice des distances minimales entre les points considérés et tous les centres est détecté (la variable "Index" a été définie à la ligne 29 de la figure 4).

Après cela, les centres de cluster sont mis à jour. Tout d'abord, une matrice pour prendre les sorties de la fonction "np.where" est créée (variable "A" à la ligne 75 de la figure 7). Cette fonction donne l'indice des points qui sont à l'intérieur des mêmes clusters (en fonction des centres de cluster). Par conséquent, la taille de la matrice "A" est modifiée en fonction du nombre de points à l'intérieur de chaque grappe.

Par la suite, un nouveau centre est créé en fonction de la valeur moyenne de chaque groupe. Il est important que ce nouveau centre ne soit pas exactement un membre d'un groupe parce qu'il est obtenu à partir de la valeur moyenne. Jusqu'à présent, nous avons appliqué l'algorithme "*K-means*".

Maintenant, une autre variable est créée pour calculer la distance entre le dernier centre créé (à partir de la valeur moyenne) et d'autres points à l'intérieur de ce groupe (variable "dcc" à la ligne 79 de la figure 7). Cette étape est réalisée en utilisant une autre boucle "for" à l'intérieur de la précédente car pour toutes les clusters, cette procédure doit être effectuée. Enfin, les derniers centres sont obtenus en calculant la distance minimale de l'étape précédente (lignes 82-84 de la figure 7). Les valeurs de poids assignées pour chaque utile sol sont insérées dans cette section du programme ( $1/d^2$ ) et les distances pondérées sont mises à la variable "dcc". Maintenant, par l'extraction de la distance minimum entre eux ("n" variable dans la ligne 83 de la figure 7), les nouveaux centres sont obtenus. Cette partie crée l'algorithme "*K-medoids*".

Cette boucle "While" sera arrêtée lorsque tous les indices au début de la boucle ("old\_Index" variable dans la ligne 67 de la figure 7) sont égaux au dernier indices (variable "Index") en utilisant une fonction "if" (ligne 86 de la figure 7).

⇒ *Jusqu'à présent, nous pouvons trouver les centres de grappes qui sont exactement dans le bâtiment résidentiel (en fonction de leurs valeurs de poids). Cependant, nous ne pouvons pas établir de marchés sur ces points et nous devrions trouver des points commerciaux proches de ces centres déterminés.*

#### Étape 4

Dans cette étape, les emplacements commerciaux les plus proches des centres de cluster sont sélectionnés. Ces emplacements commerciaux doivent être sélectionnés à partir du fichier "**Commercial.csv**" qui a été importé au début du programme (variable "y" à la ligne 91 de la figure 8).

```
88 for num7 in range(k):
89     D = numpy.zeros(shape=(1,Size_Commercial))
90     for num8 in range(Size_Commercial):
91         y = Commercial[num8][:]
92         D[0][num8] = np.linalg.norm(Centers[num7]-y)
93         Commercial_Index = np.argmin(D.min(0))
94     CENTERS[num7][:] = Commercial[Commercial_Index]
```

FIGURE 8 – Trouver des lieux commerciaux à proximité des centres qui ont été sélectionnés à l'étape précédente.

La variable "**D**" montre les sorties du calcul de distance entre les centres et les points commerciaux (ligne 92 de la figure 8). En outre, des indices de distances minimales sont extraits (variable "**Commercial\_Index**" à la ligne 93 de la figure 8) et les nouvelles places de marché dans les édifices commerciaux sont sélectionnées.

## Étape 5

Maintenant, les résultats sont tracés (plot). Chaque cluster a une couleur. Le nombre de couleurs est déterminé en fonction du nombre de grappes (ligne 95 de la figure 9).

```

95 colors = cm.rainbow(np.linspace(0, 1, k))
96 plt.figure(figsize=(10, 10))
97 for i in range(k):
98     B = np.where(Index == i)
99     plt.scatter(Data[B[:, 1], 0], Data[B[:, 1], 1], s=0.1, c=colors[i])
100    plt.scatter(Centers[i, [0]], Centers[i, [1]], s=50, c=colors[i], marker = 's', edgecolors="black")
101    plt.scatter(CENTERS[i, [0]], CENTERS[i, [1]], s=50, c=colors[i], marker = 'o', edgecolors="black")
102    plt.axis('equal')
103 plt.savefig("C:\Users\Mohsen\Documents\ArcGIS\GEO 6352\Project\Image\Cluster" + str(k) + ".png")
104 plt.show()
105 stop = timeit.default_timer()
106 print "Running Time = ", stop - start

```

FIGURE 9 – Les tracés (plot) de résultats

Dans cette plot, il y a deux centres de forme différente ;

- Les points carrés (ligne 100 de la figure 9) sont les centres qui ont été choisis parmi les bâtiments résidentiels
- Les points de cercle (ligne 101) sont les endroits commerciaux les plus proches des bâtiments résidentiels

Ces points de cercle sont les dernières places pour les marchés.

Parce que le programme est capable de fonctionner avec un nombre différent de cluster (cela sera expliqué plus tard), les résultats seront sauvegardés avec le suffixe de  $k$  nombres (ligne 103 de la figure 9).

Afin d'obtenir le temps d'exécution du programme, le temps entre la ligne 50 (variable "**start**", figure 6) et la ligne 105 (variable "**stop**", figure 9) est calculé et à la ligne 106 (figure 9), on peut voir le résultat.

## Étape 6

Les résultats (les points carrés et les points de cercle) sont enregistrés au format .csv (lignes 108-112 de la figure 10).

```

108 f = open("C:\Users\Mohsen\Documents\ArcGIS\GEO 6352\Project\SelectedMarkets" + str(k) + ".csv", 'wb')
109 out = csv.writer(f, delimiter=',', quoting=csv.QUOTE_ALL)
110 out.writerows(Centers)
111 out.writerows(CENTERS)
112 f.close()
113
114 request2 = raw_input('Do you want to test with other number of Clusters; [Y] or [N] :')
115 if request2 == 'y' or request2 == 'Y':
116     request2 = 1
117 else:
118     request2 = 0

```

FIGURE 10 – Sauvegarder les résultats

Lignes 114-118 de la figure 10, poser des questions sur le test de l'autre numéro de cluster. Si l'utilisateur ne veut pas continuer, la première boucle de la boucle (boucle "**while**" extérieure) est terminée et le programme continue.

## Étape 7

Dans la dernière section du programme, les centres sont exportés vers ArcGIS en utilisant la bibliothèque " **arcpy**" (figure 11).

Lorsque nous exportons des données vers ArcGIS, il est important de déterminer le système de coordonnées et la fonction " **arcpy.MakeXYEventLayer\_management**" est appliquée pour cette raison. Parce que nous devons créer un fichier de formes, la fonction " **arcpy.Copy Features management**" est utilisée (la fonction " **arcpy.MakeXYEventLayer management**" ne peut pas créer de fichier de formes).

```
for i in range(5,21,1):
    SelectedMarkets = "C:\\Users\\Mohsen\\Documents\\ArcGIS\\GEO 6352\\Project\\SelectedMarkets"
    SelectedMarkets_Layer = "SelectedMarkets" + str(i) + "Layer"
    SelectedMarkets_shp = "C:\\Users\\Mohsen\\Documents\\ArcGIS\\GEO 6352\\Project\\SelectedMarkets.shp"
    ## Make XY Event Layer and create feature class ##
    arcpy.MakeXYEventLayer_management(SelectedMarkets, "Field1", "Field2", SelectedMarkets_Layer)
    arcpy.CopyFeatures_management(SelectedMarkets_Layer, SelectedMarkets_shp, "", "0", "0")
```

FIGURE 11 – Exportation du résultat des marchés sélectionnés dans ArcGIS

⇒ Dans cette étude, nous avons appliqué 16 clusters de 5 clusters à 20 clusters. Ces résultats sont représentés dans la figure 24 de la section [Annexe 3](#)

## 5.2 Groupement des Marchés afin de Déterminer le Dépôt le Plus Proche

Après avoir analysé et vérifié les différents résultats du clustering, nous avons décidé de sélectionner le cluster avec 12 centres ou marchés car :

- I La grappe avec ce nombre de centres peut couvrir approximativement des régions entières de Montréal et
- II Elle peut être économiquement justifiée

Dans cette étape, le deuxième programme python (son nom est "Grouping") est utilisé et nous essayons de diviser 12 points de marché en 4 groupes avec 3 marchés dans chaque groupe.

## Étape 1

Comme le programme précédent, les données requises ont été importées (fichier "**Depot.csv**" avec 22 points et fichier "**SelectedMarkets12.csv**" avec 12 points) (lignes 16 et 20 de la figure 12).

```
16 File_path = 'C:\\Users\\Mohsen\\Documents\\ArcGIS\\GEO 6352\\Project\\Depot.csv'
17 Depot = genfromtxt(File_path, delimiter=',')
18 Size_Depot = Depot.shape[0]
19
20 SelectedMarkets_path = "C:\\Users\\Mohsen\\Documents\\ArcGIS\\GEO 6352\\Project\\SelectedMarkets12.csv"
21 SelectedMarkets = genfromtxt(SelectedMarkets_path, delimiter=',')
22 SelectedMarkets = SelectedMarkets[12:24][:]
23 SelectedMarkets = SelectedMarkets[np.lexsort((SelectedMarkets[:,1],SelectedMarkets[:,0]))]
24 Size_Markets = SelectedMarkets.shape[0]
25
26 quest1 = int(raw_input('How many points in one group do you need: '))
27 group = Size_Markets/quest1
28 colors = cm.rainbow(np.linspace(0, 1, group))
29 plt.figure(figsize=(10, 10))
30 for i in range(group):
31     plt.scatter(SelectedMarkets[quest1*i:quest1*i+quest1,0], SelectedMarkets[quest1*i:quest1*i+quest1,1])
32 plt.axis('equal')
33 plt.show()
```

FIGURE 12 – Importation des données requises dans la deuxième partie du traitement

La ligne 22 "SelectedMarkets" est définie pour les points de la ligne 12 à 24 parce que dans la section précédente nous avons sauvegardé tous les centres impliquant des centres résidentiels et commerciaux (points carrés et cercles ensemble) et les 12 premières données appartiennent aux centres résidentiels (points carrés).

À la ligne 23 de figure 12, tous les points du marché sont triés (sort) en fonction de la direction y.

À la ligne 26 de figure 12, le programme demande le nombre de points dans chaque groupe et nous choisissons 3 points. En ce moment, ce programme ne fonctionne qu'avec ce nombre de points mais il sera développé pour d'autres situations.

Entre les lignes 28 à 33, on peut voir la distribution des points. Il faut mentionner que la couleur des groupes primaires est sélectionnée en fonction de leur tri.

## Étape 2

Dans cette section, le programme calcule la distance entre chaque point avec d'autres points (variable "Mat" à la ligne 38 de la figure 13).

```
35 Mat = numpy.zeros(shape=(Size_Markets,Size_Markets))
36 for i in range(Size_Markets):
37     for j in range(Size_Markets):
38         Mat[i][j] = np.linalg.norm(SelectedMarkets[i]-SelectedMarkets[j])
39 Combination = int(comb(Size_Markets-1,quest1-1))
40 if quest1 == 2:
41     rounnum = Size_Markets-1
42     colnum = Size_Markets-1
43 else:
44     rounnum = Size_Markets - quest1 + 1
45     colnum = Combination
```

FIGURE 13 – Calcul de la distance entre les points et la valeur de la combinaison

Par conséquent, nous avons une matrice de dimension  $12 * 12$  dont les valeurs diagonales sont nulles.

Pour calculer le nombre de triangles entre les points (parce que chaque groupe a 3 points), on utilise la fonction "comb" (fonction de combinaison) (ligne 39 de la figure 13).

Dans la fonction de condition "if", le nombre de lignes et de colonnes pour les étapes suivantes est calculé.

## Étape 3

L'idée principale dans cette étape est l'organisation des groupes. Lorsque le premier point sélectionne les deux points les plus proches autour de lui, ces trois points sont supprimés de la matrice d'origine. Par conséquent, nous avons 9 points pour le regroupement. Ces points sont à nouveau triés et le nombre de combinaisons de ces points est à nouveau calculé. Maintenant, le premier point de ces 9 points trouve les deux points les plus proches et encore ces trois points sont supprimés. Ces procédures continuent jusqu'à ce que la taille de la matrice restante soit égale à la taille du groupe (figure 14).

```

47 Min_Dist = numpy.zeros(shape=(rownum,1))
48 Groups = numpy.zeros(shape=(Size_Markets,3))
49 S = numpy.zeros(shape=(Size_Markets,3))
50 v=0
51 for i in range(rownum):
52     if Size_Markets == quest1:
53         break
54     else:
55         count = 0
56         Dist = numpy.zeros(shape=(1,int(comb(Size_Markets-1,quest1-1))))
57         Index = numpy.zeros(shape=(int(comb(Size_Markets-1,quest1-1)),3))
58         for j in range(1,Size_Markets-1):
59             for k in range(j+1,Size_Markets):
60                 Dist[0][count] = Mat[0][j] + Mat[0][k] + Mat[j][k]
61                 Index[count][:] = [0,j,k]
62                 count+=1
63         Min_Dist = np.argmin(Dist.min(0))
64         Min_Index = Index[Min_Dist]
65         Min_Index = Min_Index.astype(int)
66         Groups[quest1*i:quest1*i+quest1][:] = SelectedMarkets[Min_Index]
67         SelectedMarkets = np.delete(SelectedMarkets,Min_Index,0)
68         Size_Markets = SelectedMarkets.shape[0]
69         SelectedMarkets = SelectedMarkets[np.lexsort((SelectedMarkets[:,0],SelectedMarkets[:,1]))]
70         Mat = numpy.zeros(shape=(Size_Markets,Size_Markets))
71         for a in range(Size_Markets):
72             for b in range(Size_Markets):
73                 Mat[a][b] = np.linalg.norm(SelectedMarkets[a]-SelectedMarkets[b])

```

FIGURE 14 – Groupement les points en supprimant chaque groupe après le calcul

Dans les lignes 58 - 62 différentes façons de former un triangle et un périmètre de chaque triangle sont calculées.

La distance minimale est trouvée (ligne 63 de la figure 14) et un groupe avec ces points est créé (ligne 66 de la figure 14).

Les points sélectionnés sont sortis de la matrice originale (ligne 67) et les points restants sont triés, encore (ligne 69). La distance entre les points avec un nouvel arrangement est calculée (lignes 70-73).

Maintenant, la combinaison de ces points est calculée (lignes 56 et 57 de la figure 14). Si la taille de la nouvelle matrice est égale à la taille du groupe (3), le programme arrête la boucle et le programme continue (ligne 52).

#### Étape 4

Dans cette étape, des groupes de points sont tracés (plot)(figure 15).

```

75 colors = cm.rainbow(np.linspace(0, 1, group))
76 plt.figure(figsize=(10, 10))
77 for c in range(group):
78     plt.scatter(Groups[quest1*c:quest1*c+quest1,0], Groups[quest1*c:quest1*c+quest1,1],
79     plt.axis('equal')
80 plt.scatter(Depot[:,0], Depot[:,1], s=10, c='black')
81 plt.show()

```

FIGURE 15 – Tracer (plot) les groupes

⇒ Selon ce résultat, nous pouvons aller à la dernière partie du traitement qui est la sélection du lieu de dépôt.

### 5.3 Déterminer l'Emplacement des Dépôts

En fait, cette partie du programme peut être expliquée dans la section précédente, mais parce que nous appliquons et calculons les nouvelles données qui sont "dépôt", elles sont décrites dans une section individuelle.

Cette étape commence avec la détermination d'une variable pour le calcul de la distance entre chaque groupe de marchés et points de dépôt (ligne 84 de la figure 16). Après cela, les

distances sont calculées (lignes 85-87). La dimension de la matrice des résultats est de 12 \* 22 (12 pour le nombre de marchés et 22 pour le nombre de dépôts). Il faut mentionner que la variable "**Groupes**" a les points de marché que chaque trois points consécutifs sont dans le même groupe.

```

83 Size_Groups = Groups.shape[0]
84 Dist_depot = numpy.zeros(shape=(Size_Groups,Size_Depot))
85 for i in range(Size_Groups):
86     for j in range(Size_Depot):
87         Dist_depot[i][j] = np.linalg.norm(Groups[i][:]-Depot[j][:])
88 Dist_depot_Market = numpy.zeros(shape=(group,Size_Depot))
89 Min_Dist_depot = numpy.zeros(shape=(group,1))
90 for i in range(group):
91     for j in range(Size_Depot):
92         Dist_depot_Market[i][j] = sum(Dist_depot[quest1*i:quest1*i+3,j])
93     Min_Dist_depot[i] = np.argmin(Dist_depot_Market[i][:])
94 Min_Dist_depot = Min_Dist_depot.astype(int)
95 Select_Depot = Depot[Min_Dist_depot]
96 colors = cm.rainbow(np.linspace(0, 1, group))
97 plt.figure(figsize=(10, 10))
98 for c in range(group):
99     plt.scatter(Groups[quest1*c:quest1*c+quest1,0], Groups[quest1*c:quest1*c+quest1,1],
100                Select_Depot[:,0,0], Select_Depot[:,0,1], s=50, c='red', marker = 's',
101                plt.axis('equal')
102 plt.show()

```

FIGURE 16 – Détermination de la place du dépôt en fonction de chaque groupe de marchés

Maintenant, la somme de la distance de chaque trois points (un groupe) est calculée (lignes 90-92 de la figure 16) et le minimum d'entre eux est sélectionné (ligne 93). Et finalement ce lieu de dépôt appartiendra à ce groupe.

À la fin de cette section, les points de marché et de dépôt sont tracés (lignes 96-102 de la figure 16).

⇒ *Le résultat final est représenté sur la figure 25 de la section Annexe 3.*

Comme la section précédente, les résultats sont enregistrés au format .csv et exportés comme shapefile dans ArcGIS (figure 17).

```

f = open("C:\\Users\\Mohsen\\Documents\\ArcGIS\\GEO 6352\\Project\\Final_Depot.csv",'wb')
out = csv.writer(f, delimiter=',',quoting=csv.QUOTE_ALL)
out.writerow(Select_Depot[:,0,0:3])
f.close()

Final_Depot = "C:\\\\Users\\\\Mohsen\\\\Documents\\\\ArcGIS\\\\GEO 6352\\\\Project\\\\Final_Depot.csv"
Final_Depot_Layer = "Final_Depot_Layer"
Final_Depot_shp = "C:\\\\Users\\\\Mohsen\\\\Documents\\\\ArcGIS\\\\GEO 6352\\\\Project\\\\Final_Depot.shp"
## Make XY Event Layer and create feature class ##
arcpy.MakeXYEventLayer_management(Final_Depot, "Field1", "Field2", Final_Depot_Layer, "PROJCS[")
arcpy.CopyFeatures_management(Final_Depot_Layer, Final_Depot_shp, "", "0", "0")

```

FIGURE 17 – Exportation du résultat des dépôts sélectionnés dans ArcGIS

Les fonctions utilisées pour exporter les points vers ArcGIS sont identiques à celles de la section précédente.

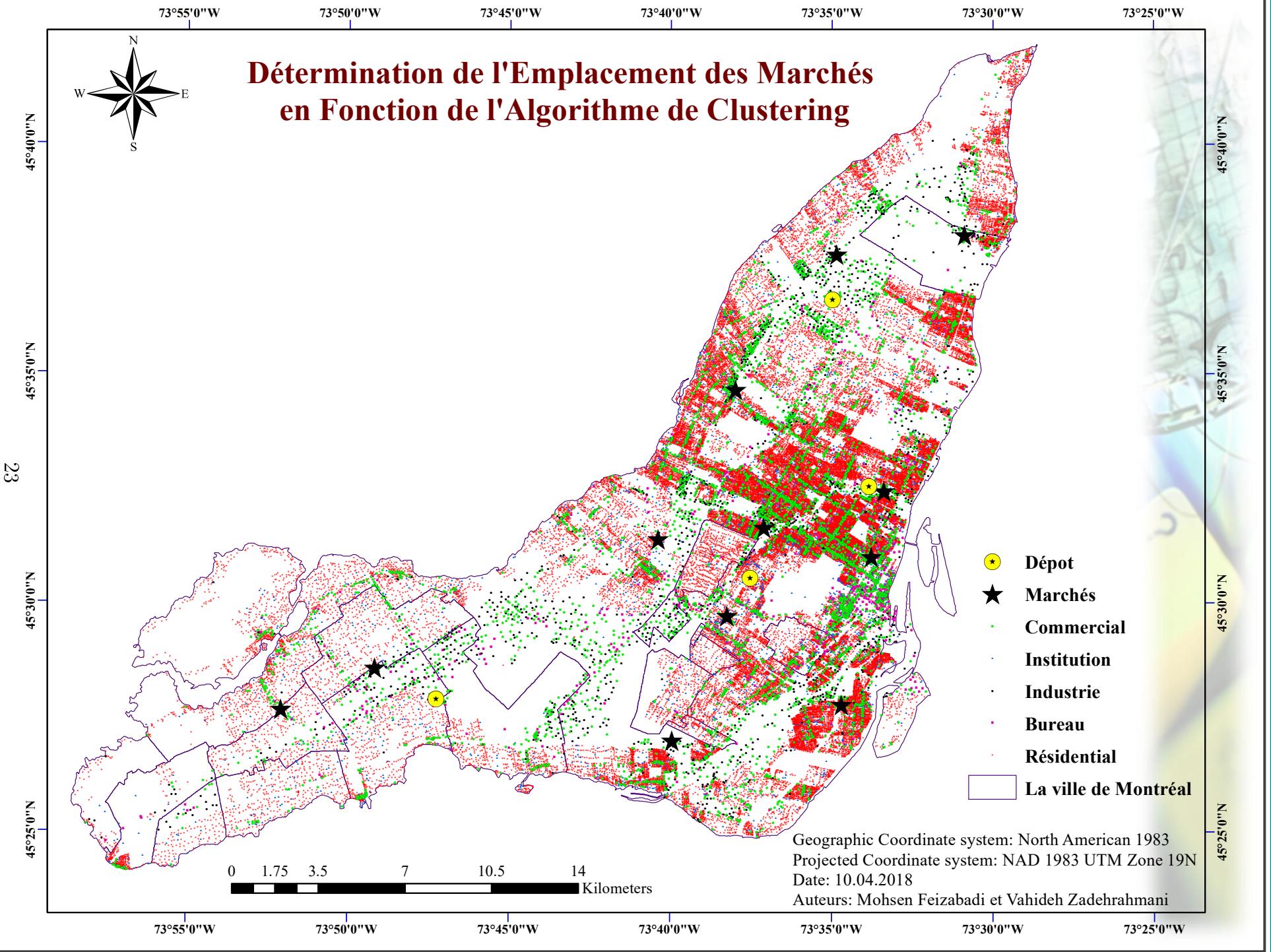
## 6 Résultats

Nous résumons notre travail dans le tableau 4 et la carte de la région avec les lieux mentionnés est représentée à la page suivante. Le tableau montre les coordonnées des marchés sélectionnés ainsi que les dépôts sélectionnés.

TABLE 4 – Coordonnées des marchés et des dépôts sélectionnés

|    | Les Marchés |         | Les Dépôts |          |         |
|----|-------------|---------|------------|----------|---------|
|    | X           | Y       | X          | Y        |         |
| 1  | 294058.6    | 5039485 | 1          | 282324.4 | 5036151 |
| 2  | 291287.4    | 5042591 | 2          | 295002.8 | 5041021 |
| 3  | 276033.6    | 5035741 | 3          | 299802.6 | 5044716 |
| 4  | 279830.1    | 5037401 | 4          | 298325.3 | 5052251 |
| 5  | 298678.7    | 5035889 |            |          |         |
| 6  | 298497.9    | 5054083 |            |          |         |
| 7  | 291834.1    | 5034446 |            |          |         |
| 8  | 299893.8    | 5041862 |            |          |         |
| 9  | 294416.3    | 5048621 |            |          |         |
| 10 | 303657.4    | 5054866 |            |          |         |
| 11 | 300399.5    | 5044518 |            |          |         |
| 12 | 295547.1    | 5043051 |            |          |         |

## Détermination de l'Emplacement des Marchés en Fonction de l'Algorithme de Clustering



## 7 Conclusion

Selon les résultats, il y a des points importants qui sont conclus ;

1. Dans cette étude, l'énoncé du problème a été fait inversement. Cela signifie que le problème a été créé sur la base des données disponibles pour la vérification et l'analyse de diverses fonctions spatiales dans Python et ArcGIS. Par conséquent, l'existence de l'incompatibilité est logique. De plus, comme nos données sont spatiales, leur distribution est très importante et peut affecter les résultats. Par exemple, la sélection de l'emplacement primaire pour les marchés (140 points) et le dépôt (22 points) a été faite afin de créer une condition idéale pour résoudre un problème spatial. Cependant, la distribution de ces points n'est pas parfaite. Par conséquent, il est possible d'observer certains emplacements inappropriés (pour le marché ou le dépôt) dans les résultats.
2. Nous avons appliqué l'algorithme de clustering (*K-medoids* ou *K-mean*) pour résoudre le problème car :
  - Nous avons besoin d'une méthode pour illustrer la répartition de la population à Montréal
  - Nous avons une grosse base de données et l'extraction de données ou la sélection de quelques données à l'intérieur d'eux a besoin d'exécuter un algorithme inclusif.
3. En utilisant les algorithmes de clustering, avoir les mêmes propriétés et caractéristiques des données analysées sont très importants. En d'autres termes, nous devons être sûrs que la sélection de chaque centre pour chaque cluster n'est pas importante car tous les points sont dans la même situation. Cependant, nous avons essayé de contrôler la sélection du centre en utilisant les "valeurs de poids" (nous avons donné ces valeurs). En fait, la valeur de poids est un seul paramètre qui modifie l'homogénéité des données et gère les données en fonction de notre objectif.
4. Selon le nombre de chaque utile sol (tableau 5), on voit que la majorité de nos données sont résidentielles (62601). Maintenant, cette question se pose que, avec ce nombre de données, est-il logique d'appliquer une valeur de poids pour chaque objet parce que, avec une grande probabilité, les centres seront choisis parmi les bâtiments résidentiels ? C'est vrai mais il n'y a aucune garantie pour sélectionner le résidentiel avec les données d'origine. Cependant, cet algorithme avec des valeurs de poids peut être testé pour d'autres utile sol. Par exemple nous pouvons donner les valeurs de poids les plus élevées pour les objets "*Bureau*" (il a le minimum de données, 900) et nous vérifions les résultats qui sont corrects ou non mais certainement dans notre cas les résultats ne peuvent pas être logiques car quelque part que la population n'est pas beaucoup et nous avons "*Bureau*", l'algorithme sélectionne ce "*Bureau*" comme place de marché mais cette sélection n'est pas bonne et économique.
5. Nous avons obtenu le temps d'exécution du programme et cela montre (avec ce nombre de données) l'algorithme n'est pas rapide même pour calculer 20 cluster cela prend 1 heure 40 minutes. Il montre que la méthode de clustering pour les données volumineuses nécessite un ordinateur performant. Cependant, il est important que, lorsque les premiers centres du cluster (qui est sélectionné aléatoirement) soit sélectionné près des derniers centres (qui est sélectionné par itération), la durée du temps diminue.

## 8 Prochaine Étude

Après avoir terminé la deuxième partie du programme python (groupe les points), il est possible de résoudre le même problème avec un nombre différent de groupes de marchés pour établir les dépôts.

En outre, l'une des fonctionnalités les plus puissantes d'ArcGIS est l'analyse de réseau et la définition de la topologie de réseau. Les codes python pour cette analyse ont été développés à travers une version différente d'ArcGIS et cette option peut être ajoutée pour trouver les marchés et les dépôts en fonction de leur temps d'accès.

## 9 Annexe

### 9.1 Annexe 1

#### Description des Données

|             | <b>Code</b> | <b>Description</b>                         |
|-------------|-------------|--|
| 255-255-000 | 100         | <b>Résidentielle</b>                       |
| 255-230-000 | 101         | Résidence de 1 logement                    |
| 255-205-000 | 102 ou 112  | Résidence ou condo de 2 à 4 logements      |
| 255-180-000 | 103 ou 113  | Résidence ou condo de 5 à 24 logements     |
| 255-155-000 | 104 ou 114  | Résidence ou condo de 25 logements et plus |
| 255-000-000 | 200         | <b>Commerciale</b>                         |
| 020-190-170 | 300         | <b>Bureau</b>                              |
| 130-000-130 | 400         | <b>Industrie</b>                           |
|             | 500         | <b>Institutionnelle</b>                    |
| 120-062-255 | 510         | Institution économique                     |
| 150-195-255 | 520         | Institution non-économique                 |

FIGURE 18 – Certaines parties des codes d'utilisation des terres. ([Retour](#))

TABLE 5 – Nombre de données pour chaque type de utile sol ([Retour](#))

|             | Nombre de données |
|-------------|-------------------|
| Résidentiel | 62601             |
| Commercial  | 5049              |
| Institution | 2609              |
| Industrie   | 1574              |
| bureau      | 900               |
| Total       | 72733             |

## 9.2 Annexe 2

### Préparation des Données

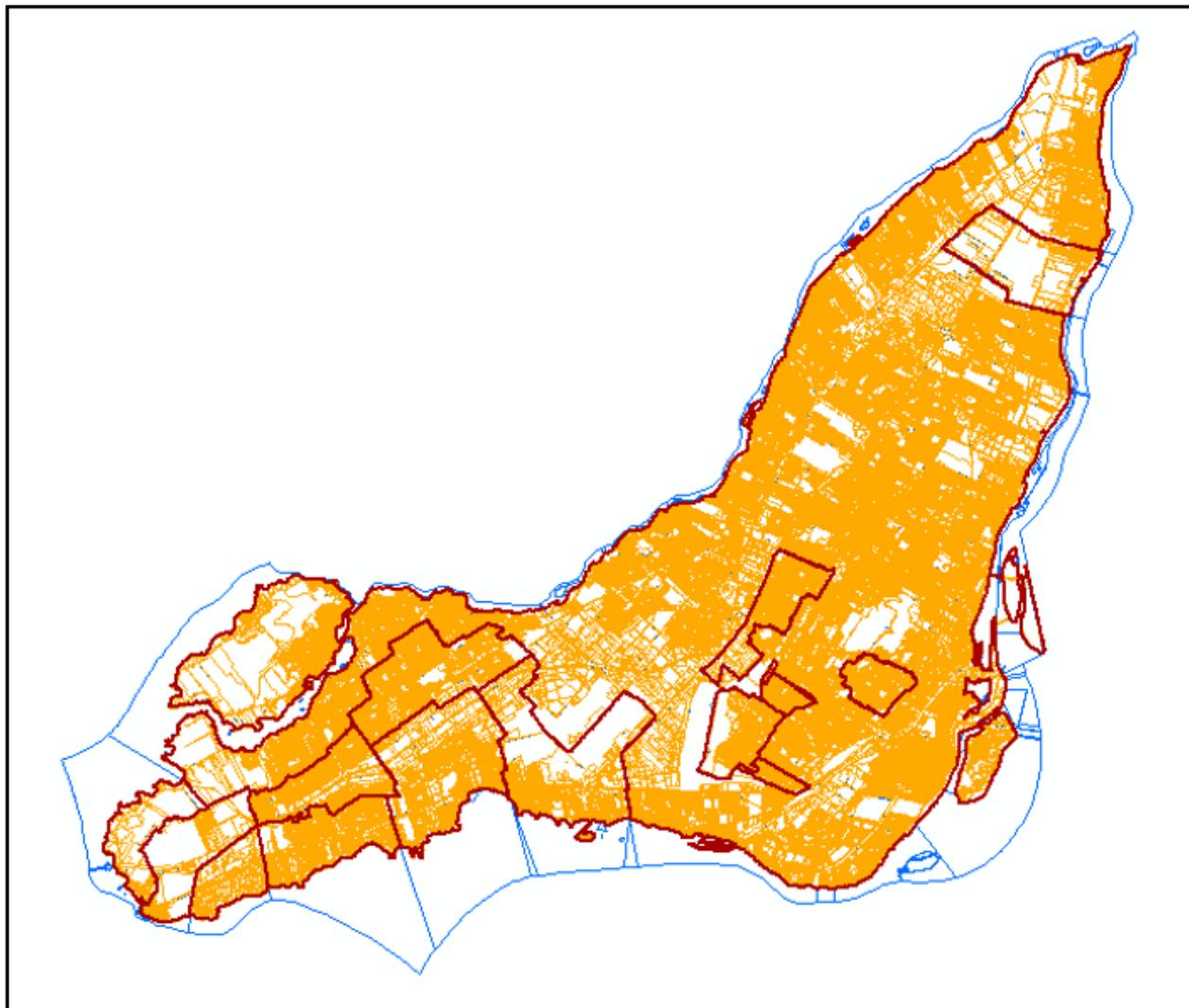


FIGURE 19 – Ville de Montréal (lignes bleues) avec ses quartiers (lignes rouges) ([Retour](#))

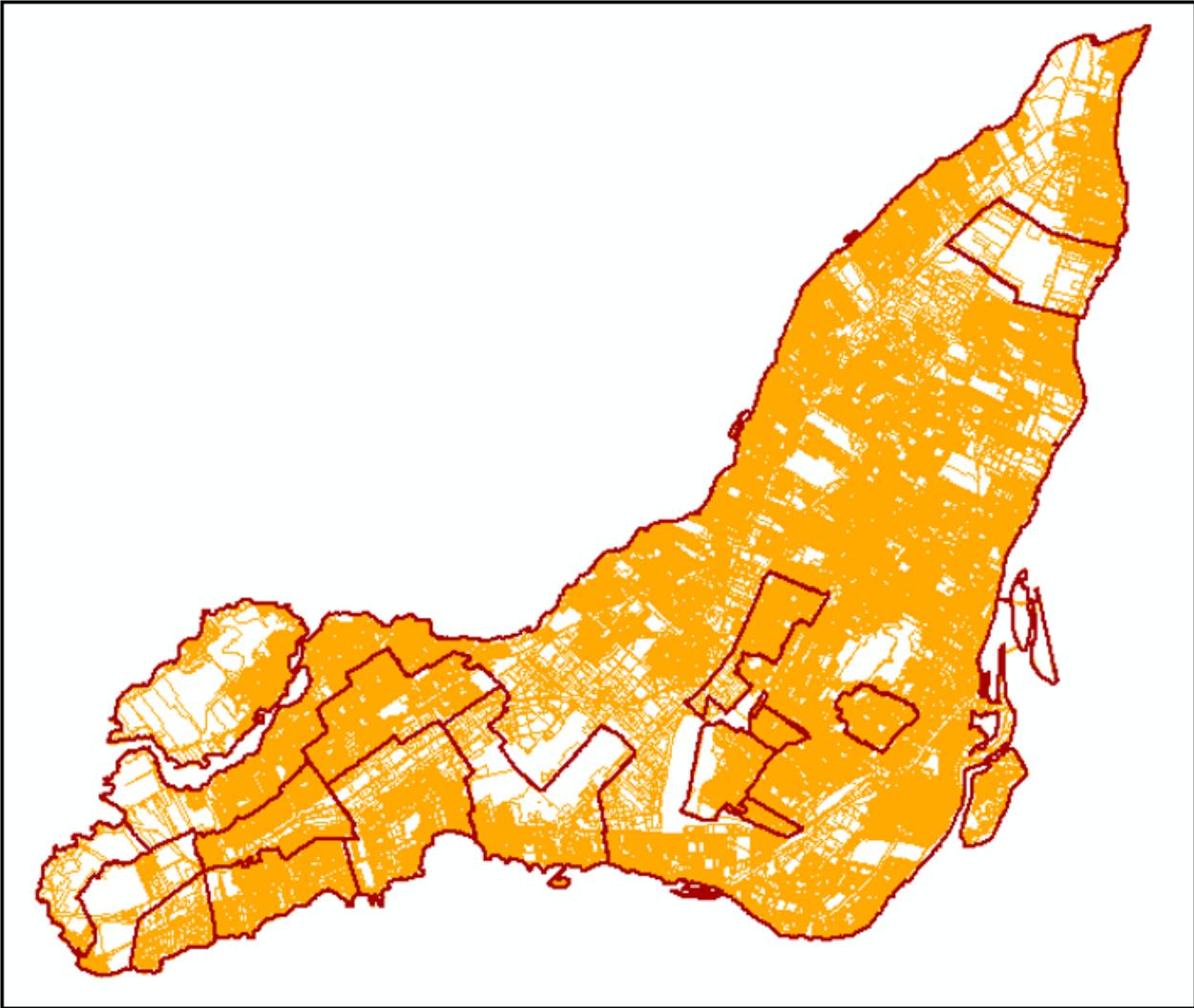


FIGURE 20 – Ville de Montréal après suppression des lignes et des polygones supplémentaires ([Retour](#))

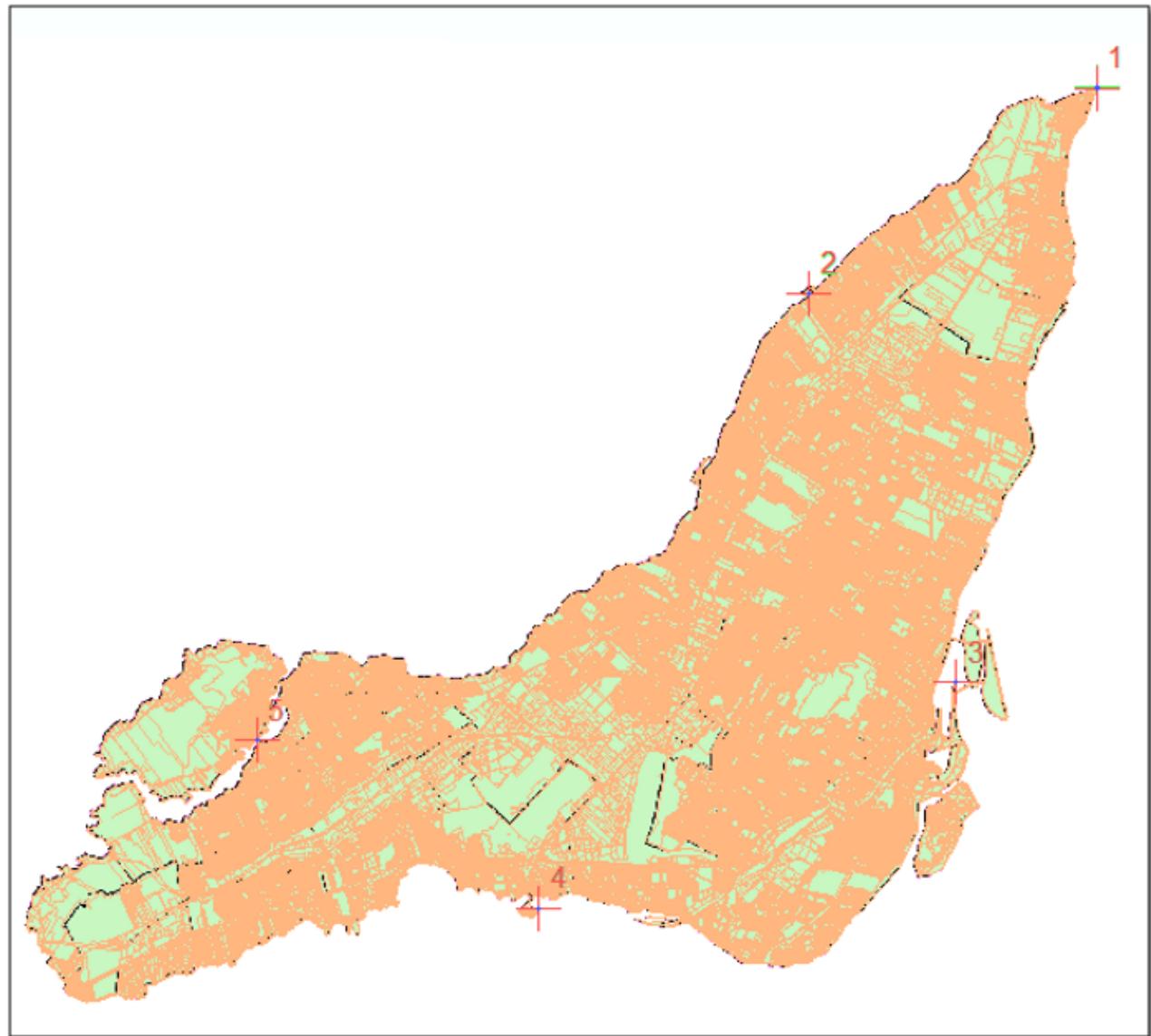


FIGURE 21 – Les résultats de la procédure de Georeferencing avec 5 points ([Retour](#))

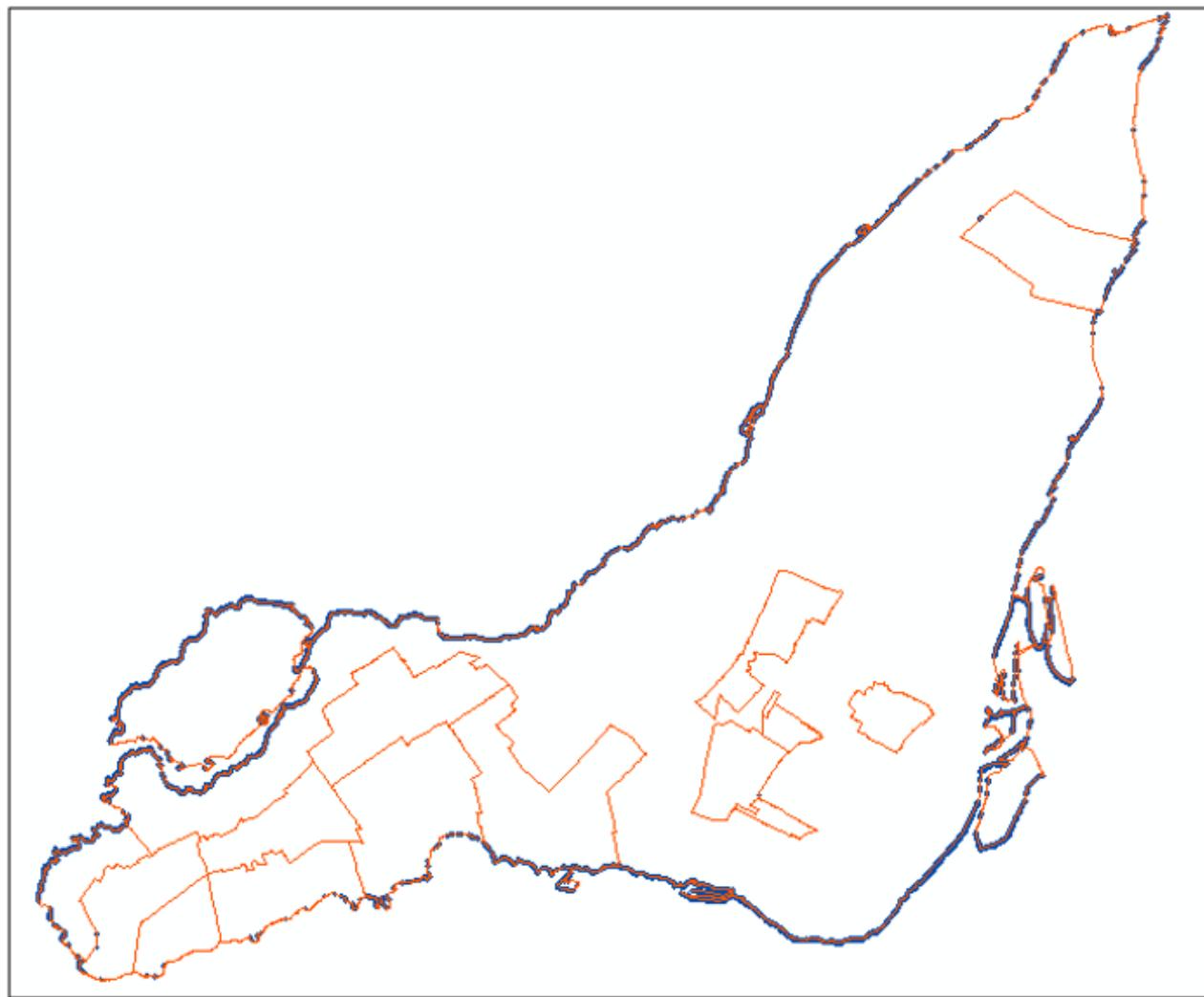


FIGURE 22 – Différence entre le polygone disponible des quartiers de Montréal et celui numérisé (ligne bleue) ([Retour](#))

### 9.3 Annexe 3

#### Analyse de Données

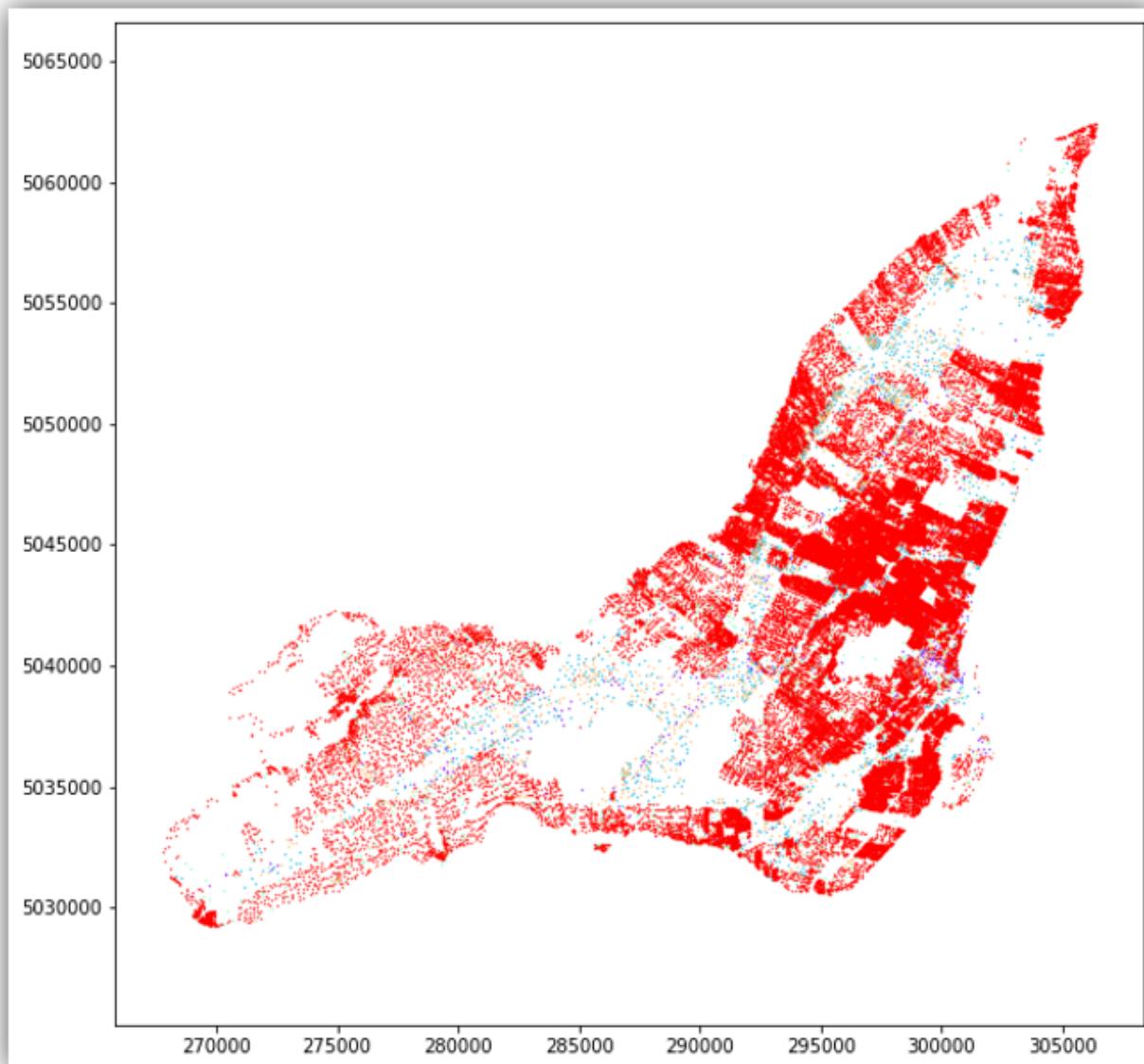


FIGURE 23 – Distribution d'emplacements résidentiels, commerciaux, institutionnels, industriels et de bureaux à Montréal.  
Différentes couleurs montrent différents types de données. ([Retour](#))

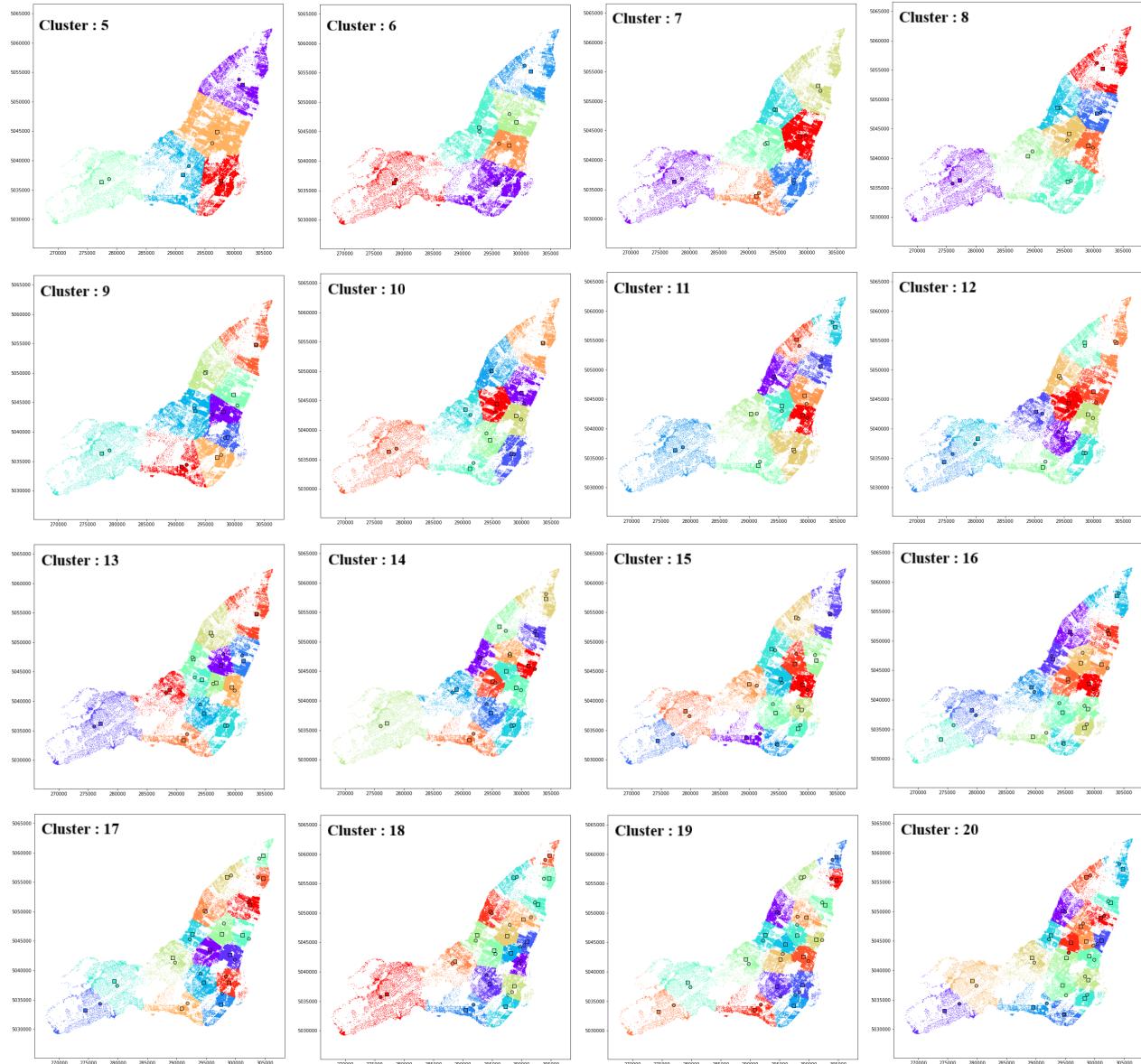


FIGURE 24 – Les résultats de différents mise en grappe de 5 grappes à 20 grappes ([Retour](#))

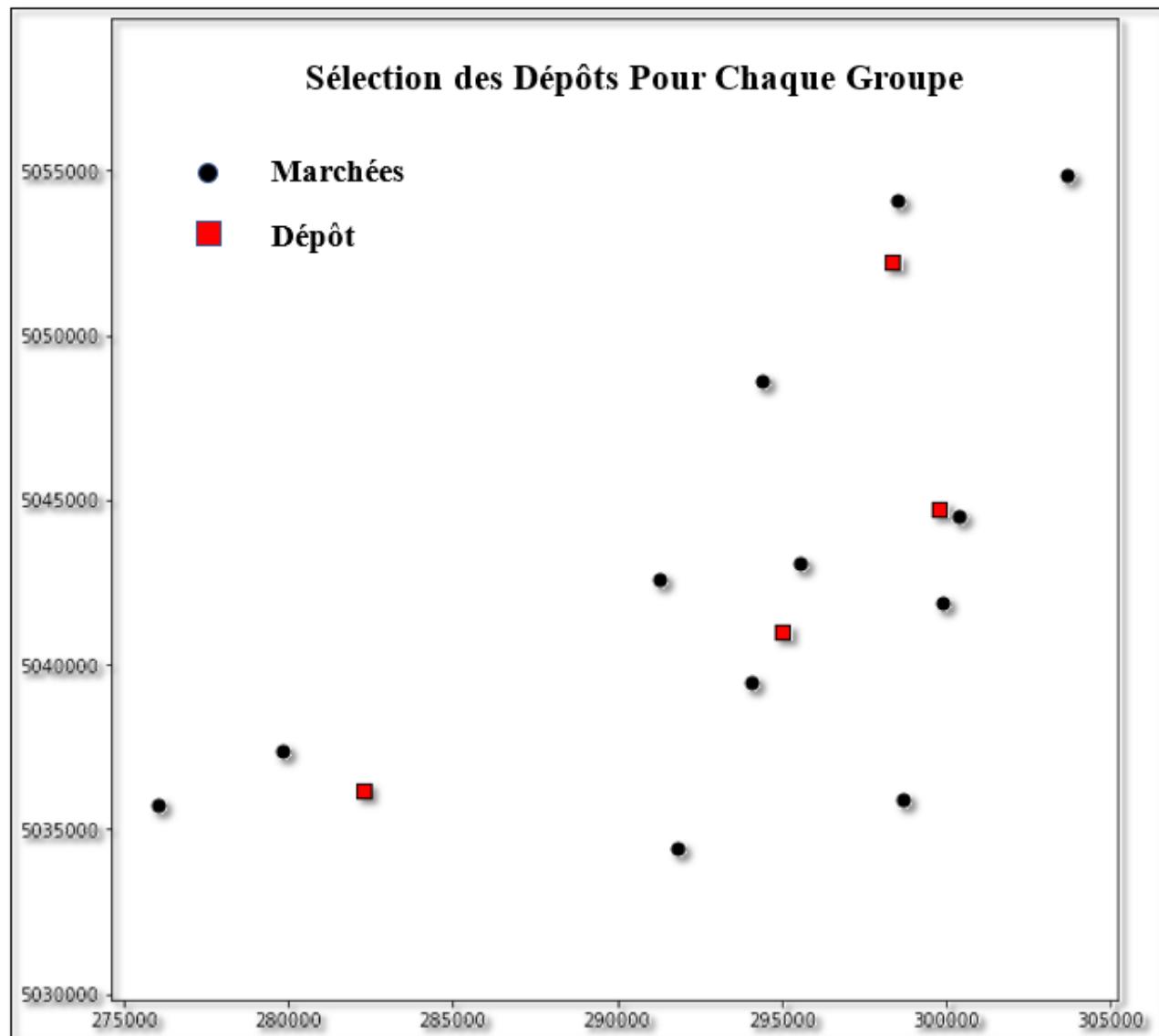


FIGURE 25 – Emplacement des dépôts pour chaque groupe de marchés (3 points) ([Retour](#))

## Références

- Äyrämö, S., & Kärkkäinen, T. (2006). Introduction to partitioning-based clustering methods with a robust example. *Reports of the Department of Mathematical Information Technology. Series C, Software engineering and computational intelligence 1/2006*. 13
- ESRI. (2017). Consulté sur <http://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/modelbuilder/what-is-modelbuilder-.htm> 5
- GIS, E. O. U. (2018). Consulté sur <https://www.geo.university/courses/applied-programming-GIS-analysis-using- arcpy> 2, 5
- MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press. 13
- Weston.pace. (2007). Consulté sur [https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering) 2, 13