

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

## Projet Recherche & Développement

2019-2020

# Mutualisation des livraisons des producteurs

**POLYTECH<sup>®</sup>**  
TOURS

**Tuteur académique**  
Azeddine CHEREF

**Étudiant**  
Paul RETAIL (DI5)

12 avril 2020



# Liste des intervenants

Nom	Email	Qualité
Paul RETAIL	<a href="mailto:paul.retail@etu.univ-tours.fr">paul.retail@etu.univ-tours.fr</a>	Étudiant DI5
Azeddine CHEREF	<a href="mailto:azeddine.cheref@univ-tours.fr">azeddine.cheref@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Paul Retail susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Azeddine Cheref susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Paul Retail, *Mutualisation des livraisons des producteurs*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2019-2020.

```
@mastersthesis{
  author={Retail, Paul},
  title={Mutualisation des livraisons des producteurs},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2019-2020}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Introduction générale du projet	1
1 Acteurs, enjeux et contexte .....	1
2 Objectifs .....	1
3 Hypothèses .....	2
4 Bases méthodologiques .....	2
<b>1 Description générale</b>	<b>3</b>
1 Environnement du projet .....	3
2 Caractéristiques des utilisateurs .....	3
3 Fonctionnalités du système .....	3
4 Structure générale du système .....	4
<b>2 Etat de l'art</b>	<b>6</b>
1 Littérature scientifique.....	6
1.1 Le problème de tournées de véhicules et ses variantes.....	7
1.2 Le problème de tournées de véhicules collaboratifs .....	7
1.2.1 Planning centralisé .....	8
1.2.2 Planning décentralisé .....	8

1.2.3	Méthode de partage des bénéfices.....	9
1.3	Etat de l'art sur notre problème .....	9
2	Cas en entreprise .....	9
2.1	TourSolver .....	10
2.2	PTV Route Optimizer .....	10
<b>3</b>	<b>Analyse et conception</b> .....	<b>11</b>
1	Modèle mono producteur.....	11
2	Modèle mutualisation producteur .....	13
3	Heuristique .....	15
<b>4</b>	<b>Mise en oeuvre</b> .....	<b>18</b>
1	Présentation du programme.....	18
2	Zoom sur le fichier heuristique.....	21
3	Tests et résultats .....	22
4	Amélioration.....	24
	<b>Conclusion</b> .....	<b>26</b>
5	Semestre 1 .....	26
6	Semestre 2.....	26
7	Ajouts et améliorations.....	26
	<b>Annexes</b> .....	<b>27</b>
<b>A</b>	<b>Specifications fonctionnelles</b> .....	<b>28</b>
1	Définition de la fonction de passage .....	28
2	Définition de la fonction génération de données.....	28
3	Définition de la fonction de recherche de résultat sans coopération .....	28
4	Définition de la fonction de clustering .....	29
5	Définition de la fonction de résolution .....	29
<b>B</b>	<b>Specifications non fonctionnelles</b> .....	<b>30</b>
1	Contraintes de développement et conception .....	30
2	Contraintes de fonctionnement et d'exploitation .....	30
2.1	Performances.....	30
2.2	Capacités.....	30
2.3	Contrôlabilité.....	30
2.4	Sécurité .....	30
3	Maintenance et évolution du système.....	31
<b>C</b>	<b>Gestion de projet</b> .....	<b>32</b>

D	Compte rendu des réunions	34
E	Bibliographie	37

# Table des figures

## 1 Description générale

1	diagramme des cas d'utilisations .....	4
2	diagramme d'activité .....	4
3	diagramme des classes.....	5
4	diagramme des classes final .....	5

## 3 Analyse et conception

1	tableau récapitulatif des heuristiques.....	16
2	algorithme de l'heuristique .....	17

## 4 Mise en oeuvre

1	architecture du programme .....	18
2	affichage des clusters.....	19
3	Une solution.....	20
4	Une solution après mutualisation .....	20
5	Tableau des résultats .....	24

## C Gestion de projet

1	Trello de ce projet .....	32
2	Gantt de ce projet .....	33

# Introduction générale du projet

L'objectif de ce document est de définir de manière technique les spécifications du projet de mutualisation des livraisons de producteurs. Il va présenter le contexte de réalisation de ce projet ainsi que ses différentes caractéristiques.

Ce document sera écrit par moi-même, Paul RETAIL élève de DI5 à Polytech Tours, qui réalisera ce projet en tant que MOE. Le projet sera encadré par Azeddine CHEREF qui représente la MOA. Il pourra être amené à relire ce cahier des spécifications.

## 1 Acteurs, enjeux et contexte

Le métier de producteurs agricoles nécessite une livraison pour distribuer leurs produits. Celle-ci est faite personnellement par chacun des producteurs en respectant une liste de contraintes. Parmi ces contraintes, on peut citer une contrainte de temps, qui oblige les livraisons à se faire dans une certaine fenêtre de temps. Il existe également une contrainte de poids limite. Les producteurs utilisent leur moyen de transport personnel pour faire les livraisons et chaque tournée doit être optimisée en fonction du poids de chaque livraison. Il existe une contrainte de détour maximum qui limite la distance parcourue par les producteurs par rapport à leur distance de livraison "habituelle". Enfin ces producteurs doivent retourner à leur point de départ en fin de livraison.

Les livraisons sont coûteuses et diminuent la marge sur la vente des produits. Afin de diminuer ce coût, nous proposons une solution de livraison coopérative qui consiste à faire travailler ensemble les producteurs, c'est-à-dire qu'un producteur A peut être amené à livrer les produits d'un producteur B (ou seulement une partie de ces produits) si de cette manière, ce coût total est inférieur au coût total engendré s'ils avaient chacun fait leurs livraisons sans coopération.

## 2 Objectifs

L'objectif de ce projet est donc de réaliser une application permettant de calculer une solution de tournées coopératives entre les différents producteurs. Cette solution sera calculée en résolvant un problème de tournées de véhicules grâce à des heuristiques.

Le problème de tournées de véhicules est une variante du problème du voyageur de commerce. Celui-ci consiste à trouver le chemin le plus court qui permet de parcourir chaque ville d'un

graphe en passant une et une seule fois dans chaque ville. Le problème de tournées de véhicules part de la même base que le problème du voyageur de commerce, sauf qu'au lieu d'avoir un seul chemin qui relie tous les points, on a  $n$  chemins qui relient tous les points. Dans notre cas  $n$  est le nombre de producteurs faisant leurs livraisons. Ici on résout donc un problème de tournées de véhicules en prenant en compte les contraintes citées dans la partie contexte de réalisation dans le but de trouver la solution satisfaisant le plus les producteurs.

Étant donnée la complexité du problème de tournées de véhicules, trouver une solution exacte prendrait trop de temps, nous avons donc choisi d'utiliser les méthodes heuristiques. Ces méthodes ne garantissent pas de trouver la meilleure solution mais elles peuvent trouver une solution approchée avec un très bon rapport qualité/temps.

Les producteurs ont la possibilité de refuser la solution trouvée par l'analyse, par exemple si cette solution les fait parcourir un trajet beaucoup plus long que celui habituel. Il sera donc nécessaire que la solution trouvée soit résistante au changement, de façon à ce que si un producteur refuse sa livraison cela ne détériore pas trop la solution complète. Pour cela nous avons envisagé, Mr. Azeddine CHEREF et moi, une solution en cluster. De cette façon, si un producteur refuse son parcours, cela n'influe que son cluster. Après avoir fait une division des producteurs en clusters, il faut donc ensuite réaliser une heuristique sur chacun des clusters trouvés afin de trouver une solution par cluster. Cette division en cluster sera faite en prenant en compte la distance entre chacun des producteurs. Cette division en cluster peut être faite avec un simple algorithme de k-means.

Durant la recherche de chemins, il faut prendre en compte le partage des coûts dans l'analyse. En effet, si un producteur fait un parcours plus grand que celui des autres le jour 1, il faudra que le jour suivant cela soit pris en compte pour que le chemin qu'il lui soit affecté soit plus court. Ainsi, on peut partager les coûts de déplacements entre les différents producteurs. On peut faire cela en changeant le poids, c'est-à-dire la participation, des producteurs en fonction de leurs parcours déjà réalisés.

L'application fonctionnera sur un seul ordinateur. Elle utilisera un fichier contenant des données sur les livraisons, qu'elle parsera pour réaliser l'analyse des données avec les paramètres choisis par l'utilisateur.

### 3 Hypothèses

Le développement de l'algorithme de recherche de chemin sera prioritaire puis celui de clustering et enfin le partage des coûts aura la priorité la plus basse.

### 4 Bases méthodologiques

Pour la gestion de projet, j'utilise trello pour créer une liste des tâches à réaliser et déjà réalisées. J'utilise Google drive pour stocker les différents fichiers que j'ai l'occasion de réaliser pendant le projet, pour pouvoir y accéder depuis différentes machines et les versionner. J'ai utilisé Visual Paradigm pour réaliser la modélisation de mon projet, avec notamment un diagramme UML de cas d'utilisation ou un diagramme de classes. Les tests du projet ont été réalisés avec des données générées et avec des données créées à la main. Le but sera de comparer les améliorations faites grâce à ces algorithmes.

# 1

## Description générale

### 1 Environnement du projet

Il n'existe pas actuellement de logiciel mis à la disposition des producteurs pour résoudre le problème de coopération de livraisons. Les producteurs font donc actuellement leurs livraisons sans prendre en compte le parcours des autres producteurs. Il existe cependant des logiciels résolvant le problème de tournées de véhicules, mais ils ne sont pas adaptés à notre cas, trop généraux ou trop complexes pour être mis à disposition des producteurs.

Mr. Azeddine Cheref a déjà réalisé un programme résolvant le problème de coopération avec une méthode exacte. Ce programme pourra en partie être repris pour réaliser la suite du projet avec des heuristiques à la place de la méthode exacte.

### 2 Caractéristiques des utilisateurs

Pour cette application, il n'y a pour l'instant pas de clients réels et donc pas d'utilisateurs à part la MOA et la MOE. Mais on pourrait envisager le fonctionnement suivant :

- Un utilisateur principal possède toutes les informations sur les producteurs et leurs listes de livraisons sous forme d'un fichier, il donne ce fichier en paramètre à l'application.
- L'application calcule les meilleures solutions en fonction des données et des paramètres renseignés par l'utilisateur.
- Les producteurs sont des utilisateurs secondaires qui ont le droit de refuser le chemin qui leur est proposé.
- L'utilisateur principal devra ensuite adapter la solution en fonction des éventuels refus.
- Les utilisateurs secondaires n'ont pas accès à l'application mais seulement au résultat que l'utilisateur principal leur divulgue.
- Aucun de ces utilisateurs n'a de connaissance particulière en informatique.

### 3 Fonctionnalités du système

Voici le diagramme des cas d'utilisations modélisant les cas d'utilisations de ce projet.

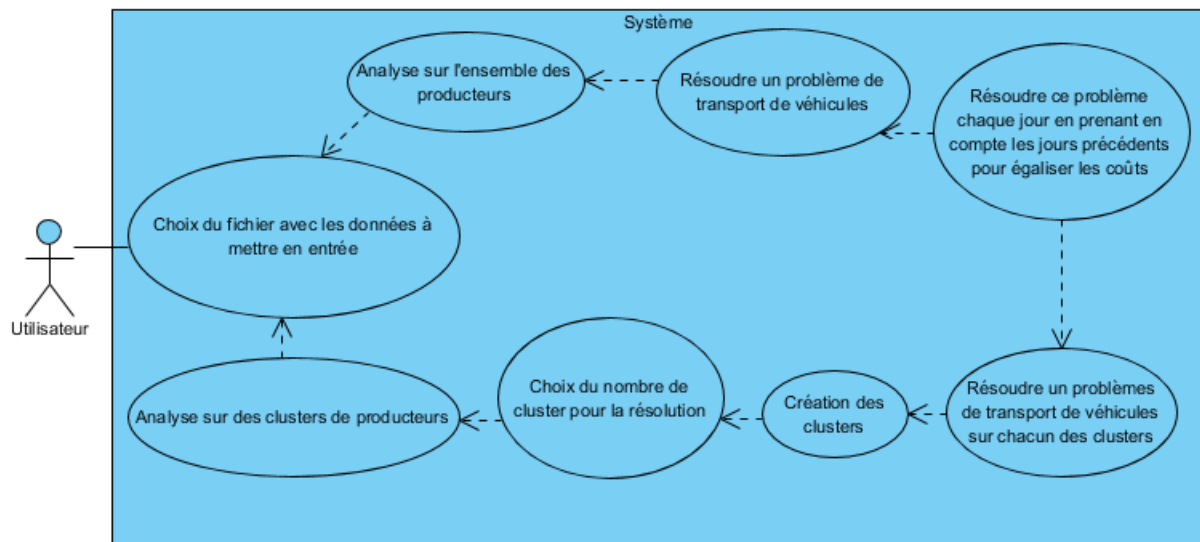


Figure 1 – diagramme des cas d'utilisations

Voici un diagramme d'activité modélisant ce projet.

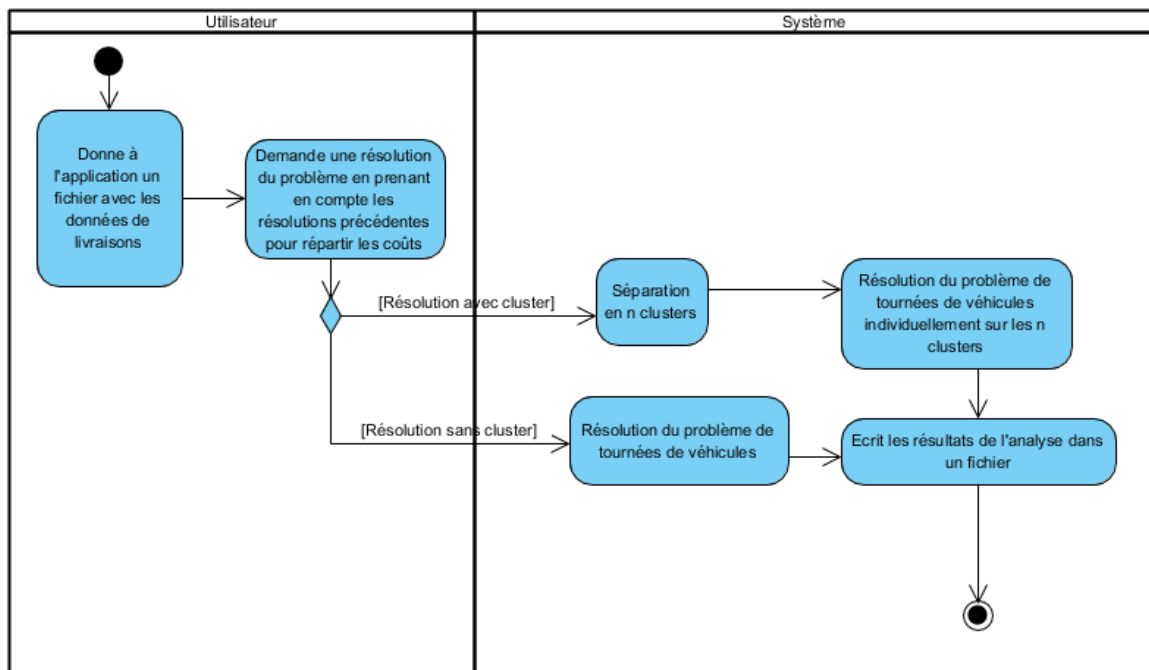
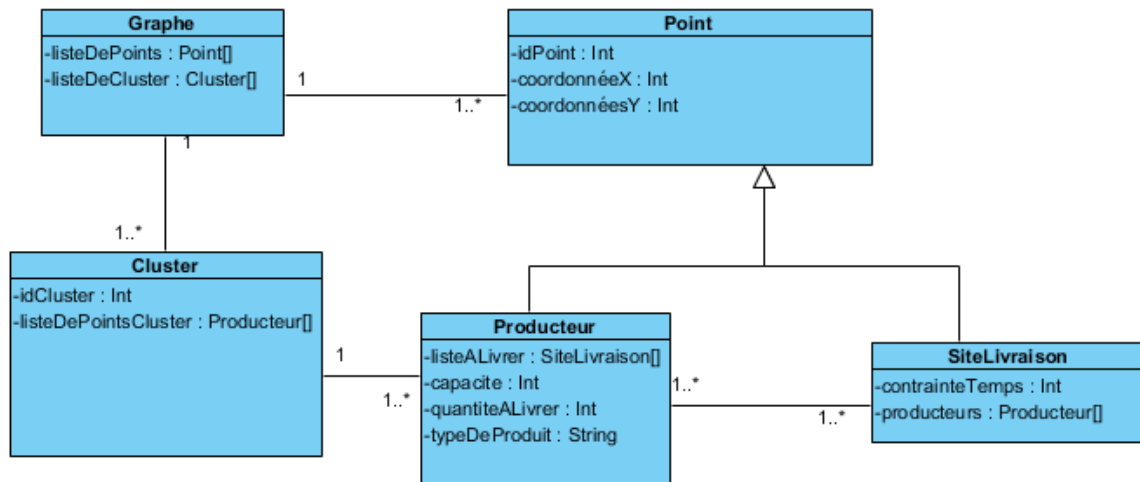


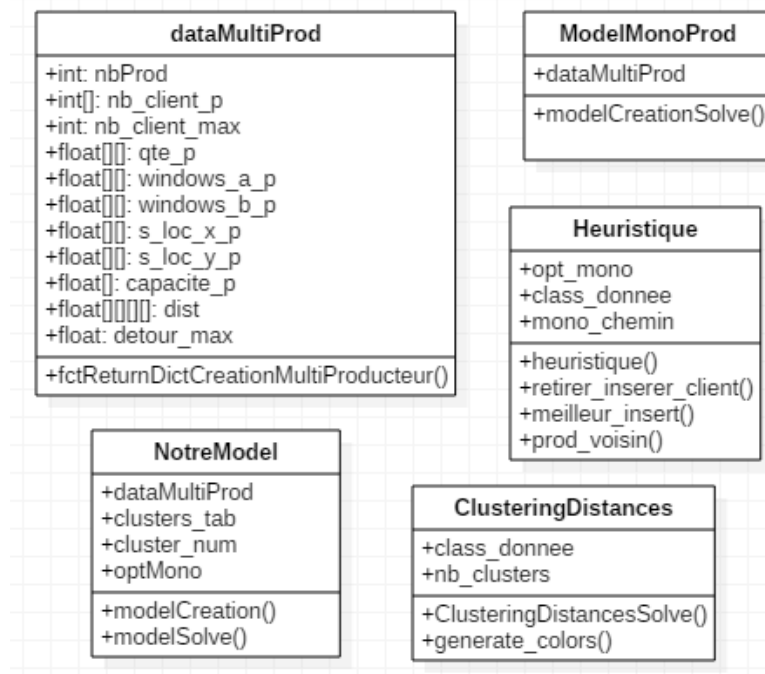
Figure 2 – diagramme d'activité

## 4 Structure générale du système

Voici un diagramme des classes modélisant les différentes classes de ce projet.

Figure 3 – *diagramme des classes*

Voici un diagramme des classes modélisant l'existant et la classe Heuristique que j'ai réalisée.

Figure 4 – *diagramme des classes final*

Toutes les données sur les producteurs et leurs clients sont résumées dans l'objet dataMultiProd sous la forme de différents tableaux. Cela permet d'accéder aisément aux données des différents points.

Par exemple, si on veut accéder à la coordonnée en x du premier client du premier producteur on fait : `s_loc_x_p[0][1]`. Car le premier producteur est le producteur 0 et le premier client du producteur 0 est à l'index 1 (l'index 0 représente le producteur lui-même).

Pour accéder à la coordonnée en y du troisième producteur il faut donc faire : `s_loc_x_p[2][0]`

# 2

## Etat de l'art

Le problème de tournées de véhicules (VRP) est l'un des problèmes les plus connus de la recherche opérationnelle. Ce problème est une variante du problème encore plus connu : le problème du voyageur de commerce, problème qui consiste à faire parcourir à un voyageur un graphe de point en faisant la plus petite distance. Sauf que dans le cas du problème de tournées de véhicules nous n'avons pas un seul voyageur mais plusieurs véhicules, il faut donc trouver les chemins les plus courts pour que ces  $n$  véhicules parcourent tous les points du graphe une et une seule fois.

Ce problème est très important dans le milieu des transports qui est très compétitif. Optimiser les livraisons permet de limiter le coût et l'émission en CO2 de celle-ci.

Étant un problème très connu, le problème de VRP possède plein de variantes. De plus, le problème de VRP est largement traité dans la littérature scientifique et il existe plusieurs variantes de ce problème, il est donc facile de trouver des articles récents et encore d'actualité sur le sujet. En cherchant avec ces mots-clés par exemple, "collaboration", "cooperation", "coalition", "alliance", "transportation", "routing", "logistics", "freight", "carrier", "shipper", on peut trouver énormément d'articles écrits durant les 30 dernières années sur le VRP et ses variantes.

On s'intéresse particulièrement au problème de VRP collaboratif. Je vais donc d'abord présenter une partie de la littérature récente sur ce sujet. Puis je vais présenter des cas d'entreprises utilisant un logiciel pour le résoudre.

### 1 Littérature scientifique

La littérature scientifique divise ce problème en plusieurs sous-problèmes s'appliquant chacun dans des situations différentes et avec des paramètres différents, chacun d'eux étant optimisé pour son cas. Dans cette partie, je vais décrire différents types de problèmes de tournée de véhicules et les recherches qui ont été faites sur ceux-ci.

Je vais d'abord faire une présentation générale du problème de tournées de véhicules, de ses paramètres et ses contraintes, ensuite je vais décrire les problèmes de tournées de véhicules collaboratifs en détail en décrivant les méthodes centralisées puis les méthodes décentralisées et les méthodes de partage des bénéfices et enfin je décrirai le problème qui nous concerne pour ce projet.

## 1.1 Le problème de tournées de véhicules et ses variantes

Le problème de tournées de véhicules est donc un problème consistant à faire parcourir à  $n$  véhicules une liste de points une et une seule fois, tout en minimisant la distance parcourue. Ce problème assez simple au premier abord, possède énormément de variantes qui permettent de spécifier le type d'étude effectuée. Dans cette partie, je vais citer puis expliquer ces différents paramètres.

Le paramètre TW (Time Windows) précise que les requêtes doivent être résolues dans une certaine fenêtre de temps. Par exemple, un colis ne peut être livré qu'entre 16h et 18h.

Le paramètre PD (Pick Up and Delivery) précise que durant la tournée, les véhicules peuvent reprendre des colis à certains points (à un dépôt par exemple). Pour plus de détails, je vous renvoie vers Savelsbergh, Martin et Sol, Massi (1995).

Les paramètres FTL ou LTL (Full Truck Load ou Less than Truck Load) précisent les types de camions utilisés pour le transport. Dans le cas du FTL, on considère que la taille d'une livraison remplit le camion. Dans le cas du LTL, on peut partir avec un camion qui n'a pas été totalement rempli. Pour plus de détails sur le FTL, je vous renvoie vers Liu R., Jiang Z., Fung R. Y., Chen F., et Liu X. (2010).

Les colis livrés peuvent être périssables et donc doivent être livrés dans une certaine tranche horaire ou dans un certain délai. Le nombre maximum de véhicules peut être limité. La distance maximale parcourue par chaque véhicule peut être limitée.

Le problème peut être posé avec différents types de véhicules pour le transport : les véhicules de transport de l'entreprise qui possèdent les marchandises transportées ou des véhicules de transports externes. La répartition des bénéfices est donc fonction du recours ou non à un transport externe.

En plus de ces paramètres, le VRP se divise en deux grandes parties :

La première est la méthode non collaborative où on cherche à maximiser le profit de chacun des participants individuellement. Chacun des  $i$  participants, où  $i$  appartient à  $(1, \dots, N)$ , cherchent à minimiser le chemin pour parcourir leurs requêtes  $R_i$ . Sachant que chaque requête à un paiement  $\pi_i(R_i)$  après la livraison et un coût  $\rho_i(R_i)$ . Je ne parlerai pas plus en détail du VRP sans collaboration car il ne touche pas à notre sujet. Sur ce point, je vous renvoie vers ce l'article suivant pour plus de détails : Marshall Fisher (1995).

La deuxième partie concerne la méthode de VRP collaborative que je vais présenter par la suite. Pour plus de précisions sur la comparaison entre les VRP collaboratifs et les VRP non collaboratifs, vous pouvez vous référer à Andrés Muñoz, Villamizar Jairo, R.Montoya, Torres Carlos, A.Vega-Mejía (2015).

## 1.2 Le problème de tournées de véhicules collaboratifs

Je vais maintenant présenter le VRP collaboratif. Contrairement au VRP non collaboratif, ici les participants ne cherchent pas à minimiser leur dépense individuelle mais la somme des dépenses de tous les participants.

Le problème peut être posé avec une autorité centrale possédant toutes les informations ou avec une autorité centrale ne possédant qu'une partie des informations. Dans le premier cas, une méthode centralisée doit être utilisée. Alors que dans le deuxième cas, une méthode décentralisée doit être utilisée. Cette autorité centrale représente le gérant des enchères dans le cas d'une méthode décentralisée avec enchère. Un exemple réel de cette autorité centrale avec toutes les informations, serait par exemple, la maison-mère d'une entreprise qui possède toutes les informations sur ses lieux de livraisons et ses véhicules.

### 1.2.1 Planning centralisé

Dans le cas d'une résolution du problème de tournées de véhicules avec une méthode de planning centralisé, le profit des participants est centralisé et on cherche à le maximiser.

Lorsque l'on utilise cette méthode, les décisions sont prises par une autorité centrale qui possède toutes les informations et les utilise pour trouver les chemins idéaux en faisant coopérer les différents livreurs. Ce cas est beaucoup utilisé en entreprise où l'autorité est centralisée et possède toutes les informations, ce qui lui permet de donner des chemins de livraisons à ses livreurs.

En utilisant la méthode de planning centralisé, en plus du simple problème de routage, il est important de considérer les dépôts. Le problème est alors décomposé en sous-problèmes pour être simplifié (on peut voir ça dans Buijs et al. (2016) ou Dai et Chen (2012)).

### 1.2.2 Planning décentralisé

Dans le cas d'une résolution du problème de tournées de véhicules avec une méthode de planning décentralisé, les participants ne partagent qu'une sous-partie de leurs requêtes (livraisons) que les autres participants peuvent choisir de prendre ou non.

Ce type de méthode est donc centré sur ce choix et sur le partage d'informations entre les participants. Ce cas d'étude est intéressant lorsque les participants ne veulent pas partager l'ensemble de leurs données, par exemple dans le cas d'une coopération entre entreprises concurrentes.

On peut diviser ce problème en deux sous-problèmes, une résolution sans enchères ou une résolution avec enchères.

#### Planning décentralisé sans enchères

La méthode de résolution utilisant un planning décentralisé sans enchère peut se diviser en trois sous-parties : la sélection d'un partenaire pour effectuer l'échange (Partner Selection - PS), la sélection des requêtes à échanger au partenaire (Request Selection - RS) puis la méthode pour effectuer cet échange (Request Exchange - RE).

La sélection de partenaires semble principalement dépendre de la synergie entre chacun des partenaires, s'ils sont géographiquement proches l'un de l'autre par exemple.

L'objectif principal de la sélection des requêtes à échanger est de supprimer les requêtes inutiles. Pour ce faire, on peut par exemple exécuter un problème d'orientation d'équipes qui va permettre de sélectionner les requêtes n'appartenant pas à la solution idéale. Pour plus de détails sur le problème d'orientation d'équipes (team orienteering problem), je vous renvoie vers I-MingChaoa, Bruce L.Golden, Edward A., Wasilc (1996).

Enfin, concernant l'échange de requêtes, l'échange par paquet simplifie le processus.

Cette méthode a l'avantage d'être moins complexe que la méthode avec enchère, mais, en contrepartie, elle génère moins de partage d'informations entre les participants.

#### Planning décentralisé avec enchères

La méthode décentralisée consiste à améliorer l'échange de requêtes à l'aide d'enchères où les participants peuvent déposer les requêtes qui ne les intéressent pas par lots. Ceux-ci sont ensuite rachetés par d'autres participants. Les premiers à avoir réalisé cette étude sont Krajewska and Kopfer (2006). La méthode de résolution utilisant une enchère est plus complexe en raison de l'organisation de cette enchère. Pour se faire, il faut qu'une autorité centrale s'occupe d'organiser l'enchère et de la gérer.

On peut diviser cette méthode en 5 étapes :

1. Le livreur décide quelles requêtes il veut mettre en enchère.
2. L'autorité s'occupant des enchères crée des groupes de requêtes et les propose aux livreurs.

3. Les livreurs placent leurs offres sur les lots de requêtes les intéressant.
4. Les lots sont alloués aux gagnants de la phase précédente.
5. Le profit des enchères est distribué entre les livreurs.

Les participants ont à la fois le rôle de vendeur et d'acheteur. La vente et l'achat de requêtes peuvent se faire individuellement ou en lot de requêtes. Plus il y a de requêtes ou de lots à vendre, plus l'enchère va prendre du temps. Pour simplifier, on peut donc limiter le nombre de requêtes mises en vente pour chaque participant.

Compte tenu de la complexité de cette méthode, elle n'a encore jamais été implémentée dans un cas réel, car il n'existe encore aucun framework ou logiciel permettant de lier les 5 étapes. La plupart des études concernant ce problème se concentrent sur l'une des 5 étapes sans s'occuper de les relier.

### 1.2.3 Méthode de partage des bénéfices

Les méthodes expliquées précédemment permettent de créer une coopération entre les livreurs afin d'augmenter leurs bénéfices. Mais une fois les parcours calculés et les requêtes redistribuées, il faut partager les bénéfices entre les participants.

Pour se faire, plusieurs méthodes existent. On peut par exemple citer la valeur de Shapley (Vanovermeire et Sörensen (2014)), la méthode proportionnelle (Özener, Ergun, and Savelsbergh (2013)) ou encore la méthode du nucléole (Guajardo et Jörnsten (2015)).

## 1.3 Etat de l'art sur notre problème

Ce projet constitue une variante du problème de tournées de véhicules : le Pick-up and Delivery. Comme expliqué plus haut, ce problème se base sur le fait que chaque participant doit passer par un point pour prendre les marchandises avant de les livrer à un ou plusieurs autres points. Ici, lorsqu'un producteur coopère avec un autre en prenant ses requêtes, il doit d'abord passer chez ce producteur pour prendre les marchandises à livrer. Ce problème est aussi une variante du problème de tournées de véhicules avec multiples dépôts, puisque chacun des producteurs peut être représenté par un dépôt. De plus, notre problème prend en compte une Time Windows pendant laquelle les produits peuvent être livrés. Les véhicules sont des LTL avec une capacité limitée.

L'un des objectifs de notre projet est de satisfaire les producteurs avec la solution qui leur est proposée. Dans le cas contraire, ils peuvent la refuser, nous devons donc intégrer la robustesse de la solution dans les critères à prendre en compte. On rend une solution plus robuste en divisant les producteurs en clusters à l'aide d'un partitionnement en k-means. En créant ces clusters, si un producteur refuse une solution cela n'impacte que la solution de son cluster et n'influe pas sur la solution des autres clusters. Les clusters peuvent servir à regrouper les producteurs qui veulent travailler ensemble. Le cluster permet ainsi restreindre le nombre de producteurs travaillant ensemble.

Il faut aussi prendre en compte le partage des bénéfices. En effet, au bout d'une première tournée, certains producteurs seront avantagés car ils auront parcouru une distance moindre. Il faut donc en tenir compte pour la prochaine création de chemins, en mettant un poids plus important sur les producteurs ayant peu participé aux solutions précédentes.

## 2 Cas en entreprise

La résolution du problème de tournées de véhicules est très importante pour optimiser les routes prises et minimiser les coûts engendrés. Les entreprises dans le milieu de la livraison ont donc beaucoup à gagner en optimisant ces coûts, certaines d'entre elles utilisent donc des

applications résolvant ce problème. L'objectif pour l'entreprise étant de faire une optimisation des coûts de l'ordre de 20 à 30% en utilisant ces logiciels.

Dans cette partie je vais vous présenter quelques logiciels utilisés en entreprise pour optimiser leurs livraisons.

## 2.1 TourSolver

Geoconcept The geoptimization company est une entreprise développant des logiciels touchant à l'optimisation de tournées, de géolocalisation ou encore de gestion de temps et de ressources. TourSolver est leur logiciel s'occupant de la création et optimisation de tournées. TourSolver fonctionne en 4 étapes simples. La première étape consiste à spécifier les paramètres de l'optimisation et le type d'activité pour laquelle le problème d'optimisation est résolu. La deuxième étape permet de définir les ressources utilisées. La troisième étape permet d'importer ou de créer les lieux de visite. La dernière étape exécute les calculs dans le cloud en visant un gain de 10-30%.

Cette solution est notamment utilisée par Domino's Pizza ou encore Veolia Transport.

## 2.2 PTV Route Optimizer

PTV Group est une entreprise réalisant des logiciels en rapport avec la mobilité, le transport et l'optimisation de ceux-ci.

PTV Route Optimizer est leur logiciel d'optimisation de tournées. Il permet de planifier des tournées, en paramétrant par exemple le type de véhicule, l'usage de dépôt ou non, les Time Windows etc.

Cette solution est notamment utilisée par DHL ou Kuehne Nagel.

# 3

## Analyse et conception

Voici une modélisation rapide du problème permettant de résumer les différentes contraintes :

- Liste de villes “départ” (producteur)
- Liste de villes “arrivée” (client)
- Toutes les villes ne doivent être visitées qu’une seule fois
- Tous les producteurs ont une liste de clients à livrer
- Un producteur peut assurer les livraisons pour un autre producteur mais il doit d’abord passer charger la marchandise
- A la fin de leurs tournées les véhicules doivent revenir à leurs points de “départ”
- Il existe un coût/distance entre chaque paire de ville (calculé à partir de leurs coordonnées respectives)
- Chaque producteur a une quantité de produits à livrer (un poids)
- Chaque producteur a un détour maximum par rapport à son chemin idéal sans coopération
- Chaque producteur a un véhicule
- Chaque véhicule a une capacité maximum
- Chaque client a une fenêtre de temps de livraison
- Un cluster est un groupe de producteurs
- Chaque cluster réunit des producteurs proches et avec des types de produits différents
- Objectif : Minimiser le coût moyen des tournées dans chaque cluster
- Objectif sur  $n$  tournées : Faire en sorte que le profit de chacun des producteurs soit similaire et proportionnel au nombre de produits vendus et à la participation aux livraisons

Ce problème est donc une variante du problème de pick-up and delivery, car on doit récupérer les livraisons à certains points avant de les déposer à d’autres points. De plus, ce problème implémente des clusters et une notion de partage de bénéfice/coût.

### 1 Modèle mono producteur

Pour trouver le chemin idéal de chaque producteur avec mutualisation, il faut d’abord trouver leur chemin idéal sans mutualisation/coopération afin de savoir s’ils dépassent le détour maximum qu’ils ont posé.

Dans cette partie, nous allons donc décrire et expliquer le modèle permettant de résoudre ce problème avec une méthode exacte.

#### Données

Voici les données dont nous disposons pour résoudre ce problème ainsi que leurs notations.

NC : Le nombre de clients du producteur plus 1 (le producteur étant le "client" 0).

C : La capacité que peut prendre le véhicule du producteur.

$Q_S, S \in \{0, \dots, NC\}$  : La quantité de produits que vend le client S.

$Dist_{i,j}$  : La distance entre le point i et le point j.

La fenêtre de temps pour la livraison vers le client S :

$Fa_S$  pour le début de la fenêtre de temps

$Fb_S$  pour la fin de la fenêtre de temps

M : gros chiffre permettant de symboliser les conditions dans certains cas (voir contrainte 2 par exemple)

### Variables

$X_{i,j}$  : Tableau à 2 dimensions de taille, contenant une valeur entre 0 et 1 (0 exclu) si l'arc entre le point i et le point j est parcouru. Sinon il fait 0.

$D_i$  : La date d'arrivée au point i,  $i \in \{0, \dots, NC\}$

R : La date de fin de tournée du producteur.

### Fonction objective

L'objectif de ce modèle est de minimiser la date de retour R :

$$\min(R - D[0])$$

### Contraintes

#### Contrainte 1 :

Contraintes obligeant le producteur à ressortir d'un point s'il y est entré, cela permet de former le parcours.

$$\forall j \in \{0, \dots, NC\},$$

$$\sum_{i=0, i \neq j}^{NC} X_{i,j} = 1$$

$$\sum_{i=0, i \neq j}^{NC} X_{j,i} = 1$$

#### Contrainte 2 :

Contrainte obligeant de passer dans un certain ordre et de fixer la date d'arrivée à chaque point.

$$\forall i \in \{0, \dots, NC\}, \forall j \in \{1, \dots, NC\},$$

$$D_j \geq D_i + Dist_{i,j} - M * (1 - X_{i,j})$$

Note : On peut ajouter une notion de temps de chargement en modifiant cette contrainte en :

$$D_j \geq D_i + Dist_{i,j} - M * (1 - X_{i,j}) + tempsdechargement$$

#### Contrainte 3 :

Contrainte permettant de vérifier que le début de la fenêtre de temps soit vérifié.

$$\forall i \in \{0, \dots, NC\},$$

$$D_i \geq Fa_i$$

#### Contrainte 4 :

Contrainte permettant de vérifier que la fin de la fenêtre de temps soit vérifiée.

$$\forall i \in \{0, \dots, NC\},$$

$$D_i \leq Fb_i$$

#### Contrainte 5 :

Contrainte permettant de calculer la durée de la tournée.  
 $\forall i \in \{0, \dots, NC\},$

$$R \geq D_i + Dist_{i,0}$$

## 2 Modèle mutualisation producteur

Dans cette partie nous allons décrire et expliquer le modèle permettant de résoudre le problème de mutualisation des producteurs avec une méthode exacte.

### Données

Voici les données dont nous disposons pour résoudre ce problème ainsi que leurs notations.

NP : Le nombre de producteurs

$NC_P, P \in \{0, \dots, NP\}$  : Le nombre de clients du producteur P.

$C_P, P \in \{0, \dots, NP\}$  : La capacité que peut prendre le véhicule du producteur P.

$Q_{P,S}, P \in \{0, \dots, NP\} \text{ et } S \in \{0, \dots, NC_P\}$  : La quantité de produits que le client S du producteur P a commandé.

$Dist_{P1,S1,P2,S2}$  : La distance entre le client S1 du producteur P1 et le client S2 du producteur P2.

La fenêtre de temps pour la livraison vers le client S du producteur P :

$Fa_{P,S}$  pour le début de la fenêtre de temps

$Fb_{P,S}$  pour la fin de la fenêtre de temps

$optMono_P$  : Le parcours idéal du producteur P pour livrer tous ses clients (trouvé grâce au modèle précédent)

$Det_P$  : Le détour maximum que le producteur P accepte.

M : gros chiffre permettant de symboliser les conditions dans certains cas (voir contrainte 7 par exemple)

### Variables

$X_{P0,P1,S1,P2,S2}$  :

Tableau à cinq dimensions de taille : nombre de producteurs (NP) \* nombre de producteurs (NP) \* nombre de clients du producteur ( $NC_P$ ) \* nombre de producteurs (NP) \* nombre de clients du producteur ( $NC_P$ ).

Il contient une valeur entre 0 et 1 (0 exclus) si le producteur P0 a parcouru l'arc entre le client S1 du producteur P1 et le client S2 du producteur P2. Sinon il fait 0.

$Y_{P0,P1,S1}$  :

Tableau à trois dimensions, de taille nombre de producteurs (NP) \* nombre de producteurs (NP) \* nombre de clients du producteur ( $NC_P$ ).

Il contient 1 si le producteur (1er dimension, P0) a visité le client (3ème dimension, S1) d'un autre producteur (2ème dimension, P1). Sinon il contient 0. P0 et P1 peuvent représenter le même producteur.

$D_{P0,P1,S1}$  : Il contient la date de livraison du producteur P0 au client S1 du producteur P1.

$R_P$  : Il contient la date de fin de tournée du producteur P.

$Tour_P$  : Il contient la durée de la tournée (date départ - date d'arrivée) du producteur P.

### Fonction objective

L'objectif de ce modèle est de minimiser la fonction suivante :

$$\sum_{P=0}^{NP} Tour_P$$

**Contraintes****Contrainte 1 :**

Contrainte limitant la variable Y et X à être binaire.

$$Y \in \{0, 1\}$$

$$X \in \{0, 1\}$$

**Contrainte 2 :**

Contrainte obligeant un producteur à ressortir d'un point s'il y est passé, cela permet de former le parcours.

$$\forall P_0, P_1 \in \{0, \dots, NP\}, \forall S_1 \in \{0, \dots, NC_{P_1}\}$$

$$\sum_{P_2=0, P_2 \neq P_1}^{NP} \sum_{S_2=0, S_2 \neq S_1}^{C_{P_2}} X_{P_0, P_1, S_1, P_2, S_2} = \sum_{P_0=0, P_2 \neq P_1}^{NP} \sum_{S_2=0, S_2 \neq S_1}^{C_{P_2}} X_{P_0, P_2, S_2, P_1, S_1}$$

**Contrainte 3 :**

Contrainte obligeant tous les sites à être visités.

$$\forall P_1 \in \{0, \dots, NP\}, \forall S_1 \in \{0, \dots, NC_{P_1}\}$$

$$\sum_{P_0=0}^{NP} Y_{P_0, P_1, S_1} = 1$$

**Contrainte 4 :**

Contrainte obligeant que le producteur soit visité avant de visiter l'un de ses clients.

$$\forall P_0, P_1 \in \{0, \dots, NP\}, \forall S_1 \in \{0, \dots, NC_{P_1}\}$$

$$Y_{P_0, P_1, 0} \geq Y_{P_0, P_1, S_1}$$

**Contrainte 5 :**

Contrainte liant la variable X et Y. Si le producteur P0 passe par le client S1 du producteur P1, alors la somme des arcs sortant de ce point avec le producteur P0 doit faire 1.

$$\forall P_0, P_1 \in \{0, \dots, NP\}, \forall S_1 \in \{0, \dots, NC_{P_1}\}$$

$$Y_{P_0, P_1, S_1} = \sum_{P_2=0, P_1 \neq P_2}^{NP} \sum_{S_2=0, S_1 \neq S_2}^{C_{P_2}} X_{P_0, P_1, S_1, P_2, S_2}$$

**Contrainte 6 :**

Contrainte qui permet d'assurer que la quantité de produits prise est inférieure à la capacité du producteur.

$$\forall P_0 \in \{0, \dots, NP\}$$

$$\sum_{P_1=0}^{NP} \sum_{S_1=0}^{C_{P_1}} Y_{P_0, P_1, S_1} * Q_{P_1, S_1} \leq C_{P_0}$$

Cette contrainte se contente de vérifier que la somme des poids des livraisons est inférieure à la capacité du véhicule. Dans le cas d'un problème de tournées de véhicules classique, cette vérification est suffisante. Mais dans notre cas, la capacité du véhicule peut augmenter ET diminuer selon que le véhicule passe chez un client (on dépose des marchandises, on gagne de la capacité) ou chez un autre producteur (on prend des marchandises, on perd de la capacité). On peut donc optimiser cette modélisation en modifiant la contrainte de capacité pour avoir une capacité par producteur et par point ( $C_{P_0, P_1, S_1}$  au lieu de  $C_{P_0}$ ) et ajouter cette contrainte :

$$C_{P_0, P_2, S_2} - Q_{P_0, P_1, S_1} - M(1 - X_{P_0, P_1, S_1, P_2, S_2}) \leq C_{P_0, P_1, S_1}$$

**Contrainte 7 :**

Contrainte permettant de calculer les dates de livraison du producteur P0 au client S1 du producteur P1.

$$\forall P0, P1, P2 \in \{0, \dots, NP\}, \forall S1 \in \{0, \dots, C_{P1}\}, \forall S2 \in \{0, \dots, C_{P2}\}, P2 \neq P0, S2 \neq 0$$

$$D_{P0, P2, S2} \geq D_{P0, P1, S1} + Dist_{P1, S1, P2, S2} - M * (1 - X_{P0, P1, S1, P2, S2})$$

Note : On peut ajouter une notion de temps de chargement en modifiant cette contrainte en :

$$D_{P0, P2, S2} \geq D_{P0, P1, S1} + Dist_{P1, S1, P2, S2} - M * (1 - X_{P0, P1, S1, P2, S2}) + tempsdechargement$$

#### Contrainte 8 :

Contrainte permettant de calculer la date de retour du producteur P0 à son point de départ (le client 0 de P0).

$$\forall P0, P1 \in \{0, \dots, NP\}, \forall S1 \in \{0, \dots, C_{P1}\}$$

$$R_{P0} \geq D_{P0, P1, S1} + Dist_{P1, S1, P0, 0} * X_{P0, P1, S1, P0, 0}$$

#### Contrainte 9 :

Contrainte permettant de vérifier que le début de la fenêtre de temps soit vérifié.

$$\forall P0, P1 \in \{0, \dots, NP\}, \forall S1 \in \{1, \dots, C_{P1}\}$$

$$D_{P0, P1, S1} \geq Fa_{P1, S1} * Y_{P0, P1, S1}$$

#### Contrainte 10 :

Contrainte permettant de vérifier que la fin de la fenêtre de temps soit vérifiée.

$$\forall P0, P1 \in \{0, \dots, NP\}, \forall S1 \in \{1, \dots, C_{P1}\}$$

$$D_{P0, P1, S1} \leq Fb_{P1, S1} * Y_{P0, P1, S1}$$

#### Contrainte 11 :

Contrainte permettant de faire en sorte que le producteur P0 passe chez le producteur P1 avant de livrer les clients de ce producteur.

$$\forall P0, P1 \in \{0, \dots, NP\}, \forall S1 \in \{0, \dots, C_{P1}\}, P1 \neq P0, S1 \neq 0$$

$$D_{P0, P1, S1} \geq D_{P0, P1, 0}$$

#### Contrainte 12 :

Contrainte permettant de calculer la durée de la tournée.

$$\forall P0 \in \{0, \dots, NP\}$$

$$Tour_{P0} = R_{P0} - D_{P0, P0, 0}$$

#### Contrainte 13 :

Contrainte permettant de respecter le détournement maximum autorisé par le producteur P0.

$$\forall P0 \in \{0, \dots, NP\}$$

$$Tour_{P0} \leq optMono_{P0} * (1 - Det_{P0})$$

### 3 Heuristique

Les modélisations ci-dessus permettant de résoudre le problème avec une méthode exacte, ne sont fonctionnelles qu'avec un nombre de données très bas. Il a donc été nécessaire de faire des recherches et réfléchir à la création d'une heuristique adaptée à notre problème.

Dans cette partie, je vais décrire les recherches que j'ai effectuées puis conclure avec l'algorithme que j'ai créé et utilisé pour développer l'heuristique.

## Recherches

Pour commencer mes recherches, j'ai utilisé l'article "Heuristiques pour les problèmes de tournées de véhicules multi-attributs" par Thibaut Vidal, Christian Prins etc (Mars 2011). Cette publication détaille l'état de l'art des heuristiques pour le problème de tournées de véhicules et les différentes variantes de celui-ci. Comme expliqué dans la partie Etat de l'art de ce rapport, les 2 variantes les plus proches de notre problème sont : les problèmes avec multiples dépôts et les problèmes de pickup and delivery (collecte et livraison).

		ESP. RECH. ECHANGE ARCS	RELAXATION ESPACES MULT. CROISEMENTS VOIS. LARGES ACCÉL. LS	TRAJEC. MIXTES SAUTS CONT. + DÉTER. ALÉA	APPR. GUIDAGE GESTION DIV. POPULATIONS ADAPT. PARAMS.	HYBRIDATION	MATH. HEUR.	COOPERATION	DÉCOMPOSITION	PARALLÉLISME
<b>VRP "CLASSIQUE"</b>										
Tarantilis (2005)	M. Adap + Tabou	X	X	X X X	X X					
Prins (2009a)	GRASP×ELS	X X	X	X X X X	X X	X	X			
Zachariadis et K. (2010a)	Tabou	X X		X X	X					
Mester et Bräysy (2007)	EA + ELS guidé	X X X		X X	X	X	X			
Nagata et Bräysy (2009b)	Génétique	X X	X	X X	X		X			
<b>MULTIPLE DEPOTS</b>										
Renaud et al. (1996)	Tabou	X		X X						
Cordeau et al. (1997)	Tabou	X X	X	X	X	X				
Pisinger et Ropke (2007)	ALNS		X X X	X X	X	X	X			
<b>FLOTTE HETEROGENE</b>										
Li et al. (2007a)	SA (R-to-R)	X X		X		X				
Prins (2009c)	Génétique	X	X X	X X	X X	X				
<b>PERIODIQUE</b>										
Cordeau et al. (1997)	Tabou	X X	X	X	X	X				
Alegre et al. (2007)	SS	X X X		X X	X X	X			X	
Hemmelmayr et al. (2009)	VNS	X	X	X X		X				
<b>LIVRAISONS FRACTIONNES</b>										
Chen et al. (2007)	SA (R-to-R)	X X	X	X		X	X X			
Boudia et al. (2007)	Génétique	X	X X	X X	X X	X				
Mota et al. (2007)	Scatter Search	X	X	X	X					
<b>TOURNEES + PRIMES</b>										
Archetti et al. (2006)	Tabu + VNS	X	X X	X X X X		X	X			
Ke et al. (2008)	ACO	X		X X	X	X				
Souffriau et al. (2010)	Path Relinking	X	X	X X	X X					
<b>BACKHAULS</b>										
Brandao (2006)	Tabou	X	X	X X	X	X				
Ropke et Pisinger (2006a)	ALNS		X X X	X X	X	X	X			
Gajpal et Abad (2009)	ACO	X X	X	X X		X				
<b>COLLECTE ET LIVRAISON</b>										
Bent et Hentenryck (2006)	SA + LNS	X X X	X	X X			X X			
Ropke et al. (2007)	ALNS		X X X	X X	X	X	X			
Cordeau et Laporte (2003)	Tabou	X X	X	X	X	X				
Parragh et al. (2010)	VNS	X X	X X	X X		X				
<b>ROUTES MULTIPLES</b>										
Alonso et al. (2007)	Tabou	X	X	X	X	X				
Olivera et Viera (2007)	M. Adap + Tabou	X X	X X	X X X X	X	X				
Salhi et Petch (2007)	Génétique	X	X X X	X X	X X	X				

Figure 1 – tableau récapitulatif des heuristiques

On s'intéresse donc, dans ce tableau issu de cet article, aux lignes "multiple dépôts" et "collecte et livraison".

J'ai finalement étudié les articles suivants : Li H. et Lim A. (2001) et Ropke S. et Pisinger D. (2006), car ils se rapprochaient le plus de notre problème.

Dans Li H. et Lim A. (2001) l'heuristique étudiée est une heuristique se basant sur la méthode Tabou et des explorations de voisinage. La méthode Tabou est une méthode avec mémoire qui évite de retomber plusieurs fois sur la même solution. Malheureusement l'algorithme se base sur le fait qu'il existe une paire point de dépôt - point de livraison. Ce qui n'est pas le cas dans notre problème, chaque dépôt (producteur) pouvant avoir plusieurs livraisons (clients).

Dans Ropke S. et Pisinger D. (2006), l'heuristique étudiée est une heuristique se basant sur l'exploration du voisinage. L'exploration de voisinage se fait à l'aide de méthodes retirant un groupe de points d'un chemin pour les insérer dans un autre chemin. Pour cela, plusieurs

méthodes existent pour retirer les points puis les réinsérer.

Méthodes pour retirer les points :

- La méthode Shaw qui calcule la similarité entre les points pour retirer un groupe de points similaires
- La méthode Random qui choisit un groupe de points aléatoirement
- La méthode Worst qui retire les points les plus coûteux sur ce chemin

Méthodes pour insérer les points :

- La méthode Basique qui insère en priorité les points augmentant le moins le coût
- La méthode avec regret qui calcule le regret avant d'insérer les points. Le regret est une mesure qui permet de connaître la différence entre insérer le point maintenant ou après avoir inséré n autres points

J'ai utilisé cet article comme modèle pour réaliser mon algorithme.

### Algorithme

Dans cette partie, je vais présenter et expliquer en détail ce que fait l'algorithme que j'ai réalisé. L'algorithme prend comme modèle celui réalisé dans Ropke S. et Pisinger D. (2006) et fonctionne par retrait puis insertion de points. Il commence par la solution initiale retournée par le modèle exact mono producteur.

A partir de cette solution, on sélectionne un chemin aléatoire (ici chemin et producteur sont similaires), on sélectionne un producteur qui est dans le chemin sélectionné précédemment (à la première itération, les 2 producteurs sélectionnés sont forcément les mêmes). On sélectionne un nombre aléatoire de points appartenant au producteur sélectionné dans le chemin sélectionné. Ces points sont dans un ordre aléatoire. On sélectionne une liste de producteurs proches du chemin sélectionné et on essaie d'insérer les points sélectionnés un par un. Chaque fois que l'on trouve une meilleure solution, on retient dans quel chemin l'insertion s'est faite, avec combien de clients et à quelle position. Une fois que tous les producteurs ont été testés, on effectue le meilleur mouvement retenu. On recommence toutes ces étapes 500 fois (voir la partie "Test et résultats" dans le chapitre 4 "Mise en oeuvre" pour la justification du nombre d'itérations).

```

m = 3
cout_initial # le cout initial de la solution
Pour i = 1..5
    P0 = un producteur aléatoire
    P1 = un des producteurs dans le chemin de P0 #P1 peut être égale à P0
    N = nombre aléatoire entre 1 et le nombre de clients de P1 dans le chemin de P0
    L = on sélectionne dans un ordre aléatoire les N clients de P1 et dans le chemin de P0
    Pliste = parmi les 5 plus proches producteurs voisins de P0 on en sélectionne m aléatoirement
    cout_min = 99999
    meilleur_solution
    meilleur_producteur
    Pour chaque producteur Px de Pliste :
        cout_producteur_Px = cout_initial
        Pour j = 1..N :
            si le producteur P1 du point L[j] est déjà inséré dans Px :
                on trouve la meilleure position où on peut insérer le point L[j] dans Px après le producteur P1
                cout_producteur_Px = cout_producteur_Px + cout de l'insertion du point L[j] dans Px
                    + coût du retrait du point L[j] de P0
            si le producteur P1 du point L[j] n'est pas déjà inséré :
                on trouve la meilleure position où on peut insérer le point L[j] dans Px
                on trouve la meilleure position où on peut insérer le producteur P1 du point L[j] dans Px avant le point L[j]
                cout_producteur_Px = cout_producteur_Px + coût de l'insertion du point L[j] dans Px
                    + coût du retrait du point L[j] de P0 + coût de l'insertion du producteur P1 dans Px
            si cout_producteur_Px est meilleur que cout_min alors :
                cout_min = cout_producteur_Px
                meilleur_solution = les j premiers points de L
                meilleur_producteur = Px
On retire les points meilleur_solution de P0
On effectue l'insertion des points (et de leur producteur P1 si nécessaire) de meilleur_solution dans meilleur_producteur
cout_initial = cout_min

```

Figure 2 – algorithme de l'heuristique

# 4

## Mise en oeuvre

Dans ce chapitre, je vais détailler le programme qui a été réalisé pour ce projet. Je vais séparer les parties réalisées par mon encadrant et les parties que j'ai développées.

### 1 Présentation du programme

Voici l'architecture générale du programme. Le programme a été réalisé en python en utilisant la version 3.6.

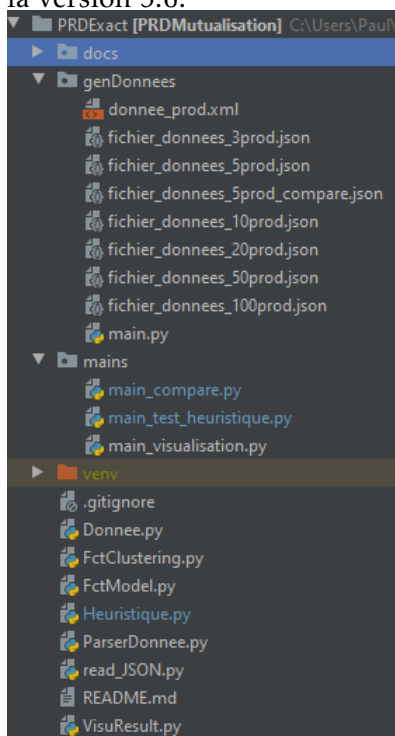


Figure 1 – architecture du programme

#### Dossier genDonnees

Dans le dossier genDonnees se trouve toute la partie servant à générer des données avec les différents fichiers générés. Le fichier main.py dans ce dossier permet de générer un fichier de données JSON en respectant des paramètres comme le nombre de producteurs, le nombre

de clients moyen par producteur etc.. Il a été créé par mon encadrant. Le fichier XML `donnee_prod.xml` est un fichier xml que j'ai créé à la main pour montrer à quoi doit ressembler un fichier xml s'il doit être parsé. Les autres fichiers sont des fichiers de données JSON que j'ai générés grâce au `main.py` en gardant une logique de nommage pour que leur contenu soit compréhensible.

### Racine

A la racine se trouve la partie la plus importante du projet avec tous les fichiers faisant les différents traitements sur les données.

Il y a d'abord le fichier `Donnee.py` réalisé par mon encadrant. Il contient les différentes fonctions et classe permettant de modéliser les données sur les producteurs. C'est la classe `"dataMultiProd"` qui permet de modéliser un groupe de producteurs et leurs clients à l'aide de plusieurs tableaux. Ces tableaux donnent chacun accès à une donnée. Par exemple le tableau à 2 dimensions `s_loc_x_p` donne accès à la coordonnée en x pour chacun des points, il faut lui préciser le numéro du producteur et le numéro du client du point dont on cherche les informations. Ce fichier contient aussi la fonction `CreationMonoProducteur` qui permet de générer des producteurs à partir de paramètres.

Les fichiers `read_JSON.py` et `ParserDonnee.py` permettent tous les deux de parser un fichier de données et retourne un objet `"dataMultiProd"`. `read_JSON.py` parse un fichier JSON et a été réalisé par mon encadrant. `ParserDonnee.py` permet de parser un fichier XML, j'ai développé ce fichier.

Le fichier `FctClustering.py` a été réalisé par mon encadrant et gère la partie clustering et son affichage. J'y ai ajouté une fonction `generate_colors` pour pouvoir afficher autant de données que l'on veut lors de l'affichage, chacun des producteurs aura une couleur différente générée aléatoirement. Voici un exemple de l'affichage avant et après avoir créé les clusters :

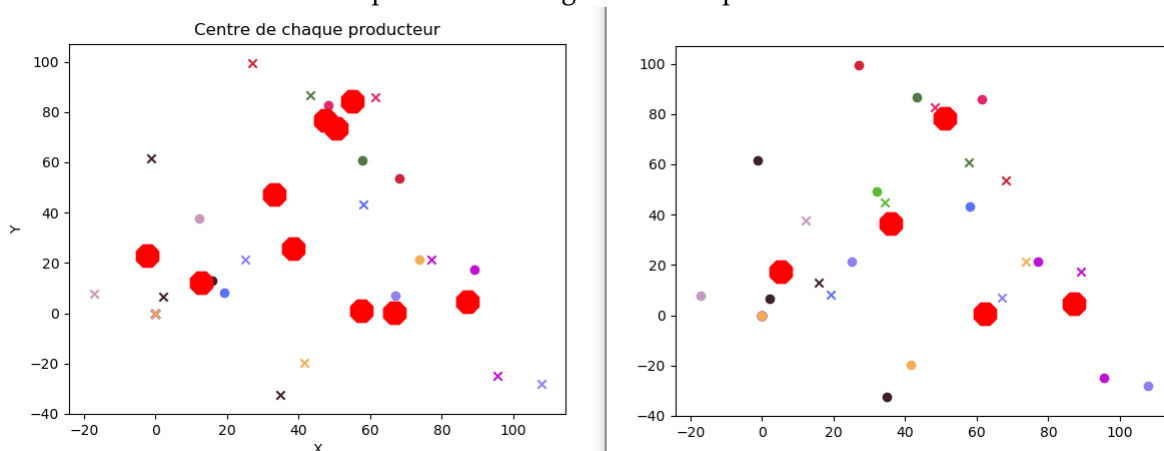


Figure 2 – affichage des clusters

Chacun des producteurs est représenté par un point et les clients par des croix. Les clients et producteurs liés ont la même couleur. Les hexagones rouges représentent le centre des clusters, avant le clustering on a un cluster par producteur.

Le fichier `VisuResult.py` a été réalisé par mon encadrant et contient des fonctions permettant d'afficher le chemin trouvé sous forme de plot, je ne l'ai pas utilisé.

Le fichier `FctModel.py` a été réalisé par mon encadrant. Il gère toutes les modélisations du problème puis résolution du problème en utilisant `DOCPLEX`. Pour plus de détails sur la modélisation voir la partie "Modèle mono producteur" et "Modèle mutualisation producteur" du chapitre 3 "Analyse et conception". Si je n'ai pas développé ce fichier, j'y ai quand même ajouté beaucoup de commentaires pour en faciliter la compréhension. J'ai aussi ajouté une partie permettant de parser le résultat rendu afin d'en créer une solution (une solution étant le chemin parcouru par chacun des producteurs).

J'ai choisi de représenter la solution sous la forme d'un tableau de tableaux. Chacun des tableaux

représente un chemin parcouru par un producteur. Chacun des points dans le chemin sont représentés par un tuple ( , ) contenant le numéro du producteur et le numéro du client lui faisant référence. De cette manière, on peut, à partir de ce tuple, accéder aux informations sur ce point dans les tableaux de dataMultiProd. Voici un exemple de solution :

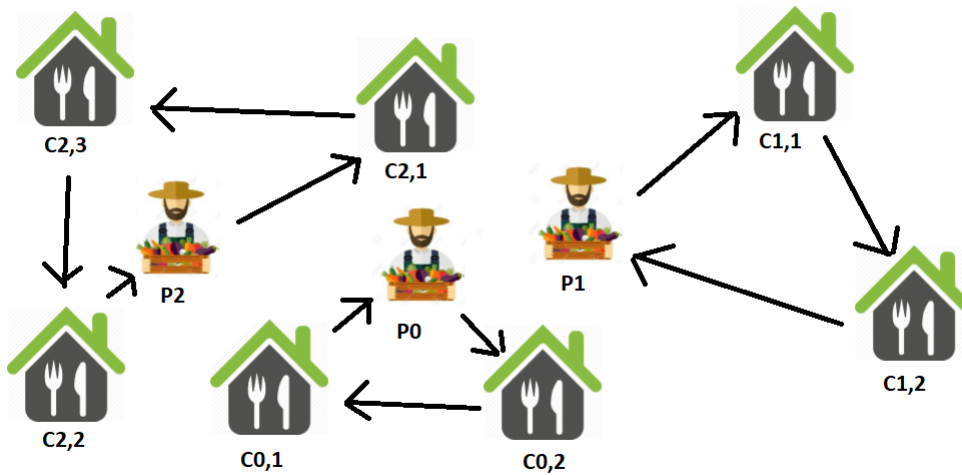
$$[[ (0,0), (0,2), (0,1) ], [ (1,0), (1,1), (1,2) ], [ (2,0), (2,1), (2,3), (2,2) ]]$$


Figure 3 – Une solution

Sur cet exemple il y a 3 chemins donc 3 producteurs. Le producteur 0 passe par son client 2, puis le 1. Le producteur 1 passe chez son client 1 puis le client 2. Le producteur 2 passe chez son client 1, 3, puis 2.

Puis voici un exemple de solution après la mutualisation :

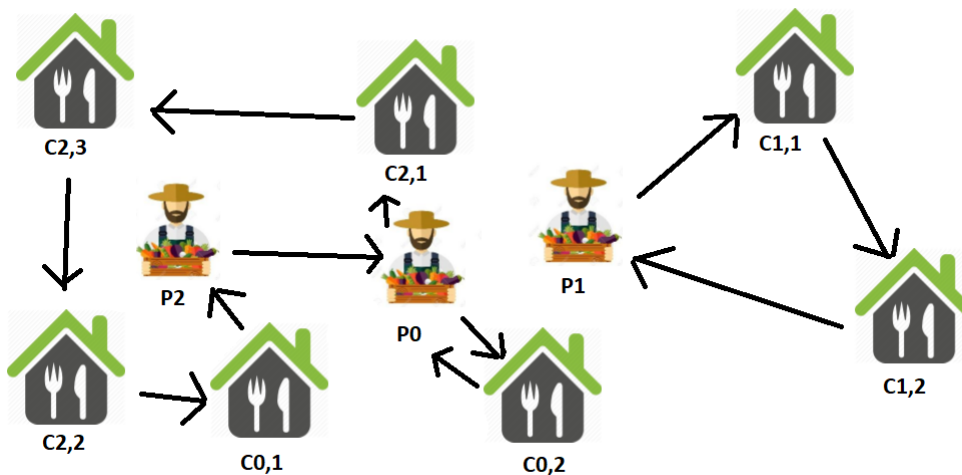
$$[[ (0,0), (0,2) ], [ (1,0), (1,1), (1,2) ], [ (2,0), (0,0), (2,1), (2,3), (2,2), (0,1) ]]$$


Figure 4 – Une solution après mutualisation

Ici le producteur 0 ne va que chez son client 1. Le producteur 1 fait toujours les mêmes livraisons. Mais le producteur 2 lui passe maintenant chez le producteur 0 afin de prendre des marchandises, il va ensuite livrer ses clients 1, 3 puis 2 et finit ses livraisons chez le client 1 du producteur 0 dont il avait pris les marchandises plus tôt.

Le fichier Heuristique.py est un fichier que j'ai développé. Il contient toutes les fonctions et classes relatives à l'heuristique. Je détaillerai ce fichier dans la partie 2 "Zoom sur le fichier heuristique".

### Dossier mains

Ce dossier contient tous les mains.

`main_compare.py` permet de comparer l'efficacité de l'heuristique par rapport à l'efficacité de la méthode exacte avec cluster. La méthode exacte avec cluster n'étant fonctionnelle qu'avec très peu de données (dû aux limitations du solver), ce test ne peut être effectué qu'avec le fichier de test `fichier_donnees_5prod_compare.json` qui a été créé pour cela. Il suffit de préciser le path de ce fichier en paramètre de la commande pour l'utiliser.

`main_test_heuristique` permet de lancer l'heuristique plusieurs fois et en retourne des chiffres comme le temps pour trouver la meilleure solution ou la valeur de celle-ci. Il suffit de préciser le path du fichier voulu en paramètre de la commande pour l'utiliser. De plus, il est possible de préciser le cas que l'on souhaite étudier en 2ème paramètre. Les cas possible sont :

- cas 0 : le cas neutre, sans changements supplémentaires
- cas 1 : le cas avec des livraisons "urgentes" qui doivent être réalisées par leur producteur de départ
- cas 2 : le cas avec une limite de passage chez d'autres producteurs par chemin (limité à 3)

Enfin `main_visualisation` est un main à finaliser avec pour objectif de visualiser les chemins parcourus sur un graphe.

### Dossier docs

Ce dossier contient la documentation générée par Sphinx.

Le sous-dossier `source` contient tous les fichiers nécessaires à cette génération de documentation.

Le sous-dossier `build` contient la documentation générée sous forme de fichiers `.html`.

## 2 Zoom sur le fichier heuristique

Dans cette partie je vais décrire plus en détail le fichier `Heuristique.py` que j'ai réalisé.

Tout le contenu du fichier `Heuristique.py` est contenu dans la classe du même nom. Cette classe est construite en prenant le contenu de la classe `dataMultiProd` pour récupérer toutes les données sur les clients et producteurs, mais elle prend aussi la solution et la fonction objectif retournée par la méthode exacte mono producteur. Cela permet d'utiliser la solution comme point de départ à l'heuristique mais aussi d'utiliser la fonction objectif pour trouver le détournement max de chacun des producteurs.

Une fois l'objet `Heuristique` construit, on peut lancer la fonction méthode principale de cet objet, la méthode `heuristique`. Cette méthode prend en paramètre le cas que l'on souhaite étudier.

Cette fonction lance donc l'heuristique comme décrite dans le paragraphe "Algorithme" de la partie 3 "Heuristique" du chapitre 3 "Analyse et conception".

Lorsque les producteurs et clients à modifier ont été sélectionnés aléatoirement, cette fonction va chercher quelle est la meilleure insertion possible en utilisant la méthode `meilleur_insert` qui prend en paramètre :

- le client que l'on veut insérer
- le producteur auquel on va retirer des points de son chemin
- le producteur auquel appartenait les points modifiés originellement
- une copie du chemin dans lequel on veut faire l'insertion
- la distance parcourue dans la copie du chemin
- la capacité restante du producteur parcourant la copie du chemin

Ces informations vont nous servir à trouver le meilleur endroit où insérer le client dans le chemin en tenant en compte du besoin d'insérer son producteur avant et en faisant attention aux contraintes de capacité et de détournement maximum. Nous avons choisi de ne pas prendre en compte la contrainte de Time Windows. Cette méthode retournera un tableau contenant l'index où le client est inséré et le coût de cette insertion, si son producteur est inséré le tableau contiendra aussi ses informations. Si l'insertion est impossible en raison du non respect des contraintes,

cette méthode retournera à la place : 0 si la contrainte de poids n'est pas respectée, 1 si la contrainte de détour max n'est pas respectée, 2 si lorsque nous sommes dans le cas 2 et le chemin a déjà le nombre maximum de producteurs.

La méthode principale va récupérer ces informations et comparer pour chacun des chemins où l'on essaie d'insérer des clients, lequel donne le meilleur résultat. Une fois le meilleur chemin sélectionné et les meilleures insertions choisies, la méthode `retirer_inserer_client` s'occupe d'effectuer les changements.

Lorsque ces modifications sont faites, la méthode principale recommence les traitements précédents sur la nouvelle solution créée, tout en retenant toujours la meilleure solution trouvée.

Une fois toutes les itérations effectuées, la méthode s'arrête et retourne un tableau contenant : la meilleure solution créée, le poids de cette solution ainsi que le temps que l'algorithme a mis pour la trouver.

### 3 Tests et résultats

Dans cette partie, je vais lister les différents tests utilisateur que j'ai réalisés pour évaluer l'heuristique.

#### Trouver le bon nombre d'itérations

Les premières mouvements de clients ont tendance à empirer la solution, ce n'est qu'après que les clients et producteurs soient bien "mélangés" que l'on commence à avoir des meilleures solutions que notre solution initiale.

La question est donc : combien de mélanges/d'itérations faut-il effectuer avant d'atteindre une bonne solution ?

Pour répondre à cette question j'ai effectué des tests sur plusieurs données générées aléatoirement, avec 20, 50 et 100 producteurs.

Sur chacun de ces tests, j'ai mis le nombre d'itérations à 3000 sachant que ce nombre d'itérations est beaucoup trop important, il va me permettre de trouver l'itération où la meilleure solution est trouvée.

J'ai donc trouvé les meilleures itérations suivantes :

- 20 producteurs : 87, 27, 16, 22, 34, 17, 112, 23, 15, 31, 10, 12, 23, 58, 21, 21, 52, 18, 157, 15
- 50 producteurs : 76, 88, 53, 63, 85, 84, 268, 79, 76, 60, 64, 49, 65, 60, 47, 91
- 100 producteurs : 168, 196, 149, 238, 170, 179, 123, 164, 280, 222, 147, 292, 197, 220, 139, 229, 463, 159, 197

On constate que le nombre d'itérations nécessaire pour trouver la meilleure solution est plus important avec 100 producteurs, ce qui est logique car avec plus de producteurs, il faut plus de temps pour "mélanger" les clients.

Comme on cherche un nombre d'itérations maximum, on n'a besoin de regarder que les nombres itérations trouvées avec 100 producteurs. On a donc un minimum de 123 et un maximum de 463.

Par rapport à ce résultat, j'ai donc décidé de fixer le nombre d'itérations à 500.

#### Comparaison heuristique/cluster

Pour faire cette comparaison j'ai utilisé le `main_compare.py` qui permet de comparer les résultats de la méthode avec cluster et la méthode avec heuristique. Pour lui donner un fichier contenant des données sur les producteurs, il suffit de lui préciser son path dans les paramètres du run.

J'ai utilisé le fichier `fichier_donnees_5prod_compare.json` pour ces tests. Il contient des données sur 5 producteurs qui ont chacun entre 2 et 4 clients. Ce fichier contient très peu de données car avec plus de producteurs ou de clients, la méthode exacte utilisée par la méthode avec cluster atteint sa limite.

Le résultat obtenu par la méthode exacte sans mutualisation est : 1018.75.

Le résultat obtenu en séparant les producteurs en clusters où l'on résout avec une méthode exacte pour chacun d'eux est : 810.60. On a donc une amélioration de 20.4%. Comme c'est une méthode exacte on obtiendra toujours ce résultat.

Le résultat de l'heuristique, lui, varie. Sur 10 tests on obtient :

714.46, 647.58, 696.97, 638.34, 708.36, 622.87, 657.19, 703.28, 606.73, 652.58

Ce qui nous donne une moyenne de : 664.83 et une amélioration moyenne de 34.74%. On constate donc qu'en plus de fonctionner sur des données plus importantes que la méthode avec cluster, les résultats de l'heuristique sont meilleurs.

### Comparaison des différents cas

Dans cette partie, je vais tester les résultats de l'heuristique sur différentes données avec 5, 10, 20, 50 et 100 producteurs (tous les producteurs ont 5 clients). Je vais aussi tester les résultats des différents cas sachant que :

- cas 0 : le cas neutre, sans changements supplémentaires
- cas 1 : le cas avec des livraisons "urgentes" qui doivent être réalisées par leur producteur de départ
- cas 2 : le cas avec une limite de passage chez d'autres producteurs par chemin (limité à 3)

Pour réaliser ces tests, j'ai utilisé le main `main_test_heuristique` avec les fichiers `fichier_donnees_5prod.json`, `fichier_donnees_10prod.json`, `fichier_donnees_20prod.json`, `fichier_donnees_50prod.json` et `fichier_donnees_100prod.json`.

Le premier paramètre du run est le path du fichier et le 2ème le cas que l'on souhaite étudier (0 par défaut, 1 ou 2).

Voici le tableau des résultats avec les pourcentages d'amélioration par rapport à la solution sans mutualisation. Dans les colonnes "Résultats" je ne mets que les 5 premiers résultats pour ne pas encombrer le tableau mais la moyenne et le calcul du temps sont faits sur 100 itérations. Le temps correspond au temps pour trouver la meilleure solution :

	cas 0		cas 1		cas 2	
nb prod	Résultats	Moyenne et temps sur 100	Résultats	Moyenne et temps sur 100	Résultats	Moyenne et temps sur 100
5	33.43 24.29 27.73 25.4 28.86	28.59  0.18 s	0.46 8.11 5.12 9.0 8.18	4.89  0.03 s	32.33 30.36 28.04 29.17 25.68	29.23  0.22 s
10	3.96 6.76 15.07 2.25 5.03	10.43  0.06 s	3.38 1.91 7.36 5.63 6.42	5.64  0.06 s	10.25 10.01 8.3 4.99 5.56	9.82  0.05 s
20	17.19 20.87 21.5 16.32 23.13	18.48  0.06 s	13.87 12.51 9.59 10.46 13.25	11.74  0.15 s	17.55 14.38 18.31 11.55 17.1	17.25  0.03 s
50	28.55 26.51 25.51 28.38 29.37	27.41  0.33 s	16.29 16.1 17.06 18.07 17.19	17.48  0.72 s	22.83 28.36 23.87 24.65 26.07	25.96  0.30 s
100	37.35 37.76 34.72 36.08 35.4	35.78  1.54 s	24.67 23.69 25.04 24.46 23.21	24.25  3.06 s	33.26 33.54 33.36 33.05 33.51	32.98  0.75 s

Figure 5 – Tableau des résultats

On constate donc que l'heuristique dans le cas 0 est très efficace avec 5 producteurs, très peu efficace avec 10 producteurs et son efficacité progresse plus on a de producteurs. Les performances importantes avec 5 producteurs peuvent être remises en cause par le fait qu'il n'y ait pas assez de données pour que l'expérience soit concluante.

On constate que le cas 1 fait baisser énormément les résultats par rapport au cas 0 surtout sur les tests avec moins de producteurs. Le cas 2, lui, n'a qu'un effet minimum sur les résultats avec des baisses de quelques pourcents seulement par rapport au cas 0.

Comme nous pouvons le constater sur ce tableau, les temps pour trouver les meilleures solutions sont extrêmement petits (de l'ordre de la seconde).

En analysant les données, on peut aussi voir qu'un problème se pose. En effet, les résultats ne sont pas stables. Si on regarde les résultats avec 10 producteurs sur le cas 0 par exemple, on peut voir un résultat à 2.25 et un autre à 15.07, ce qui est une énorme variation. L'explication probable est que si les premières décisions de l'heuristique ne peuvent amener à un bon chemin dès le début de son fonctionnement, elle peut avoir du mal à trouver une bonne solution. Comme l'heuristique ne revient jamais en arrière, une mauvaise décision prise au cours de son fonctionnement (dû au hasard par exemple), entraîne un résultat dégradé. Comme notre heuristique fait une descente vers un minimum local, si le minimum local est mauvais elle n'a que très peu de moyens d'en trouver un autre.

## 4 Amélioration

Dans cette partie, nous allons discuter des améliorations réalisables pour perfectionner l'heuristique et le modèle.

### Améliorer le modèle

On peut améliorer le modèle en mettant en place les différentes contraintes ajoutées dans la partie modélisation. On peut aussi ajouter la notion de temps de livraison pour rendre le modèle plus réaliste.

### Améliorer la gestion de la capacité

Si la contrainte de capacité a été mise en place, elle peut être améliorée. En effet, contrairement à un problème de tournées de véhicules classique où la capacité est remplie au maximum au départ et ne fait que réduire au fur et à mesure du chemin en déposant les livraisons chez les clients. Dans notre cas la quantité de produits transportés peut diminuer mais aussi augmenter. En effet, si un producteur passe chez un client, il dépose des marchandises mais s'il passe chez un autre producteur, il en charge. Il faudrait donc modéliser la capacité, non pas par un nombre pour chaque producteur, mais par un tableau qui évolue au fur et à mesure du parcours du producteur. Ceci rend les vérifications pour les insertions et retraits très complexes.

Prenons comme exemple un chemin de producteur (avec une capacité de 65) :  $[P_0, C_{0,1}, C_{0,2}, P_1, C_{1,1}, C_{1,2}, C_{0,3}, C_{0,4}]$ .

Et un tableau symbolisant l'évolution du poids du chargement du producteur en fonction de l'étape de son parcours :  $[50, 45, 30, 65, 45, 24, 14, 0]$ .

Si on essaie d'insérer une liste de clients, par exemple, un point demandant une charge de 16 et un autre de 20. On commence par trouver les endroits où on peut l'insérer. On peut trouver assez facilement que l'on peut l'insérer à toutes les positions sauf entre  $P_0$  et  $C_{0,1}$  et entre  $P_1$  et  $C_{1,1}$ . C'est l'insertion du deuxième point qui va poser problème et compliquer la chose, car il doit être placé autour du premier point en sachant que son insertion va augmenter le poids entre lui et le premier point ajouté. Pour finir, l'insertion du producteur de ces deux points demande de vérifier encore plus de zone afin d'être bien inséré.

### Améliorer l'heuristique

Comme nous avons pu le voir dans la partie précédente, l'heuristique donne en moyenne de très bons résultats en un temps minimal mais est très peu stable. La manière la plus simple pour améliorer cette heuristique serait donc de régler le problème de stabilité.

Pour cela, on pourrait par exemple lancer l'heuristique plusieurs fois pour avoir des résultats différents et sélectionner seulement le meilleur chemin trouvé. Cela nous permet d'avoir plusieurs optimum locaux et de sélectionner le meilleur d'entre eux. Comme notre heuristique est déjà très rapide, la lancer plusieurs fois n'est pas très coûteux.

Une autre manière d'améliorer l'heuristique serait de mettre en place des swaps pour améliorer les solutions. Mais avec toutes les contraintes et notamment le fait que l'on doit passer par le producteur avant de passer par ses clients rendent les swaps compliqués à implémenter et pas forcément très efficace.

Une autre manière d'améliorer l'heuristique serait d'implémenter d'autre type d'insertion ou de retrait comme présenté dans le paragraphe "Recherches" de la partie 3 "Heuristique" du chapitre 3 "Analyse et conception". Par exemple un retrait avec la méthode Shaw et une insertion avec la méthode Regret me paraissent efficaces.

# Conclusion

Ce projet se passe en 2 semestres. Pour le premier semestre je me suis concentré sur la partie recherche et état de l'art du projet. Le semestre 2 a permis de réaliser la partie développements et tests.

## 5 Semestre 1

Pendant ce premier semestre, je me suis concentré sur la partie recherches du projet. J'ai lu de nombreux articles sur les problèmes de tournées de véhicules et ses différentes variantes. J'ai aussi rédigé plusieurs documents aidant à réaliser ce projet comme l'état de l'art, le cahier des spécifications ainsi que ce rapport.

## 6 Semestre 2

Pendant le second semestre, je me suis concentré sur la partie développements et tests. Après des recherches supplémentaires sur les heuristiques sur le problème de tournées de véhicules et une prise en main de l'existant développé par mon encadrant, j'ai développé l'algorithme réalisant l'heuristique. Je l'ai ensuite testé pour voir les résultats obtenus. En conclusion, l'heuristique donne de bons résultats mais il peut être amélioré pour les stabiliser.

## 7 Ajouts et améliorations

En plus des améliorations citées dans la partie amélioration du chapitre précédent, les points suivants pourraient être ajoutés au projet :

- ajouter la gestion de la contrainte de temps
- résoudre le problème sur plusieurs jours pour équilibrer les charges des producteurs
- ajouter une classe pour mieux modéliser les résultats
- ajouter une classe pour visualiser les résultats
- rendre le problème plus réaliste en ajoutant des contraintes comme les cas 1 et 2 le font déjà

## Annexes

# A

## Specifications fonctionnelles

Notre programme se lancera avec une ligne de commande où l'on peut préciser des paramètres comme le nom du fichier contenant les données. Cela lancera :

- une fonction qui s'occupera de parser le fichier donné en entrée ou une fonction de génération de producteurs et livraisons aléatoires si aucun fichier n'ont été donné en paramètre
- une fonction qui calculera la meilleure solution pour chaque producteur sans coopération
- une fonction de clustering
- une fonction qui appliquera l'heuristique sur les données parsées

### 1 Définition de la fonction de passage

Cette fonction s'occupera du passage des informations données par le document. Elle prendra en entrée un path vers le document. Elle ouvrira ce document pour le parser et créer les objets nécessaires à l'analyse à partir de ces informations. Ce document doit contenir toutes les informations sur les producteurs et leurs livraisons. Cela peut être soit un document JSON soit un document XML.

### 2 Définition de la fonction génération de données

Cette fonction s'occupera de générer des producteurs et des livraisons aléatoirement que l'on pourra utiliser pour tester nos algorithmes. Cette fonction se déclenchera par exemple si aucun fichier n'est donné en entrée.

### 3 Définition de la fonction de recherche de résultat sans coopération

Cette fonction s'occupera de calculer le meilleur chemin pour chacun des producteurs sans les faire coopérer ensemble. Ces résultats vont servir à faire une comparaison avec une analyse avec coopération afin d'en déduire le pourcentage d'amélioration. Ces résultats vont aussi servir à calculer le détour maximum accepté par les producteurs. On utilisera une méthode exacte ici.

#### 4 Définition de la fonction de clustering

Cette fonction s'occupera du clustering des points du graphe. Elle prendra en paramètre le nombre de clusters que l'utilisateur souhaite créer. Pour effectuer son clustering, elle prendra en compte la distance entre les producteurs. K-means est un exemple d'algorithme pouvant effectuer cette tâche. Cette fonction fait partie de l'existant.

#### 5 Définition de la fonction de résolution

Cette fonction appliquera l'algorithme de résolution du problème de tournées de véhicules en utilisant une heuristique. Elle retournera tous les résultats qu'elle a trouvés : le poids de la meilleure solution trouvée, la meilleure solution sous forme d'une liste de chemins et le temps pour trouver cette meilleure solution.

# B

## Specifications non fonctionnelles

### 1 Contraintes de développement et conception

L'application va être développée en python et en utilisant l'environnement de développement Pycharm. La bibliothèque CPLEX (DOCPLEX) a été utilisée par mon encadrant pour développer les méthodes exactes. Docplex ne fonctionnant pas sur la version 3.7 de python j'utiliserai la version 3.6 .

### 2 Contraintes de fonctionnement et d'exploitation

#### 2.1 Performances

Le temps de réponse de l'application dépendra de la quantité de données fournies par l'utilisateur. Pour optimiser le temps de réponse de l'application, nous avons privilégié des heuristiques à la place de recherche de solutions exactes.

#### 2.2 Capacités

L'application doit être capable de gérer un grand nombre de données pour son analyse. Même si ces données ne sont individuellement pas lourdes, elles restent nombreuses. Cette capacité dépend aussi du nombre de clusters utilisés car chaque cluster divise le problème en sous-problèmes avec une population plus petite et donc plus facile à résoudre.

#### 2.3 Contrôlabilité

Pour que l'utilisateur puisse suivre le déroulement de l'exécution de l'algorithme, il sera peut-être nécessaire d'ajouter une partie affichage sur l'application. Dans cette partie, des informations sur l'exécution du programme seront affichées au fur et à mesure de celle-ci.

#### 2.4 Sécurité

L'application a un utilisateur unique. Comme il n'existe pas de version en ligne, il n'y a donc pas de sécurité nécessaire. De plus, aucune demande à ce sujet n'a été faite de la part de la MOA.

### 3 Maintenance et évolution du système

Dans une optique de future reprise du code pour une amélioration ou maintenance, celui-ci sera commenté et documenté. En effet, si un client est trouvé plus tard pour ce projet il pourrait être modifié. La documentation sera générée avec l'outil Sphinx en utilisant les commentaires de fonctions et de classes écrit dans le code.

# C

## Gestion de projet

Pour ma gestion j'ai réalisé les choses suivantes :

- mise en place d'un Trello afin de gérer toutes les tâches et leurs dates limite
- création d'un Gantt afin de réaliser un planning
- création d'un git pour le contrôle de versions
- différents documents pour noter les comptes rendus de réunions ou marquer les décisions prises durant le projet

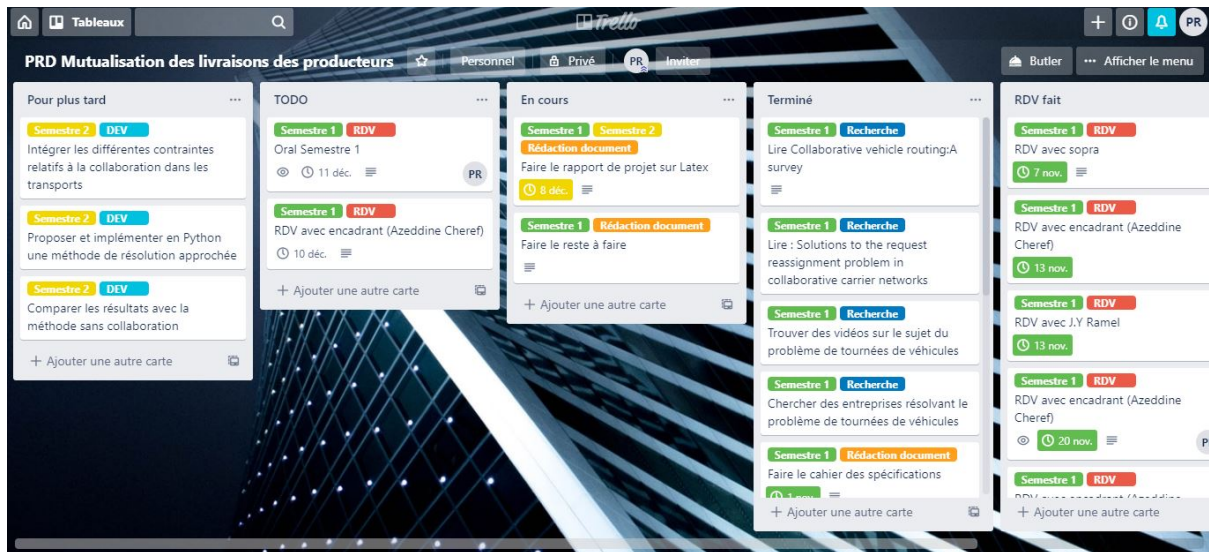


Figure 1 – Trello de ce projet

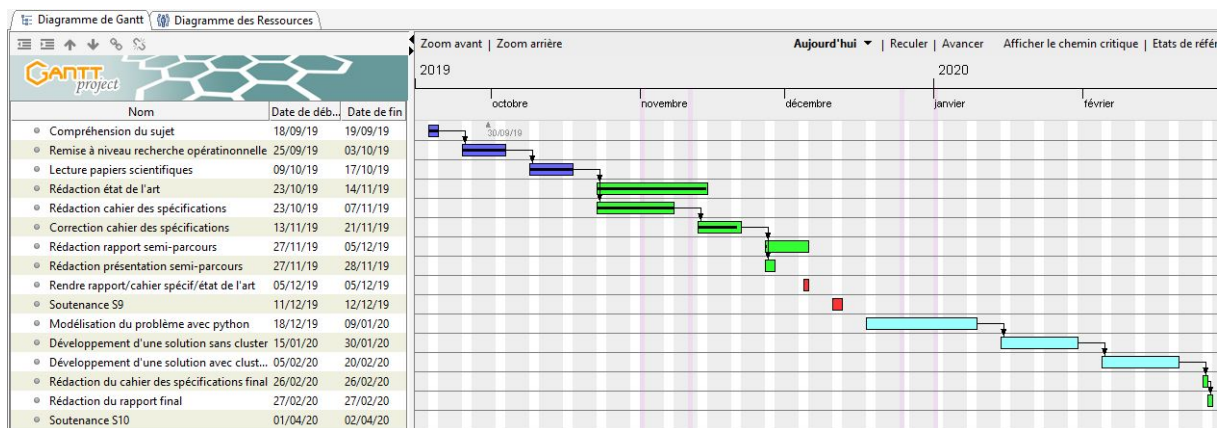


Figure 2 – Gantt de ce projet

# D

## Compte rendu des réunions

Ce document a pour objectif de faire un récapitulatif succinct des différentes réunions que j'ai eu avec mon encadrant afin de garder une trace des décisions prises.

### **Réunion 19/09 : première réunion**

présentation plus en détail du sujet  
2 documents à lire donnés par mon encadrant

### **Réunion 03/10 : deuxième réunion**

plus de détails sur le projet : il se divise en 2 parties, avec et sans clusters

### **Réunion 23/10 : troisième réunion**

précisions sur l'état de l'art (plan)

### **Réunion 13/11 : quatrième réunion**

Dans cette réunion nous avons parlé de :

- réaliser ce projet dans un but de recherche
- j'utiliserai des heuristiques pour trouver la meilleure solution sur des clusters
- j'utiliserai potentiellement l'interface réalisée par le projet de 4a
- si je trouve un solveur intégrable dans le code je peux l'utiliser pour me concentrer sur tout ce qu'il y a "autour"

Comme nous en avons discuté durant cette réunion, je vais maintenant me concentrer sur l'état de l'art et trouver 2-3 exemples d'entreprises utilisant des outils ou applications résolvant le problème de tournées de véhicules pour leurs activités.

Je lui ai envoyé mon gantt

### **Réunion 20/11 : cinquième réunion**

Discussion sur l'état de l'art

Mise en place d'une réunion la semaine prochaine où je vais rendre mon état de l'art

Discussion sur la manière de résoudre le problème

### **Réunion 27/11 : sixième réunion**

Relecture et correction de l'état de l'art

### **Réunion 04/12 : septième réunion**

Validation de la version finale de l'état de l'art

Relecture et correction du cahier des spécifications

**Réunion 10/12 : septième réunion**

Présentation orale blanche

**Réunion 23/01 : huitième réunion**

Discussion et explication sur la modélisation du problème

**Réunion 30/01 : neuvième réunion**

Correction du modèle mathématique réalisé

Début des discussions sur l'heuristique à réaliser

**Réunion 05/02 : dixième réunion**

Discussions sur les heuristiques à utiliser

Sélection de quelques papiers : voir "heuristiques"

Discussions sur les bornes que l'on peut ajouter à l'algorithme (cercle autour de l'utilisateur correspondant à sa distance mono prod \* son détour max)

**Réunion 13/02 : onzième réunion**

Présentation des heuristiques recherchées

Proposition d'une première heuristique à implémenter pour la prochaine réunion

**Réunion 20/02 : douzième réunion**

Discussion sur la première heuristique à réaliser

Premier jet pour l'algorithme de cette heuristique

**Réunion 05/03 : treizième réunion**

Lecture de l'algorithme réalisé

Discussion sur les contraintes (fenêtre de temps, capacité, détour max) sur comment les implémenter : nous avons décidé que les implémenter n'était pas prioritaire (voir la partie amélioration dans le rapport pour voir comment nous avons pensé pouvoir réaliser l'implémentation)

Pour la prochaine réunion :

- finir l'algorithme (faire une boucle)
- plus commenter le code
- mettre en place un fichier de données (générées par le code envoyé par mail)

**Réunion 12/03 : treizième réunion**

Discussion sur des cas plus "réel", nous avons résumé le problème en trois cas :

- le cas neutre, sans changements supplémentaires
- le cas avec des livraisons "urgentes" qui doivent être réalisées par leur producteur de départ
- le cas avec une limite de passage de chez d'autres producteurs par chemin (limité à 3)

Pour la prochaine réunion :

- écrire les résultats sur un fichier (dans un dictionnaire que l'on écrit sur le fichier JSON)
- trouver un bon nombre d'itérations en testant
- faire la visualisation
- faire des scripts pour lancer plusieurs fois un fichier et sortir tous les résultats
- faire un script pour comparer heuristique avec cluster (seulement pour 3 et 5 producteurs)

Les tests à faire :

- sur 5 producteurs (généré) faire la comparaison entre la méthode exacte avec cluster et l'heuristique sans cluster (faire plusieurs tests et faire une moyenne)
- le cas 1 sur 5 - 10 - 20 - 50 - 100 prod (faire plusieurs tests et faire une moyenne)
- le cas 2 sur 5 - 10 - 20 - 50 - 100 prod (faire plusieurs tests et faire une moyenne)
- le cas 3 sur 5 - 10 - 20 - 50 - 100 prod (faire plusieurs tests et faire une moyenne)

**Réunion 26/03 : quatorzième réunion (sur teams)**

Discussion sur le cahier des tests réalisé

Validation de l'ensemble du code

Début de la rédaction du rapport final

# E

## Bibliographie

- Margaretha Gansterer, Richard F. Hartl (2018). Collaborative vehicle routing: A survey. *European Journal of Operational Research*, 268, 1–12
- Liu R., Jiang Z., Fung R. Y., Chen F., & Liu X. (2010). Two-phase heuristic algorithms for full truckloads multi-depot capacitated vehicle routing problem in carrier collaboration. *Computers & Operations Research*, 37(5), 950–959.
- Savelsbergh, Martin & Sol, Massi (1995). The General Pickup and Delivery Problem
- Marshall Fisher (1995). Chapter 1 Vehicle routing. *Handbooks in Operations Research and Management Science*, 8, 1-33
- Andrés Muñoz-Villamizar, Jairo R., Montoya-Torres, Carlos A. Vega-Mejía (2015). Non-Collaborative versus Collaborative Last-Mile Delivery in Urban Systems with Stochastic Demands. *Procedia CIRP*, 30, 263-268.
- Buijs P., Alvarez J. A. L., Veenstra M. & Roodbergen K. J. (2016). Improved collaborative transport planning at Dutch logistics service provider Fritom. *Interfaces*, 46(2), 119–132
- I-Ming Chao, Bruce L. Golden, Edward A. Wasil (1996). The team orienteering problem. *European Journal of Operational Research*, 88(3), 464-474
- Dai B. & Chen H. (2012). Mathematical model and solution approach for carriers collaborative transportation planning in less than truckload transportation. *International Journal of Advanced Operations Management*, 4, 62–84
- Krajewska M. & Kopfer H. (2006). Collaborating freight forwarding enterprises. *OR Spectrum*, 28(3), 301–317
- Vanovermeire C. & Sörensen K. (2014). Integration of the cost allocation in the optimization of collaborative bundling. *Transportation Research Part E: Logistics and Transportation Review*, 72, 125–143
- Özener O., Ergun Ö., & Savelsbergh M. (2013). Allocating cost of service to customers in inventory routing. *Operations Research*, 61(1), 112–125
- Guajardo M. & Jörnsten K. (2015). Common mistakes in computing the nucleolus. *European Journal of Operational Research*, 241(3), 931–935
- geoconcept.com
- toursolver.com
- ptvgroup.com
- Brochure PTV RouteOptimizer
- Vidal T., Crainic T. G., Gendreau M. & Prins C. (2011). Heuristiques pour les problèmes de tournées de véhicules multi-attributs
- Li H. & Lim A. (2001). A Metaheuristic for the Pickup and Delivery Problem with Time Windows

in International Journal of Artificial Intelligence Tools 12(02):160-167

Ropke S. & Pisinger D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows in Transportation Science 40(4):455-472

# Mutualisation des livraisons des producteurs

## Résumé

L'objectif de ce projet est de réaliser une application résolvant un problème de tournées de véhicules en collaboration afin de minimiser les dépenses liées aux livraisons pour les producteurs agricoles. Le problème de tournées de véhicules est un problème d'optimisation où on doit faire parcourir tous les points d'un graphe à plusieurs véhicules. L'objectif étant de satisfaire les producteurs il faut donc que les nouvelles livraisons ne soient pas trop longues par rapport aux anciennes. Ces calculs sont réalisés grâce à des heuristiques résolvant le problème en suivant les diverses contraintes.

## Mots-clés

producteur, collaboration, optimisation, tournées de véhicules, heuristique

## Abstract

The goal of this project is to develop an application that is capable of resolving a collaborative vehicle routing problem for agricultural producers. It needs to create solutions that fit the need of the producers and doesn't make them do a huge detour. Those problems are solved using heuristic that compute the solution and follow the multiple constraints.

## Keywords

producer, collaboration, optimization, vehicle routing problem, heuristic