Optimisation de livraison par drone

Rapport de projet

Cours de Développement Logiciel – Réalisé par : EL AAMRANI Ahmed –
Encadré par : M. Peletan Loïc

Optimisation de livraison par drone

Sommaire

I. Introduction

Motivations

En sortant de classes préparatoires, mon rapport avec la programmation était un peu flou. Mais comme je souhaite entamer ensuite un double diplôme en « data science », j'étais conscient dès le début de l'importance de maîtriser parfaitement la programmation. Soucieux d'améliorer mes capacités dans cette discipline, j'ai essayé dans un premier temps de travailler dur la matière d'Algorithmique et Programmation. Mais déjà premier coup dur avec un contrôle dans lequel j'ai eu finalement une note pas très encourageante. Ensuite, j'avais le choix de la passer en rattrapage en mi-février, mais au final j'y ai renoncé. Et pour but : passer le temps que je devais lui consacrer pour ce projet de Développement Logiciel. Et je peux dire à ce stade que je ne regrette absolument pas mon choix !

Méthode de travail

En commençant de programmer pendant les petites classes, j'ai essayé dans un premier temps d'utiliser en parallèle des fonctions tests. Mais après quelques séances, j'avais à peine effectué le 1/10 du travail demandé. Je commençais donc à m'inquiéter sur la possibilité de finir le travail à temps. Et ça m'a amené à vouloir progresser à tout prix, sans même vérifier ce que je fais, en cherchant juste à valider l'évaluation automatique. Ainsi après avoir perdu un temps énorme dû à mon manque de pratique, je vous ai donc une fois demandé une aide pour une question et c'est là que vous m'avez réorienté vers la bonne voie. Et dès lors le projet a pris tout son sens.

Intérêt porté au projet

Au début, j'avançais trop lentement. Mon contact avec le projet était très abstrait. Mais une fois que j'ai commencé à réaliser des fonctions test, mon rapport avec ce projet a totalement changé et a pris une dimension beaucoup plus concrète. Je trouvais vraiment du plaisir à avancer, en se rendant compte à chaque fois de mes erreurs et en les corrigeant. Pendant presque une semaine, je ne m'intéressais qu'à ce beau projet d'optimisation. J'avais l'impression de jouer à un jeu vidéo tellement je m'y trouvais. Dans ce « jeu », le but est certes dans un premier temps de collecter le plus d'expérience et d'informations possibles au fil du chemin. Mais, le but ultime est surtout de savourer le moment d'arrivée à la fin et de jouer contre deux grands monstres (ici, les 2 dernières fonctions).

Et ce qui m'a énormément aidé et motivé était la problématique relative à ce projet : une problématique relativement concrète et qui fait même l'objet d'études des grandes entreprises d'aujourd'hui, à l'instar d'Amazon. De plus, le projet était très bien encadré, et je tiens à vous remercier Monsieur de la clarté dont vous avez fait preuve dans la définition des différentes tâches du problème. J'avais tout à ma disposition pour pratiquer, manier Python de façon plus avancée que ce que j'ai eu l'occasion de faire dans le passé et de découvrir notamment les classes de fonctions. En résumé, si j'avais le choix, j'aurais aimé faire 3 ou 4 projets de ce type mais cette fois en plus « avancée ».

II. Contexte

Le 7 décembre 2016, **Amazon** affirme livrer son premier colis par drone. Cette nouvelle marque le début d'une ère où le géant mondial de l'e-commerce souhaite par une **rupture technologique** acquérir un avantage concurrentiel durable, et encore plus marquant que celui qu'il n'a actuellement. En effet d'après l'US Census Government, Amazon s'est accaparé aux Etats-Unis à lui seul **51% de la croissance du e-commerce** au dernier quadrimestre 2015. Sur l'ensemble de l'année, c'est 42% de la croissance captée par Amazon contre 22% en 2014. Autrement dit, Amazon accélère sa croissance et sa prise de parts de marché. Cela s'appelle écraser la concurrence !

Devant ces différents enjeux remarquables et croissants, Amazon s'est donc lancé dans une nouvelle idée qui est celle de la **livraison par drone** et qui pourrait révolutionner le marché de l'e-commerce. Et pour affronter ses besoins croissants, il s'est avéré nécessaire d'optimiser la livraison de n clients par un ou plusieurs drones en réduisant le coût global de la solution.

Ce problème est connu sous le nom de « **problème de tournées de véhicules** ». Ici, nous étudierons une heuristique du problème qui porte le nom de l'algorithme de « **Clarke and Wright** ». Celle-ci, ne fournit pas la solution optimale du problème. Néanmoins en pratique, cet algorithme donne de « bons » résultats c'est-à-dire des résultats qui diffèrent peu de la solution optimale. De plus, cet algorithme a 2 manières de l'implémenter, une séquentielle et une autre parallèle, dont on fera la comparaison.

D'après les dernières recherches dans ce domaine, cette heuristique « Clarke and Wright » pourrait être améliorée par **l'algorithme de Lin-Kernighan**, heuristique pour le « problème de voyageur de commerce », et qui pourrait être appliquée à chacune des tournées individuellement afin d'améliorer le coût global de la solution.

La solution en code telle qu'elle nous est proposée est donc constitué de 3 parties. La 1ère partie porte le nom de « pre_processing » et nous a permis d'implémenter les briques de bases du problème, surtout en termes de classes comme la classe « Drone », « Livraison », ou encore « Solution ». Ensuite, la 2nde partie « post_processing » nous a permis d'afficher la solution sous forme graphique. En s'aidant de ces 2 parties, la dernière section « processing » sert à développer le cœur du problème.

III. Quelques démonstrations mathématiques

Dans cette section, on a choisi de présenter deux démonstrations mathématiques, la première plutôt simple de la fonction *cost_a*(). Quant à la deuxième, elle est plus compliquée. Et on a tâché à présenter les deux démonstrations pour faire le parallèle entre les deux façons de calculer le coût d'un trajet.

1. Démonstration de la fonction *cost_a()*

Dans cette fonction, nous connaissons les caractéristiques du drone et on souhaite calculer le coût pour aller du point a au point b. Or, Dans la fonction qui précède celle − là dans le fichier processing. py, nous avons calculé la puissance requise qui dépend du drone, de la vitesse relative du drone par rapport à l'air et de la masse volumique. Donc, l'énergie est donnée par,

$$Energie = Puissance * \Delta t$$

$$Energie = Puissance * \frac{distance}{drone.speed}$$

Car drone. speed est la norme de la vitesse du drone par rapport au sol.

On vérifiera que cette norme est bien strictement positive

De plus, on peut calculer la distance à travers la position du point − a et du point − b.

Ce qu'il nous manque, dans ce cas là, est la norme de la vitesse du drone par rapport à l'air qui sert comme argument de la fontion puissance.

Or, on a d'après la loi de composition des vitesses :

$$\vec{V}_{drone\backslash sol} = \vec{V}_{drone\backslash air} + \vec{V}_{air\backslash sol}$$

Or, nous connaissons $\vec{V}_{drone\backslash sol}$ *dont la norme est drone. speed et la direction est donné par le vecteur* $\frac{\overrightarrow{AB}}{\|\overrightarrow{AB}\|}$. *De plus nous connaissons* $\vec{V}_{air\backslash sol}$ *dont les coordonnées sont données par wind. x et wind. y.*

Donc on peut déduire les coordonnées du vecteur,

$$\vec{V}_{drone\backslash air} = \vec{V}_{drone\backslash sol} - \vec{V}_{air\backslash sol}$$

Et ensuite, nous pourrions calculer sa norme qu'on pourra injecter ensuite dans la fonction drone_power_consumption.

2. Démonstration de la fonction cost_b

On a d'après la loi de composition des vitesses :

$$\vec{V}_{drone\backslash sol} = \vec{V}_{drone\backslash air} + \vec{V}_{air\backslash sol} \qquad (1)$$

Or notons $\vec{V}_{drone\backslash sol} = x\,\vec{\imath}$; $\vec{V}_{drone\backslash air} = d\,\vec{\jmath}$ *et* $\vec{V}_{air\backslash sol} = \vec{k}$ *avec* $x > 0$ *et* $d > 0$

Avec $\vec{\imath}$ *et* $\vec{\jmath}$ *des vecteurs unitaires, et comme le problème est plan, on peut noter :*

$$\vec{\imath} = \begin{pmatrix} a \\ b \end{pmatrix} \text{ avec } a^2 + b^2 = 1$$

$$\vec{\jmath} = \begin{pmatrix} s \\ t \end{pmatrix} \text{ avec } s^2 + t^2 = 1$$

$$Et\ \vec{k} = \begin{pmatrix} u \\ v \end{pmatrix}$$

$\vec{\imath}$ *est dans ce cas le vecteur déplacement et peut* être *calculée* à *partir des coordonnées des points a et b*

Or dans le cadre du problème pro.cost − b, nous connaissons :

$$d = drone.speed$$

$$u = wind.x$$

$$v = wind.y$$

$$u^2 + v^2 = (wind.x)^2 + (wind.y)^2 = (wind.speed)^2 = w^2$$

$\vec{\imath}$ *est dans ce cas le vecteur déplacement normé et peut* être *calculée* à *partir des coordonnées des points de départ et d'arrivée*

Les inconnues du problème sont donc le vecteur $\vec{\jmath}$ *et la variable* "x" *qui est la norme du vecteur vitesste relative du drone par rapport au sol.*

En projetant (1) *sur les axes du plan, nous avons :*

$$\begin{cases} ax = d * s + u & (2) \\ bx = d * t + v & (3) \end{cases}$$

Or le but est de de supprimer des équations (2)*et* (3) *les inconnues s et t. Ainsi,*

$$\begin{cases} (ax - u)^2 = (d * s)^2 \\ (bx - v)^2 = (d * t)^2 \end{cases}$$

En sommant les deux équations, en tenant compte de $s^2 + t^2 = 1$ *on a,*

$$(ax-u)^2+(bx-v)^2=d^2$$

Or on a aussi $a^2+b^2=1$ *donc,*

$$x^2-2(au+bv)x+w^2-d^2=0$$

On obtient ainsi une équation du second degré dont le discrimant est,

$$\Delta=4(au+bv)^2-4(w^2-d^2)$$

Or, dans les conditions que l'on prend dans la condition pro.cost − b on a,

$$drone.speed > safety-factor * wind.speed \quad avec \quad safety-factor > 1$$

Ainsi $d^2 > w^2$ *donc* $\Delta > 0$

Donc on a,

$$x_1 = au+bv+\frac{1}{2}\sqrt{\Delta} \quad ou \quad x_2 = au+bv-\frac{1}{2}\sqrt{\Delta}$$

Or dans les mêmes conditions cité plus haut, on a $x_2 < 0$ *ce qui est impossible car on avait supposé x une norme* $(donc\ x > 0)$

Donc on obtient la norme de la vitesse relative du drone par rapport au sol à partir de la norme de la vitesse relative par rapport à l'air, la norme de la vitesse du vent, et les points de départ et d'arrivée.

Et donc le coût du trajet peut être calculée par la fonction drone − power − consumption qui calcule la puissance nécessaire au fonctionnement du drone en fonction des caractéristiques du drone, de la vitesse relative du drone par rapport à l'air qui est ici drone.speed et la masse volumique de l'air qui est fixé et donc le cout énergétique est :

*drone_power_consumption(drone, drone.speed, rho=1.3) * distance / x1*

avec distance : la distance entre les 2 points a et b et x1 $(en\ m.s^{-1})$ *qu'on vient de calculer*

$$et\ \frac{distance}{v1}\ est\ le\ temps\ mis\ par\ le\ drone\ durant\ le\ trajet$$

Finalement, nous pourrons ajouter que la fonction cost − b est plus réaliste que la fonction cost − a, car pour des raisons de sécurité l'avion cherche à conserver sa vitesse relative par rapport à l'air, donc c'est plutôt cette dernière qui est fixée à priori.

IV. Etudes de cas

1. Choix du meilleur drone

Problématique :

Un employeur va implanter un nouveau dépôt dans une zone à forte demande. Le secteur de livraison est de forme rectangulaire. Il s'étend sur une longueur de 12km d'est en ouest et sur 7km du nord au sud. Le dépôt se situera au centre de cette zone. Dans une journée typique on s'attend à devoir livrer entre 50 et 60 clients répartis aléatoirement dans cette zone. Chaque client sera typiquement livré d'une quantité de marchandise comprise entre 5 et 60 unités. D'un point de vue météorologique, la zone est sous dominante ouest avec une vitesse de vent typiquement comprise entre 2 m.s-1 et 3,5 m.s-1. Cet employeur doit choisir entre trois modèles de drone dont les caractéristiques sont les suivantes :

- Drone 1 : capacité = 70 unités ; vitesse = 7,7 $m.s^{-1}$; SCx = 0,009 m^2
- Drone 2 : capacité = 200 unités ; vitesse = 10,5 $m.s^{-1}$; SCx = 0,012 m^2
- Drone 3 : capacité = 450 unités ; vitesse = 13,2 $m.s^{-1}$; SCx = 0,018 m^2

Quel modèle de drone doit-on lui recommander ?

Solution :

On commence dans un premier temps par importer tous les modules dont on aura besoin dans l'implémentation du problème. Ensuite, on définit les 3 drones en faisant appel à la classe DRONE et en prenant comme premier argument la capacité du drone. Le deuxième argument correspond à la vitesse du drone en norme et le troisième au coefficient de trainée.

En parallèle, on définit les caractéristiques du vent qui se limitent ici à la vitesse selon les 2 axes du plan car on suppose le problème plan c'est-à-dire qu'on néglige les phases de décollage et d'atterrissage. Or dans notre problème, la zone est sous dominante ouest et on peut donc estimer que le vent avance d'ouest en est, c'est-à-dire vx= *np.random.uniform*(2, 3.5) et vy=0. Cela est dû au fait que la vitesse en norme varie d'après l'énoncé de 2 à 3.5 m/s et que la fonction *np.random.uniform()* renvoie un réel aléatoire dans l'intervalle souhaité, la probabilité de renvoi d'un réel donné est uniforme.

De plus, dans une journée typique, on s'attend à devoir livrer entre 50 et 60 clients répartis aléatoirement dans cette zone. D'où l'utilisation de la fonction *np.random.randint()* car le nombre de client est un entier naturel.

Chaque client sera typiquement livré d'une quantité de marchandise comprise entre 5 et 60 unités. Ce caractère aléatoire de la quantité de marchandise à livrer est tenu en compte dans la fonction *problem.generate_random_clients()* qui génère un nombre de clients aléatoire selon les

conditions imposées à cette fonction. De plus, on prend **dans toute la suite** la fonction *pro.cost_b()* qui modélise mieux ce qui se passe réellement.

Dans cette boucle, on somme les économies réalisées par le drone dans chaque problème indépendant. On somme également le coût du trajet donné par l'algorithme de Clarke and Wright. A la fin de la boucle, on peut calculer la moyenne d'économies qui est le rapport entre le coût total des n problèmes indépendants (donné par cette heuristique) sur le coût d'une approche brutale dans laquelle on effectue des allers-retours entre chaque client et le dépôt (ce coût est égal à la somme du coût total et des économies réalisés)

On pourra aller regarder le script ***Choosing_the_best_drone.py*** (l'utilité de tous les scripts attachés qui suivent résident dans leur exécution). Mais pour afficher les différents résultats, il faudra jouer sur les paramètres drone et version de l'algorithme. Ainsi, on peut modifier à la ligne 17 le drone avec lequel on souhaite travailler. Et on peut modifier à la ligne 27 du code la version avec laquelle on aimerait implémenter le problème ainsi que la ligne d'affichage du résultat qu'il conviendra de modifier en fonction du drone et de la version choisis.

Résultats :

Pour une approche « séquentielle » :

```
Drone1. La moyenne en séquentiel pour 500 problèmes indépendants est 0.36383687699427975

Drone2. La moyenne en séquentiel pour 500 problèmes indépendants est 0.7074128239427805

Drone3. La moyenne en séquentiel pour 500 problèmes indépendants est 0.795708859273979
```

Pour une approche « parallèle » :

```
Drone1. La moyenne en parallèle pour 500 problèmes indépendants est 0.3863111255450713

Drone2. La moyenne en parallèle pour 500 problèmes indépendants est 0.734527425564438

Drone3. La moyenne en parallèle pour 500 problèmes indépendants est 0.8166761712758153
```

Pour le nombre de problèmes indépendants, on a choisi de prendre 500 problèmes indépendants dans ce cas pour garder un temps de calcul raisonnable (de l'ordre de 4-5 minutes pour cette étude). Les résultats nous montrent clairement qu'en terme d'efficacité et de rendement, on a d'abord le drone 3 plus le drone 2, puis le 1er drone. Remarquons le pourcentage très remarquant de rendement du drone 2 qui permet de diviser les coûts de l'entreprise par un facteur 5 ! Cela peut s'expliquer du fait que la capacité des drones augmente lorsque l'on passe du 1er au 2ème puis au 3ème drone, ce qui n'est pas vrai en absolu mais qui sera toujours vrai si on compare 2 drones qui ne diffèrent que par leur capacité de stockage.

Donc si nous devons faire un choix, c'est bien le **drone 3** qu'il faudra choisir.

Comparaison de l'approche séquentielle et l'approche parallèle de l'algorithme de Clarke and Wright :

On voit bien dans un premier temps que le pourcentage d'économie de la version parallèle de l'algorithme Clarke and Wright est nettement supérieure à la version séquentielle. Cette constatation n'est pas contre-intuitive car dans la version séquentielle les itinéraires sont construits successivement alors que dans la version parallèle, plusieurs itinéraires peuvent être construits en même temps. Et donc construire plusieurs itinéraires en même temps permet une certaine « amélioration » de la solution du problème.

Cependant, cette affirmation n'est pas toujours vrai et dépend des caractéristiques du problème qu'on cherche à résoudre. Mais, dans la majorité des cas « classiques » qu'on a pu voir dans la suite de ce projet, la version « parallèle » paraît plus efficace.

2. Partition de la liste des clients

a) Pb250_a

Problème :

Le dépôt d'un employeur possède 2 modèles de drone dont les caractéristiques sont :

- Drone 1 : capacité = 180 unités ; vitesse = 10,3 $m.s^{-1}$; SCx = 0,013 m^2.
- Drone 2 : capacité = 510 unités ; vitesse = 12,5 $m.s^{-1}$; SCx = 0,024 m^2.

Dans cette étude nous considèrerons qu'il n'y a pas de vent. Dans un premier temps, nous utiliserons l'algorithme de Clarke et Wright pour résoudre le problème décrit dans le fichier « pb250_a.csv » en utilisant uniquement le drone 1 puis uniquement le drone 2.

Solution du problème séparé :

Dans cette première partie, nous allons voir les performances de chaque drone à lui seul dans la résolution du pb_250_a.csv. Il s'agit là d'un problème simple, comme on peut le voir sur le fichier « ***partitioning_pb250_a_1.py*** », et dans lequel on a défini les deux instances drones comme suggérées dans l'énoncé, et on a considéré que la vitesse du vent est nulle. On a défini ensuite les instances paramètres et problème, ce qui permet d'appellee directement la fonction *pro.clarke_and_wright()* qu'on affiche ensuite à l'aide *post.plot_problem().*

Pour afficher les résultats, il s'agit de modifier l'instance drone choisi à la ligne 17 du code selon que l'on veut travailler avec le drone 1 ou 2, ainsi que de modifier la version du problème avec version= « sequential » ou « parallel » à la ligne 19 puis d'exécuter le code. On obtient les résultats suivants.

Résultats :

- Drone 1 – Version séquentielle : (Voir Annexe 1)

```
Initialising Clarke & Wright sequential version... done !
Building deliveries... done !
Solution name : Drone1 - Partition. Number of deliveries = 74.
 Total cost = 1.12483e+06, total savings = 2.43056e+06
```

Donc on trouve un pourcentage d'économie de 68,4% et un coût total de 1 124 830 U (on choisit ici de représenter le coût avec une unité arbitraire U).

- Drone 1 – Version parallèle : (Voir Annexe 2)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone1 - Partition. Number of deliveries = 75.
 Total cost = 1.10200e+06, total savings = 2.45339e+06
```

Donc on trouve un pourcentage d'économie de 69,0% et un coût total de 1 102 000 U.

- Drone 2 – Version séquentielle : (Voir Annexe 3)

```
Initialising Clarke & Wright sequential version... done !
Building deliveries... done !
Solution name : Drone2 - Partition. Number of deliveries = 22.
 Total cost = 1.43666e+06, total savings = 8.23053e+06
```

On trouve un pourcentage d'économie de 85,1% et un coût total de 1 436 660 U.

- Drone 2 – Version parallèle : (Voir Annexe 4)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone2 - Partition. Number of deliveries = 23.
 Total cost = 1.14249e+06, total savings = 8.52471e+06
```

On trouve un pourcentage d'économie de 88,2% et un coût total de 1 142 490 U.

On remarque que les pourcentages d'économies sont nettement plus élevés pour le drone 2 que pour le drone 1. Par contre le cout total calculé avec l'algorithme de Clarke and Wright est légèrement plus faible pour le drone 1 que pour le drone 2. Cette incohérence au premier degré n'est en pas une car le coût du trajet dépend des caractéristiques du drone employé. Pour la suite, le pourcentage d'économie n'est donc plus parlant et c'est le coût réel du trajet qui est déterminant.

Problème partitionné :

On cherche à diviser la liste de clients en deux pour voir si on parvient à réduire le coût du problème et faire mieux que ce que réalise le drone 1 (par l'heuristique parallèle).

Dans ce qui suit, on choisira la fonction parallèle qui affiche des résultats meilleurs dans la majorité des cas. Il conviendra de partitionner la liste des clients en 2 sous-listes, comme on le voit sur la carte c'est-à-dire des clients situés dans la zone x>0 et y>0 et les autres qui sont dans la zone x<0 et y<0.

Implémentation :

Le code ici est plus compliqué si on veut afficher le problème avec les 2 solutions. En effet, la fonction *pro.clarke_and_wright()* ne retourne pas une instance solution. D'où, la nécessité d'aller créer les 2 instances livraisons avec la fonction *parallel_build_deliverise()*. Les solutions respectives peuvent à son tour être définis. Et on crée ainsi les 2 instances solutions qu'on met ensuite dans une même liste de solutions pour pouvoir les afficher. Ce code est consultable dans le fichier python ***partitioning_pb250_a_2.py***. Les lignes 29 et 30 permettent d'afficher le résultat en termes de coût, tandis que la suite permet d'afficher les 2 solutions dans un même graphique.

- Si on prend le drone 1 pour les clients situés au nord-est et le drone 2 pour les clients situés au sud-ouest, on obtient : (Voir Annexe 5)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone1 - Partition. Number of deliveries = 44. Total cost = 5.96904e+05,
 total savings = 6.58474e+04
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone2 - Partition. Number of deliveries = 11. Total cost = 6.20932e+05,
 total savings = 7.24422e+06
```

On trouve donc un coût total de 1 217 836 U.

- De la même façon, si on choisit le drone 1 pour la région sud-ouest et le drone 2 pour la région nord-est, on obtient : (Voir Annexe 6)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone1 - Partition. Number of deliveries = 13. Total cost = 5.31703e+05,
 total savings = 1.27034e+06
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone2 - Partition. Number of deliveries = 31. Total cost = 5.04975e+05,
 total savings = 2.38766e+06
```

On trouve donc un coût total de 1 036 678 U et on fait donc mieux que 1 102 000 U c'est-à-dire une économie relative de 6,3% ce qui n'est pas négligeable du tout.

Il conviendra donc de choisir, d'après cette étude, le drone 1 pour servir la région du sud-ouest et le drone 2 pour livrer la région du nord-est.

Analyse de résultats :

On est donc parvenu à réduire le coût total de la livraison en combinant les 2 drones. Cela peut s'expliquer du fait que le coût dépend des caractéristiques des drones qui diffèrent ici légèrement. Et comme le problème est divisé en 2 petits problèmes, chaque drone est plus performant que l'autre dans une région. Ici, le drone 2 a une plus grande capacité que le drone 1, par contre il a un coefficient de traînée plus grand synonyme d'un coup plus important que le drone 1 pour une même livraison. Or dans la région du sud-ouest, la demande de chaque client est en moyenne de l'ordre de 30-40 (le drone 1 ayant une capacité maximale de 180 et le drone 2 de 510), le drone 1 présentant de meilleures caractéristiques aérodynamiques (le coût est proportionnel au coefficient de traînée qui est ici plus faible) permet d'avoir de meilleures économies dans la région du sud-ouest. Cependant, dans la zone du nord-est où la demande moyenne s'élève à 110-120, le drone 1 sera obligé à effectuer majoritairement des aller-retours et son efficacité aérodynamique (moins de frottement) s'avère ici beaucoup plus faible que sa faiblesse logistique ; d'où la nécessité de prendre le drone 2 pour la région du nord-est à cause de meilleure rentabilité.

b) Pb250_b

Problème séparé :

Passons maintenant au problème pb250_b.csv. Si on implémente ce problème de façon classique avec un seul drone comme on l'a fait dans ***partitioning_pb250_a_1.py*** en changeant juste l'instance problème, on obtient les résultats suivants

On trouve pour la solution du problème « parallèle » pour le drone 1 : (Voir Annexe 7)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone1 - Partition. Number of deliveries = 132. Total cost = 1.98689e+06,
 total savings = 1.49692e+06
```

Soit un coût total de 1 986 890 U.

De même, on trouve pour la solution « parallèle » avec le drone 2 : (Voir Annexe 8)

```
Initialising Clarke & Wright parallel version... done !
Building deliveries... done !
Solution name : Drone2 - Partition. Number of deliveries = 42. Total cost = 2.01175e+06,
 total savings = 7.46084e+06
```

Soit un coût total de 2 011 750 U. On voit bien que le coût total est très similaire entre les 2 performances.

Problème partitionné :

La question qui se pose est : Peut-on faire mieux ? D'après la discussion à l'issue de l'optimisation du problème pb250_a.csv, on a vu que le drone 1 est plus performant quand il s'agit de livraisons à basses demandes. L'idée est donc comme on a fait précédemment de partitionner la liste des clients à la base de la demande de chaque client. Dans le problème pb250_b, cela ne revient pas juste à une partition géographique car on n'arrive pas à distinguer deux zones de clients (voir annexe7). On créera donc 2 listes, une avec des livraisons à basses demandes, et l'autre avec des demandes plus élevées. Cependant quelle limite doit-on prendre pour diviser la liste des clients ?

Implémentation :

On fait donc une boucle pour essayer de voir dans quels cas on a une performance optimale. Le but de cette démarche c'est donc d'avoir le coût minimal de la livraison qu'on peut avoir en divisant la liste des clients sur la base de la demande ; ainsi que la limite de partition associé (par exemple si la limite de partition est de 100 kg (unité arbitraire), cela signifie que le drone 1 aura à

livrer toutes les commandes chacune inférieure à 100 kg et le drone 2 se chargera des commandes supérieures à 100 kg). Or une simple analyse du problème pb250_b affirme que la demande des clients varie entre 6 kg et 150 kg. Donc on commence par créer une boucle dont l'indice « i » porte sur la limite de partition. Ensuite, on crée les 2 listes de clients du problème partitionné. Et au lieu juste d'appeler les fonctions *pro.clarke_and_wright()* pour les 2 problèmes séparés, on a donc créé nous-même les différentes fonctions intermédiaires pour pouvoir calculer la somme du coût du trajet associé au 1er drone et celle associé au 2nd drone. Et ainsi, à chaque parcours de la boucle, on affiche la limite de partition et le meilleur coût réalisé.

Ce code pourra être consulté dans le fichier ***partitioning_pb250_b.py***

Voici les 10 dernières lignes sur les 145 lignes du résultat affiché !

```
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
5 2011748.72695
150 1986892.78814

Process finished with exit code 0
```

Cohérence avec le comportement asymptotique :

Quand on prend la limite minimale, c'est comme si on affectait tous les clients au drone 2 et on a donc un coût de 2 011 750 U. Par contre, si on prend la limite maximale, cela revient à n'utiliser que le drone 1 et on a ainsi le même coût que celui qu'on a déjà cité c'est-à-dire 1 986 890 U.

Analyse de résultats :

Ce qui est frappant dans ce résultat, c'est quand on partitionne la liste des clients en 2, on ne fait JAMAIS mieux qu'avec un seul drone avec cette heuristique puisque quelque soit la limite de partition, le meilleur coût dans les 144 premières possibilités est celui associé au trajet effectué simplement par le drone 2. Et ce n'est qu'à la dernière étape de la boucle qu'on peut faire mieux. Cette dernière étape correspond à une mission dans laquelle seul le drone 1 est en charge. Cela peut s'expliquer par le fait que dans ce problème, les clients sont dispatchés presque de façon aléatoire et ne permettent pas une partition meilleure du problème dans laquelle on pourrait regrouper les clients par leur demande comme on a pu le faire dans pb250_a. Et donc si on divise le problème en 2, l'efficacité de la solution sera plus faible car les trajets effectués en moyenne sont plus longs.

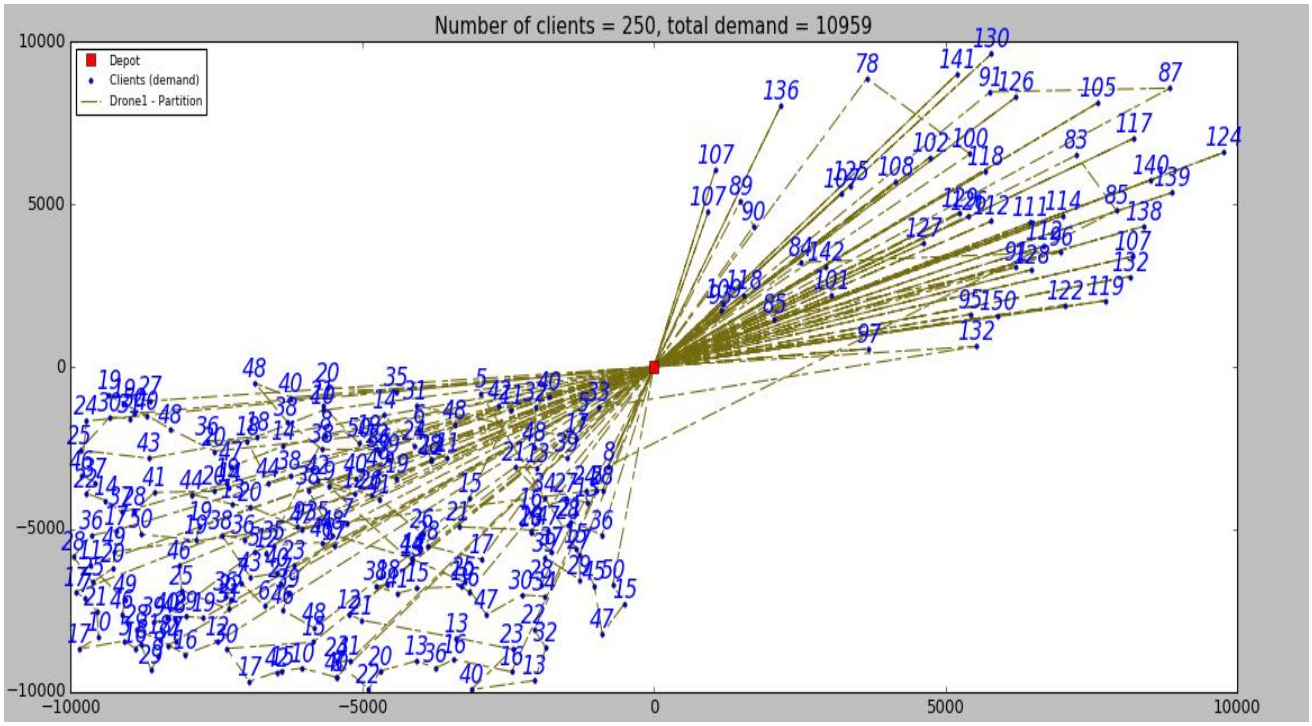
V. Conclusion

Dans cette étude, nous avons donc étudié les caractéristiques de quelques problèmes pas très compliqués avec un nombre de clients qui ne dépasse pas 250 clients, et on a vu que le temps mis pour le calcul n'est pas du tout négligeable. Il convient donc déjà dans un premier temps d'optimiser sa propre implémentation, en condensant les lignes de code et en réduisant le temps de calcul. Ensuite, il faudra chercher à améliorer cette heuristique qui comme on l'a dit au début de notre rapport n'est pas la solution optimale du problème. Et puis dans un dernier temps, il s'agira de traiter d'autres problématiques non abordées dans cette heuristique comme la contrainte de temps, la gestion d'autonomie, la gestion de plusieurs dépôts, la gestion des no-fly zones …

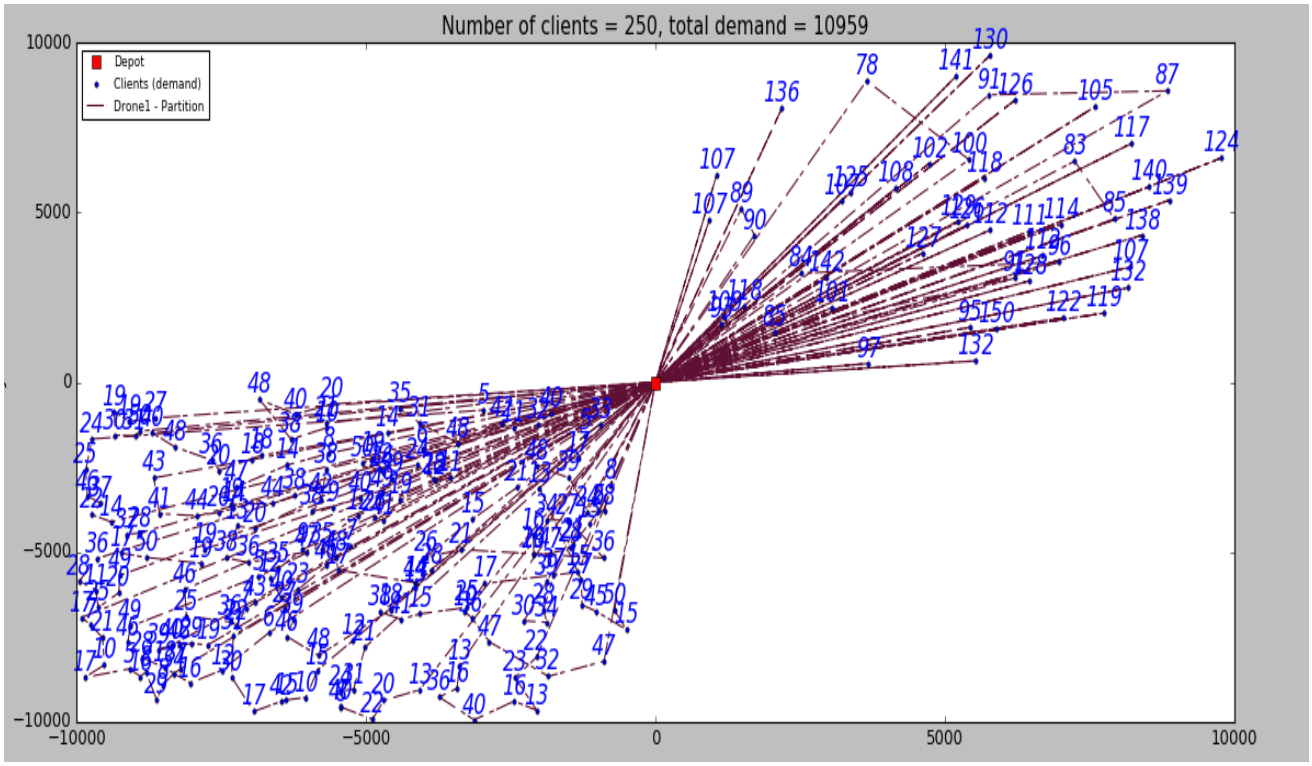
VI. Annexe

Partition de la liste des clients Pb250_a – Problème séparé

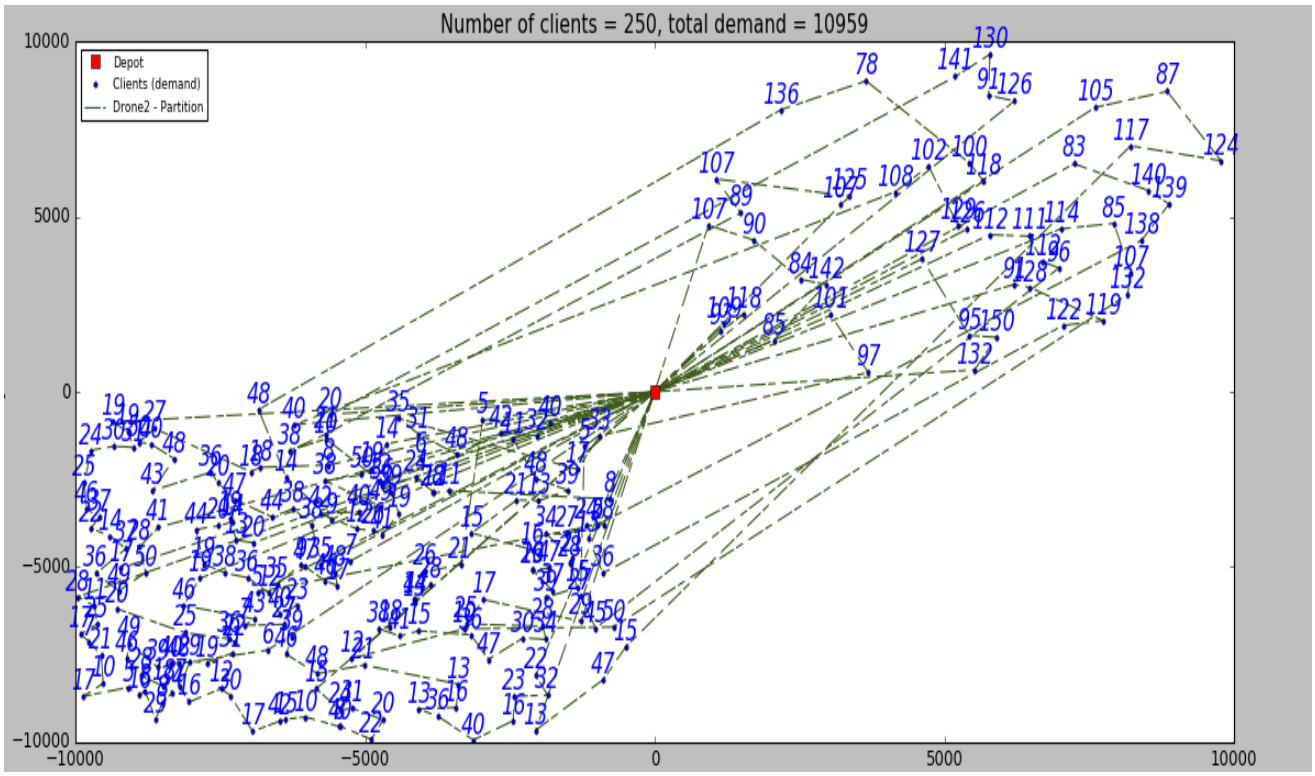
Annexe 1 : Drone1 – Version séquentielle :



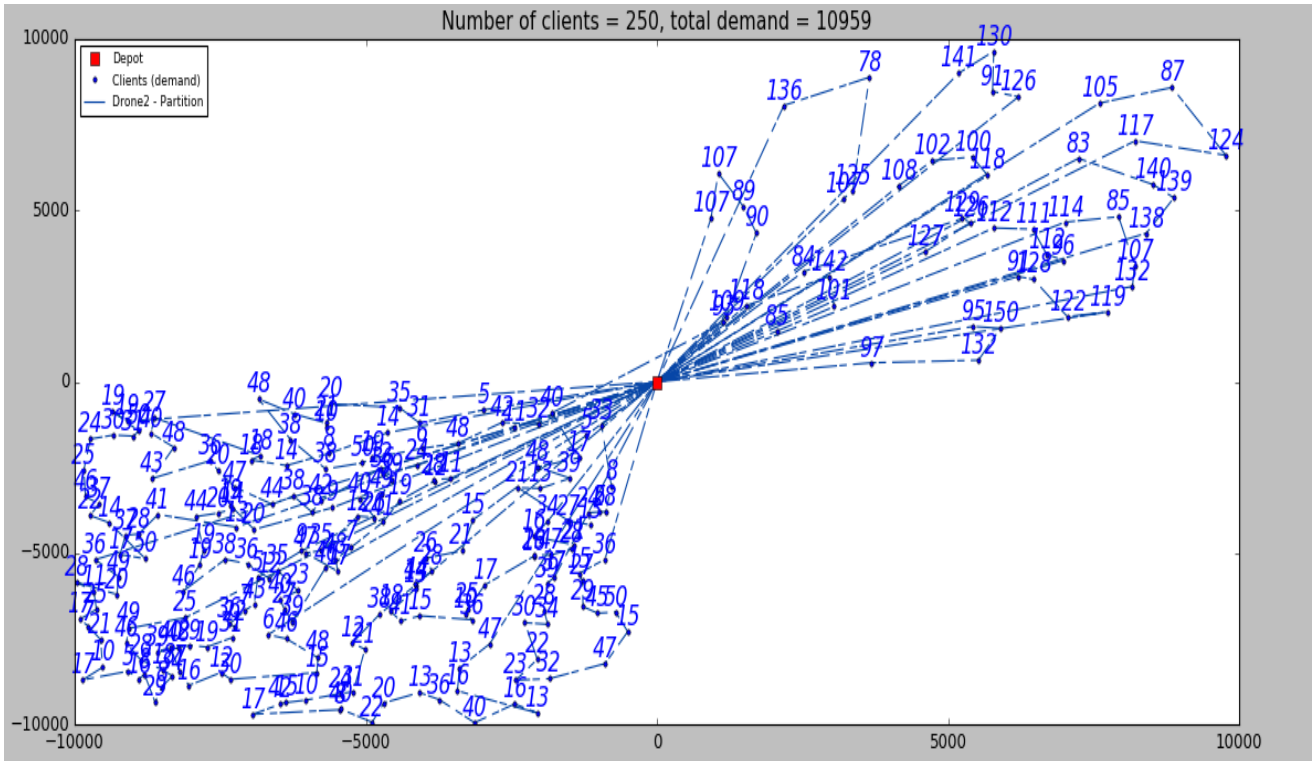
Annexe 2 : Drone 1 – Version parallèle :



Annexe 3 : Drone 2 – Version séquentielle :

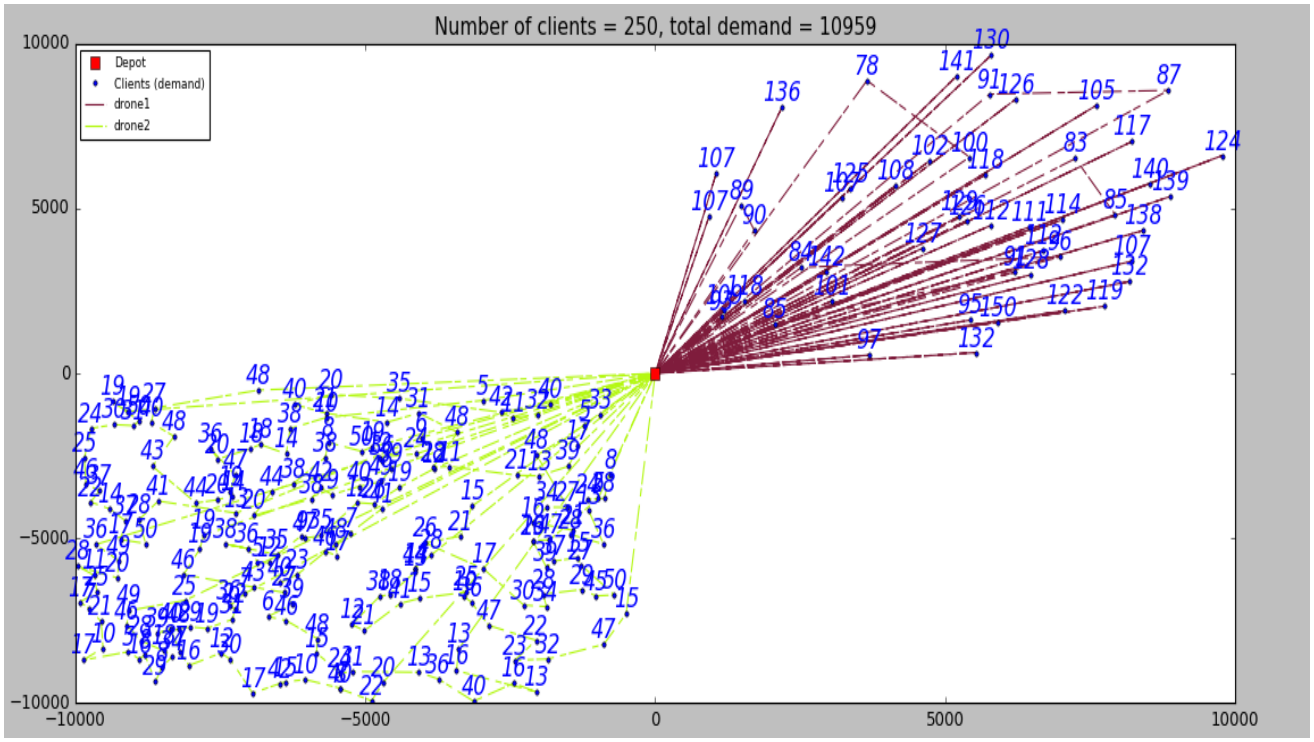


Annexe 4 : Drone 2 – Version parallèle :

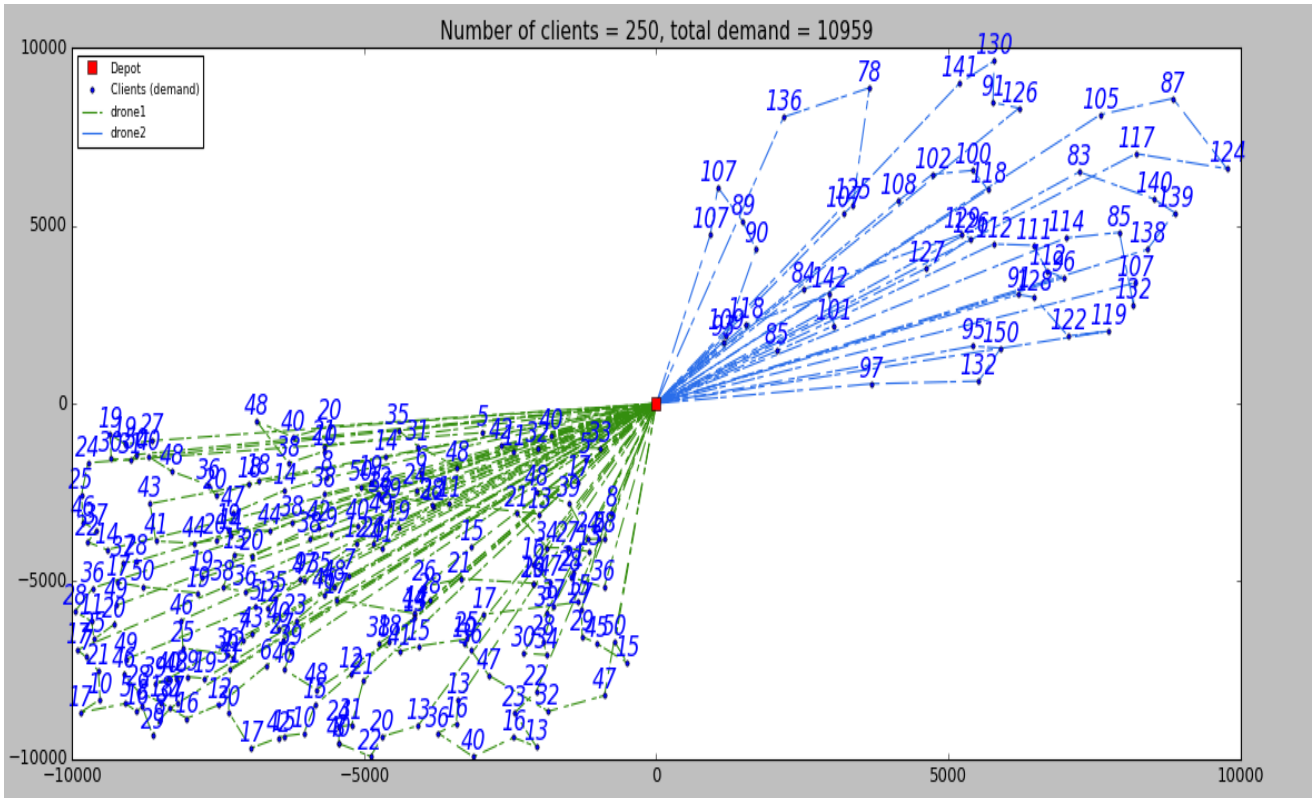


Partition de la liste des clients Pb250_a – Problème mixte

Annexe 5 : Drone 1 au nord-est et drone 2 au sud-ouest

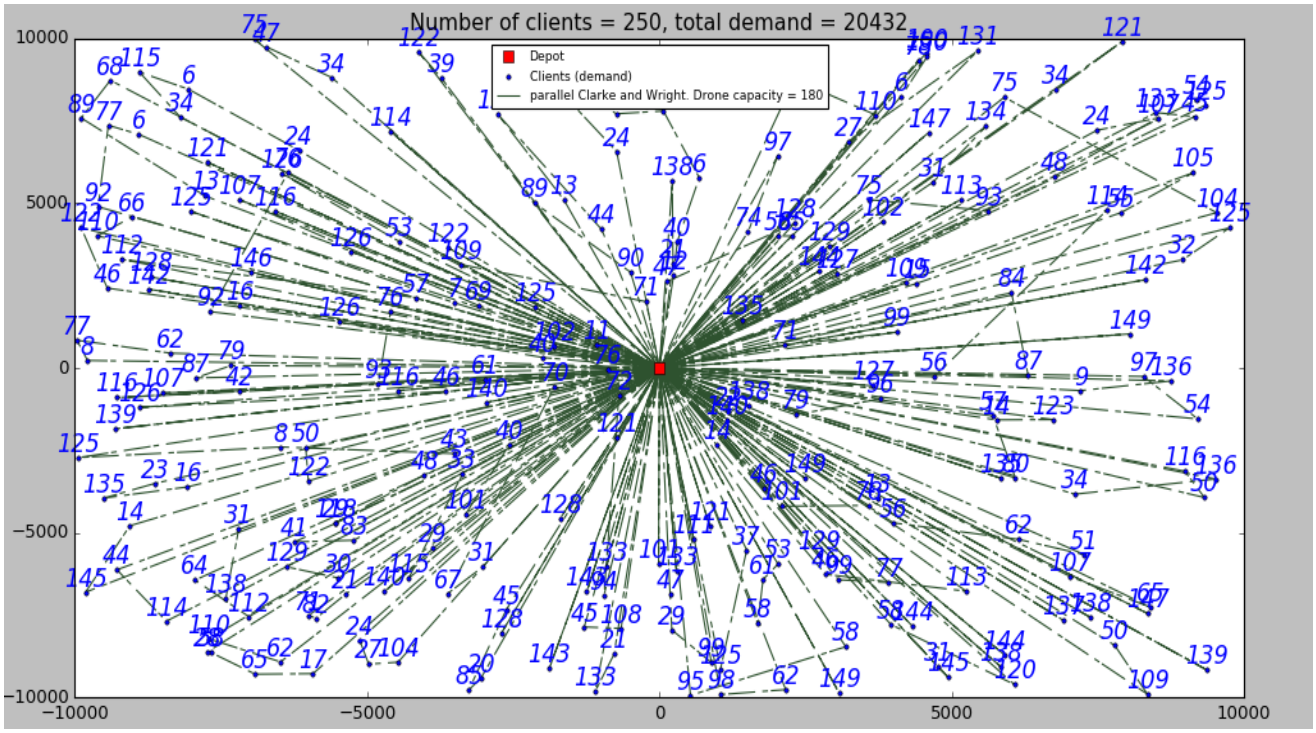


Annexe 6 : Drone 1 au sud-ouest et drone 2 au nord-est



Partition de la liste des clients Pb250_b – Problème séparé

Annexe 7 : Drone 1 – Version parallèle :



Annexe 8 : Drone 2 – Version parallèle :