

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement
2017-2018

Livraison multicritères de produits de chimiothérapie

Tuteur académique
Alexis ROBBES

Étudiant
Benoît VACHER (DI5)

2 avril 2019



Liste des intervenants

Nom	Email	Qualité
Benoît VACHER	benoit.vacher@etu.univ-tours.fr	Étudiant DI5
Alexis ROBBES	alexis.robbes@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Benoît Vacher susnommé l'auteur.

L'Ecole Polytechnique de l'Université de Tours est représentée par Alexis Robbes 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 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

Benoît Vacher, *Livraison multicritères de produits de chimiothérapie*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Vacher, Benoît},
  title={Livraison multicritères de produits de chimiothérapie},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Liste des tableaux	vi
1 Introduction	1
1 Acteurs, enjeux et contexte	1
2 Objectifs	1
3 Bases méthodologiques	2
2 Description Générale	3
1 Environnement du projet	3
2 Caractéristiques des utilisateurs	3
3 Fonctionnalités du système	3
3.1 Structure générale du système	4
3 Etat de l'art	6
1 Mise en lot.....	6
1.1 Présentation des algorithmes	6
1.2 K-Means	6
1.3 Agglomerative.....	7

1.4	Autres	8
2	Routing de la livraison	8
3	Workshop RAMOO 2018.....	9
4	Analyse et conception	11
1	Analyse du problème de livraison	11
2	Modélisation du problème de livraison	11
3	Analayse des algorithmes de partitionnement	12
3.1	Analyse de l'algorithme agglomératif	13
3.1.1	Single	13
3.1.2	Average	14
3.1.3	Ward.....	15
3.2	Comparaison K-Means / Agglomératif	15
4	Représentation des données	16
5	Mise en oeuvre	20
1	Outils et librairies.....	20
2	Implémentation	20
2.1	Structure	20
2.2	Mise en lots	21
2.3	Livraison	21
2.4	Interface	21
2.4.1	Mise en lots	22
2.4.2	Chemins pour la livraison	22
2.4.3	Livraison	23
3	Resultats.....	23
4	Qualité	24
4.1	Vérificateur de solutions.....	24
4.2	Tests.....	25
4.3	Outils	25
6	Conclusion	26
1	Planning.....	26
2	Qualité	27
3	Ce qui fonctionne	27
4	Possibilités d'améliorations	28
5	Conclusion personnelle	28

Annexes	29
A Spécifications	30
1 Contexte de la réalisation.....	30
1.1 Objectifs	30
1.2 Bases méthodologiques.....	31
2 Description générale.....	31
2.1 Environnement du projet	31
2.2 Caractéristiques des utilisateurs	31
2.3 Fonctionnalités du système	32
2.4 Structure générale du système	33
3 Description des interfaces externes du logiciel	33
3.1 Interfaces matériel/logiciel	33
3.2 Interfaces homme/machine.....	33
3.3 Interfaces logiciel/logiciel	34
4 Spécifications fonctionnelles.....	34
4.1 Optimisation de la mise en lots	35
4.2 Optimisation de la livraison.....	35
4.3 Affichage des résultats.....	36
5 Spécifications non fonctionnelles.....	36
5.1 Contraintes de développement et conception.....	36
5.2 Diagramme de séquence.....	36
5.3 Performances.....	37
5.4 Capacités	37
5.5 Contrôlabilité	38
5.6 Sécurité	38
5.7 Intégrité.....	38
B Gestion de projet	39
C Fiches de tests	41
D Documentations	42
Bibliographie	51

Table des figures

1 Introduction

1	Cycle en v.....	2
---	-----------------	---

2 Description Générale

1	Diagramme de cas d'utilisation	4
2	Diagramme de composants	5

3 Etat de l'art

1	Dendrogramme de 13 points équidistants.....	7
2	Deux façons de choisir les lots avec l'agglomerative clustering	7
3	Problème de TSP.....	8
4	Modélisation du TSP [2].....	9

4 Analyse et conception

1	Linkage : single.....	14
2	Linkage : average	14
3	Linkage : ward	15
4	Diagramme en boites algorithmes de partitionnement.....	16
5	Représentation 3D des produits.....	17
6	Représentation 2D des produits - x/y.....	18
7	Représentation 2D des produits - x/date due.....	18
8	Représentation 2D des produits - y/date due.....	19

5 Mise en oeuvre

1	Interface	22
2	Interface - mise en lots	22
3	Interface - chemins pour la livraison	23
4	Interface - livraison	23

6 Conclusion

1	Gantt S9	26
2	Gantt S10	27

A Spécifications

1	Cycle en v	31
2	Diagramme de cas d'utilisation	32
3	Diagramme de composants	33
4	Fonctions du livrable	34
5	Diagramme de séquence du livrable	37

B Gestion de projet

1	Gantt S9	39
2	Gantt S10	40



Liste des tableaux

C Fiches de tests

1	Test 1 du Validateur de solution	41
2	Test 2 du Validateur de solution	41

1

Introduction

1 Acteurs, enjeux et contexte

À Tours, il y a 3 hopitaux qui sont concernés par la livraison de produits de chimiothérapie : le CHU de Bretonneau, de Trousseau, et de Clocheville. C'est le CHU de Bretonneau qui livre ces produits. Il est nécessaire d'être le plus efficace dans cette livraison, car les produits de chimiothérapie ont une courte durée de vie. Les différentes étapes pour la production/livraison sont les suivantes :

- Prescription par un service de cancérologie
- Validation
- Stérilisation par l'URCC
- Préparation par l'URCC
- Vérification (contrôle analytique) par l'URCC
- Livraison
- Administration par un service de cancérologie

URCC = Unité de Reconstitution Centralisée des Cytotoxiques

Les produits ont un temps de stabilité limité lié aux éléments chimiques utilisés, il est donc nécessaire de gérer au mieux la livraison pour qu'ils soient utilisés dans les délais. Pour optimiser la livraison il faut : optimiser la mise en lots des produits pour que les produits qui doivent arriver au même endroit et/ou au même moment soient envoyés ensemble, et optimiser les tournées des livreurs pour rendre leurs déplacements le plus efficace possible. Etant donné que la vie de patients est en jeu, le but est limiter les retards.

2 Objectifs

Ce projet vise à répondre à un besoin d'optimisation de la logistique en proposant une application d'aide à la décision pour la mise en lot et la définition des tournées des livreurs.

3 Bases méthodologiques

1. Les outils

- GitLab
- GanttProject

La gestion de projet a été effectuée suivant un modèle de cycle en V.

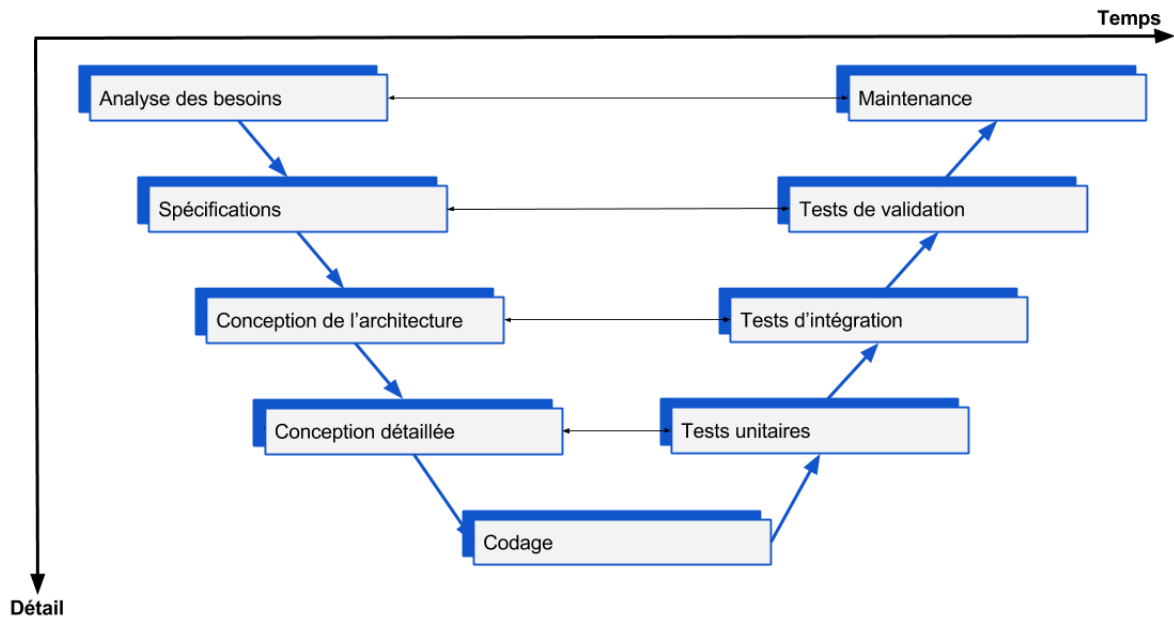


Figure 1 – Cycle en v

2

Description Générale

1 Environnement du projet

Ce projet vient en complément du sujet de thèse d'Alexis ROBBES. Seuls les données testées sont fournies. Le reste du projet est entièrement indépendant, et aucun existant n'a été fourni. Le produit sera fonctionnel sur Windows, et le tout devra être disponible gratuitement (donc utilisation de logiciels/bibliothèques/etc. gratuits). En plus d'être gratuit, le code sera open source.

2 Caractéristiques des utilisateurs

Il n'y aura qu'un client, Alexis ROBBES, qui possède les caractéristiques suivantes :

- Connaissance approfondie du sujet
- Connaissances en informatique

Le client aura accès à la totalité du travail rendu, autant du produit de vue interface, qu'à l'intégralité du code source réalisé. Il aura donc la possibilité de porter des modifications sur n'importe quel composant.

3 Fonctionnalités du système

L'utilisateur doit avoir la possibilité de choisir les jeux de données qu'il souhaite analyser, que ces données soient traitées, puis que différents graphiques permettant d'expliquer la/les solution(s) soient affichés. Des résultats seront sauvegardés sur des fichiers.

Pour la partie livraison, l'utilisateur doit attribuer un livreur pour chaque lot. Ensuite, pour chaque livreur, il doit choisir parmi un ensemble de solutions comment se fera le trajet du premier lot. Ce choix influera sur les solutions disponibles pour le trajet du deuxième lot, etc. En effet, il est possible que le premier trajet se fasse vite au dépend du retard d'un seul produit, ou que le premier trajet ne soit plus long mais qu'il livre tous les produits en avance. En fonction des cas il se peut que le deuxième trajet parte avec plus ou moins de retard, influant sur les solutions envisageables.



Figure 1 – Diagramme de cas d'utilisation

3.1 Structure générale du système

L'utilisateur aura pour rendu un système capable de lui afficher, pour un problème donné, un ensemble de graphiques/d'indicateurs sur la/les solution(s) envisageable(s). Etant donné que l'optimisation de la mise en lot est réalisée en Python, et que l'optimisation de la livraison est réalisée en Julia, ces deux parties ont leur propre interface/affichage. Il est à noter que l'on peut appeler du code Python depuis Julia. Il est donc possible que le livrable final combine les deux parties en un seul logiciel.

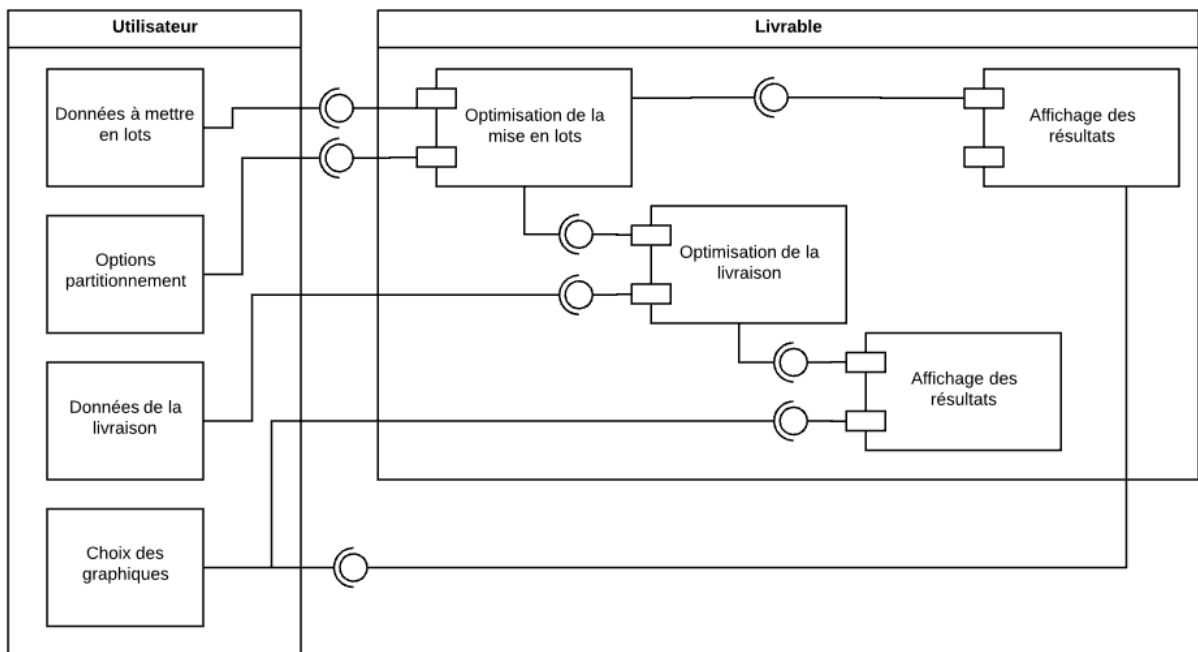


Figure 2 – Diagramme de composants

3

Etat de l'art

1 Mise en lot

Afin d'être livrés, les produits doivent être mis en lots. Chaque lot doit contenir des produits les plus "liés" possible. On entend par lié que ce sont des produits proches d'un point de vue géographique (livrés au même hôpital par exemple), ou proche en terme d'heure de livraison (produits qui doivent être livrés environ au même moment). Afin de définir ces lots, il est possible d'utiliser différents algorithmes. Nous allons étudier ces algorithmes afin de définir lesquels sont les plus intéressants dans notre cas.

1.1 Présentation des algorithmes

Il existe un large panel d'algorithmes permettant de résoudre notre problème. Cependant, nous nous focaliserons sur les deux algorithmes suivants :

- K-Means
- Agglomerative

Ces algorithmes utilisent des techniques différentes, et donc nous apportent des résultats différents.

Voici une courte explication de comment chacun fonctionne.

1.2 K-Means

Le principe est le suivant : l'algorithme pose un nombre k de repères dans le nuage de points, où k est le nombre de lots souhaités. Chaque point du nuage de points va être associé au repère le plus proche, créant des sous-ensembles. Une fois que tous les points sont attribués, les repères sont déplacés pour se situer sur le centre de leur sous-ensemble. Les opérations sont répétées jusqu'à ce que plus aucun repère ne bouge.

Cet algorithme est rapide, mais on peut se retrouver avec des lots de tailles très variables. De plus, il est semi-aléatoire, car les repères originaux sont placés aléatoirement. Enfin, il est nécessaire de spécifier le nombre de lots voulus avant de lancer l'algorithme.

1.3 Agglomerative

L'agglomerative clustering est un algorithme de la famille des algorithmes hiérarchiques. Le principe est de lier les deux points les plus proches du nuage pour créer un lot. On ne va plus prendre en compte la distance entre ces deux points lors des prochaines itérations. On recommence ensuite la procédure jusqu'à ce qu'on obtienne un seul lot contenant tous les points du nuage. On obtient ensuite un graphique nommé dendrogramme, qui nous permet de savoir comment créer les lots en fonction de ce que l'on souhaite.

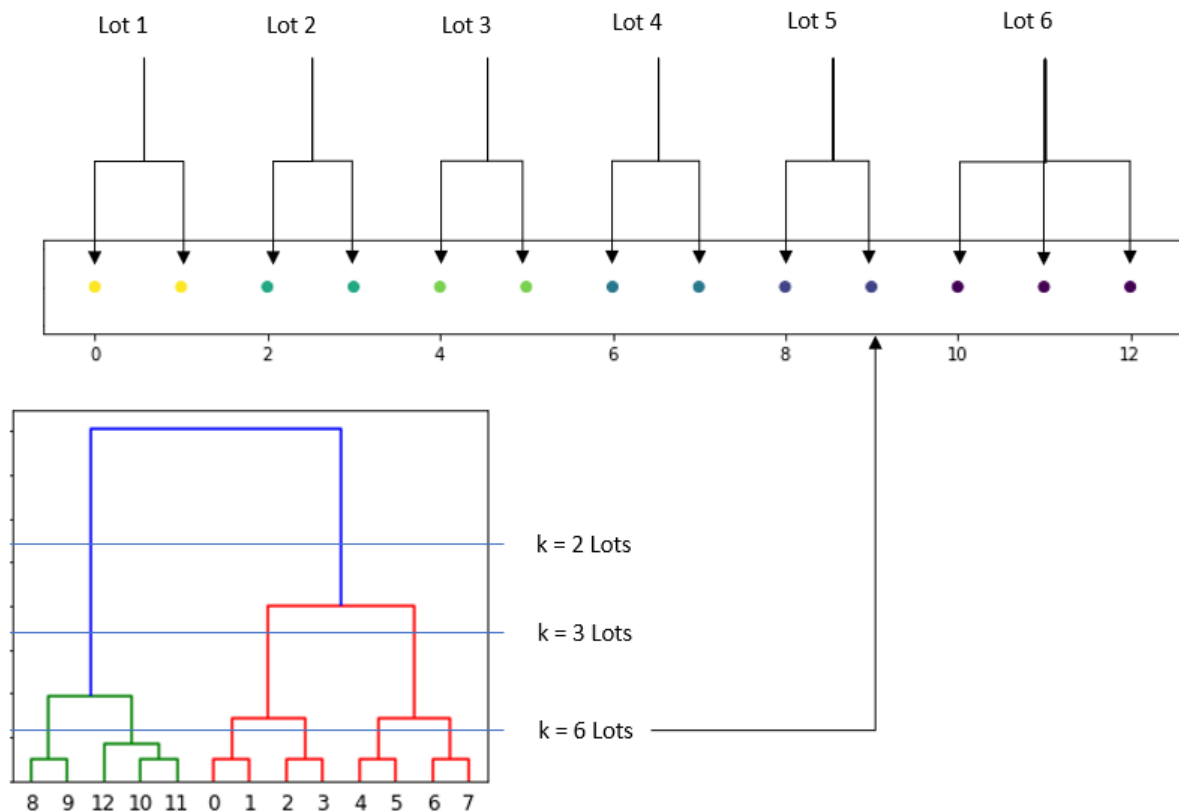


Figure 1 – Dendrogramme de 13 points équidistants

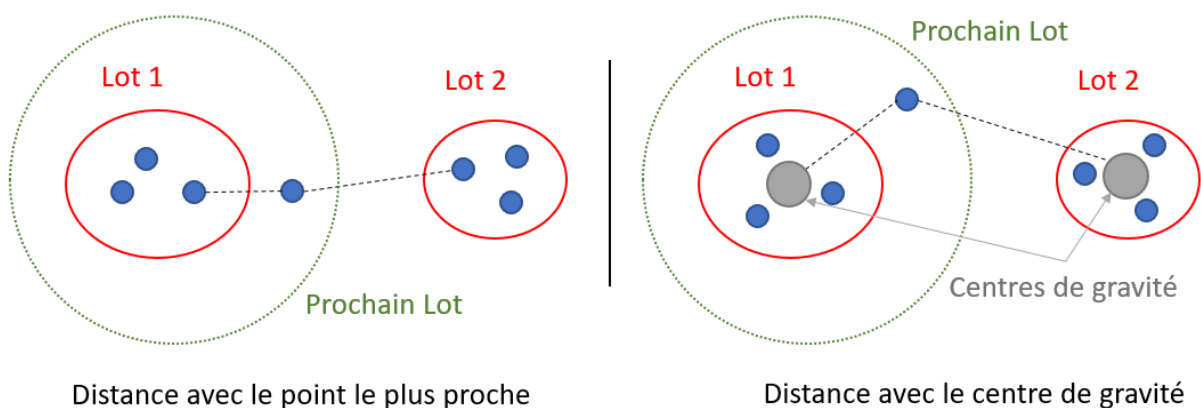


Figure 2 – Deux façons de choisir les lots avec l'agglomerative clustering

Il est possible de paramétrer cet algorithme de beaucoup de façons. Il est notamment nécessaire de choisir comment sont calculées les distance, entre autre si la distance entre un point et un lot est :

- la distance entre ce point et le point du lot le plus proche
- la distance entre ce point et le point du lot le plus loin
- la distance entre ce point et le centre de gravité du lot
- etc.

Dans notre exemple nous avons utilisé le centre de gravité. Nous analyserons les résultats de chaque méthode dans la partie analyse.

1.4 Autres

Il existe beaucoup d'autres algorithmes/familles d'algorithmes qui pourraient être utilisés, dont voici une liste non exhaustive [3] [1] :

- Spectral
- Affinity propagation
- DBSCAN
- Gaussian
- SOM

Tous ces algorithmes peuvent être utilisés, mais il n'est pas possible de tous les étudier, il est donc nécessaire de faire un choix. Ces algorithmes possèdent certaines caractéristiques qui font qu'ils n'ont pas été étudiés : impossibilité de choisir le nombre de lots à obtenir, lots finaux ayant des tailles trop variables, fonctionnels pour des problèmes spécifiques, etc.

2 Routing de la livraison

Une fois que les produits ont été mis en lot, il faut livrer chaque lot. Ce problème est un problème de TSP : "Travelling Salesman Problem". Un TSP standard est constitué comme suit :

- Données : un ensemble de points à relier
- Objectif : relier tous les points afin de faire le trajet le plus court
- Contraintes : il y a un unique trajet qui contient tous les points, et on ne passe qu'une seule fois sur chaque point

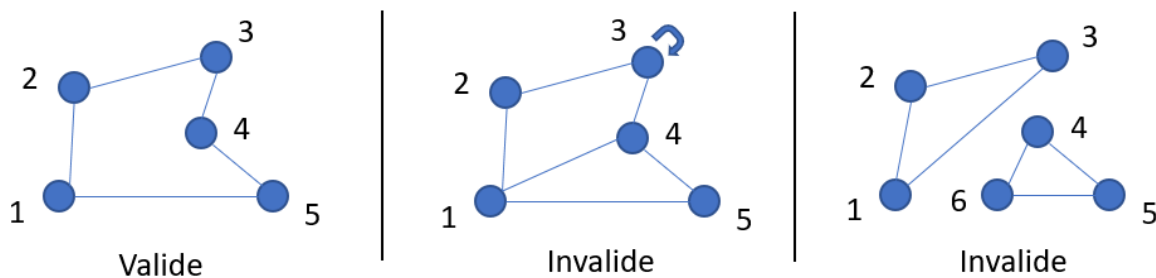


Figure 3 – Problème de TSP

Sur l'image précédente, le trajet de gauche est valide : tous les points ne sont traversés qu'une fois. Sur l'exemple du milieu, il y a 2 problèmes : les points 1 et 4 sont connectés à plus de 2 autres points, et un sous-tour unitaire (le point 3 pointe sur lui-même). L'exemple de droite n'est pas valide car il présente 2 boucles indépendantes, ne respectant pas la contrainte d'un unique trajet contenant tous les points.

Une des représentations les plus simples du problème est la suivante.

Minimize

$$K = \sum_{i,j \in V} c_{ij} x_{ij}$$

subject to

$$\begin{cases} \sum_{i \in V} x_{ij} = 1 & (j \in V \setminus \{0\}), \\ \sum_{j \in V} x_{ij} = 1 & (i \in V \setminus \{0\}), \\ \sum_{i,j \in U} x_{ij} \leq |U| - 1 & (U \subset V), U \neq \emptyset, \\ x_{ij} \in \{0, 1\} & (i, j \in V). \end{cases}$$

Figure 4 – Modélisation du TSP [2]

Dans cette représentation, nous avons :

- c_{ij} : distance entre 2 points
- x_{ij} : =1 si les points sont reliés, =0 sinon
- V : l'ensemble des points

Nous voyons sur cette modélisation que nous souhaitons minimiser le chemin parcourant les points. Les deux premières contraintes nous assurent que, pour chaque point, il n'y a qu'un arc entrant, et qu'un arc sortant. La troisième contrainte permet de spécifier qu'il ne doit y avoir qu'un seul chemin parcourant les points (U est un sous-ensemble de V).

Cette modélisation est très basique, et elle doit être modifiée afin d'être adaptée à notre problème. Notre modification majeure est que nous résolvons un problème bi-objectif, c'est à dire que nous ne minimons pas qu'un, mais deux objectifs. La modélisation du TSP appliquée à notre problème est disponible dans la partie analyse du rapport.

3 Workshop RAMOO 2018

À l'occasion de ce PRD, l'étudiant et son encadrant ont assisté à un workshop nommé "Recent Advances in Multi-Objective Optimization" (RAMOO) [4], qui se tenait dans les locaux de l'université de Nantes. Des chercheurs du domaine ont apporté leur expertise sur les récentes avancées, et sur un ensemble de points propres à la recherche opérationnelle multi-objectifs. Etant donné que ce PRD tente de résoudre un problème bi-objectif, ce workshop avait un intérêt, notamment pour avoir une vue de ce qui est faisable, fait et à faire dans le domaine. Voici une liste non-exhaustive de certaines conférences qui se sont déroulées et des points qui sont ressortis :

- "Quantifying the hardness of the enumeration of Pareto optima : a theoretical framework with application to scheduling problems" - Vincent T'kindt [6]
 - Il est possible de quantifier le pire cas d'un problème NP-hard en utilisant des algorithmes à temps exponentiel

- Outre les méthodes de résolution de problème d'optimisation combinatoire que sont le Branch-and-Bound et le Branch-and-Check, il est possible d'utiliser un Branch-and-Reduce
- Il est possible d'utiliser la technique du Sort & Search, qui consiste à trier nos données afin d'optimiser notre recherche de solution
- "Duty Rostering for Physicians at a Department of Orthopedics and Trauma Surgery : Decision Support using Mathematical Optimization" - Clemens Thielen [5]
 - Problème de répartition en groupe du personnel chirurgical d'un hôpital : obtenir une répartition en respectant au maximum les souhaits de chacun (jours de congés, heures de travail, etc.)
 - Ici le choix a été d'utiliser une résolution mono-objectif (utilisation de sommes pondérées pour ne pas résoudre en multi-objectifs). Néanmoins, comme l'a spécifié Clemens Thielen, il est possible de passer en multi-objectif (ce qui va être fait)
 - En effet, certaines contraintes peuvent être vues comme un objectif qu'il est impératif de ne pas dépasser
 - En passant du triage manuel à l'utilisation d'algorithmes d'optimisation, le temps d'assignement a été très grandement amélioré, et il n'y a presque plus, voir plus aucun, souhaits non respectés

4

Analyse et conception

1 Analyse du problème de livraison

Chaque produit est groupé dans un lot. Nous obtenons donc un ensemble de lots qu'il faut livrer. Le problème qui suit est appliqué à un lot, il faut donc exécuter ceci pour chaque lot.

Nous avons un nombre n_j de produits à livrer. Chaque produit doit être livré à une heure donnée d_i , et ne doit pas dépasser une certaine heure limite \tilde{d}_i . Il y a n_s services, et chaque produit est lié à l'un de ces services.

Nous voulons que les produits soient livrés en optimisant l'heure de retour à l'hôpital de fabrication des produits (donc en optimisant le temps de trajet), et en optimisant le retard cumulé de chaque produit.

En réalisant ce problème en bi-objectif, notre livrable d'aide à la décision va permettre à l'utilisateur de varier les solutions qu'il souhaite pour chaque trajet. S'il sait que le lot actuel doit arriver vite car le lot suivant doit livrer des produits qui atteignent bientôt leur durée de vie, il va pouvoir choisir de favoriser l'heure de retour sur le trajet actuel.

2 Modélisation du problème de livraison

Si l'on suit notre analyse, et que nous utilisons les contraintes nécessaires, nous obtenons le modèle mathématique suivant :

Paramètres

n_s : nombre de services

n_j : nombre de produits à livrer

$I = \{0, \dots, n_j\}$: l'ensemble de tous les produits à livrer

$S = \{0, \dots, n_s\}$: l'ensemble de tous les services

$p_{ss'}$: distance entre les services s et s'

d_i : heure dû du produit i

\tilde{d}_i : heure limite du produit i

α_0 : heure de départ du véhicule

UB : borne maximum du temps nécessaire pour livrer tous les produits

Variables

$$x_{ss'} = \begin{cases} 1 & \text{si les services } s \text{ et } s' \text{ sont reliés} \\ 0 & \text{sinon} \end{cases}$$

C_s : heure d'arrivée au service s

T_i : retard pour le produit i

Objectifs

$$\min \sum_{i \in I \setminus \{0\}} T_i \quad (1)$$

$$\min C_0 \quad (2)$$

Contraintes

$$\sum_{s \in S} x_{ss'} = 1 \quad \forall s' \in S \quad (3)$$

$$\sum_{s' \in S} x_{ss'} = 1 \quad \forall s \in S \quad (4)$$

$$C_s \geq C_{s'} + p_{ss'} - UB(1 - x_{ss'}) \quad \forall (s, s') \in S \times S \setminus \{0\} \quad (5)$$

$$T_i \geq C_{s_i} - d_i \quad \forall (i, s) \in I \setminus \{0\} \times S \quad (6)$$

$$\tilde{d}_i \geq C_{s_i} \quad \forall (i, s) \in I \setminus \{0\} \times S \quad (7)$$

$$C_0 \leq UB \quad (8)$$

$$C_s \geq \alpha_0 + p_{0s} \quad \forall s \in S \setminus \{0\} \quad (9)$$

$$C_s \leq C_0 - p_{s0} \quad \forall s \in S \setminus \{0\} \quad (10)$$

$$x_{ss'} \in \{0, 1\}, C_s \in \mathbb{R}^+, T_s \in \mathbb{R}^+ \quad \forall (s, s') \in S \times S \setminus \{0\} \quad (11)$$

(3) Contrainte de flux entrant

(4) Contrainte de flux sortant

(5) L'heure d'arrivée en s est égale à l'heure d'arrivée au service précédent + le temps $s \rightarrow s'$. Elimine également les sous-tours de plus de 2 éléments

(6) Le retard est la différence heure prévue, heure réelle

(7) Obligation d'arriver avant l'heure limite

(8) On revient au service 0 avant la borne maximum

(9) On ne peut pas arriver avant de partir du service 0

(10) On arrive à un service avant l'heure de retour - le temps de trajet $s \rightarrow s'$

(11) $x_{ss'}$ est une variable booléenne, C_s et T_s sont des variables réelles positives

Les contraintes (8) et (10) ne sont pas obligatoires, en revanche elles nous permettent d'affiner notre recherche en diminuant la taille de l'espace des solutions. Ce sont des inégalités valides.

3 Analyse des algorithmes de partitionnement

Comme énoncé plus tôt, nous allons analyser 2 algorithmes :

- K-Means
- Agglomerative

Avant de comparer les algorithmes, nous allons étudier comment configurer l'algorithme agglomératif afin d'obtenir les résultats qui nous conviennent le plus.

3.1 Analyse de l'algorithme agglomératif

Nous avons énoncé plus tôt que cet algorithme peut se paramétrer, notamment pour définir comment est calculé la distance entre différentes entités (produit/lot ou lot/lot). Nous travaillons en Python, avec la bibliothèque sklearn [1]. Cette bibliothèque nous fournit la fonction suivante :

```
AgglomerativeClustering(n_clusters, affinity, memory, connectivity,
                        compute_full_tree, linkage)
```

Nous allons garder certains paramètres avec la valeur par défaut. Néanmoins nous allons paramétrer :

- `n_clusters` : le nombre de lots voulus
- `linkage` : comment est calculée la distance lot/lot

Le linkage ici est la partie la plus importante. Nous allons tester 3 types de liaisons :

- `single` : prend la distance la plus petite entre les produits des 2 lots
- `average` : prend la distance entre les centres de gravité de chaque lot
- `ward` : regroupe les 2 lots pour lesquels le lot résultant aura la variance la plus faible (distance entre les produits du lot la plus faible)

Enfin, pour tester l'algorithme nous utilisons 4 jeux de données. Le but de ces jeux de données est de pouvoir observer, dans des cas spécifiques, si le l'algorithme configuré par notre paramètre créé des lots cohérents avec nos attentes. Dans la majorité des cas, nous souhaitons des lots dont les produits les plus distants sont le plus proche possible. En effet, nous voulons éviter d'avoir des lots dont les points sont très loin les uns des autres.

Pour ces tests, nous étudions des cas dans 1 dimension. Il y a 13 produits, et nous souhaitons obtenir 6 lots. Notre axe des abscisses est notre représentation temporelle : heure de fin de production.

Voici les jeux de données :

- Valeurs suivant la suite de Fibonacci
- Ecart entre les produits de 1
- Ecart entre les produits augmentant de 1 à chaque nouveau produit
- Produits placés aléatoirement

3.1.1 Single

Ici, les lots liés sont ceux dont la distance des 2 produits les plus proches est minimale.

En mode single, nous observons pour les produits "+1" un lot de grande taille, puis une succession de lots de taille 1. En effet, avec cette méthode, et étant donné que les produits sont équidistants, l'algorithme va suivre ce cheminement :

- Les produits sont équidistants, on prend donc la première paire : 0-1 -> lot1
- La distance entre le lot1 et le produit 2 = distance entre 1 et 2
- Donc les produits et le lot sont toujours équidistants, on prend la première paire : lot1-2 -> lot2
- etc.

Enfait, les produits vont être rajoutés dans l'ordre dans le lot, puis, lorsqu'il restera juste assez de produits seuls pour avoir notre nombre de lots, tous les produits restants restent seuls. On peut remarquer que dans le cas de "Fibonacci" et de "+n" nous sommes face au même problème : un groupe de tous les produits possibles, puis les derniers produits seuls. Avec des produits aléatoires, l'algorithme fonctionne comme voulu.

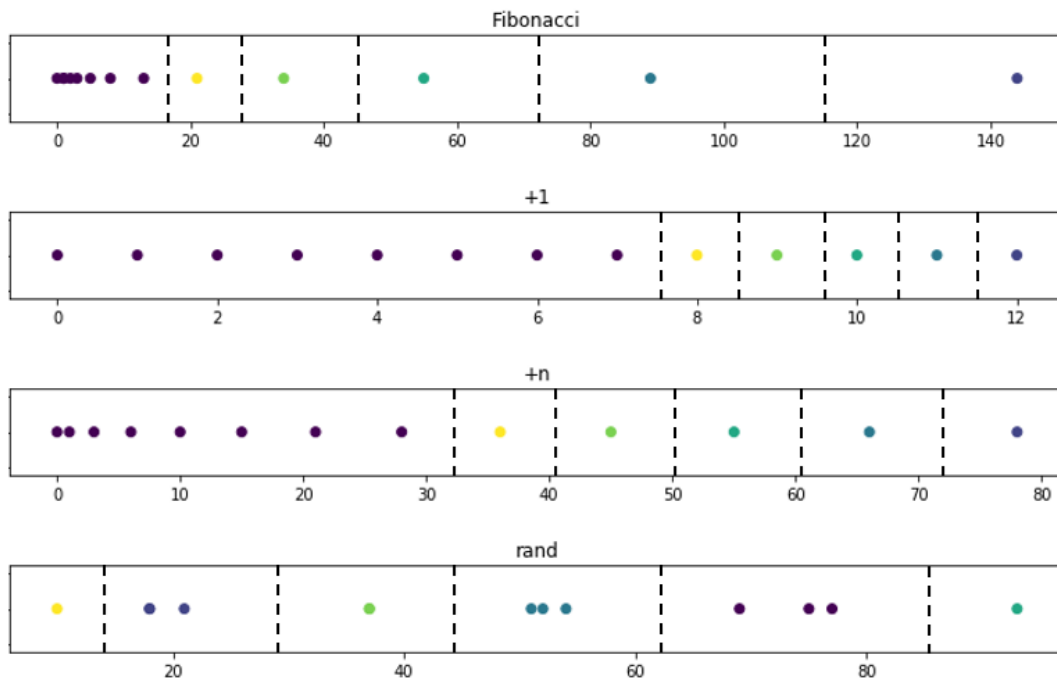


Figure 1 – Linkage : single

3.1.2 Average

Dans ce mode, on ne prend non plus la distance entre les 2 produits les plus proches de 2 lots, mais la distance entre leur centre de gravité.

Ici, pour les données de "+1", les résultats sont plus concluants : nous avons 5 lots de 2 produits

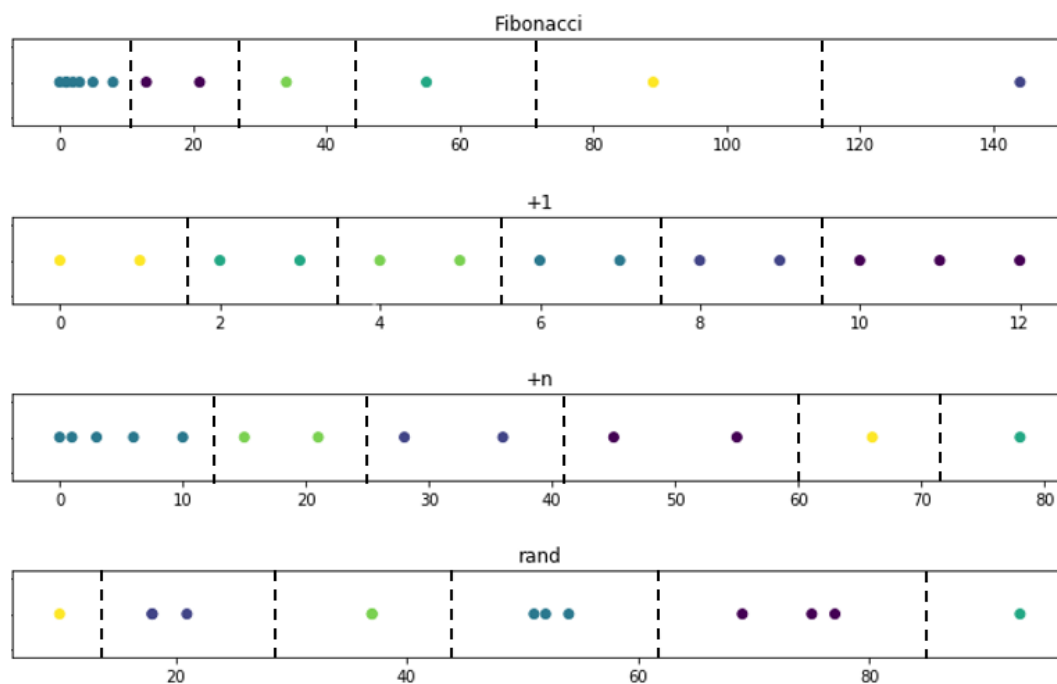


Figure 2 – Linkage : average

et 1 lot de 3 produits. C'est ce type de groupement que nous attendons. Sur les autres jeux de

données nous observons le même genre de résultat : ce mode de fonctionnement de l'algorithme est plus efficace que le précédent.

3.1.3 Ward

Ce mode ne calcule plus la distance entre des produits, mais fait la liaison qui va générer le lot avec la plus faible variance possible. La variance est un indicateur de dispersion : plus la variance est faible, plus les points sont regroupés.

Nous obtenons, pour les instances "+1", "+n" et "rand" les mêmes résultats qu'en mode "average".

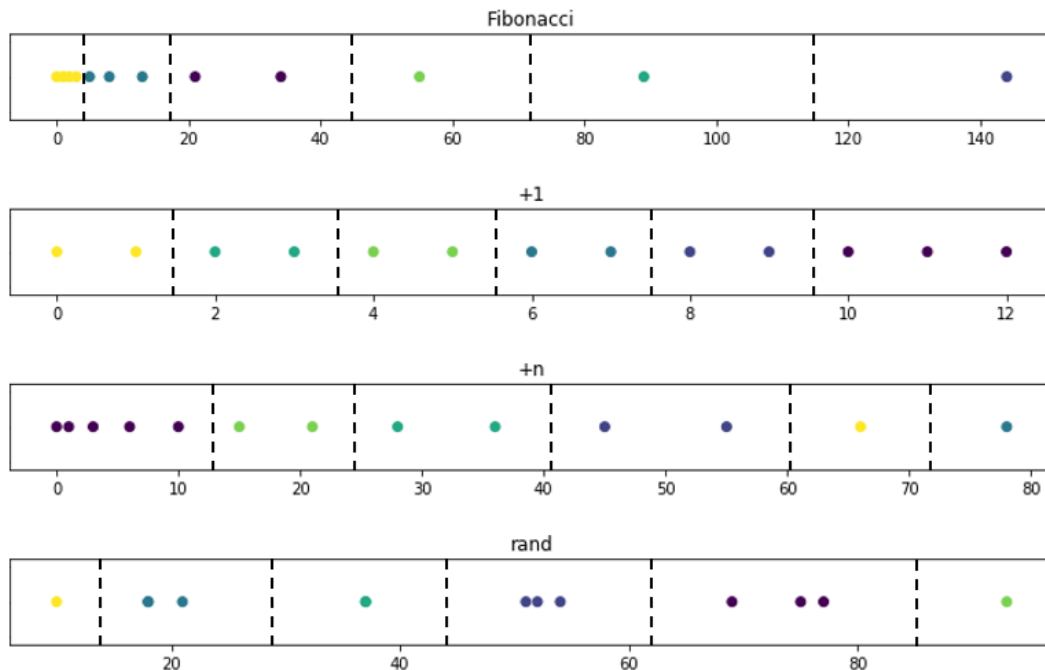


Figure 3 – Linkage : ward

Néanmoins, une différence est notable pour l'instance "Fibonacci". Voici comment cette instance a été découpée par les 2 méthodes :

- Average : 7/2/1/1/1/1
- Ward : 5/3/2/1/1/1

C'est donc en fonctionnement "ward" que l'instance a été découpé avec des lots de taille le plus similaire. C'est donc en mode "ward" que nous avons eu les meilleurs résultats, et c'est de cette façon que nous utiliserons notre algorithme dans la suite du projet.

3.2 Comparaison K-Means / Agglomératif

Afin de nous décider sur l'algorithme à utiliser, nous avons effectué des tests sur les deux algorithmes avec les mêmes instances. Afin de résumer les informations essentielles nous utilisons des diagrammes en boîtes. Les graphiques affichés dans ce rapport ont été réalisés avec une instance de 200 produits à livrer en 40 lots. Dans la figure 4, le graphique 1 est celui de K-Means, et le graphique 2 est celui de l'algorithme agglomératif. Nous voyons qu'avec K-Means, nous obtenons ici des lots qui atteignent les 10 produits, alors qu'il y a au maximum 9 produits par lot avec l'agglomératif. Le 3ème quartile de l'agglomératif est également légèrement plus bas que celui du K-Means. Nous pouvons remarquer que dans l'ensemble, l'agglomératif donne

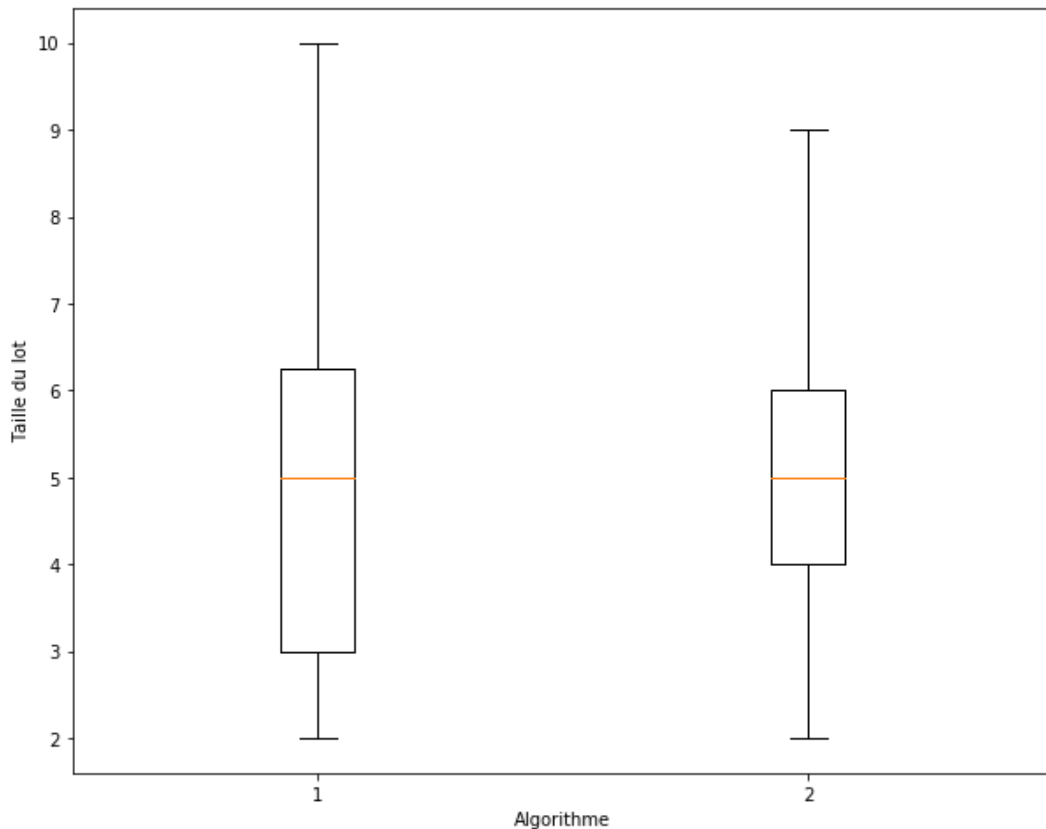


Figure 4 – Diagramme en boîtes algorithmes de partitionnement

des lots de taille plus homogènes que le K-Means. Les deux algorithmes nous donnent des résultats presque similaires, mais l'homogénéité et la tendance à produire des lots plus petits de l'agglomératif ont pour conséquence que c'est l'algorithme que nous allons développer dans l'application finale.

4 Représentation des données

La Figure 5 représente des produits à livrer. Les axes horizontaux représentent les positions géographiques en x et y des différents produits. L'axe vertical représente l'heure due de chaque produit. Plus un produit est haut, plus son heure due est importante. Sur la figure, nous avons représenté un exemple contenant 4 hopitaux à livrer, pour un total de 200 produits. Chaque couleur correspond à un lot, mais étant donné le nombre de lots (40), il est difficile de discerner chaque lot. Néanmoins, on se rend bien compte en voyant ces données qu'il y a énormément de produits à livrer, et on comprends donc pourquoi il est difficile d'optimiser ceci sans l'aide d'un solveur.

Pour comprendre un peu mieux le découpage qui a été produit, nous allons analyser nos données en 2D.

La Figure 6 représente le découpage seulement selon les axes géographiques. Chaque produit est lié aux autres produits d'un même lot par un trait. Au milieu de chaque trait se trouve un rond, qui représente le centre de gravité du lot. On peut voir que tous les produits à livrer dans l'hôpital le plus loin sont en lot entre eux, donc chaque livraison pour aller livrer cet hôpital ne contiendra aucun produit à destination d'un autre hôpital. L'hôpital en haut à droite est également un peu en retrait puisqu'un seul lot mélange ses produits et ceux d'un autre. En

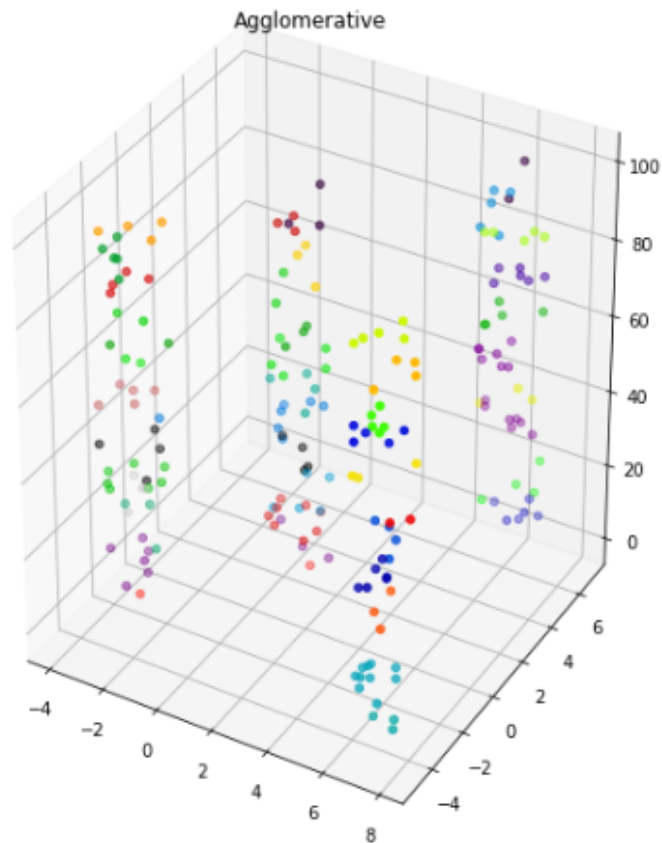


Figure 5 – Représentation 3D des produits

revanche, les 2 hôpitaux restants sont très mêlés puisqu'on peut voir un important amas de liaisons.

Les deux dernière figures, **Figure 7** et **Figure 8**, représentent le découpage vu en fonction des dates dues. Ici, on voit mieux comment les lots sont constitués. Premièrement, on peut voir que comme dit précédemment, les lots n'excèdent pas les 9 produits, et qu'il sont en majorité constitués d'environ 5 produits. Ensuite, on peut voir sur la **Figure 8** que les 2 hôpitaux à proximité sont en effet liés puisqu'une grande partie de leurs produits sont dans des lots communs.

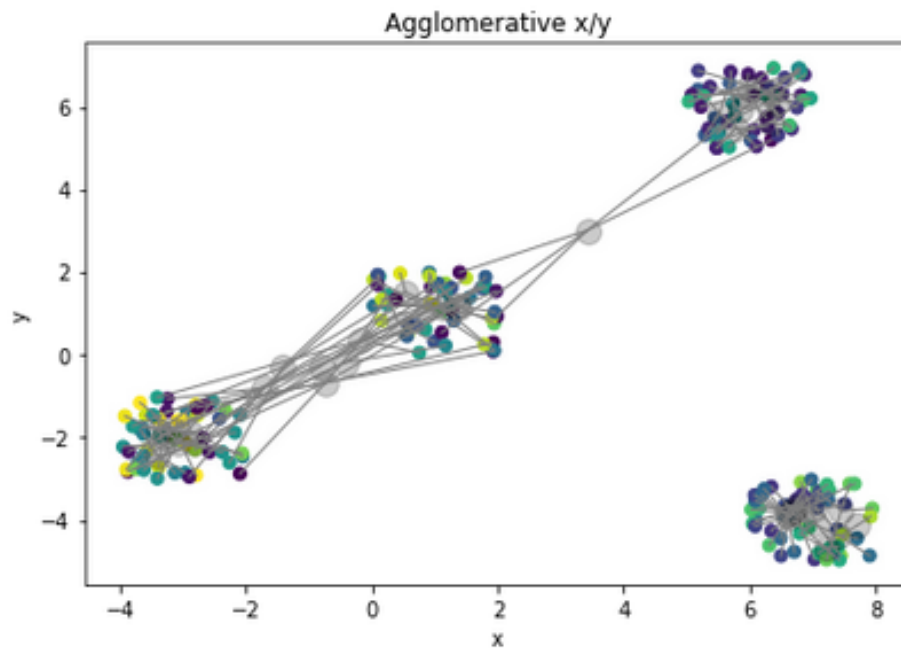


Figure 6 – Représentation 2D des produits - x/y

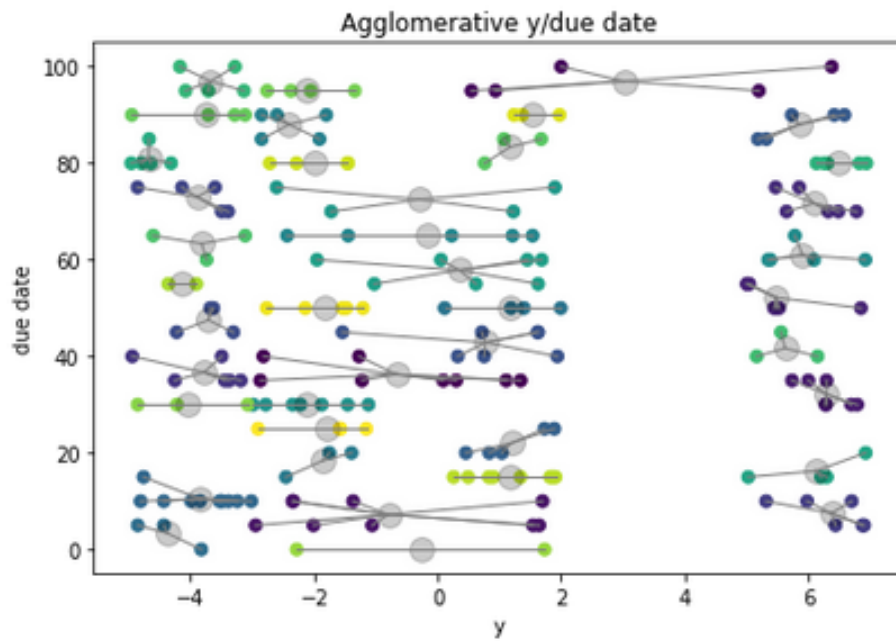


Figure 7 – Représentation 2D des produits - $x/\text{date due}$

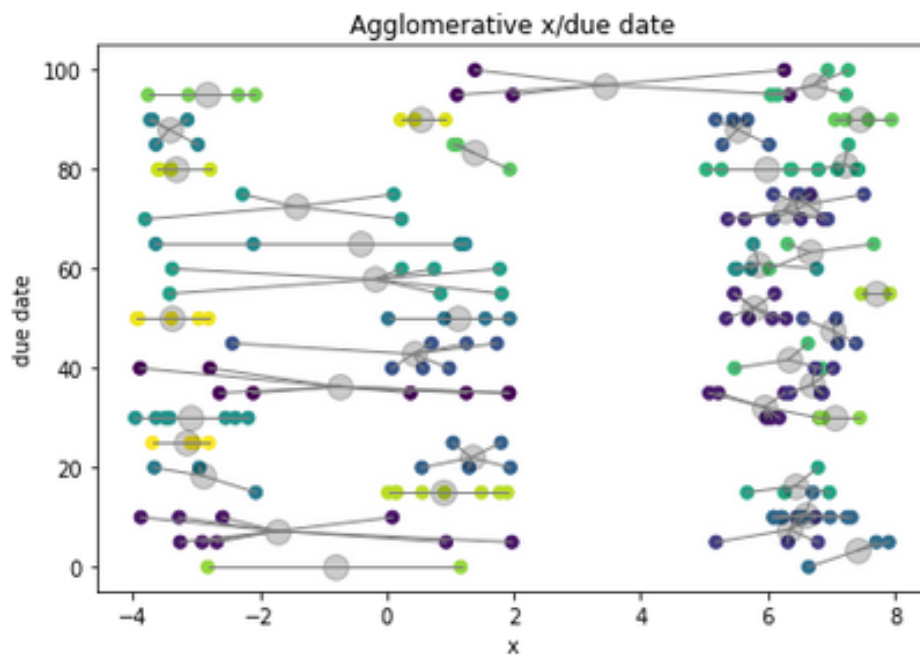


Figure 8 – Représentation 2D des produits - y/date due

5

Mise en oeuvre

1 Outils et librairies

Le logiciel qui a été créé utilise 2 langages :

- Python 3.7.0 pour la majorité du code
- Julia 0.6.4 pour le solveur

Avec chaque langage, nous utilisons différentes librairies :

- Python
 - sklearn 0.20.0 pour le découpage en lots
 - matplotlib 3.0.0 pour l’affichage des résultats du solveur
- Julia
 - CPLEX 0.4.0 qui est notre solveur (en version essai)
 - voptGeneric 0.1.1 pour l’utilisation de CPLEX

Si nous utilisons une version antérieure de Julia, c’est car le github vOptGeneric précise que son fonctionnement n’est garanti qu’avec Julia 0.6.4. L’avantage de vOptGeneric, c’est que lorsque l’on a codé notre problème, si nous avons envie de changer de solveur, nous pouvons le faire en une ligne.

2 Implémentation

2.1 Structure

Nous avons suivi un design pattern MVC (Modèle Vue Controlleur), le programme est donc constitué comme suit :

- Une fenêtre principale, dont différentes parties sont découpées en sous-classes, le tout faisant partie de la "Vue"
- Des classes "Modèle", qui contiennent le coeur de notre logiciel comme l’algorithme de découpage et les fonctions d’écriture/lecture de fichiers
- Un ensemble de classes dites "Controlleur", qui gèrent la cohésion entre les interactions de la fenêtre, et l’appel des fonctions situées dans le modèle

Cette structure permet, notamment, de nous assurer que si l'interface devait être changée, par exemple pour utiliser un autre outil que celui utilisé actuellement, les modifications n'affecteront pas les autres parties du programme. Il sera ainsi facile de modifier l'interface puis de la ré-intégrer dans le logiciel.

2.2 Mise en lots

La mise en lot avait déjà été étudiée au premier semestre, c'est donc le code qui avait été développé à ce moment qui a été utilisé. Il n'a fallu faire que quelques modifications pour pouvoir être utilisable dans notre logiciel.

2.3 Livraison

Des premiers scripts avaient été réalisés au semestre 9, mais ils ne suivaient pas la dernière version de notre modèle mathématique, comme l'utilisation de contraintes d'inégalités valides. Il y a également des optimisations logicielles qui ont été faites. Lorsque que l'on calcule le trajet pour livrer un seul produit, nous ressortons directement les valeurs calculées "à la main", sans passer par l'utilisation de notre solveur. Dans le cas où il y a 2 produits à livrer, nous faisons la même chose en calculant les 2 trajets possibles et en prenant celui avec le meilleur résultat dans la somme des retards (le temps de trajet étant forcément identique). Quand il y a plus de 2 produits, nous faisons appel au solveur.

S'il y a appel au solveur, nous l'utilisons en mode "Epsilon constraint". Cette méthode permet, puisqu'on est en bi-objectif, d'obtenir non pas 1 mais plusieurs solutions. Nous avons défini 2 objectifs dans le solveur : minimiser la somme des retards et minimiser le temps de trajet. Le solveur va tenter de résoudre le problème en nous trouvant le meilleur temps de trajet. Une fois fait, il va recommencer la résolution du problème mais en autorisant un résultat sur cet objectif un peu moins bon, mais en permettant par la même, si possible, d'améliorer la somme des retards. S'il trouve en effet une nouvelle solution il va recommencer afin de trouver toutes les solutions, qui vont former le "Front de Pareto". C'est cet ensemble de solutions que l'on montrera à notre utilisateur sur l'interface. Mais dans la majorité des cas, il n'y a qu'une seule solution trouvable. Il arrive qu'il y ai 2 solutions, mais il devient très rare qu'il y ai 3 solutions ou plus.

2.4 Interface

Comme on peut le voir sur la [Figure 1](#), il y a 3 parties distinctes dans l'interface : la partie mise en lots, la partie chemins pour la livraison, et la partie résolution et affichage des résultats de la livraison.

Chacune des parties possède sa propre classe dans le code, et il est facile de pouvoir en modifier une, la fenêtre s'adaptera aux modifications. Seuls les parties chemins de la livraison et résolution communiquent entre elles. La partie liée à la mise en lot est totalement indépendante.

Si jamais un champs devait être invalide, par exemple parce qu'un chemin de dossier ne contient pas les fichiers nécessaires, l'application ne plantera pas, mais aucun signal n'a été développé. En conséquence de quoi, si une erreur est faite, il ne se passera juste rien. Mais tout ce qui est nécessaire pour cette détection est présent dans le code, notamment par la présence et l'envoi d'erreurs.

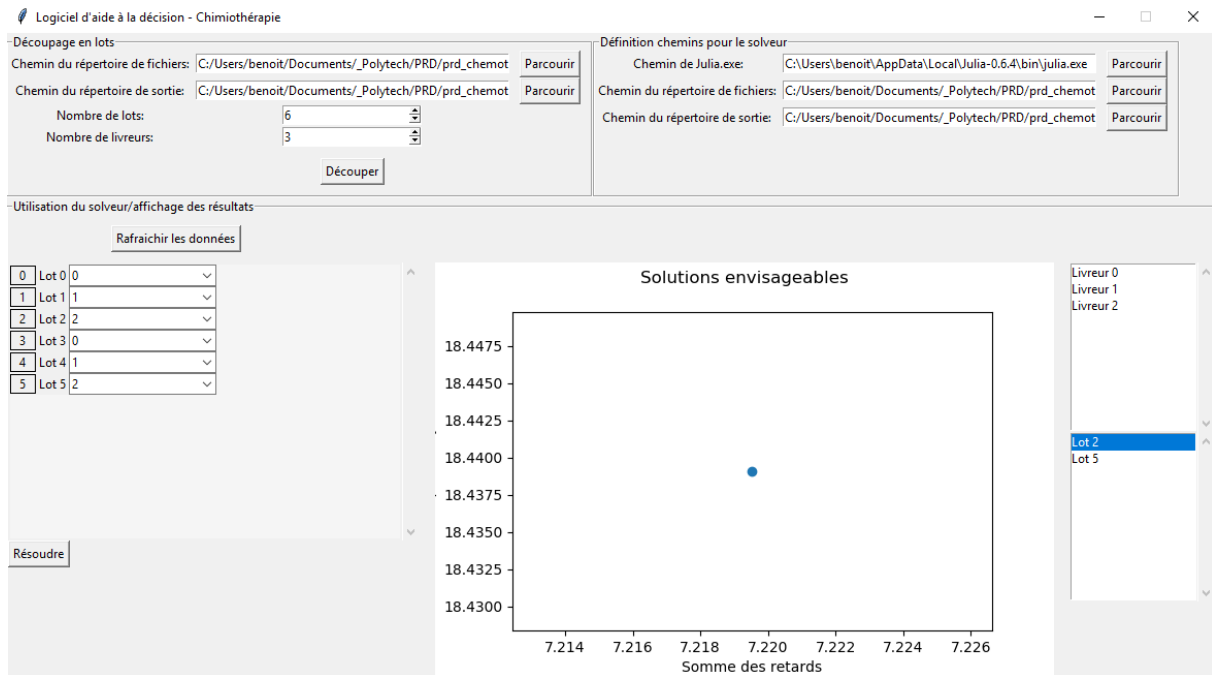


Figure 1 – Interface

2.4.1 Mise en lots

Le but de cette partie est de spécifier dans "Chemin du répertoire de fichiers" le dossier qui contient l'ensemble des fichiers nécessaires au découpage en lots, et de spécifier dans "Chemin du répertoire de sortie" là où les fichiers générés vont être stockés. Il faut ensuite spécifier le nombre de lots voulus et le nombre de livreurs disponibles. Une fois que toutes les informations sont renseignées, il faut appuyer sur "Découper" pour lancer le découpage en lots de notre instance.

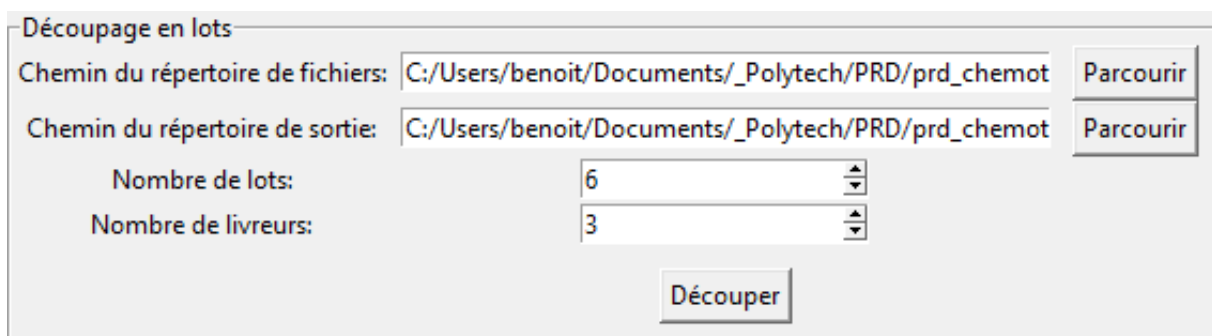


Figure 2 – Interface - mise en lots

2.4.2 Chemins pour la livraison

Ici, il faut commencer par rentrer le chemin vers julia.exe, qui est utilisé pour la résolution de nos problèmes d'acheminement. De base, ce champ vaut la valeur présente sur la Figure 3, en remplaçant juste "benoit" par le nom d'utilisateur courant, car c'est ici qu'il est installé si on suit l'installation par défaut qui est conseillée. Il faut ensuite, comme pour la partie précédente, spécifier le dossier contenant les fichiers d'entrée et le dossier de sortie. Ici, le dossier d'entrée doit être le même que le dossier de sortie de la partie précédente, puisque que c'est les fichiers précédemment créés que nous allons utiliser.

Définition chemins pour le solveur

Chemin de Julia.exe:

Chemin du répertoire de fichiers:

Chemin du répertoire de sortie:

Figure 3 – Interface - chemins pour la livraison

2.4.3 Livraison

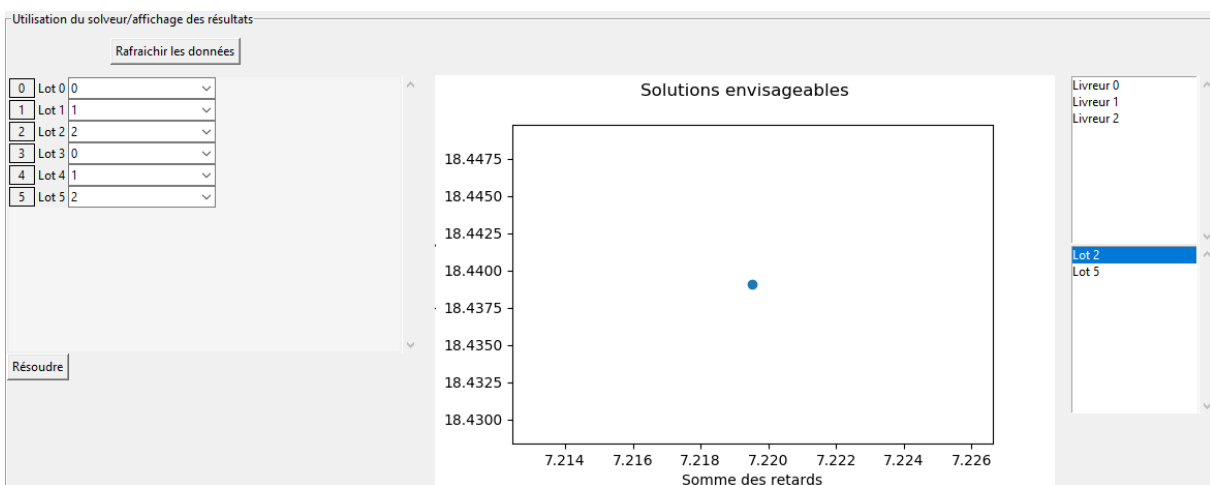


Figure 4 – Interface - livraison

Cette dernière partie permet de visualiser nos résultats, de modifier les affectations livreur/lot, et de lancer la résolution de notre problème. En haut à gauche, il y a un bouton "Rafraichir les données", qui sert à actualiser nos données avec les fichiers trouvés dans les dossiers spécifiés précédemment. Si vous modifiez des fichiers ou des répertoires de dossier, appuyez sur ce bouton pour vous assurer que les données sont bien actualisées. En dessous, il y a une liste de lots, avec pour chaque lot la possibilité d'attribuer un livreur. Dans cet exemple, nous avons l'attribution suivante :

- Livreur 0 : lots 0 et 3
- Livreur 1 : lots 1 et 4
- Livreur 2 : lots 2 et 5

Quand les attributions vous conviennent, vous pouvez appuyer sur "Résoudre". Il va falloir un certain temps pour que le solveur résolve l'ensemble des trajets. Une fois fini, vous aurez à droite de l'écran la possibilité de cliquer sur un livreur, puis de cliquer sur l'ensemble des lots qu'il livre. Vous aurez alors sur le graphique du milieu un ensemble de points correspondants à des trajets de livraison. Il y a deux coordonnées : "Somme des retards", qui équivaut à la somme des retards sur chaque produit à livrer, et "Temps de trajet", qui est le temps total de trajet pour livrer le lot.

3 Resultats

Les résultats finaux et de chaque étape sont sauvegardés au format JSON.

```
{
  "start": 0,
  "distances": [[0.0, 5.0, 5.3],[5.0, 0.0,
1.4],[5.3, 1.4, 0.0]],
  "dueDates": [14.0, 12.0, 12.0],
  "deadLines": [9999.0, 9999.0, 9999.0],
  "jobsServices": [1, 2, 3, 4],
  "solution": [{
    "path": [[0, 0, 1],[1, 0, 0],[0, 1, 0]],
    "z": [0.0, 11.79937]}
]}
```

Voici un exemple de résultat pour la livraison du premier lot d'un livreur. C'est ce format que lit le logiciel pour afficher les résultats sous forme de graphiques. Nous avons les informations suivantes :

- start : l'heure à laquelle la livraison du lot commence
- distances : la distance entre chaque service du trajet
- dueDates : les dates dues de chaque produit (le premier est à 0 car il représente le service de départ)
- deadLines : les deadlines de chaque produit
- jobsServices : dans quel service se trouve chaque produit. Dans ce cas, on a tout simplement autant de services que de produit, donc chaque service ne contient qu'un seul produit
- solution : la solution contient deux éléments :
 - path : correspond au trajet à parcourir. On commence dans le premier tableau, et on cherche le 1. On se dirige au 1 correspondant, et on recommence. Ici, on va de 1 en 3 qui nous amène en 2 qui finalement nous ramène en 1
 - z : ici se trouve les valeurs de nos deux objectifs, le premier est la somme des retards, et le deuxième est le temps de trajet. Ici, il n'y a aucun retard

Sur la machine testée, les temps de calculs ne dépassent pas les 1 minute par lot pour des lots de moins de 9 produits. Lorsque l'on passe à 10 produits, le temps de calcul explose et on peut dépasser les 5 minutes par lot. Il est donc préférable de ne pas dépasser les 10 produits par lot fin de ne pas avoir des temps de calculs beaucoup trop longs. Dans nos spécifications nous avons demandé un temps de calcul inférieur à 10 minutes. A condition d'avoir environ une dizaine de lots, les temps sont bons. Dans les cas contraires, l'application dépassera certainement les 10 minutes. Lorsque comparé à l'algorithme développé par Alexis Robbes, les temps de calcul du solveur sont beaucoup plus grands. Il faut donc soit trouver un moyen de les réduire (notamment en ajoutant des contraintes d'inégalités valides), soit remplacer l'utilisation du solveur par l'implémentation de son algorithme.

4 Qualité

4.1 Vérificateur de solutions

Une classe a été créée spécialement pour tester les solutions retournées par notre solveur. Cette class va tester si la solution respecte les différentes contraintes. Par exemple, elle va tester si le chemin d'une solution passe bien par tous les points, une unique fois. Dans le cas où notre solution est correcte, un tableau vide est retourné. Dans le cas contraire, une liste d'erreurs est retournée. Cette liste contient des erreurs qui sont définies dans un énumérateur.

4.2 Tests

Il n'y a que deux tests unitaires qui ont été réalisés, et ces deux tests vérifient que notre classe de vérification de solutions retourne bien une liste vide lorsque notre solution est correcte, et qu'elle renvoie les bonnes erreurs si certaines se présentent. Les fiches de test des tests sont retrouvables dans les annexes.

Les autres parties n'ont pas eues de tests unitaires dû à un manque de temps, donc elles n'ont été testées qu'à la main, en mode debug ou par la vérification manuelle des entrées/sorties, notamment pour les classes de lecture et d'écriture de fichiers.

4.3 Outils

Afin d'assurer la partie qualité qualité du code, certains éléments ont été introduits :

- Gitlab pour le versionnement du code
- Des tests unitaires
- Une documentation développeur et utilisateur
- Un diagramme de classes UML ainsi que d'autres diagrammes UML (séquence, fonctions, etc.)
- Une documentation python afin de documenter l'ensemble des fonctions et leur utilité

GitLab a été utilisé pour versionner le code, mais aussi pour gérer les différentes tâches à gérer. En effet, le système d'"issues" est très pratique puisqu'il permet d'attribuer à une tâche des personnes, des dates de début/fin, des labels personnalisés, etc. Avec ce système, on entre plus dans le détail que le Gantt. Le Gantt permet par exemple de définir la tâche "Développement du script lié au solveur", et sur GitLab on postera différentes "issues" comme "Bug de la contrainte 3", "Réaliser la fonction de lecture de fichier", etc.

Pour ce qui est de la documentation, il y a deux types de documentations en plus de la doc Python. La doc Python explique pour chaque méthode/fonction quels sont les paramètres en entrées et les sorties possibles. Il y a également une documentation développeur, qui explique comment est organisé le programme, quels sont les rôles de chaque classe, et comment installer les bonnes versions de chaque langage/librairie. Enfin, la documentation utilisateur explique comment installer et utiliser le logiciel pour une personne avec peu de connaissances en informatique. On va donc de la partie téléchargement des bons fichiers aux bons endroits, jusqu'à l'installation des bonnes bibliothèques, en passant par le rôle de chaque zone de texte de l'interface. Enfin, dans cette documentation est présente une description des différents fichiers attendus en entrée de notre programme.

6

Conclusion

1 Planning

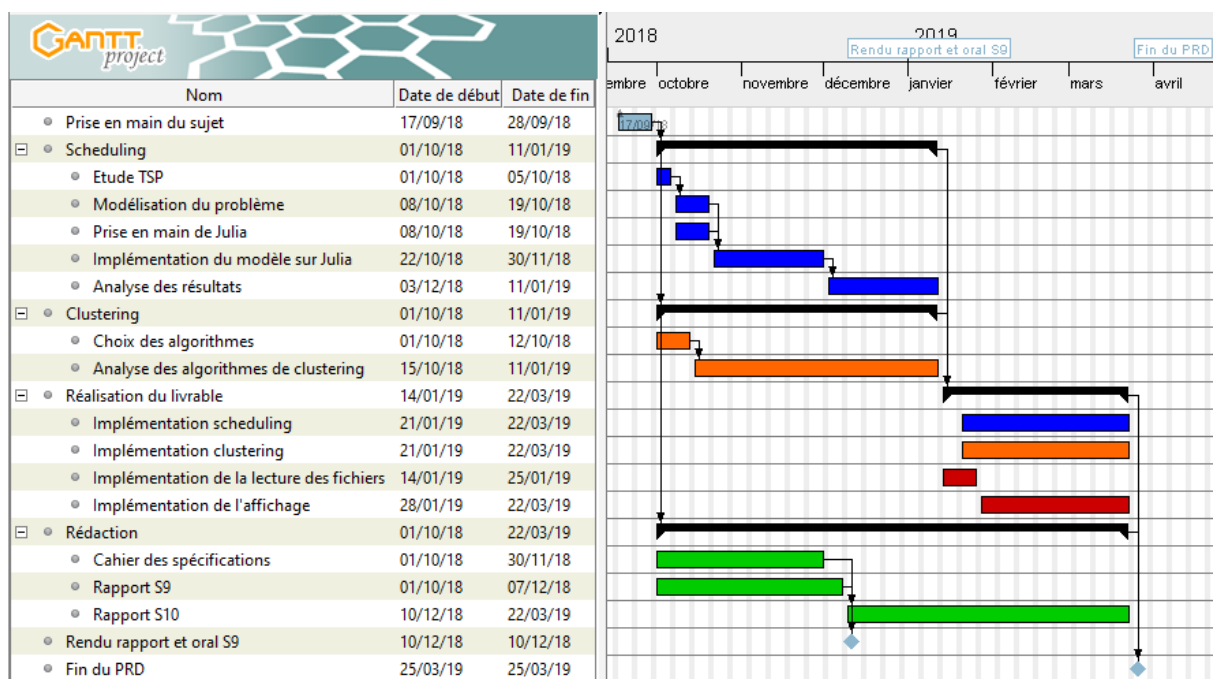


Figure 1 – Gantt S9

Voici les gantt du projet. Il y a différentes couleurs qui représentent des parties différentes :

- Bleu : optimisation de la livraison
- Orange : optimisation de la mise en lots
- Rouge : interface et lecture des fichiers
- Vert : rédaction des différents rapports

Le rapport est rendu autour du jalon "Rendu rapport et oral S10". La Figure 1 est le gantt qui avait été réalisé au courant du semestre 9, donc dans la première partie du projet. Le gantt réel du projet est la Figure 2. Les différences sont notables dans la partie réalisation du livrable. Il était à l'origine prévu que les fonctions d'écritures/lectures de fichiers soient terminées au plus

tôt, et que le reste du temps serait alloué à l'implémentation des différentes parties. Il s'est en réalité avéré que les fonctions de traitement de fichiers ont été beaucoup plus longues à réaliser, et ont notamment demandé de nombreuses modifications, d'où le temps de tâche plus long que prévu. C'est l'inverse pour les tâches d'implémentation de l'existant dans le livrable, cela à été beaucoup rapide que prévu, ce qui a laissé plus de temps pour travailler sur l'interface du logiciel. Pour la partie livraison, il y a eu de nombreuses améliorations au fur et à mesure, ce qui explique pourquoi cette tâche a duré presque tout le semestre 10.

Le projet a été réalisé dans les temps, et il n'y a pas eu beaucoup d'imprévus, ce qui a permis de suivre dans les grandes lignes le planning réalisé dans la première partie du projet.

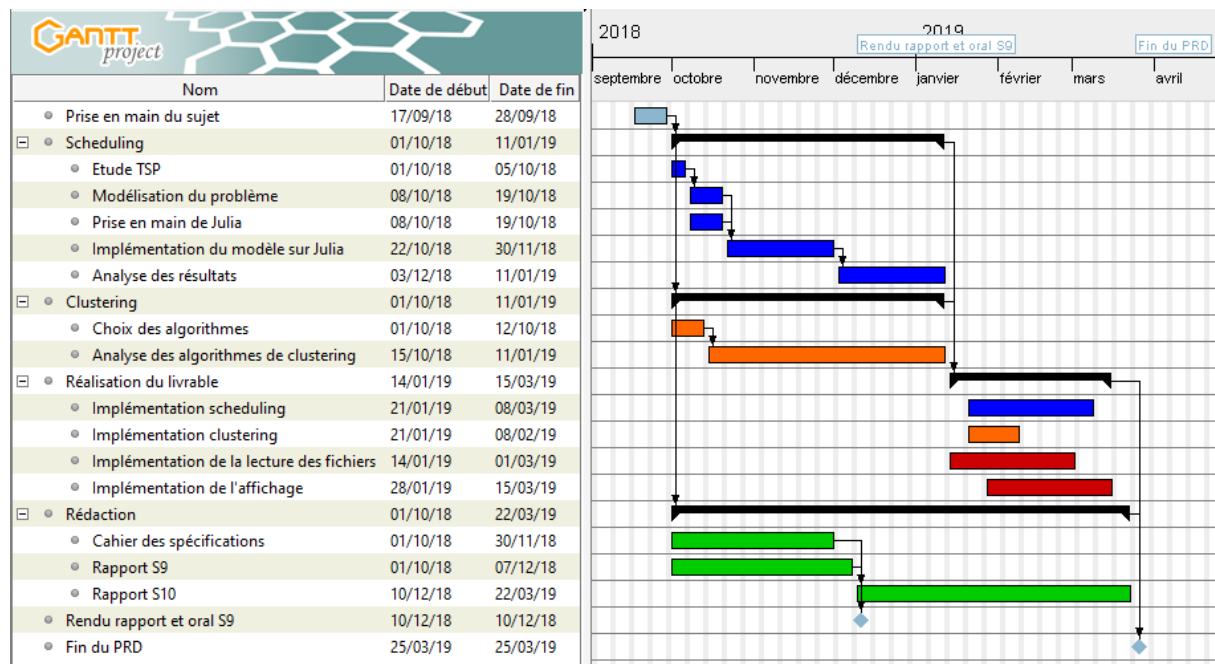


Figure 2 – Gantt S10

2 Qualité

L'ensemble des documentations attendues ont été fournies (utilisateur, développeur, etc.), et différents diagrammes sont disponibles, comme le diagramme des classes, afin d'avoir une idée de l'architecture du programme. Différents tests ont été mis en place, mais malheureusement, le manque de temps a fait que ces tests ne sont pas assez nombreux.

Le code est uniforme en terme de documentation, nommages, et il a été réalisé de manière à être facile à reprendre.

3 Ce qui fonctionne

Le logiciel fourni fonctionne, et permet à la fois de découper nos produits en lots, puis de calculer la trajectoire optimale de livraison de chaque lot. On pourra déplorer l'impossibilité, en cas de présence de plusieurs trajectoires possibles, le fait de pouvoir choisir laquelle prendre (le logiciel va choisir celle avec le meilleur temps de trajet, et l'utilisateur ne peut actuellement pas choisir d'en prendre une autre). Mais étant donné que dans de nombreux cas nous n'obtenons qu'une seule solution, ce problème n'empêche pas l'utilisation du logiciel.

4 Possibilités d'améliorations

Il y a de nombreuses choses que l'on peut améliorer/rajouter dans le livrable :

- Pouvoir choisir parmi une liste de trajets pour chaque livraison de lot
- Obtenir des graphiques/statistiques sur les solutions
- Améliorer/optimiser le modèle mathématique et son implémentation pour améliorer les temps de calcul

Il y a aussi la possibilité de retoucher à la partie interface de l'application pour la rendre plus ergonomique.

5 Conclusion personnelle

Ce projet a été un bon moyen de travailler toutes les étapes d'un projet, en passant de l'analyse, au développement, aux tests, etc. J'ai également appris beaucoup de la recherche opérationnelle et de sa mise en application, étant donné que j'avais des connaissances mais que je n'avais pas fait beaucoup de mise en pratique. Il est regrettable que je n'ai pas touché à la partie attribution livreurs/lots automatique et optimisée.

Annexes

A

Spécifications

1 Contexte de la réalisation

À Tours, il y a 3 hopitaux qui sont concernés par la livraison de produits de chimiothérapie : le CHU de Bretonneau, de Trousseau, et de Clocheville. C'est le CHU de Bretonneau qui livre ces produits. Il est nécessaire d'être le plus efficace dans cette livraison, car les produits de chimiothérapie ont une courte durée de vie. Les différentes étapes pour la production/livraison sont les suivantes :

- Prescription par un service de cancérologie
- Validation
- Stérilisation par l'URCC
- Préparation par l'URCC
- Vérification (contrôle analytique) par l'URCC
- Livraison
- Administration par un service de cancérologie

URCC = Unité de Reconstitution Centralisée des Cytotoxiques

Les produits ont un temps de stabilité limité lié aux éléments chimiques utilisés, il est donc nécessaire de gérer au mieux la livraison pour qu'ils soient utilisés dans les délais. Pour optimiser la livraison il faut : optimiser la mise en lots des produits pour que les produits qui doivent arriver au même endroit et/ou au même moment soient envoyé ensemble, et optimiser les tournées des livreurs pour rendre leurs déplacement le plus efficace possible. Etant donné que la vie de patients est en jeu, le but est limiter les retards.

Ce projet est réalisé en parallèle avec le sujet de thèse d'Alexis ROBBES. Par conséquent, il est le client du PRD.

1.1 Objectifs

Ce projet vise à répondre à un besoin d'optimisation de la logistique en proposant une application d'aide à la décision pour la mise en lot et la définition des tournées des livreurs.

1.2 Bases méthodologiques

Afin que le projet se déroule dans les meilleures conditions, nous utilisons un certain nombre d'outils :

- GitLab
- GanttProject

Nous gérons le projet en suivant un cycle en V.

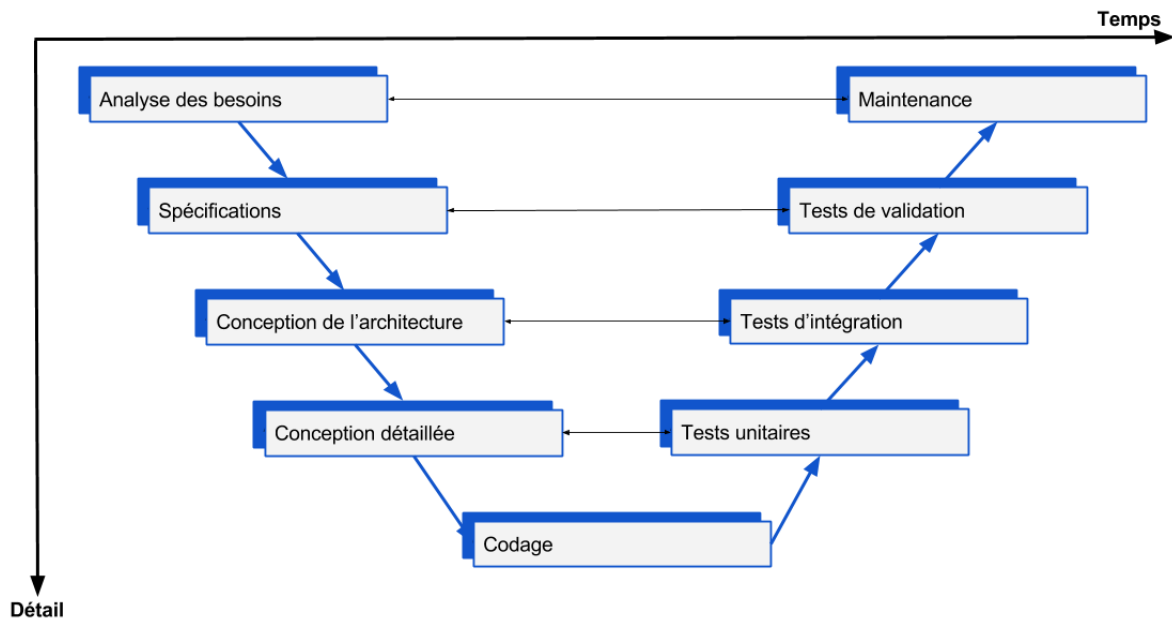


Figure 1 – Cycle en V

2 Description générale

2.1 Environnement du projet

Ce projet vient en complément du sujet de thèse d'Alexis ROBBES. Seuls les données testées sont fournies. Le reste du projet est entièrement indépendant, et aucun existant n'a été fourni. Le produit sera fonctionnel sur Windows, et le tout devra être disponible gratuitement (donc utilisation de logiciels/bibliothèques/etc. gratuits). En plus d'être gratuit, le code sera open source.

2.2 Caractéristiques des utilisateurs

Il n'y aura qu'un client, Alexis ROBBES, qui possède les caractéristiques suivantes :

- Connaissance approfondie du sujet
- Connaissances en informatique

Le client aura accès à la totalité du travail rendu, autant du point de vue interface, qu'à l'intégralité du code source réalisé. Il aura donc la possibilité de porter des modifications sur n'importe quel composant.

Néanmoins, le livrable doit pouvoir être utilisé par du personnel d'hôpital par exemple, il faut donc que l'ergonomie de l'interface permette à une personne n'ayant aucun lien avec le projet de comprendre et de pouvoir l'utiliser.

2.3 Fonctionnalités du système

L'utilisateur doit avoir la possibilité de choisir les jeux de données qu'il souhaite analyser, que ces données soient traitées, puis que différents graphiques permettant d'expliquer la/les solution(s) soient affichés. Des résultats seront sauvegardés sur des fichiers.

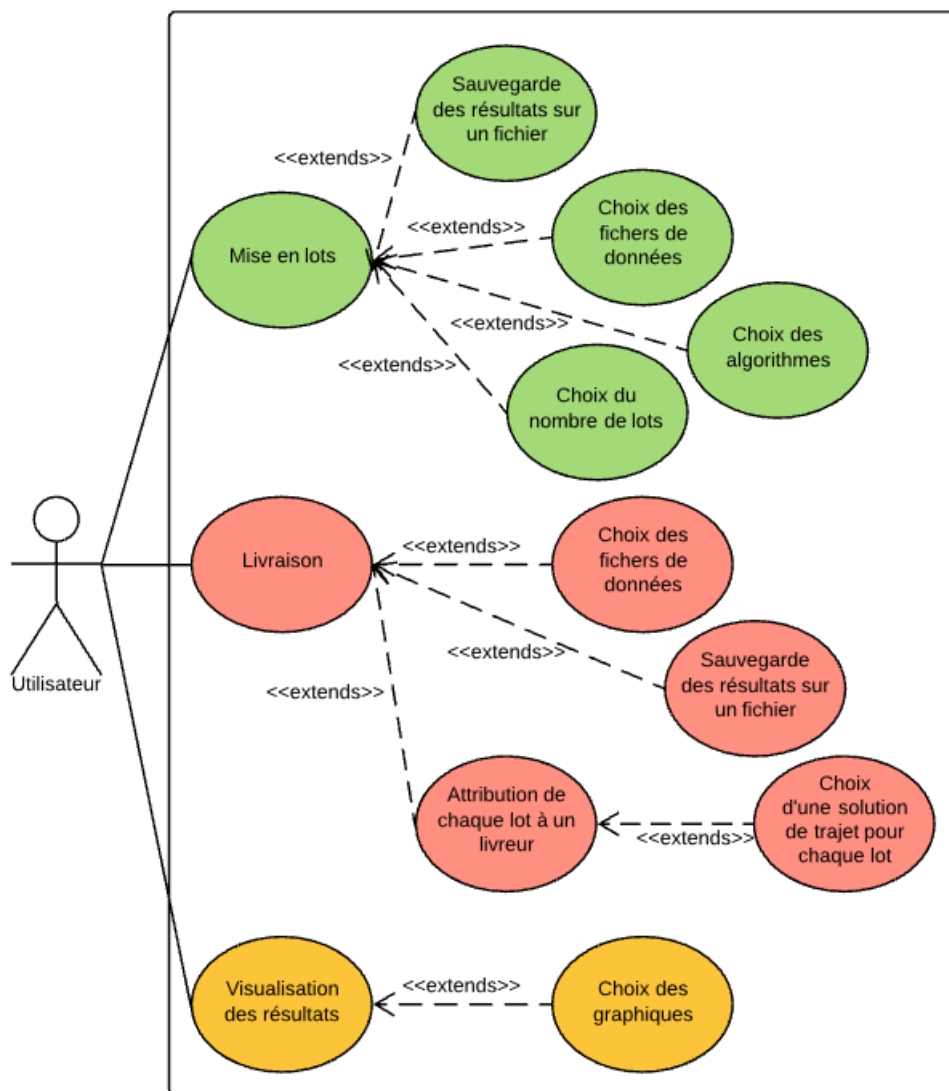


Figure 2 – Diagramme de cas d'utilisation

Pour la partie livraison, l'utilisateur doit attribuer un livreur pour chaque lot. Ensuite, pour chaque livreur, il doit choisir parmi un ensemble de solutions comment se fera le trajet du premier lot. Ce choix influera sur les solutions disponibles pour le trajet du deuxième lot, etc. En effet, il est possible que le premier trajet se fasse vite au dépend du retard d'un seul produit, ou que le premier trajet ne soit plus long mais qu'il livre tous les produits en avance. En fonction des cas il se peut que le deuxième trajet parte avec plus ou moins de retard, influant sur les solutions envisageables.

2.4 Structure générale du système

L'utilisateur aura pour rendu un système capable de lui afficher, pour un problème donné, un ensemble de graphiques/d'indicateurs sur la/les solution(s) envisageable(s). Etant donné que l'optimisation de la mise en lot est réalisée en Python, et que l'optimisation de la livraison est réalisée en Julia, ces deux parties ont leur propre interface/affichage. Il est à noter que l'on peut appeler du code Python depuis Julia. Il est donc possible que le livrable final combine les deux parties en un seul logiciel.

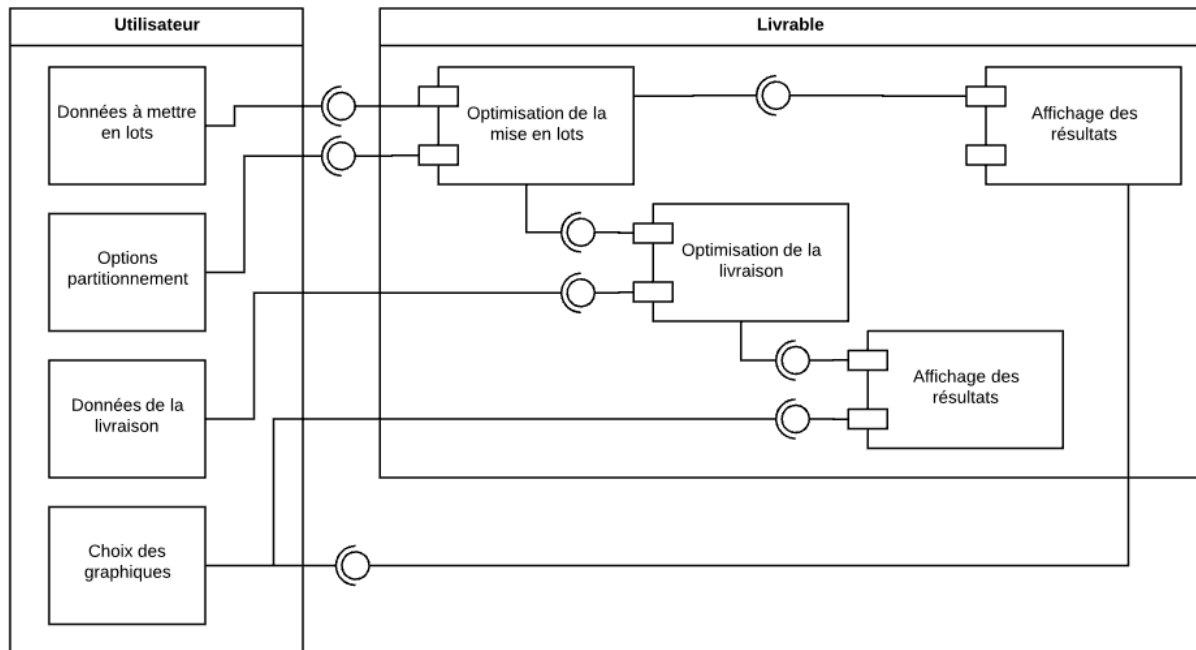


Figure 3 – Diagramme de composants

3 Description des interfaces externes du logiciel

3.1 Interfaces matériel/logiciel

Rien de ce qui ne sera rendu n'aura de besoins particuliers en terme matériel :

- Pas de communication avec des éléments externes à l'ordinateur qui exécute le programme
- Pas de nécessité de connexion internet

3.2 Interfaces homme/machine

Il n'y a que peu de contraintes sur l'interface proposée. Celle-ci doit contenir les informations suivantes :

- Possibilité de sélectionner des fichiers comme sources de données
- Affichage de résultats d'analyse sous forme de graphiques et/ou d'indicateurs
- Pour la partie livraison, choix manuel de l'association livreur/lot, puis choix manuel de la solution à utiliser (trajet à effectuer)
- Après le choix d'une solution de trajet pour la livraison d'un lot, choix de la solution du lot suivant

3.3 Interfaces logiciel/logiciel

Le livrable ne nécessitera que deux types d'interactions avec des éléments externes : choisir un/des fichier(s) d'entrée afin d'exécuter l'analyse, et sauvegarder certains résultats sur des fichiers externes.

4 Spécifications fonctionnelles

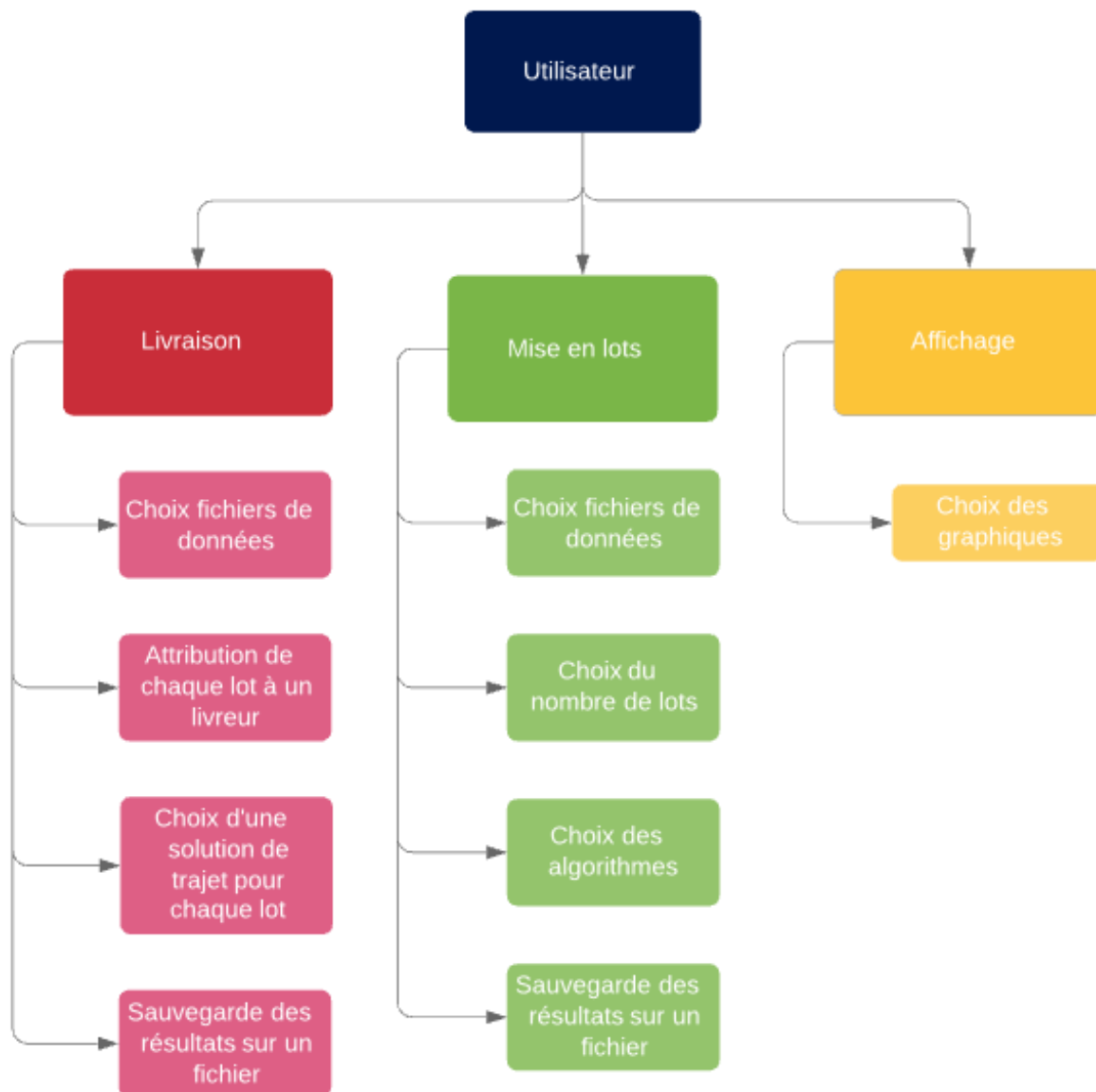


Figure 4 – Fonctions du livrable

Il y a 3 parties dans ce projet, qui sont : l'optimisation de la mise en lot, l'optimisation de la livraison, et l'affichage des résultats.

Mise en lots :

- Choix fichiers de données : choix du fichier qui contient la liste des produits à mettre en lots, ainsi que leurs 3 coordonnées : position en x, y et heure de livraison attendue

- Choix du nombre de lots : possibilité sur l'interface de définir le nombre de lots voulus en sortie
- Choix des algorithmes : possibilité de choisir l'algorithme utilisé pour calculer le partitionnement
- Sauvegarde des résultats sur un fichier : sauvegarde de l'ensemble des lots créés et les produits qu'ils contiennent

Livraison :

- Choix fichiers de données : choix des fichiers qui contiennent l'ensemble des données nécessaires (cf. section suivante)
- Attribution de chaque lot à un livreur : attribution de l'ensemble des lots au livreur souhaité (1 livreur par lot).
- Choix d'une solution de trajet pour chaque lot : proposition à l'utilisateur d'un ensemble de trajets possibles, et les conséquences associées (retards de livraison, date de retour). Chaque trajet influence les trajets suivant
- Sauvegarde des résultats sur un fichier : sauvegarde de l'ensemble des trajets définis et des associations lots/livreurs

4.1 Optimisation de la mise en lots

Présentation de la fonction :

- Nom : optimisation de la mise en lot
- Rôle : à partir d'un ensemble de produits, retourner un ensemble de lots de produits
- Priorité : primordiale

Description :

- Entrée : fichier contenant une liste de produits de 3 dimensions : position géographique x, y et heure à laquelle le produit est attendu
- Sortie : liste des lots créés et les produits associés, le tout sauvegardé sur un fichier
- Possibilité de choisir le ou les algorithmes utilisés

4.2 Optimisation de la livraison

Présentation de la fonction :

- Nom : optimisation de la livraison
- Rôle : à partir d'un ensemble de produits, de services, et de données liées, établir le chemin le plus efficace pour tout livrer à temps
- Priorité : primordiale

Description :

- Entrée : fichiers contenant :
 - Les produits et leur service associé
 - La distance entre chaque service
 - L'heure à laquelle le produit est attendu
 - L'heure limite à laquelle le produit peut-être livré
- Sortie : ensemble des trajets obtenues, et valeurs associées (retards de livraison, date de retour), le tout sauvegardé sur un fichier
- Cette fonction est exécutée pour chaque lot défini en sortie de la fonction "optimisation de la mise en lot"
- Cette fonction requiert l'interaction de l'utilisateur pendant son exécution :

- Attribution d'un livreur à chaque lot
- Pour chaque livreur, et pour chacun de ses lots, choisir, parmi une liste de trajets proposés, celui voulu

4.3 Affichage des résultats

Il y a en réalité deux fonctions d'affichage : une pour la livraison et une pour la mise en lot. Mais étant donné que la seule différence entre les 2 est le type de graphique affiché, nous ne spécifions ici qu'une fonction, et nous détaillons dans la partie "sortie" les différences entre les graphiques possibles. Présentation de la fonction :

- Nom : affichage des résultats
- Rôle : afficher un ensemble de graphiques basés sur les résultats des fonctions précédentes
- Priorité : secondaire

Description :

- Entrée : résultats obtenus par les 2 fonctions précédentes
- Sortie : ensemble de graphiques créés à partir des données d'entrée :
 - Livraison :
 - Représentation des chemins calculés
 - Données des chemins calculés (temps de trajet, retards, etc.)
 - Fronts de Pareto des solutions
 - Mise en lot :
 - Histogramme des tailles de lot
 - Représentation sur plan 2D des lots créés
 - Boîtes à moustache des tailles de lot

Ce qui va être livré est un outil d'aide à la décision, il y aura donc plusieurs solutions envisageables et elles seront toutes affichées à l'utilisateur, et c'est à lui que reviendra le choix de sélectionner celle qui l'intéresse.

5 Spécifications non fonctionnelles

5.1 Contraintes de développement et conception

Il y a des contraintes pour chaque partie :

- Recherche opérationnelle sur les livraisons
 - Développement en Julia v0.6.4
 - Utilisation de vOpt solver
 - Utilisation des solveurs cplex ou gurobi
- Machine learning sur les mises en lot
 - Développement en Python 3
- Interface de visualisation des graphiques
 - Développement en Python 3 / Julia

5.2 Diagramme de séquence

Voici le diagramme de séquence du livrable. Avec ce diagramme nous voyons comment utiliser le livrable dans son entièreté, de la partie mise en lots jusqu'à l'affichage de l'ensemble des trajets de chaque livreur.

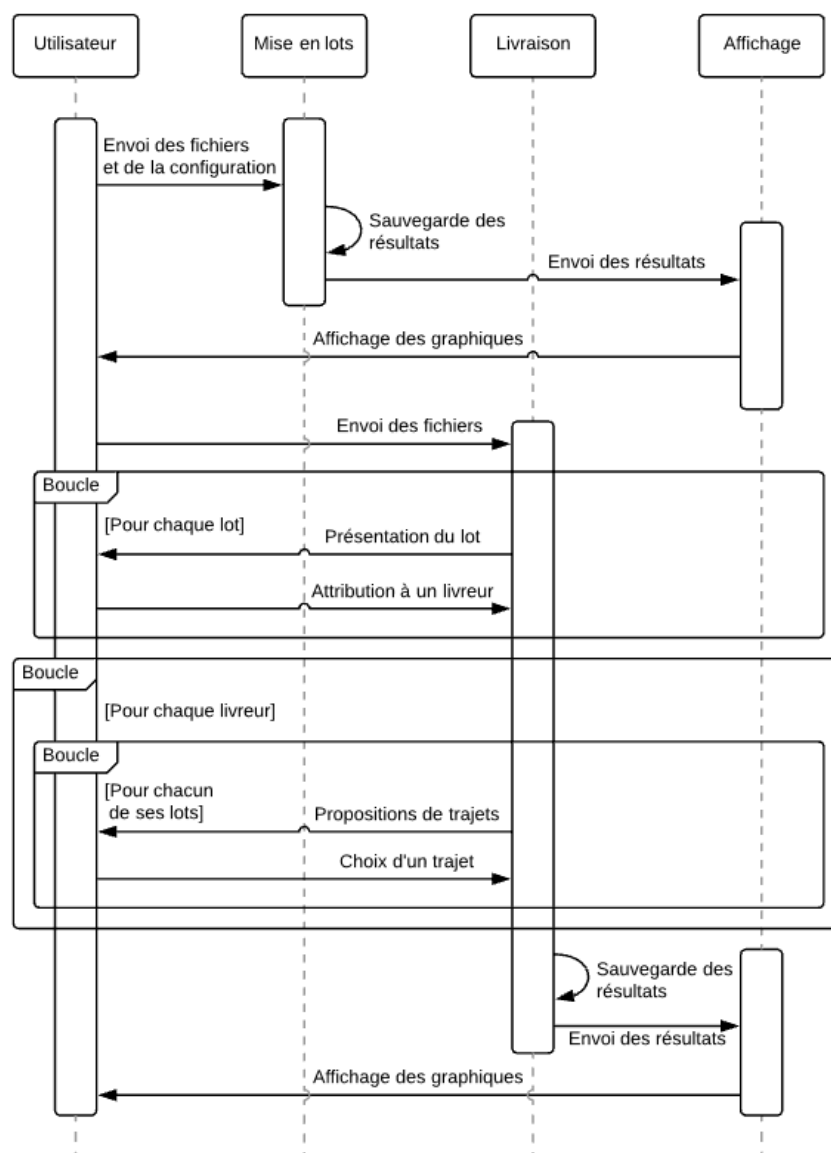


Figure 5 – Diagramme de séquence du livrable

5.3 Performances

Nous nous attendons à un ensemble d'algorithmes capables de calculer la mise en lot plus l'assignement des livraisons en moins de 10 minutes.

5.4 Capacités

Le livrable n'a pas de contraintes liés aux capacités :

- Il est indépendant, donc pas de problème si 2 instances sont lancées en même temps
- Le stockage des données se fait sur des fichiers textes à la fin de chaque étape du traitement, mais les seules contraintes sont celles liées au matériel (pas de taille maximum, donc limité seulement en cas de dépassement de la taille de la mémoire disponible)

5.5 Contrôlabilité

Aucun système de log n'est prévu.

5.6 Sécurité

Il n'y a qu'un seul type d'utilisateur, et rien n'est à sécuriser. Il n'y a donc aucun système de sécurisation des données.

5.7 Intégrité

Aucun système de protection des crash ne sera mis en place. En cas de crash, les calculs devront être réalisés de nouveau.

B

Gestion de projet

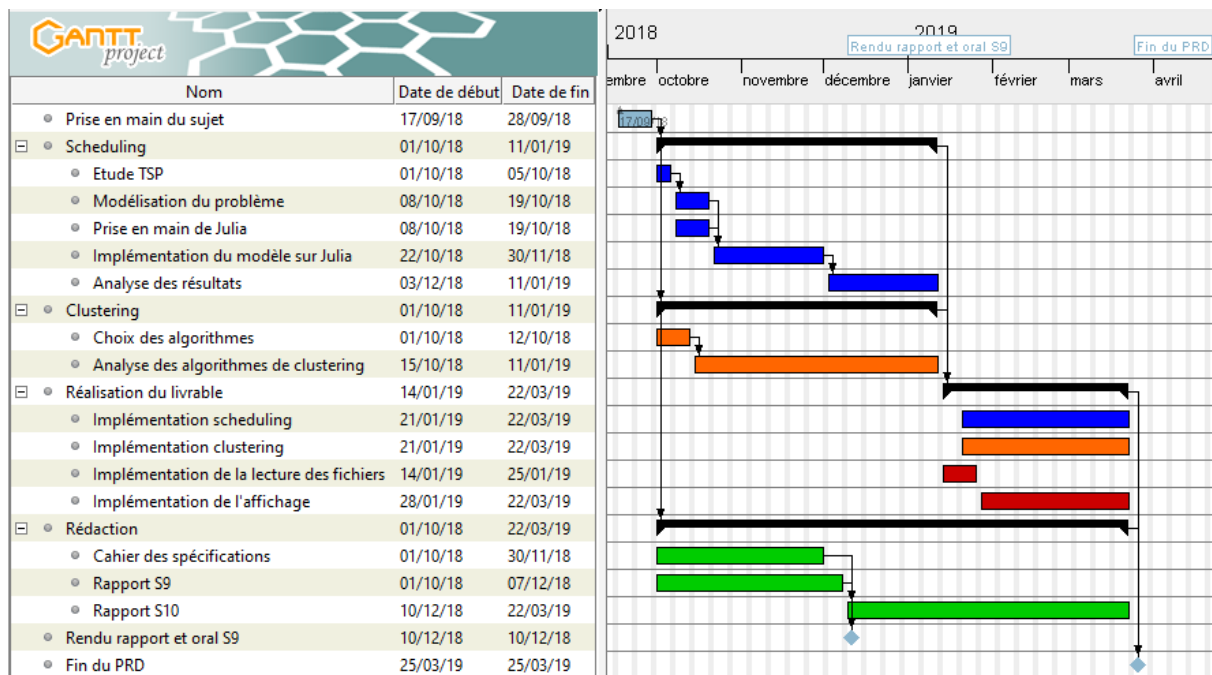


Figure 1 – Gantt S9

La **Figure 1** est le gantt qui avait été réalisé au courant du semestre 9, donc dans la première partie du projet. Le gantt réel du projet est la **Figure 2**. Les différences sont notables dans la partie réalisation du livrable. Il était à l'origine prévu que les fonctions d'écritures/lectures de fichiers soient terminées au plus tôt, et que le reste du temps soit alloué à l'implémentation des différentes parties. Il s'est en réalité avéré que les fonctions de traitement de fichiers ont été beaucoup plus longues à réaliser, et ont notamment demandé de nombreuses modifications, d'où le temps de tâche plus long que prévu. C'est l'inverse pour les tâches d'implémentation de l'existant dans le livrable, cela a été beaucoup rapide que prévu, ce qui a laissé plus de temps pour travailler sur l'interface du logiciel.

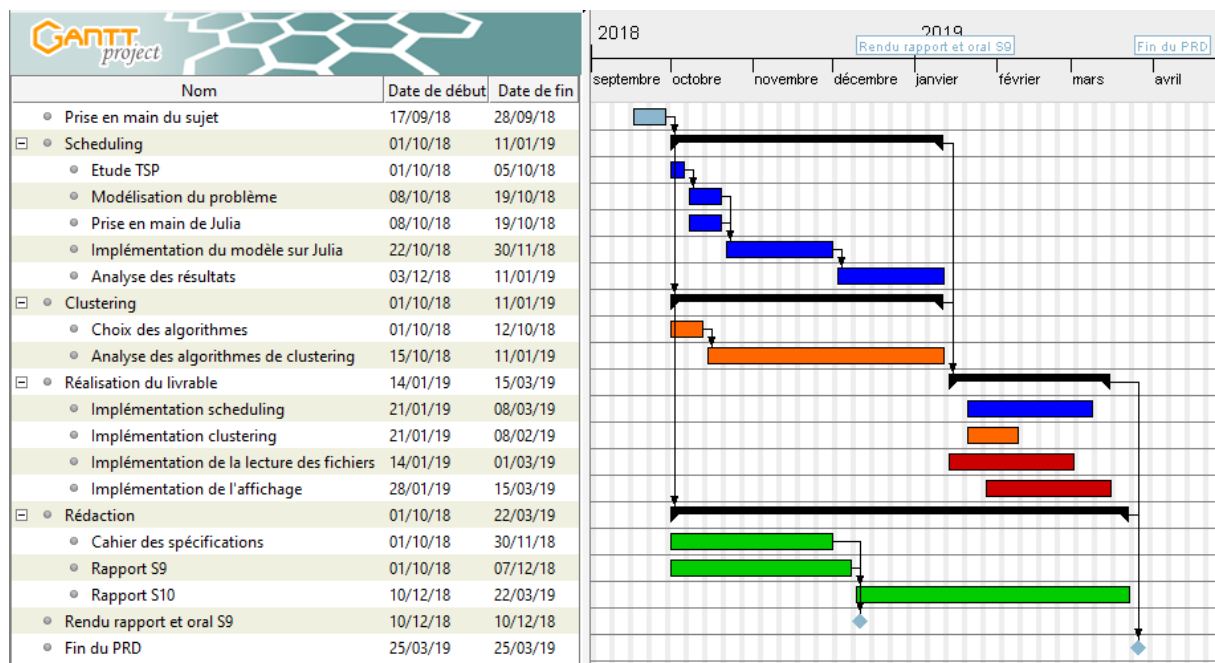


Figure 2 – Gantt S10



Fiches de tests

Table 1 – *Test 1 du Validateur de solution*

Projet	Séquence de test	Identification du test
Chimiothérapie	Fichier solution	test_correct_solution
Description	Fonctionnement du validateur de solution sur résultat valide	
Résultat attendu	Résultat obtenu	Statut
[]	[]	OK

Table 2 – *Test 2 du Validateur de solution*

Projet	Séquence de test	Identification du test
Chimiothérapie	Fichier solution	test_incorrect_solution
Description	Fonctionnement du validateur de solution sur résultat invalide	
Résultat attendu	Résultat obtenu	Statut
["points_not_on_path", "point_not_connected"]	["points_not_on_path", "point_not_connected"]	OK

D

Documentations

DOCUMENTATION DEVELOPPEUR

Ce document présente l'ensemble des actions nécessaires afin d'obtenir un environnement de développement permettant de faire fonctionner le projet.

Il est à noter que le projet n'a été testé que sous Windows 10, et il n'est pas garanti que l'utilisation d'une version antérieure de l'un des composants ne nuise pas au bon fonctionnement du logiciel.

VERSIONS ET INSTALLATION

PYTHON

La majorité du programme repose sur du code en python. Le projet a été réalisé avec Python 3.7.0. Deux librairies externes sont utilisées :

- Sklearn 0.20.0
- Matplotlib 3.0.0

Un fichier nommé « install_packages.bat » permet de lancer l'installation automatique de ces libraires. Il n'est pas nécessaire d'avoir mis python dans le PATH des variables d'environnement du système pour que le script fonctionne.

Tkinter est utilisé pour l'interface graphique, mais il est fourni de base avec la majorité des noyaux python.

JULIA

Le solveur est utilisé par un script Julia. Julia est en version 0.6.4. Il se peut que le programme ne marche pas et/ou que l'installation des différentes dépendances doive être faite différemment sur une version plus récente de Julia. Certaines dépendances sont utilisées :

- CPLEX 0.4.0
- vOptGeneric 0.1.1

Comme pour la partie python, le lancement du batch « install_packages.bat » va installer les librairies nécessaires au fonctionnement du solveur. Il faut néanmoins s'assurer que Julia soit dans le PATH pour que la commande fonctionne. Si vous avez installé Julia dans son répertoire d'origine (« C:\Users\[UTILISATEUR]\AppData\Local\Julia-0.6.4 »), il est possible d'utiliser « set_julia_path.bat » afin de mettre Julia dans les variables d'environnement. Il faut lancer ce script en mode administrateur. Sinon, il est possible de le faire à la main.

Pour que l'installation se passe bien, il faut avoir au préalable installé CPLEX sur son ordinateur, avec une licence valide. Le logiciel a été réalisé avec la version d'essai, qui est gratuite et trouvable sur le site d'IBM.

ARCHITECTURE DU PROJET

Le projet a été développé en suivant le design pattern MVC, il y a donc 3 catégories de classes :

- les classes vues pour l'interface
- les classes modèles pour le code fonctionnel
- les classes contrôleurs pour la liaison vue/modèle

Nous allons vous lister les différentes classes présentes, leur position dans le pattern et leur rôle.

VUE

MAINWINDOW

Cette classe est la classe principale de la vue. Elle contient l'ensemble des autres classes qui gèrent l'interface, et son utilité est d'appeler l'ensemble de ces autres classes afin qu'elles affichent ce qu'il faut. Les 3 classes que MainWindow appelle contiennent des éléments graphiques, et MainWindow fait en sorte que leur contenu soit affiché dans la bonne partie de l'écran.

CLUSTERINGFRAME

C'est une des 3 classes qui affichent du contenu. Cette classe affiche le contenu lié au découpage en lots des produits.

ROUTINGRESOURCESFRAME

Cette classe contient l'ensemble des zones de textes utilisés dans l'interface pour stocker des informations pour la partie livraison. On y retrouve notamment la localisation de nos fichiers d'entrées, le chemin du répertoire de sortie, etc.

ROUTINGFRAME

La dernière classe de la vue et la plus conséquente. Elle contient tous les éléments utiles à l'utilisation du solveur, à l'affichage des résultats, etc. C'est également dans cette classe qu'est instancié la classe RoutingController.

MODELE

CLUSTERING

C'est la classe qui découpe un ensemble de produits en lots. Une fois le découpage fait, elle contient certaines informations comme les centres de gravité de chaque lot, la liste des produits et leur lot, etc. Quand on a fourni les informations nécessaires, il suffit d'appeler « execute » pour que le découpage soit exécuté.

CLUSTERINGPARSER

Dans notre programme, on prend un certain nombre de fichiers en entrée, que l'on découpe en des plus petits fichiers, un pour chaque lot, comprenant uniquement les valeurs des produits concernés. Cette classe permet la lecture de chacun des fichiers, et leur découpage.

SOLUTIONTEST

C'est avec cette classe que l'on test si une solution est valide ou non. Elle test la solution fournie et test différentes choses, afin de s'assurer que la solution que l'on a calculé est valide par rapport à notre modèle mathématique.

VEHICLESPARSER

C'est la classe qui permet de lire et écrire dans les fichiers liés aux liaisons lot/livreur.

SOLVERERROR

Enumérateur qui contient la liste des erreurs testées par Solutiontest.

FILENAMEENUM

Enumérateur qui contient la liste des noms de fichiers utilisés dans le logiciel.

CONTROLEUR

ROUTINGCONTROLLER

RoutingController est la classe qui gère l'ensemble des interactions possibles entre le modèle et la vue pour la partie livraison. On y retrouve notamment les fonctions d'écriture/lecture des fichiers liés à cette partie, et la fonction qui lance le script du solveur avec les paramètres souhaités.

CLUSTERINGCONTROLLER

C'est la classe contrôleur liée à la mise en lots. Cette classe contient une unique fonction, qui exécute le découpage en fonction des paramètres fournis.

DOCUMENTATION UTILISATEUR

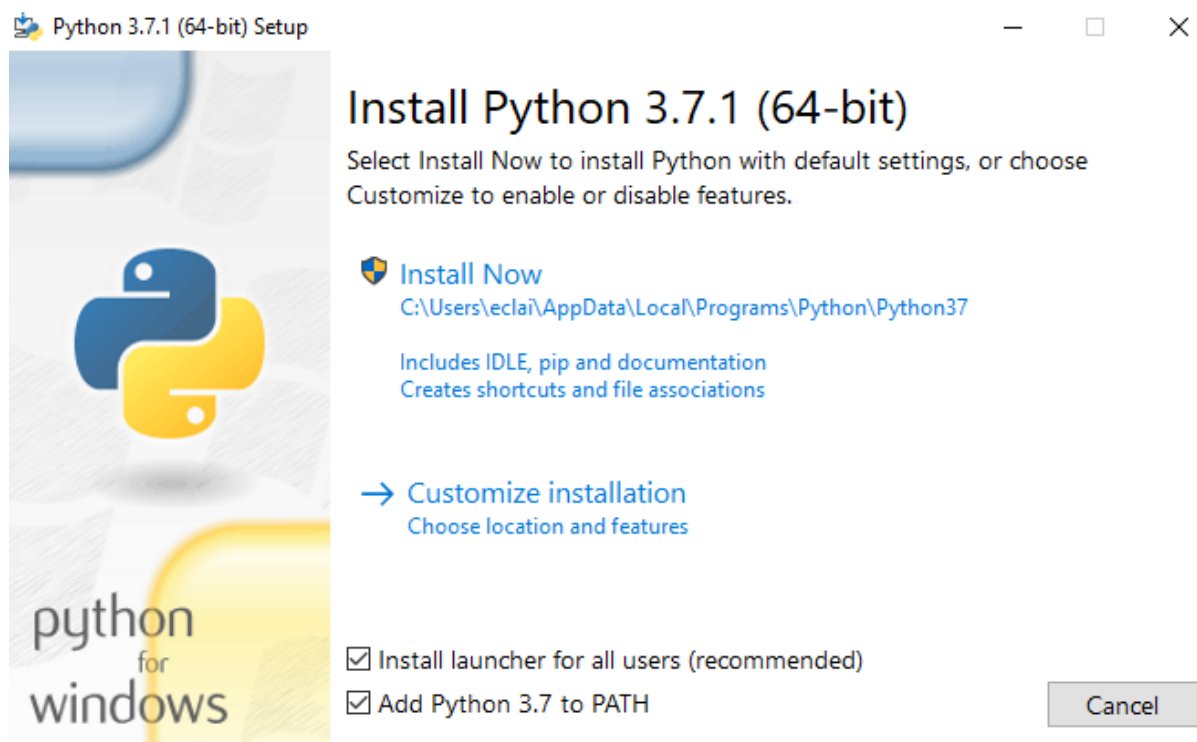
Ce document présente la façon d'installer le logiciel, et les différentes parties de l'interface graphique.

Le fonctionnement du logiciel sur un système autre que Windows 10 n'est pas garanti.

INSTALLATION

PYTHON

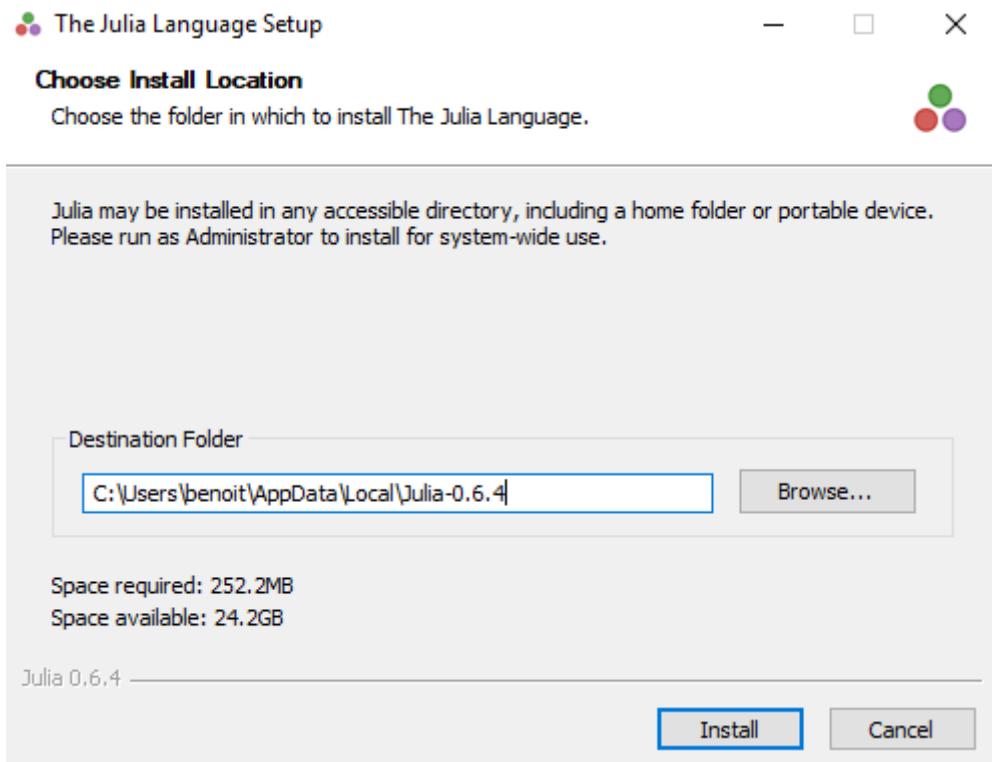
La première chose à installer est python. Vous pourrez trouver le fichier à télécharger sur <https://www.python.org/downloads/>. Le logiciel est fonctionnel avec Python 3.7.0 et les versions suivantes.



Dans cette fenêtre, il est recommandé de cocher « Add Python 3.7 to PATH ». Ensuite, cliquez sur « Install Now ». Une fois que l'installation est terminée, vous pouvez fermer la fenêtre.

JULIA

Ensuite, il faut installer Julia 0.6.4. Téléchargez-le depuis cette page <https://julialang.org/downloads/oldreleases.html>. Lorsque vous lancez l'installateur, cette fenêtre devrait apparaître :



Il est fortement recommandé de laisser le chemin d'installation par défaut pour pouvoir simplifier l'installation de ce qui suit. Vous pouvez ensuite lancer l'installation puis fermer la fenêtre une fois que tout est complété.

CPLEX

Rendez-vous sur le site <https://www.ibm.com/analytics/cplex-optimizer>, et téléchargez « IBM ILOG CPLEX Optimization Studio », en version d'essai. Il n'y a pas de contraintes particulières à son installation, il suffit de le télécharger et de l'installer.

FINALISATION DE L'INSTALLATION

Une fois que Python, Julia et CPLEX ont été installés, il va falloir installer différents modules nécessaires au bon fonctionnement du logiciel. Dans le dossier du programme, il y a un dossier « installation ». Lancez le fichier « set_julia_path » en mode Administrateur (cliquer droit > exécuter en tant qu'administrateur). Si vous n'aviez pas laissé le chemin d'installation par défaut pour Julia, vous avez plusieurs possibilités :

1. Cliquez droit sur le fichier > modifier, puis remplacez « C:\Users\%username%\AppData\Local\Julia-0.6.4\bin » par votre chemin d'installation
2. Modifier vos variables d'environnement vous-même afin d'y inclure Julia

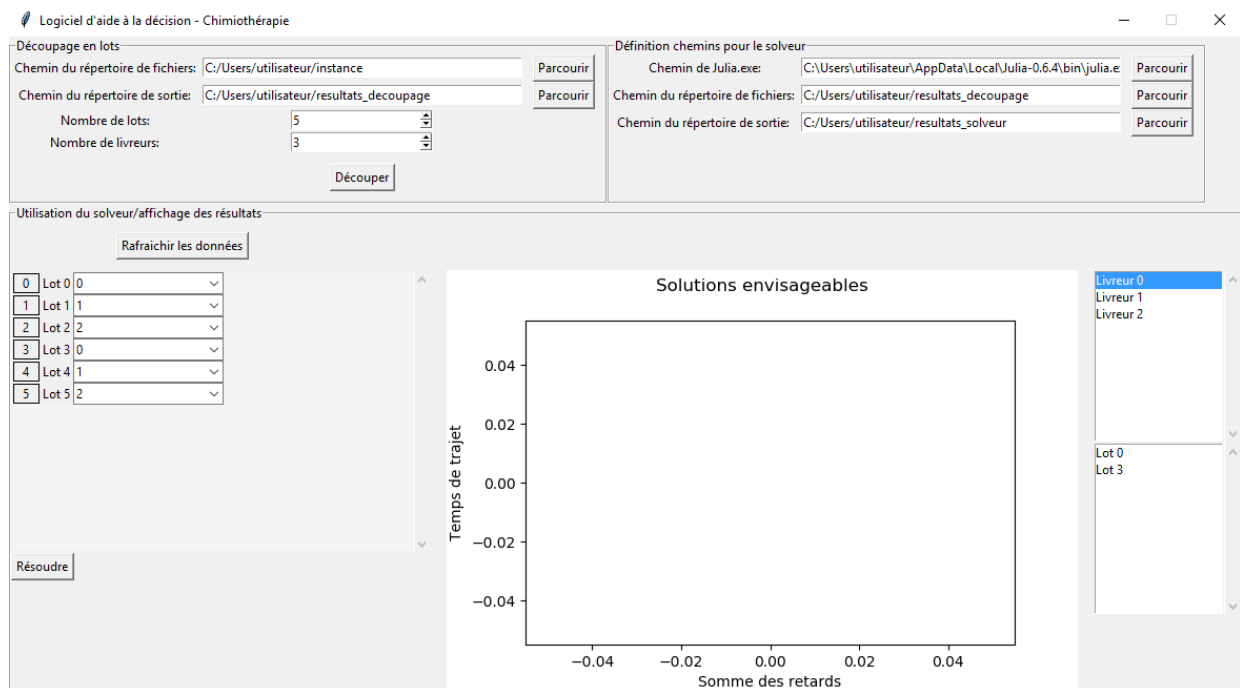
Une fois que vous avez effectué l'une de ces actions, vous pouvez lancer « install_packages ». Pas besoin de le lancer en tant qu'administrateur. Quand le message « Environnement fonctionnel » apparaît, c'est que l'installation est terminée, et que le logiciel est prêt à être utilisé.

UTILISATION

Afin de lancer le logiciel, il faut lancer le fichier « Chimiothérapie », qui se trouve à la racine du projet.

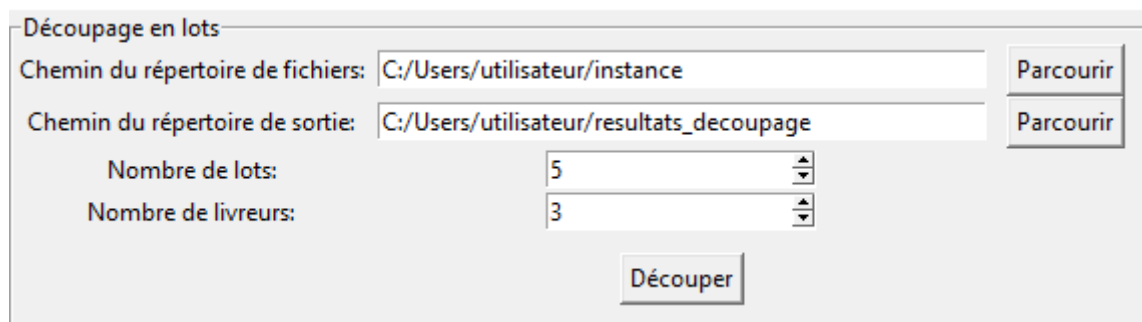
INTERFACE

Voici l'interface générale :



Nous allons détailler ses 3 sous parties.

DECOUPAGE EN LOTS



Le but de cette partie est de spécifier dans « Chemin du répertoire de fichiers » le dossier qui contient l'ensemble des fichiers nécessaires au découpage en lots (voir chapitre « Fichiers types »), et de spécifier dans « Chemin du répertoire de sortie » là où les fichiers générés vont être stockés. Il faut ensuite spécifier le nombre de lots voulus et le nombre de livreurs disponibles. Une fois que toutes les informations sont renseignées, il faut appuyer sur « Découper » pour lancer le découpage en lots de notre instance.

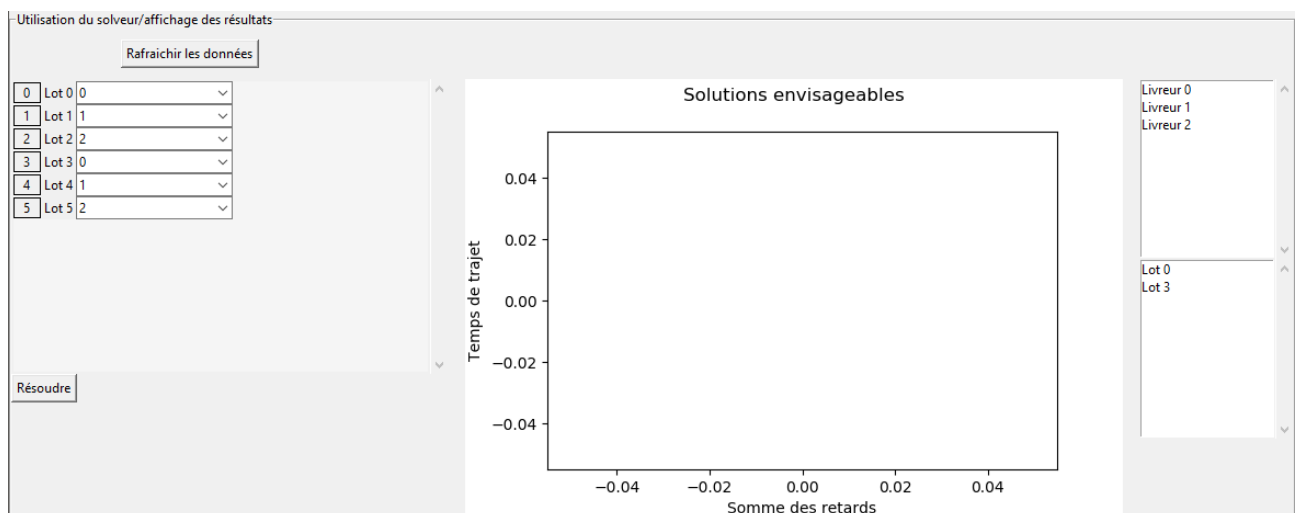
DEFINITION DES CHEMINS POUR LE SOLVEUR

Définition chemins pour le solveur

Chemin de Julia.exe:	C:\Users\utilisateur\AppData\Local\Julia-0.6.4\bin\julia.exe	Parcourir
Chemin du répertoire de fichiers:	C:/Users/utilisateur/resultats_decoupage	Parcourir
Chemin du répertoire de sortie:	C:/Users/utilisateur/resultats_solveur	Parcourir

Ici, il faut commencer par rentrer le chemin vers julia.exe, qui est utilisé pour la résolution de nos problèmes d'acheminement. De base, ce champ vaut « C:\Users\nom_d_utilisateur\AppData\Local\Julia-0.6.4\bin\julia.exe », car c'est ici qu'il est installé si on suit l'installation par défaut qui est conseillée. Il faut ensuite, comme pour la partie précédente, spécifier le dossier contenant les fichiers d'entrés et le dossier de sortie. Ici, le dossier d'entrée doit être le même que le dossier de sortie de la partie précédente, puisque que c'est les fichiers précédemment créés que nous allons ici utiliser.

UTILISATION/AFFICHAGE



Cette dernière partie permet de visualiser nos résultats, de modifier les affectations livreur/lot, et de lancer la résolution de notre problème. En haut à gauche, il y a un bouton « Rafraichir les données », qui sert à actualiser nos données avec les fichiers trouvés dans les dossiers spécifiés précédemment. Si vous modifiez des fichiers ou des répertoires de dossier, appuyez sur ce bouton pour vous assurer que les données sont bien actualisées.

En dessous, il y a une liste de lot, avec pour chaque lot la possibilité d'attribuer un livreur. Dans cet exemple, nous avons l'attribution suivante :

- Livreur 0 : lots 0 et 3
- Livreur 1 : lots 1 et 4
- Livreur 2 : lots 2 et 5

Quand les attributions vous conviennent, vous pouvez appuyer sur « Résoudre ». Il va falloir un certain temps pour que le solveur résolve l'ensemble des trajets. Une fois fini, vous aurez à droite de l'écran la possibilité de cliquer sur un livreur, puis de cliquer sur l'ensemble des lots qu'il livre. Vous aurez alors sur le graphique du milieu un ensemble de points correspondant à des trajets de livraison. Il y a deux coordonnées : « Somme des retard », qui équivaut à la somme des retards sur chaque produit à livrer, et « Temps de trajet », qui est le temps total de trajet pour livrer le lot.

FICHIERS TYPES

Dans le dossier que vous spécifiez dans « Chemin du répertoire de fichiers » doivent se trouver les fichiers suivants :

- deadlines.txt : liste de la deadline de chaque produit
- distances.txt : matrice de distances produit/produit ou service/service
- dueDates.txt : l'heure due de chaque produit
- jobLocalisation.txt : le service dans lequel se trouve chaque produit (de base est la suite : 1, 2, 3, etc.)
- localizations.txt : la position de chaque produit ou de chaque service

Un exemple de dossier type est présent dans le répertoire du logiciel.

Le découpage de ces fichiers va résulter en la création de nombreux fichiers, qui vont ensuite être utilisés par le solveur. Chaque étape du processus génère différents fichiers, qui se trouvent dans les différents dossiers spécifiés par l'utilisateur.

Bibliographie

- [1] scikit-learn DEVELOPERS. 2.3. *Clustering - scikit-learn 0.20.1 documentation*. URL : <https://scikit-learn.org/stable/modules/clustering.html#clustering>.
- [2] Maximilian JÄGER. *The Traveling Salesman Problem - A Quantum Computing Approach for Bounded-Degree Graphs*. Août 2018. URL : <http://www.math.uni-leipzig.de/~grad/soa/Jaeger-TSP.pdf>.
- [3] *List of algorithms*. URL : https://en.wikipedia.org/wiki/List_of_algorithms.
- [4] *Recent Advances in Multi-Objective Optimization*. Nov. 2018. URL : <https://moo.univie.ac.at/>.
- [5] Clemens THIELEN. *Duty Rostering for Physicians at a Department of Orthopedics and Trauma Surgery : Decision Support using Mathematical Optimization*. Nov. 2018. URL : <https://moo.univie.ac.at/wp-content/uploads/2018/11/11-rostering.pdf>.
- [6] Vincent T'KDINT. *Quantifying the hardness of the enumeration of Pareto optima : a theoretical framework with application to scheduling problems*. Nov. 2018. URL : https://moo.univie.ac.at/wp-content/uploads/2018/11/1-Tkindt_RAM00.pdf.

Livraison multicritères de produits de chimiothérapie

Benoît Vacher

Encadrement : Alexis Robbes

Objectifs

Ce projet a pour but de proposer une solution au problème de mise en lots et de livraison de produits de chimiothérapie sur les sites du CHRU de Tours. Il faut:

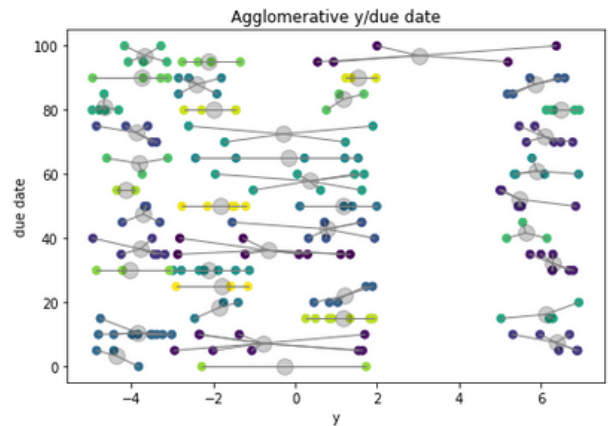
- Découper des produits en lots
- Attribuer un livreur à chaque lot
- Optimiser le trajet de livraison de chaque lot

Solution

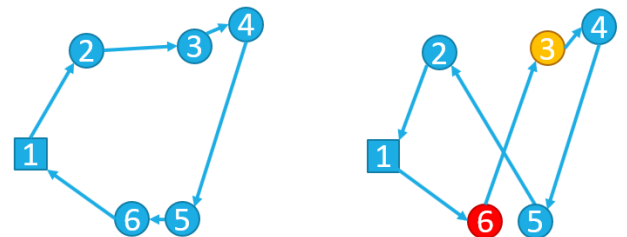
La solution fournie est un logiciel d'aide à la décision. Ce logiciel est capable de découper un ensemble de produits en lots, puis, après que l'utilisateur ait attribué un livreur à chaque lot, il calcule les trajets possibles pour livrer chaque lot

Conclusion

Le logiciel fonctionne, mais il y a moyen de l'améliorer afin de le rendre plus ergonomique et plus efficace en vitesse de calcul

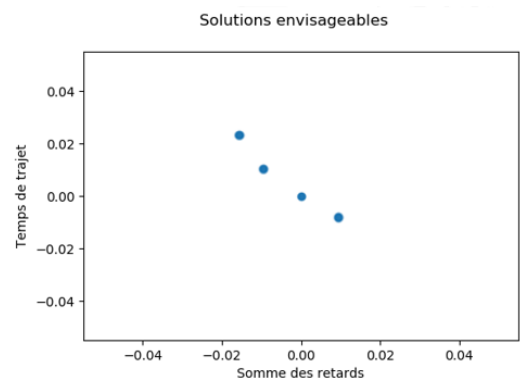


Représentation de 200 produits découpés en 40 lots: position géographique/date due



Minimiser l'heure de retour Minimiser la somme des retards des produits

Différents trajets de livraison en fonction des objectifs



Solutions de livraison d'un lot: somme des retards/temps de trajet

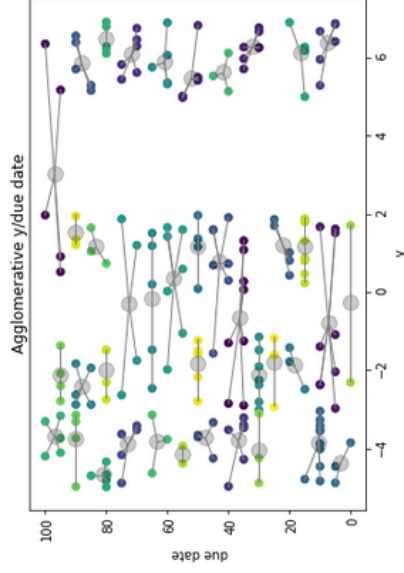
Livraison multicritères de produits de chimiothérapie

Benoît Vacher

Encadrement : Alexis Robbes

Objectifs

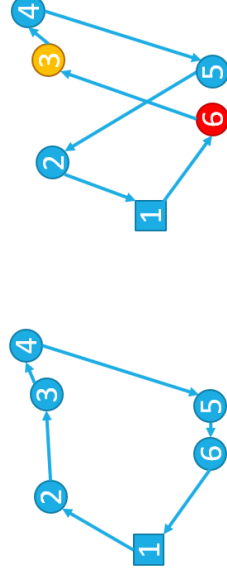
- Ce projet a pour but de proposer une solution au problème de mise en lots et de livraison de produits de chimiothérapie sur les sites du CHRU de Tours. Il faut :
- Découper des produits en lots
 - Attribuer un livreur à chaque lot
 - Optimiser le trajet de livraison de chaque lot



Représentation de 200 produits découpés en 40 lots: position géographique/date due

Solution

La solution fournie est un logiciel d'aide à la décision. Ce logiciel est capable de découper un ensemble de produits en lots, puis, après que l'utilisateur ai attribué un livreur à chaque lot, il calcule les trajets possibles pour livrer chaque lot



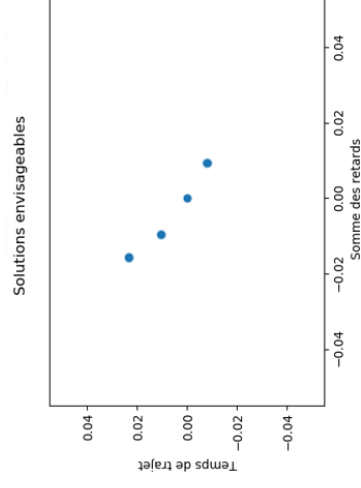
Minimiser l'heure de retour

Minimiser la somme des retards des produits

Différents trajets de livraison en fonction des objectifs

Conclusion

Le logiciel fonctionne, mais il y a moyen de l'améliorer afin de le rendre plus ergonomique et plus efficace en vitesse de calcul



Solutions de livraison d'un lot:

somme des retards/temps de trajet



Livraison multicritères de produits de chimiothérapie

Résumé

Optimisation de la mise en lot de produits de chimiothérapie, et optimisation du trajet de livraison de chaque lot

Mots-clés

Voyageur de commerce, Algorithmes de partitionnement, K-Means, Partitionnement agglomératif, Multi-Objectif

Abstract

Optimization of batching of chemotherapy products, and optimization of the delivery path of each batch

Keywords

Travelling salesman problem, Clustering algorithms, K-Means, Agglomerative clustering, Multi-Objective