



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

**Projet Recherche & Développement  
2021-2022**

# **Logistique des circuits courts / Production-Distribution**



**POLYTECH<sup>®</sup>  
TOURS**

**Entreprise**

**Polytech**



**Tuteur entreprise**

**Jean Charles BILLAUT**

**Étudiant**

**Thomas DUVAL (DI5)**

**Tuteur académique**

**Jean Charles BILLAUT**

11 avril 2022

# Liste des intervenants

## Entreprise

Polytech  
64 avenue Jean Portalis  
37200 Tours, France  
[polytech.univ-tours.fr](http://polytech.univ-tours.fr)



Nom	Email	Qualité
Thomas DUVAL	<a href="mailto:email@univ-tours.fr">email@univ-tours.fr</a>	Étudiant DI5
Jean Charles BILLAUT	<a href="mailto:jean-charles.billaut@univ-tours.fr">jean-charles.billaut@univ-tours.fr</a>	Tuteur académique, Département Informatique
Jean Charles BILLAUT	<a href="mailto:jean-charles.billaut@univ-tours.fr">jean-charles.billaut@univ-tours.fr</a>	Tuteur entreprise



# Avertissement

Ce document a été rédigé par Thomas DUVAL susnommé l'auteur.

L'entreprise Polytech est représentée par Jean Charles BILLAUT susnommé le tuteur entreprise.

L'Ecole Polytechnique de l'Université de Tours est représentée par Jean Charles BILLAUT 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 assume 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

Thomas DUVAL, *Logistique des circuits courts / Production-Distribution*, Projet Recherche & Développement, Ecole Polytechnique de l'Université de Tours, Tours, France, 2021-2022.

```
@mastersthesis{
  author={DUVAL, Thomas},
  title={Logistique des circuits courts / Production-Distribution},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2021-2022}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	v
<b>1 Introduction</b>	<b>1</b>
1 Acteurs, enjeux et contexte .....	1
2 Objectifs .....	2
3 Hypothèses .....	2
4 Bases méthodologiques.....	3
<b>2 Description générale</b>	<b>4</b>
1 Environnement du projet .....	4
2 Caractéristiques des utilisateurs .....	4
3 Fonctionnalités du système .....	5
4 Structure générale du système.....	5
<b>3 État de l'art / Veille technologique</b>	<b>7</b>
1 Problème de l'ordonnancement de la production .....	7
1.1 Planification d'opérations de récolte de fermes à fermes .....	7
1.2 RCPSP .....	8
2 Problème de distribution de la récolte .....	8
2.1 Problème CRS .....	9

<b>4</b>	<b>Analyse et conception</b>	<b>14</b>
1	Analyse .....	14
1.1	Hypothèses utilisées .....	14
1.2	Spécifications.....	14
1.2.1	Spécifications fonctionnelles .....	14
	Lecture d'instances.....	14
	Résolution d'instance .....	14
	Ecriture d'instance .....	14
	Sauvegarde de résultat .....	15
1.2.2	Spécifications non fonctionnelles .....	15
2	Modélisation proposée.....	15
2.1	Paramètres .....	15
2.2	Variables .....	16
2.3	Contraintes.....	16
2.3.1	Contraintes de la fonction objectif .....	16
2.3.2	Contraintes sur la première machine : la phase de pousse.....	17
2.3.3	Contraintes sur la seconde machine : la phase de récolte .....	18
2.3.4	Contraintes sur les demandes.....	19
2.3.5	Contraintes de liaison entre z et x.....	20
2.3.6	Contraintes de consommation .....	20
2.3.7	Contraintes de planification de la distribution.....	21
2.3.8	Contraintes d'accélération.....	22
2.4	Fonction objectif.....	23
<b>5</b>	<b>Mise en oeuvre</b>	<b>24</b>
1	Outils et librairie utilisés.....	24
2	Éléments d'implémentation, choix techniques .....	24
3	Analyse des résultats, évaluation, qualité .....	28
<b>6</b>	<b>Bilan et conclusion</b>	<b>29</b>
1	Bilan du semestre 9 .....	29
2	Bilan du semestre 10.....	29
3	Bilan sur la qualité .....	29
4	Bilan auto-critique.....	30
	<b>Annexes</b>	<b>31</b>
<b>A</b>	<b>Planification, gestion de projet</b>	<b>32</b>
1	Evolution du projet .....	32
2	Description des tâches.....	35

2.1	Tâche 1 : Etat de l'art .....	35
2.2	Tâche 2 : Rédaction du rapport PRD1.....	35
2.3	Tâche 3 : Description du projet.....	35
2.4	Tâche 4 : Cahier de spécifications .....	35
2.5	Tâche 5 : Analyse et conception du modèle mathématique .....	36
2.6	Tâche 6 : Rédaction du rapport PRD2.....	36
2.7	Tâche 7 : Génération d'instances.....	36
2.8	Tâche 8 : Modèle de simulation.....	36
2.9	Tâche 9 : Implémentation du modèle .....	36
2.10	Tâche 10 : Création d'heuristiques .....	36
2.11	Tâche 11 : Expérimentations.....	37
2.12	Tâche 12 : Préparation soutenance.....	37
<b>B</b>	<b>Cahier de Spécifications</b> .....	<b>38</b>
1	Spécifications Fonctionnelles .....	38
1.1	Fonctionnalités à développer .....	38
1.2	Définition de la fonction 1 : Lecture d'instance .....	38
	Présentation de la fonction 1 : .....	38
	Description de la fonction 1 : .....	38
1.3	Définition de la fonction 2 : Ecriture d'instance .....	38
	Présentation de la fonction 2 : .....	38
	Description de la fonction 2 : .....	39
1.4	Définition de la fonction 3 : Résolution d'instance .....	39
	Présentation de la fonction 3 : .....	39
	Description de la fonction 3 : .....	39
1.5	Définition de la fonction 4 : Sauvegarde de résultat .....	39
	Présentation de la fonction 4 : .....	39
	Description de la fonction 4 : .....	39
2	Spécifications non fonctionnelles .....	39
2.1	Contraintes de développement et conception .....	39
2.2	Contraintes de fonctionnement et d'exploitation.....	40
2.2.1	Performances.....	40
2.2.2	Capacités .....	40
2.2.3	Contrôlabilité .....	40
2.2.4	Sécurité .....	40
2.3	Contraintes d'évolution.....	40
<b>C</b>	<b>Cahier du développeur</b> .....	<b>41</b>
1	Diagrammes architecturaux et UML .....	41
1.1	Structure du projet.....	41
1.2	Architecture de résolution d'une instance .....	42

<b>D</b>	<b>Document d'installation</b>	<b>44</b>
<b>E</b>	<b>Document d'utilisation</b>	<b>45</b>
<b>F</b>	<b>Cahier de test</b>	<b>46</b>
1	Tests unitaires .....	46
1.1	Tests du générateur d'instance .....	46
1.2	Tests du Solveur .....	47
2	Scénario de test .....	47
<b>G</b>	<b>Bibliographie</b>	<b>50</b>
<b>H</b>	<b>Acronymes</b>	<b>51</b>

# Table des figures

## 2 Description générale

2.1 Diagramme de la structure du logiciel .....	6
---	---

## 3 État de l'art / Veille technologique

3.1 Description de la taille des instances .....	11
3.2 Temps de résolution des méthodes proposées.....	12
3.3 Ecart avec la borne inférieure en pourcentage .....	13

## 5 Mise en oeuvre

5.1 Tableau du temps d'exécution d'une solution en fonction des instances et de la méthode utilisée.....	28
5.2 Tableau de la qualité d'une résolution en fonction des instances et de la méthode utilisée.....	28

## A Planification, gestion de projet

A.1 Le diagramme de Gantt prévisionnel.....	32
A.2 Le diagramme de Gantt à la fin du semestre 9 .....	33
A.3 Le diagramme de Gantt prévisionnel à la fin du semestre 10 .....	33
A.4 Le diagramme de Gantt final à la fin du projet .....	34
A.5 Tableau pour le diagramme de Gantt prévisionnel.....	34
A.6 Tableau pour le diagramme de Gantt à la fin du semestre 9 .....	34
A.7 Tableau pour le diagramme de Gantt prévisionnel pour le semestre 10 .....	34
A.8 Tableau pour le diagramme de Gantt à la fin du projet .....	35

**C Cahier du développeur**

C.1 UML du programme avec les principales fonctions .....	42
C.2 Le diagramme de résolution d'une instance .....	43

**F Cahier de test**

F.1 Exemple de résolution faisable .....	48
F.2 Exemple de résolution infaisable .....	48
F.3 Exemple d'erreur de résolution.....	49

# 1

## Introduction

### 1 Acteurs, enjeux et contexte

Dans ce projet, seuls la **Maitrise d'Ouvrage (MOA)** représentée par M. Billaut et la **Maitrise d'Oeuvre (MOE)** représentée par moi seront présentes. En effet, bien que ce sujet de recherche se base sur des besoins concrets, aucun client n'est présent à ce moment-là de la réalisation.

Grâce à l'automatisation toujours croissante des moyens de production dans le milieu agricole, la production de nourriture en France dépasse désormais largement les besoins[3]. Mais de nos jours, de nouveaux critères d'achats de produits apparaissent dans tous les secteurs y compris dans le monde agricole et les grandes exploitations ne répondent plus forcément à toutes ces nouvelles demandes. Ainsi dans les années 1960, le commerce équitable fait son apparition mettant en avant d'autres valeurs comme la parité, les conditions de travail ou l'écologie plutôt qu'uniquement l'aspect du produit. On peut également parler de l'apparition de l'agriculture biologique en 1999 mettant là encore l'accent sur une production écologique des produits.

Dernièrement, ce sont les circuits courts qui font leur apparition notamment dans le milieu agricole. De nombreuses raisons peuvent expliquer cet élan des consommateurs pour ce mode de production : les produits sont plus variés et souvent moins chers, l'agriculteur est mieux rémunéré et la production de proximité à petite échelle pollue moins. Les circuits courts sont caractérisés par un nombre d'intermédiaires très faibles (aucun ou un seul intermédiaire). L'avantage pour un agriculteur de réduire le nombre d'intermédiaires est de pouvoir générer de meilleurs revenus, cependant cela ajoute une contrainte d'organisation supplémentaire pour pouvoir livrer sa production, mais aussi pour prévoir l'utilisation des champs afin d'éviter une sous-exploitation ou une sur-exploitation pouvant mener à des pertes financières que ce soit à cause d'une production trop faible, à cause de pertes de rendements ou à cause de retards.

## 2 Objectifs

L'objectif principal du projet est donc de réaliser un programme permettant de planifier automatiquement à la fois des plannings de productions mais aussi des plannings de livraisons afin de respecter les délais et les quantités des livraisons des clients.

La première partie du problème est un problème d'ordonnancement. Le but est de positionner toutes les tâches nécessaires à l'obtention de la production au moment voulu pour pouvoir la livrer. La complexité de cette partie est que les tâches de production de nourriture ne possèdent pas des temps constants : en effet certains aliments peuvent se récolter ou se conserver plus longtemps ce qui permet d'allonger la durée des tâches tandis que d'autres aliments peuvent se récolter plus tôt ce qui permet de raccourcir la durée des tâches. Cependant, ces modifications ne se font pas sans conséquences, dans le cas où la durée optimale n'est pas respectée, des pertes ont lieu. Un dernier paramètre peut néanmoins modifier la durée d'une tâche mais cette fois sans conséquence pour la production : la main d'œuvre à affecter en fonction des tâches et qui peut accélérer certaines parties d'une tâche lorsque la main d'œuvre augmente.

La deuxième partie du problème est un problème de tournée. Le but est de livrer la production obtenue dans la première partie en minimisant les coûts. Les coûts peuvent augmenter pour plusieurs raisons comme à cause des retards ou bien de la distance parcourue par le véhicule de l'agriculteur permettant d'estimer l'usure du véhicule. Ce problème se rapproche du problème de tournée de véhicule dont le principe est d'effectuer différents circuits à partir d'un point d'origine afin de desservir toutes les destinations. Cependant, pour simplifier les calculs et étant donné la faible probabilité qu'un agriculteur fasse plusieurs tournées dans la même journée, je me rapprocherai d'un problème du voyageur de commerce ou cette fois-ci chaque destination doit être atteinte en un seul circuit.

Dans un premier temps, le but est de créer un modèle fonctionnel permettant de résoudre des instances à l'optimal. Cependant, à cause de la grande complexité du problème, nous avons décidé dans un second temps de développer des heuristiques, c'est-à-dire des méthodes permettant d'obtenir des solutions non optimales mais en beaucoup moins de temps. Le but est alors de lancer des idées qui seront développées dans de futurs projets pour trouver un équilibre entre la recherche d'un bon résultat et un temps de calcul raisonnable.

## 3 Hypothèses

Différentes hypothèses doivent être posées concernant la fiabilité du modèle et sa capacité à être transposé dans un cas concret. En effet, la production agricole est influencée par de très nombreux facteurs. Il est donc nécessaire de prendre en compte que tous les facteurs suivants seront négligés dans la résolution d'un problème :

- La météo
- La saisonnalité
- L'indisponibilité de main d'oeuvre
- Les arrêts maladie

De plus nous poserons différentes hypothèses concernant les données des instances :

- On suppose que les commandes reçues par l'agriculteur permettent la livraison en une seule journée. Cela revient à partir du principe que l'agriculteur refuserait une commande s'il a déjà trop de livraisons dans la même journée ou si la distance pour livrer est trop grande
- On suppose que la capacité du véhicule de l'agriculteur est adapté aux offres de livraison qu'il reçoit et donc qu'il n'existe pas de limite de volume ou de poids pour le véhicule pour un circuit

## 4 Bases méthodologiques

J'ai utilisé différents logiciels pour m'aider dans ma gestion de projet.

- Bitrix24 qui permet de créer des listes de tâches, d'organiser des plannings et de visualiser l'avancement des tâches.
- Google drive qui permet de stocker en ligne une partie des documents permettant d'y accéder en ligne
- Diagrams.net qui permet de créer des diagrammes (**Unified Modeling Language (UML)** et autres)

Une méthodologie agile sera utilisée. Cette méthodologie semble adaptée étant donnée la facilité avec laquelle il est possible d'échanger avec la **MOA** en cas de problème ou d'incertitude mais aussi parce que cela permet de s'assurer de la validité du développement du logiciel au cours de la réalisation.

# 2

## Description générale

### 1 Environnement du projet

Il n'existe aucun existant que ce soit un logiciel sur le marché, un logiciel en cours de création ou simplement un modèle permettant de répondre aux attentes du problème. En effet, il est possible de trouver des logiciels permettant de résoudre certaines parties du problème mais pas dans sa globalité.

J'ai donc pu décider personnellement de l'environnement dans lequel le projet sera développé. Pour faciliter la lecture du code, j'ai choisi un langage simple et connu : python, en version 3.8. Cette version date du 14 octobre 2019 et sera maintenue au moins jusqu'en octobre 2024. Il s'agit donc d'une version suffisamment récente et qui pourra facilement être téléchargée. Un autre avantage de cette version est qu'elle est compatible avec windows 7 contrairement aux versions suivantes. Ainsi, les risques d'incompatibilités sont réduits. Le système d'exploitation utilisé est windows étant donné que c'est le plus répandu. Pour le solveur qui permet de calculer des solutions pour les instances, il a d'abord été envisagé d'utiliser Cplex Community Edition. Il s'agit d'un solveur gratuit dans la limite d'une utilisation non commerciale et un des meilleurs solveurs actuellement. Cependant, c'est finalement le solveur GLPK qui a été retenu. Il s'agit également d'un solveur très efficace mais également complètement gratuit d'utilisation. Ce solveur a été privilégié car il offre la possibilité de vérifier très rapidement si une instance possède au moins une solution réalisable.

### 2 Caractéristiques des utilisateurs

Comme expliqué précédemment, il n'y a pas de clients pour le moment étant donné que le projet est encore en phase de conception. Les utilisateurs sont donc limités aux différents acteurs du projet.

### 3 Fonctionnalités du système

Le système possède plusieurs fonctionnalités dont la nécessité de réalisation est catégorisé en 3 niveaux :

- primordial : fonction clef sans laquelle le programme ne peut fonctionner
- important : fonction importante pour permettre une utilisation simple du programme
- facultatif : fonction qui ajoute une fonctionnalité n'étant pas nécessaire au fonctionnement du programme

Le programme sera développé sous la forme de scripts python. La fonctionnalité principale du programme est la capacité à interagir avec le solveur GLPK afin de permettre la résolution d'instances mais aussi l'affichage des résultats et même des informations liés à la résolution pendant l'exécution. Il s'agit donc d'une fonction primordiale qui sera également la partie la plus longue à développer. Pour pouvoir grandement faciliter l'utilisation de l'application, une fonction devra permettre la création d'un fichier d'instance et une autre fonction devra permettre la lecture de ce fichier pour pouvoir utiliser l'instance dans le solveur. Etant donné que la manipulation à la main de tels fichiers peut s'avérer compliquée et qu'il est préférable de pouvoir enregistrer des instances, la fonction de lecture de fichier et la fonction d'écriture de fichier sont des fonctions primordiales. Les fonctions permettant de lancer des heuristiques ne sont pas nécessaires pour l'intégrité du programme, mais elles sont tout de même d'une grande utilité pour permettre la résolution d'instances de plus grande taille. Pour cette raison, ces fonctions sont importantes. Enfin, une fonctionnalité devra permettre à l'utilisateur de sauvegarder la solution dans un fichier. Cette fonction est facultative.

Dans le cas où l'implémentation de ces fonctionnalités serait terminée suffisamment tôt, il serait envisagé d'ajouter une interface graphique. Dans ce cas, une discussion avec la MOA serait nécessaire afin de prévoir l'ajout d'autres fonctionnalités comme par exemple un menu permettant de sélectionner l'action à effectuer ou un moyen de visualiser la solution d'une instance.

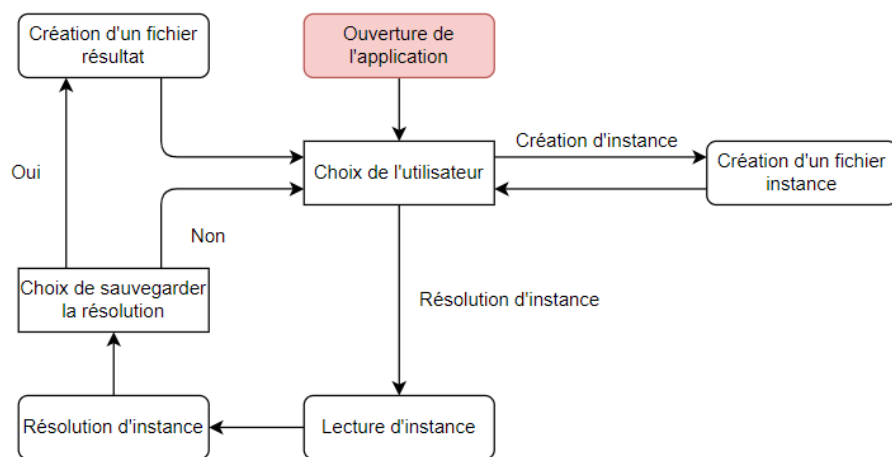
### 4 Structure générale du système

Malgré le fait que ce projet se concentre uniquement sur la recherche de la meilleure manière de résoudre une instance, un modèle global a été réfléchi pour avoir une vision du résultat final.

Après avoir lancé le logiciel, l'utilisateur devrait avoir le choix de créer une instance ou de lancer une résolution à partir d'un fichier instance. Dans le second cas et après la résolution, il aurait la possibilité de sauvegarder le résultat ou non.

Les choix et les différentes fonctions appelées sont résumés dans le diagramme ci-dessous.

Pour ce projet, les sections développées seront donc uniquement la création d'un fichier instance, la lecture d'instance et la résolution d'instance.



**Figure 2.1** – *Diagramme de la structure du logiciel*

# 3

## État de l'art / Veille technologique

### 1 Problème de l'ordonnancement de la production

Les problèmes de production sont des problèmes très connus. Le but est dans la plupart du temps de résoudre un problème d'ordonnancement de tâches permettant de les réaliser sur la période de temps voulus en respectant des contraintes de temps. Cependant les spécificités du monde agricole rendent ces études difficiles à transcrire dans ce domaine. Dans cette partie, je travaillerai à mettre en place une documentation des différents travaux réalisés et qui se rapproche du problème de production du sujet.

#### 1.1 Planification d'opérations de récolte de fermes à fermes

Cette étude de son nom anglais "Scheduling Contractors' Farm-to-Farm Crop Harvesting Operations"[1] publiée récemment en 2018. Cette étude s'intéresse à la planification d'opérations de récoltes dans différentes fermes par un exploitant extérieur à la production agricole. Le modèle permet en effet à un exploitant tiers de distribuer des machines sur différentes fermes dans le but de permettre des récoltes se faisant en plusieurs tâches (chaque tâche possédant un type de machine unique). L'exemple donné est un ensemble de fermes cultivant du soja. Les différentes tâches ordonnées sont la fauchaison, le mouillage et la récolte du colza. La phase de semence n'est pas présente dans cette étude. On retrouve des similitudes avec le sujet du projet comme par exemple la variable  $n$  du modèle de base qui représente le nombre d'opérations différentes (3 dans l'exemple) et qui vaut dans le modèle du sujet 2 car on peut considérer deux tâches à savoir la semence et la récolte, ainsi on contourne la limitation de l'absence de semaison. De plus on peut retirer du modèle de base toutes les données, les variables et les contraintes en lien avec les distances entre les différentes fermes puisque l'on considère un seul site de production. En revanche, la manière dont la production est modélisée est loin de correspondre au modèle à construire puisqu'aucun temps de pousse n'est ajoutable de manière à pouvoir laisser des cultures mûrir plus longtemps ou moins longtemps si besoin. Pour cette raison, on ne s'intéressera pas au modèle proposé mais plutôt aux méthodes de résolution proposées. Deux méthodes sont proposées à savoir une résolution exacte ou l'utilisation de la procédure **Tabu Search** ou **Recherche Tabou (TS)**. En réalité, une troisième méthode est proposée mais celle-ci est trop éloignée du modèle à réaliser à cause de toutes les modifications effectuées pour ressembler au mieux au modèle présent dans l'étude.

Il apparaît que la méthode de résolution exacte soit peu performante pour résoudre des problèmes de grandes tailles. Le document spécifie un maximum de 6 types de production et 3 tâches en tout pour réussir à obtenir un résultat en temps raisonnable. La procédure **TS** est une métaheuristique qui a pour avantage de permettre d'éviter de converger trop vite dans le premier optimum local afin d'explorer d'autres solutions. Cette heuristique utilise une mémoire des dernières solutions trouvées afin d'interdire le retour à des optimums déjà explorés ce qui force la découverte de nouveaux optimums augmentant ainsi les chances d'obtenir une meilleure solution.

## 1.2 RCPSP

Le **Resource-Constrained Project Scheduling Problem** ou **Problème d'ordonnancement de projet sous contraintes de ressources (RCPSP)** est un problème d'ordonnancement abordé dans le document "Planification et ordonnancement de projets sous contraintes de ressources complexes[2]. Il s'agit d'un problème dans lequel on cherche à minimiser la durée totale d'un projet. Un projet est défini comme un ensemble d'activités possédant différentes caractéristiques listées ci-dessous :

- $n$  : nombre d'activités
- $A$  : ensemble des activités (avec  $n$  activités plus une activité de début et une activité de fin)
- $m$  : nombre de ressources
- $R$  : ensemble des ressources (avec  $m$  ressources)
- $p_i$  : durée d'exécution de l'activité  $i \in A$
- $b_k$  : capacité de la ressource  $k \in R$
- $r_{ik}$  : consommation de l'activité  $i \in A$  sur la ressource  $k \in R$
- $E_{ii'}$  : relation d'antériorité directe entre l'activité  $i \in A$  et l'activité  $i' \in A$
- $E_{ii'}^*$  : relation d'antériorité directe et indirecte entre l'activité  $i \in A$  et l'activité  $i' \in A$
- $h$  : temps de planification
- $T$  : ensemble des moments discrets allant de 0 (début du projet) jusqu'à  $h$  (fin du projet)
- $ES_i$  : date de début au plus tôt de l'activité  $i \in A$
- $LS_i$  : date de début au plus tard de l'activité  $i \in A$

Un certain nombre de points communs peuvent être mis en évidence entre le modèle présenté dans le document et le modèle que l'on cherche à créer pour gérer les tâches du problème de planification des opérations agricoles. En effet, on va avoir plusieurs activités pouvant être classées en deux catégories : les activités de semences et les activités de récoltes dont l'antériorité correspondra à la semence de la ressource. Un autre détail important présenté dans le modèle et qui correspond aux attentes est l'impossibilité de préemption ce qui signifie qu'une activité doit être terminée si elle est commencée.

## 2 Problème de distribution de la récolte

Les problèmes de distribution sont également des problèmes très connus. Ces problèmes peuvent soit se rapprocher des problèmes de voyageurs de commerces, soit des problèmes de tournées. Dans le cas de ce projet, le problème du voyageur de commerce sera privilégié car on considère que l'agriculteur ne peut faire qu'un seul circuit par jour.

## 2.1 Problème CRS

Dans la thèse publié en 2013 et dont le titre est "Contributions à la chaîne logistique numérique : conception de circuits courts et planification décentralisée." [3] et plus particulièrement dans le premier chapitre, l'auteur introduit le problème **Conception de Réseau de Service (CRS)** dont les données sont les suivantes :

- $F$  : nombre d'agriculteurs
- $C$  : nombre de clients
- $H$  : nombre de plateformes
- $P$  : nombre de produits
- $T$  : nombre de périodes
- $N_t$  : nombre de sous-périodes dans la période  $t \in T$
- $H^{max}$  : nombre maximal de plateformes ouvertes
- $S_{pft}$  : offre maximale de l'agriculteur  $f \in F$ , pour le produit  $p \in P$ , pour une sous-période à la période  $t \in T$
- $D_{pct}$  : demande du client  $c \in C$  pour le produit  $p \in P$  pour une sous-période de la période  $t \in T$
- $C_{fi}$  : coût fixe d'envoi depuis l'agriculteur  $f \in F$  jusqu'au client ou à la plateforme  $i \in C \cup H$
- $C_{hc}$  : coût fixe de livraison du client  $c$  par la plateforme  $h \in H$
- $c_{hi}$  : coût unitaire de livraison depuis la plateforme  $h \in H$  jusqu'au client où la plateforme  $i \in C \cup H$
- $l_{pct}$  : coût unitaire de la pénalité pour la non satisfaction de la demande du client  $c \in C$  pour le produit  $p \in P$  à la période  $t \in T$

Les données énoncées sont très similaires à celles attendues avec néanmoins quelques ajustements liés à la présence d'un unique agriculteur et à l'absence de plateformes intermédiaires entre l'agriculteur et les clients. De plus, il peut s'avérer peu utile et peu flexible de conserver la séparation des périodes en sous-périodes dans l'usage que l'on souhaite faire du logiciel. Ainsi, il est possible de réécrire les données de manière simplifiée sous la forme suivante :

- $C$  : nombre de clients
- $P$  : nombre de produits
- $T$  : nombre de périodes
- $S_{pt}$  : offre maximale de l'agriculteur, pour le produit  $p \in P$ , pour une période  $t \in T$
- $D_{pct}$  : demande du client  $c \in C$  pour le produit  $p \in P$  pour une période  $t \in T$
- $C_c$  : coût unitaire d'envoi depuis l'agriculteur jusqu'au client  $c \in C \cup H$
- $l_{pct}$  : coût unitaire de la pénalité pour la non satisfaction de la demande du client  $c \in C$  pour le produit  $p \in P$  à la période  $t \in T$

Le modèle que l'on obtient ainsi peut également varier un peu si l'on souhaite faciliter la modification des coûts de retards. On pourrait envisager de remplacer la valeur  $C_c$  par la distance de l'agriculteur aux clients et en ajoutant une valeur unitaire de coût de distance. Le résultat du coût serait alors la valeur de la distance multipliée par la valeur scalaire du coût de distance. Enfin il faut quand même noter une différence importante entre le modèle présenté et celui qui doit être utilisé puisque l'agriculteur doit être en mesure de se déplacer jusqu'à des clients, mais également de se déplacer de clients en clients. Or, il n'existe pas dans ce modèle de données permettant de quantifier ces coûts. Une autre adaptation qu'il sera nécessaire de réaliser et de modifier la donnée  $S_{pt}$  puisque l'offre proposée par l'agriculteur sera créée avec le problème de production dont l'état de l'art a été présenté dans la partie précédente.

Avec ces données, l'auteur est donc en mesure de créer un **Mixed Integer Program ou Programme Linéaire à Variables Mixtes (MIP)** en définissant les variables suivantes :

- $x_{pfit}$  : quantité du produit  $p \in P$  qui transite à chaque sous-période de la période  $t \in T$ ,

- depuis l'agriculteur  $f \in F$  vers le client ou la plateforme  $i \in C \cup H$
- $x_{phct}$  : quantité du produit  $p \in P$  qui transite à chaque sous-période de la période  $t \in T$ , entre la plateforme  $h \in H$  et la plateforme ou le client  $c \in C \cup H$
  - $z_{pct}$  : ventes perdues par rapport à la demande du client  $c \in C$  pour le produit  $p \in P$  à chaque sous-période de la période  $t \in T$
  - $y_{fit} : \begin{cases} 1 & \text{si l'agriculteur } f \in F \text{ livre la plateforme ou le client } i \in C \cup H \\ & \text{à la période } t \in T \\ 0 & \text{sinon} \end{cases}$
  - $y_{hct} : \begin{cases} 1 & \text{si la plateforme } h \in H \text{ livre le client } c \in C \text{ à la période } t \in T \\ 0 & \text{sinon} \end{cases}$
  - $y_h : \begin{cases} 1 & \text{si la plateforme } h \in H \text{ est ouverte} \\ 0 & \text{sinon} \end{cases}$

Les variables  $y_{fit}$ ,  $y_{hct}$  et  $y_h$  possèdent donc des variables binaires. Les quantités de produits en transit  $x_{pfit}$  et  $x_{phct}$  sont des entiers uniquement positifs ainsi que les quantités de produits non livrées  $z_{pct}$ . Le but est donc de maximiser les livraisons tout en minimisant les coûts de non livraisons et de distances.

La fonction objective à minimiser est donc la fonction définie ci-dessous dans l'équation (1) telle qu'elle est écrite dans le document :

$$\begin{aligned} \text{Min} \sum_{t \in T} N_t \times ( & \sum_{f \in F, i \in C \cup H} D_{fi} \times y_{fit} + \sum_{h \in H, c \in C} C_{hc} \times y_{hct} + \\ & \sum_{h \in H, i \in C \cup H, p \in P} c_{hi} \times x_{phit} + \sum_{c \in C, p \in P} l_{pct} \times z_{pct} ) \end{aligned} \quad (1)$$

En plus de cette fonction objective à minimiser, un certain nombre de contraintes sont définies pour garantir l'exactitude de la résolution du modèle. La première contrainte est de vérifier que le nombre de plateformes ouvertes simultanément ne dépasse pas le nombre total de plateformes pouvant être ouvertes.

$$\sum_{h \in H} y_h \leq H^{max} \quad (2)$$

La contrainte suivante permet de vérifier que les quantités de chaque produit livré ne dépassent pas la quantité des stocks des agriculteurs à tout moment.

$$\sum_{i \in C \cup H} x_{pfit} \leq S_{pft} \quad \forall f \in F, p \in P, t \in T \quad (3)$$

Deux contraintes sont ensuite utilisées pour s'assurer que les livraisons allant des agriculteurs aux clients ou aux plateformes intermédiaires(4) et les livraisons allant des plateformes au clients(5) ne dépassent pas les limites voulues que ce soit pour ne pas dépasser la limite d'une demande ou ne pas livrer une entité n'ayant pas à être livrée au moment  $t$ .

$$x_{pfit} \leq S_{pft} \times y_{fit} \quad \forall f \in F, i \in C \cup H, p \in P, t \in T \quad (4)$$

$$x_{phct} \leq D_{pct} \times y_{hct} \quad \forall h \in H, c \in C, p \in P, t \in T \quad (5)$$

Une contrainte permet de calculer le manque de livraison correspondant au total attendu moins la quantité livrée.

$$\sum_{j \in F \cup H} x_{pjct} + z_{pct} = D_{pct} \quad \forall c \in C, p \in P, t \in T \quad (6)$$

Une autre contrainte permet d'assurer que les produits livrés à une plateforme sont par la suite redistribués.

$$\sum_{j \in F \cup H} x_{pjht} = \sum_{i \in C \cup H} x_{phit} \quad \forall h \in H, p \in P, t \in T \quad (7)$$

Les deux contraintes suivantes permettent d'empêcher la livraison de produits dans des plateformes fermées ou depuis des plateformes fermées.

$$x_{pfht} \leq S_{pft} \times y_h \quad \forall f \in F, h \in H, p \in P, t \in T \quad (8)$$

$$x_{phh't} \leq \min\left(\sum_{f \in F} S_{pft}, \sum_{c \in C} D_{pct}\right) \times y_h \quad \forall h \in H, p \in P, t \in T \quad (9)$$

Les deux équations qui suivent permettent d'empêcher à un produit d'être transporté plus de deux fois.

$$\sum_{h' \in H} x_{ph'ht} \leq \sum_{c \in C} x_{phct} \quad \forall h \in H, p \in P, t \in T \quad (10)$$

$$\sum_{h' \in H} x_{ph'ht} \leq \sum_{f \in F} x_{pfht} \quad \forall h \in H, p \in P, t \in T \quad (11)$$

Pour résoudre le problème exposé dans l'étude, l'auteur introduit trois techniques de résolution. La première technique de résolution est le **Branch & Cut (B&C)** qui est une technique permettant de trouver un optimum global. Cette technique est notamment utilisée par le solveur Cplex. Elle consiste à créer un arbre de possibilité où les branches représentent des solutions. Lorsque la valeur objective à un nœud devient trop faible pour permettre de trouver des optimums à partir du nœud, une coupe est effectuée permettant de ne pas calculer toutes les solutions dépendant de ce nœud. Ainsi, une partie du temps de calcul est préservé pour permettre une augmentation des performances de la résolution. La deuxième technique est la **Décomposition de Benders (DB)** dont le principe est la séparation du problème principal en deux problèmes nommés problème maître et problème esclave. Le problème maître fonctionne en premier en prenant une variable  $z$  permettant d'approcher la valeur objective que l'on peut obtenir avec le problème esclave. Une fois que le problème maître est résolu, il ne reste plus qu'à résoudre le problème esclave plus simple que le problème d'origine puisque les variables du problème maître sont fixées. On peut également noter qu'il est possible d'itérer plusieurs fois cette méthode en transformant un problème esclave en un sous-problème maître avec un sous-problème esclave. La troisième technique est la **Procédure de Pentes Dynamiques (PPD)**. Le principe est de trouver des facteurs linéaires qui permettent de conserver les contraintes tout en modifiant la valeur de la fonction objective. Le but est alors d'itérer un grand nombre de fois jusqu'à un nombre d'itérations défini ou jusqu'à une convergence des solutions du problème.

Ces trois méthodes ont été comparées à l'aide de plusieurs ensembles d'instances. Le document présente deux ensembles, mais on ne s'intéressera qu'au premier étant donné que le second a pour but principal d'évaluer les performances en fonction du nombre de plateformes. Le premier ensemble est constitué d'un corpus de 12 instances pour 3 paramètres différents permettant l'identification de chaque instance par un sigle de trois lettres. Le premier paramètre représente la taille de l'instance. Les différentes options pour ce paramètre sont résumées dans le tableau suivant provenant de l'étude.

Type d'instance	F	C	H	P	T
Petite (P)	20	20	5	4	12
Moyenne (M)	50	50	5	10	12
Grande (G)	50	100	5	16	12

Figure 3.1 – Description de la taille des instances

Ensuite, le deuxième paramètre est la répartition des agriculteurs qui peut être réalisée de manière aléatoire (A) ou groupée sur des sous-zones (G) et enfin le troisième paramètre permet de considérer la répartition des productions qui peut être mixée (M) ou partitionnée (P)

Ces 12 instances ont été testées avec les 3 méthodes proposées plus haut. Les résultats sont contenus dans le tableau ci-dessous.

Instance	B&C (CPLEX 2%)	B&C (CPLEX 5%)	Décomposition de Benders (5%)	PPD
P-GP	29,63	30,10	10 544,27	81,55
P-GM	53,44	50,08	5,42	66,54
P-AP	31,28	34,78	3,55	71,77
P-AM	35,59	34,23	4,18	79,92
M-GP	1 691,00	1 757,76	11 108,06	2 406,87
M-GM	3 084,00	3 225,31	33,61	1 972,07
M-AP	2 033,59	1 986,58	10 806,20	2 218,38
M-AM	904,07	919,77	162,17	2 549,91
G-GP	3 260,97	3 312,11	185,17	7 241,53
G-GM	6 241,31	5 801,11	293,97	6 060,13
G-AP	5 065,45	5 025,09	107,51	7 960,56
G-AM	5 591,98	5 383,32	76,58	6 518,57

Figure 3.2 – Temps de résolution des méthodes proposées

Le constat réalisé est que la **DB** est globalement la plus rapide des méthodes de résolution du problème. Cependant, il faut également s'intéresser à la qualité de la solution obtenue. Le tableau récapitulatif ci-dessous permet de montrer la qualité en fonction de la méthode et de l'instance.

Le constat réalisé sur les résultats 3.3 est que le **PPD** obtient très souvent une meilleure approximation de l'optimum global.

On peut donc en conclure que dans le cas où l'on souhaite obtenir une solution rapidement, la **DB** est la meilleure option. En revanche, si on souhaite obtenir un très bon résultat sans contraintes de temps, alors le **PPD** est meilleur. Enfin, dans le cas où on cherche à réaliser un compromis entre rapidité et qualité, la solution du **B&C** est la meilleure.

Instance	B&C (CPLEX 2%)	Décomposition de Benders (5%)	PPD
<i>P</i> -GP	1,26	3,96	0,81
<i>P</i> -GM	0,90	2,45	0,49
<i>P</i> -AP	0,86	1,68	0,53
<i>P</i> -AM	1,08	2,67	0,56
<i>M</i> -GP	1,03	3,62	0,66
<i>M</i> -GM	1,13	2,85	0,53
<i>M</i> -AP	1,04	3,13	0,55
<i>M</i> -AM	1,41	4,10	0,62
<i>G</i> -GP	0,01	0,06	0,01
<i>G</i> -GM	0,02	2,30	0,01
<i>G</i> -AP	0,01	0,03	0,01
<i>G</i> -AM	0,02	0,97	0,01

Figure 3.3 – Ecart avec la borne inférieur en pourcentage

# 4

## Analyse et conception

### 1 Analyse

#### 1.1 Hypothèses utilisées

Des hypothèses de conceptions ont été réalisées dans le but de diminuer très légèrement la complexité de la résolution d'heuristiques. En effet, il a été considéré qu'il était toujours optimal de commencer une tâche dès que possible et qu'il était toujours optimal de faire en sorte que la durée de pousse soit la plus courte autorisée.

#### 1.2 Spécifications

##### 1.2.1 Spécifications fonctionnelles

L'application sera composée de 4 fonctionnalités principales. Un résumé de ces fonctionnalités dont la description est réalisée en annexe est écrit ci-dessous.

##### **Lecture d'instances**

Le but de cette fonction est de permettre d'importer des instances automatiquement depuis des fichiers textes. L'instance obtenue en sortie sera ensuite utilisée dans le solveur.

##### **Résolution d'instance**

Cette fonction a pour but d'effectuer la résolution d'une instance. Pour cela, elle utilisera principalement la fonction de lecture d'instance afin d'obtenir les données d'une instance, puis utilisera le solveur pour obtenir le résultat de l'analyse.

##### **Ecriture d'instance**

Le but de cette fonction est de permettre de créer des instances afin d'obtenir un fichier valide pour la lecture d'instances sans que l'utilisateur n'ait besoin de le modifier directement ce qui pourrait amener à des erreurs de lectures ou des malentendus sur les données.

### Sauvegarde de résultat

Cette fonction a pour but d'effectuer la sauvegarde du résultat obtenu par le solveur.

#### 1.2.2 Spécifications non fonctionnelles

Le programme est développé en python 3.8 sur l'environnement Spyder et avec un système d'exploitation Windows. J'utiliserai également le solveur GLPK utilisant le langage GLPM pour permettre la résolution d'instances. L'application est développée sous la forme de scripts python. Il pourra bénéficier d'une interface graphique réalisée avec arena16 si le temps le permet. Etant donné la complexité du problème, les capacités de l'application seront limitées à un maximum de 10 livraisons différentes. Des conventions de nommage sont utilisées à la fois dans la partie python et dans la partie GLPK.

## 2 Modélisation proposée

Dans cette partie est développée la modélisation mathématique du problème tel qu'il a été implémenté dans le programme.

### 2.1 Paramètres

Le tableau ci-dessous récapitule les différents paramètres des instances que l'on souhaite résoudre. La taille d'une instance est principalement définie par  $m$ ,  $n$  et  $T$  dont l'indice de variation est respectivement  $i$ ,  $j$  et  $t$  dans la suite lorsqu'ils ne sont pas spécifiés.

paramètre	description
$n$	nombre de tâches
$m$	nombre de machines (prévu pour fonctionner avec une valeur de 2)
$T$	horizon de la prévision
$HV$	grande valeur
$ra_j$	date de début au plus tôt de la tâche $j$
$rb_j$	date de début au plus tard de la tâche $j$
$p1a_j$	durée de pousse minimale du produit $j$
$p1c_j$	durée de pousse maximale du produit $j$
$p2max_j$	durée de récolte maximale du produit $j$
$Q_i$	quantité de ressource disponible pour la machine $i$
$a_j$	relation entre $q_j$ et $Y1$ : $Y1(j) = a \times q(j)$ , autrement dit la relation entre l'espace occupé par les plantations et la quantité de ressource à affecter
$pi_j$	récompense pour une unité de $j$
$tpc_j$	coût de retard pour le produit $j$ peu importe la quantité
$hc_{it}$	coût d'embauche à la période $t$ , sur la machine $i$
$dde_j$	demande du produit $j$
$tt_{jj}$	temps de trajet entre le premier lieu $j$ et le second lieu $j$
$QVh$	capacité du véhicule (en tonnes)
$deadline_j$	date de rendu du produit $j$

## 2.2 Variables

Pour pouvoir résoudre le problème un certain nombre de variables ont été créés et ces dernières sont récapitulées dans le tableau suivant :

variable	description
$p1_j$	durée de la tâche j pour la phase de pousse
$q_j$	quantité du produit j à produire
$p2_j$	durée de la tâche j pour la phase de récolte
$Y_{ij}$	quantité de ressource allouée à la tâche j pour la machine i
$x_{ijt}$	vaut 1 si la tâche j est en cours sur la machine i au temps t, vaut 0 sinon
$z_{ijt}$	vaut 1 si la tâche j démarre sur la machine i au temps t, vaut 0 sinon
$conso_{ijt}$	consommation de ressource de la tâche j au temps t sur la machine i
$R_{it}$	ressource supplémentaire nécessaire sur la machine i au temps t
$zY2_{qj}$	variable pour la linéarisation de $p2 \times Y2$
$tY2_{qj}$	variable pour la linéarisation de $p2 \times Y2$
$prod\_p2Y2_j$	variable pour la linéarisation de $p2 \times Y2$
$xR_{j_1j_2j_3}$	vaut 1 si la livraison du produit $j_1$ précède la livraison du produit $j_2$ pour la route $j_3$ , vaut 0 sinon
$zB_{j_1j_2}$	vaut 1 si le produit $j_1$ est livré à la tournée $j_2$ , vaut 0 sinon
$tB_j$	date de départ de la tournée j
$D_j$	date de livraison du produit j
$Tardiness\_Cost$	coût de retard total
$TD_j$	coût de retard de la tâche j
$Hiring\_Cost$	coût de l'embauche
$TheReward$	récompense des ventes

## 2.3 Contraintes

Les sections suivantes ont pour but d'expliquer toutes les contraintes qui ont été utilisées dans le modèle.

### 2.3.1 Contraintes de la fonction objectif

Différentes contraintes ont été réalisées pour simplifier l'écriture de la fonction objectif. Dans un premier temps, on cherche à obtenir l'argent perdu.

Pour cela, on va chercher pour chaque tâche le temps de retard obtenu grâce à la contrainte suivante qui soustrait la date de rendu par la date voulue :

$$\forall j \in [1, n] \quad TD_j \geq D_j - deadline_j \quad (1)$$

Il ne reste plus ensuite qu'à multiplier le temps de retard par la coût pour une unité de temps pour chaque produit ce qui correspond à la contrainte suivante :

$$\forall j \in [1, n] \quad Tardiness\_Cost = tpc_j \times TD_j \quad (2)$$

On va ensuite chercher le coût lié aux embauches ayant pour but d'augmenter la capacité de récolte. Cette contrainte correspond à la formule suivante :

$$Hiring\_Cost = \sum_{i=1}^m \sum_{t=0}^T hc_{i,t} \times R_{i,t} \quad (3)$$

Enfin, on va également chercher à calculer l'argent gagné grâce aux ventes de chaque produit. Pour cette contrainte, il suffit simplement de regarder les quantités vendues par rapport au prix unitaire du produit comme cela est écrit ci-dessous :

$$TheReward = \sum_{j=1}^n pi_j \times q_j \quad (4)$$

On peut également ajouter une contrainte supplémentaire. Cette contrainte ne permet pas de modifier le modèle mais permet d'accélérer le processus de résolution en supprimant le nombre de valeurs que peut prendre une variable. Pour cette première fonction superflue, on considère que le montant rapporté par les ventes ne peut pas être inférieur à la demande de chaque produit multiplié par le prix unitaire. On définit donc la contrainte ainsi :

$$TheReward \geq \sum_{j=1}^n pi_j \times dde_j \quad (5)$$

Cette contrainte permet d'apporter une rapidité légèrement supérieure de la résolution pour des instances de petites tailles testées.

### 2.3.2 Contraintes sur la première machine : la phase de pousse

Tout d'abord, on cherche à s'assurer que la durée de pousse est bien comprise entre les bornes spécifiées. Cela amène aux deux contraintes suivantes :

$$\forall j \in [1, n] \quad p1a_j \leq p1_j \quad (6)$$

$$\forall j \in [1, n] \quad p1_j \leq p1c_j \quad (7)$$

On va ensuite spécifier sur la variable  $x$  les moments où la phase de pousse pour chaque produit est en cours tout en vérifiant que la durée correspond bien à celle déterminé par les contraintes (6) et (7). La contrainte est la suivante :

$$\forall t \in [ra_j, T] \quad x_{1,j,t} = p1_j \quad (8)$$

Enfin, on définit la quantité de ressources (la superficie du terrain) à allouer à chaque produit grâce à la contrainte suivante :

$$\forall t \in [ra_j, T] \quad Y_{1,j} = a_j \times q_j \quad (9)$$

## 2.3.3 Contraintes sur la seconde machine : la phase de récolte

De la même façon que pour la contrainte (8), on spécifie les moments où la phase de récolte pour chaque produit est en cours. La contrainte obtenue est la suivante :

$$\forall t \in [ra_j, T] \quad x_{2,j,t} = p2_j \quad (10)$$

On va ensuite calculer la valeur de la variable  $p2_j$  à partir de  $Y_{2,j}$  et  $p2max_j$  comme spécifié sur l'équation suivante :

$$p2max_j \times q_j = p2_j \times Y_{2,j} \quad (11)$$

Cependant, cette équation ne peut pas être transformée en contrainte. En effet,  $Y_{2,j}$  et  $p2_j$  sont des variables or il est impossible de multiplier des variables entre elles dans un problème de type **MIP**. Pour contourner le problème, on utilise une méthode de linéarisation pour pouvoir effectuer tout de même la multiplication. Tout d'abord, on introduit la variable  $prod\_p2Y2$  qui synthétise le produit impossible à réaliser. La contrainte reprend donc l'équation précédente pour se transformer comme on peut le voir ci-dessous :

$$\forall j \in [1, n] \quad prod\_p2Y2 = p2_j \times Y_{2,j} \quad (12)$$

On va ensuite décomposer la valeur de  $Y_{2,j}$  en puissance de 2 et enregistrer cette décomposition sous la forme d'une matrice de variables binaires :

$$\forall j \in [1, n] \quad \sum_{qq=0}^3 tY2_{qq,j} \times (2^{qq}) = Y_{2,j} \quad (13)$$

Le principe est alors de définir la valeur de  $zY2_{qq,j}$  comme étant égale à 0 si  $tY2_{qq,j}$  vaut 0 ou égale à  $p2_j \times 2^{qq}$  si  $tY2_{qq,j}$  vaut 1. Pour cela, on utilise les 3 contraintes ci-dessous :

$$\forall qq \in [0, 3], \quad \forall j \in [1, n] \quad zY2_{qq,j} \leq p2_j \times (2^{qq}) \quad (14)$$

$$\forall qq \in [0, 3], \quad \forall j \in [1, n] \quad zY2_{qq,j} \geq p2_j \times (2^{qq}) - HV \times (1 - tY2_{qq,j}) \quad (15)$$

$$\forall qq \in [0, 3], \quad \forall j \in [1, n] \quad zY2_{qq,j} \leq HV \times tY2_{qq,j} \quad (16)$$

Il ne reste plus alors qu'à définir la variable  $prod\_p2Y2$  introduite dans la contrainte (12) comme étant la somme des termes de  $tY2_{qq,j}$  ce qui est fait dans la contrainte suivante :

$$\forall j \in [1, n] \quad prod\_p2Y2_j = \sum_{qq=0}^3 zY2_{qq,j} \quad (17)$$

Même si cette méthode de linéarisation permet effectivement de réaliser des multiplications, il faut quand même noter une limite qui empêche les valeurs de  $Y_{2,j}$  de dépasser 15 à cause de la contrainte (13).

Enfin, étant donné l'équation que l'on cherche à résoudre, il est possible de déduire deux contraintes superflues à savoir :

$$\forall j \in [1, n] \quad p2_j \leq p2max_j \times q_j \quad (18)$$

$$\forall j \in [1, n] \quad Y_{2,j} \leq p2max_j \times q_j \quad (19)$$

Sur certaines instances comme par exemple l'instance "donnees\_base\_accelere.dat", un gain de temps de 40% à 50% a pu être observé grâce à la présence de ces deux contraintes.

## 2.3.4 Contraintes sur les demandes

La première contrainte à regarder est que la quantité livrée soit égale à la quantité demandée du produit. Pour cela, on utilise la contrainte suivante :

$$\forall j \in [1, n] \quad q_j \geq d_{de_j} \quad (20)$$

On va ensuite vérifier que le début de chaque tâche  $j$  sur la machine 2 et les ressources allouées à cette tâche soient suffisantes pour ne pas dépasser la deadline de livraison. La contrainte correspondante est la suivante :

$$\forall j \in [1, n] \quad \sum_{t=ra[j]}^T t \times z_{2,j,t} + p_{2j} \leq deadline_j \quad (21)$$

Ensuite, il faut contraindre les ressources allouées par rapport aux maximums qui sont définis. Pour la machine 1, le maximum n'est pas extensible (car la superficie des champs est limitée), mais pour la machine 2 il est possible d'augmenter les ressources allouées (en embauchant de la main d'œuvre supplémentaire par exemple). Ces deux contraintes sont les suivantes :

$$\forall t \in [0, T] \quad \sum_{j=1}^n cons_{01,j,t} \leq Q_1 \quad (22)$$

$$\forall t \in [0, T] \quad \sum_{j=1}^n cons_{02,j,t} \leq Q_2 + R_{2,t} \quad (23)$$

Deux contraintes superflues peuvent encore être ajoutées pour obtenir un résultat similaire mais plus rapide à savoir les deux contraintes suivantes qui vérifient qu'aucune tâche ne dépasse seule les ressources allouables :

$$\forall j \in [1, n] \quad Y_{1,j} \leq Q_1 \quad (24)$$

$$\forall j \in [1, n] \quad Y_{2,j} \leq Q_2 + R_{2,t} \quad (25)$$

Les performances obtenues sur de petites instances d'une durée de résolution d'environ 100 secondes sont très bonnes : sans les contraintes précédentes, le temps d'exécution peut être multiplié par 10 voire plus dans certains cas comme pour l'instance déjà utilisée précédemment où le temps d'exécution est passé d'une centaine de seconde à 3500 soit pratiquement une heure.

On va à présent contraindre le début de chaque tâche sur la machine 2 à la fin de la tâche sur la machine 1 de sorte qu'il n'y ait pas de temps d'attente possible entre les deux. Cette contrainte est définie ci-dessous :

$$\forall j \in [1, n] \quad \sum_{t=ra_j}^T t \times z_{2,j,t} = \sum_{t=ra_j}^T t \times z_{1,j,t} + p_{1j} \quad (26)$$

Enfin, on vérifie que le début de chaque tâche se fait bien entre les bornes  $ra_j$  et  $rb_j$ . La contrainte correspondant est la suivante :

$$\forall j \in [1, n] \quad \sum_{t=ra_j}^{rb_j} z_{1,j,t} = 1 \quad (27)$$

2.3.5 Contraintes de liaison entre  $z$  et  $x$ 

La variable  $z_{i,j,t}$  défini le moment de début d'une tâche et la variable  $x_{i,j,t}$  défini les moments où une tâche est en cours. Si en général l'une des notations est utilisée et pas l'autre en fonction du besoin, il est nécessaire dans la modélisation de ce problème d'avoir les deux. En effet, la variable  $z$  a déjà été utilisée dans les paragraphes précédents et la variable  $x$  est nécessaire dans le paragraphe suivant. Pour lier les deux variables, on utilise les 3 contraintes suivantes :

$$\forall i \in [1, m], \forall j \in [1, n], \forall t \in [1, T] \quad z_{i,j,t} \geq x_{i,j,t} - x_{i,j,t-1} \quad (28)$$

$$\forall i \in [1, m], \forall j \in [1, n] \quad z_{i,j,0} \geq x_{i,j,0} \quad (29)$$

$$\forall i \in [1, m], \forall j \in [1, n] \quad \sum_{t=0}^T z_{i,j,t} = 1 \quad (30)$$

## 2.3.6 Contraintes de consommation

La variable  $conso_{m,j,t}$  permet de connaître les moments où les ressources allouées à une tâche sont effectives. Pour cela, on va dans un premier temps définir les deux contraintes suivantes pour la machine 1 qui permettent d'avoir une valeur égale avec  $Y_{1,j}$  si la tâche est en cours au moment  $t$  :

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{1,j,t} \geq Y_{1,j} - Q_1 \times (1 - x_{1,j,t}) \quad (31)$$

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{1,j,t} \leq Y_{1,j} + Q_1 \times (1 - x_{1,j,t}) \quad (32)$$

On fait de même pour la machine 2 ce qui donne les 2 contraintes suivantes :

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{2,j,t} \geq Y_{2,j} - HV \times (1 - x_{2,j,t}) \quad (33)$$

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{2,j,t} \leq Y_{2,j} + HV \times (1 - x_{2,j,t}) \quad (34)$$

On doit également définir la valeur de la consommation à 0 pour chaque tâche lorsque celle-ci n'est ni en cours sur la machine 1, ni sur la machine 2 ce qui donne respectivement les deux contraintes suivantes :

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{1,j,t} \leq Q_1 \times (x_{1,j,t} + x_{2,j,t}) \quad (35)$$

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{2,j,t} \leq HV \times x_{2,j,t} \quad (36)$$

Enfin, on cherche à immobiliser les ressources sur la machine 1 lorsqu'une tâche est en cours sur la machine 2. En effet, lorsque la récolte est en cours, le champ est toujours occupé et ne peut donc pas servir pour accueillir de nouvelles semences tant que la récolte n'est pas terminée. Cette nécessité d'immobilisation des champs est défini par les 2 contraintes ci-dessous :

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{1,j,t} \geq Y_{1,j} - Q_1 \times (1 - x_{2,j,t}) \quad (37)$$

$$\forall j \in [1, n], \forall t \in [0, T] \quad conso_{1,j,t} \leq Y_{1,j} + Q_1 \times (1 - x_{2,j,t}) \quad (38)$$

## 2.3.7 Contraintes de planification de la distribution

Dans un premier temps, on va contraindre la livraison d'un produit à être comprise dans une seule tournée. Il s'agit également de pouvoir déterminer un nombre maximal de tournée puisque dans ce cas, le nombre maximal de tournée est égal au nombre de produits à livrer. Cette contrainte s'écrit de la manière suivante :

$$\forall j \in [1, n] \quad \sum_{k=1}^n zB_{j,k} = 1 \quad (39)$$

On cherche ensuite à déterminer le moment du départ d'une tournée. Les éléments à prendre en compte sont les dates de fin de récolte de chaque produit de la tournée à considérer. La contrainte correspondant s'écrit de la manière suivante :

$$\forall j \in [1, n], \quad \forall k \in [1, n] \quad tB_k \geq \sum_{t=0}^T t \times z_{2,j,t} + p2_j - HV \times (1 - zB_{j,k}) \quad (40)$$

On vérifie que dans chaque tournée, le producteur est le lieu de départ et le lieu d'arrivée ce qui correspond respectivement aux 2 contraintes suivantes :

$$\forall k \in [1, n] \quad \sum_{j=1}^n xR_{0,j,k} \leq 1 \quad (41)$$

$$\forall k \in [1, n] \quad \sum_{i=1}^n xR_{i,0,k} \leq 1 \quad (42)$$

D'une manière similaire, on vérifie que chaque client à livrer n'est présent qu'une seule et unique fois comme point de départ et comme point d'arrivée d'un trajet au sein d'une tournée.

$$\forall i \in [1, n], \quad \forall k \in [1, n] \quad \sum_{j=1, j \neq i}^n xR_{i,j,k} = zB_{i,k} \quad (43)$$

$$\forall j \in [1, n], \quad \forall k \in [1, n] \quad \sum_{i=1, i \neq j}^n xR_{i,j,k} = zB_{j,k} \quad (44)$$

Il est également possible d'ajouter la contrainte superflue qui force le nombre de points de départ à être égal au nombre de points d'arrivée pour un trajet dans chaque tournée. Cette contrainte supplémentaire est la suivante :

$$\forall u \in [1, n], \quad \forall k \in [1, n] \quad \sum_{i=0}^n xR_{i,u,k} = \sum_{j=0}^n xR_{u,j,k} \quad (45)$$

Cette contrainte ne modifie pas la fonction objectif mais permet d'obtenir une très faible diminution du temps d'exécution pour les instances de petites tailles testées.

Enfin, il faut calculer le moment de la livraison de chaque produit. Si un produit est le premier d'une tournée, alors il sera livré à la date de départ plus le temps de parcours qui aura été nécessaire. Cette contrainte est la suivante :

$$\forall j \in [1, n], \quad \forall k \in [1, n] \quad D_j \geq tB_k + tt_{0,j} - HV \times (1 - zB_{j,k}) \quad (46)$$

On va ensuite effectuer le même raisonnement pour chaque produit en prenant en compte le moment de livraison du produit précédent et en y ajoutant la distance entre les deux points de livraison. Cela donne la contrainte suivante :

$$\forall i \in [1, n], \quad \forall j \in [1, n], \quad \forall k \in [1, n | i \neq j] \quad D_j \geq D_i + tt_{i,j} - HV \times (1 - xR_{i,j,k}) \quad (47)$$

Enfin, on doit vérifier qu'une tournée ne puisse pas commencer alors qu'une autre est toujours en cours. Pour cela, on utilise la contrainte suivante :

$$\forall k \in [2, n] \quad tB_k \geq tB_{k-1} + \sum_{i=0}^n \sum_{j=0, i \neq j}^n tt_{i,j} \times xR_{i,j,k-1} \quad (48)$$

### 2.3.8 Contraintes d'accélération

En plus de toutes les autres contraintes citées, on a également ajouté les contraintes suivantes qui permettent d'obtenir des résultats similaires mais en réduisant le temps de calcul. Ce genre de contraintes est similaire aux autres contraintes superflues présentées précédemment et sont appelées des coupes. La première coupe concerne les valeurs de  $x_{i,j,t}$  pour la machine 1. En effet, même si on ne connaît pas à l'avance les valeurs qui vaudront 1, on peut déterminer les valeurs qui sont fixées à 0 à cause des bornes  $ra_j$  et  $rb_j$  additionné au temps maximum de pousse  $p1c_j$ . La contrainte est donc la suivante :

$$\forall j \in [1, n] \quad \sum_{t=0}^{ra_j} x_{1,j,t} + \sum_{t=rb_j+p1c_j+1}^T x_{1,j,t} = 0 \quad (49)$$

La vitesse de résolution peut être multipliée par 5 grâce à cette coupe.

On peut utiliser le même raisonnement pour la machine 2 en considérant que la récolte d'un produit ne peut pas commencer avant la somme entre la date de début au plus tôt et le temps minimum de la phase de pousse. Ainsi, on obtiens la contrainte ci-dessous :

$$\forall j \in [1, n] \quad \sum_{t=0}^{ra_j+p1a_j} x_{2,j,t} = 0 \quad (50)$$

Cependant, cette contrainte affecte négativement le temps de calcul de petites instances. Il serait donc nécessaire de vérifier son utilité sur un ensemble plus large d'instances comportant notamment des instances plus grandes.

Un autre moyen pour essayer de réduire le temps de résolution est de vérifier que la somme pour chaque tâche des  $z_{i,j,t}$  vaut 2 puisque chaque tâche commence une seule et unique fois sur chaque machine, et que la somme des  $x_{i,j,t}$  est égal au temps de pousse et de récolte. Les deux contraintes correspondantes sont les suivantes :

$$\forall j \in [1, n] \quad \sum_{i=1}^m \sum_{t=ra_j}^T z_{i,j,t} = 2 \quad (51)$$

$$\forall j \in [1, n] \quad \sum_{i=1}^m \sum_{t=ra_j}^T x_{i,j,t} = p1_j + p2_j \quad (52)$$

La première coupe permet d'obtenir des résultats environ 2 fois plus rapides et la seconde des résultats environ 3 fois plus rapides également.

Il est aussi possible de reprendre la contrainte (51) pour spécifier explicitement que si la somme des débuts de chaque tâche doit valoir 2, il faut également que chaque tâche possède un départ sur chaque machine. On peut donc ajouter cette contrainte pour la machine 2 comme le montre la contrainte ci-dessous :

$$\forall j \in [1, n] \quad \sum_{t=ra_j+p1a_j}^T z_{2,j,t} = 1 \quad (53)$$

Cette coupe permet d'obtenir une vitesse de résolution environ 3 plus rapide.

Enfin, deux dernières coupes ont été ajoutées pour minorer la date d'arrivée à un client  $j$  toujours dans le but d'augmenter les performances de la résolution. Dans le premier cas, on calcule le minorant en prenant la date de début de la tâche pour la machine 2 à laquelle on ajoute le temps de récolte de la tâche et la valeur de distance entre le client et le producteur. Ce minorant peut s'avérer égal dans le cas où le client  $j$  est le premier à être livré. Dans le second cas, le minorant est calculé de manière beaucoup plus large en additionnant la date de début de la phase de pousse du produit avec sa durée de pousse minimum, la valeur minimum que peut prendre  $p2j$  à savoir 1 et la valeur de distance entre le client et le producteur. Pour que le minorant puisse être égalé, il faut cette fois-ci que la livraison du client  $j$  se fasse au plus tôt possible. Les deux contraintes sont respectivement les suivantes :

$$\forall j \in [1, n] \quad D_j \geq \sum_{t=0}^T t \times z_{2,j,t} + p2j + tt_{0,j} \quad (54)$$

$$\forall j \in [1, n] \quad D_j \geq ra_j + p1a_j + 1 + tt_{0,j} \quad (55)$$

Les bénéfices de ces deux contraintes sont une vitesse environ 20 fois plus grande pour la première et une vitesse légèrement plus grande pour la deuxième.

## 2.4 Fonction objectif

Le but de la fonction objectif est de maximiser les gains. Le calcul du bénéfice prévu est calculé en retirant les divers coûts à l'argent gagné par la vente des produits. Le calcul est le suivant :

$$\max \text{ LesBenefits} = \text{TheReward} - (\text{Tardiness\_Cost} + \text{Hiring\_Cost}) \quad (56)$$

# 5

## Mise en oeuvre

Cette partie a pour but d'expliquer les différentes réalisations et les résultats obtenus.

### 1 Outils et librairie utilisés

Pour effectuer des résolutions de modèle, le solveur GLPK dans sa version 4.65 a été utilisé.

On peut également noter l'utilisation de librairies python à savoir :

1. random : permet de gérer l'aléatoire
2. os : permet de gérer les outils du système et notamment faire de la gestion de fichiers
3. time : permet de gérer le temps durant l'exécution du programme
4. subprocess : permet de lancer des sous processus pouvant notamment exécuter des résolution par le solveur GLPK en parallèle

Toutes ces librairies font partie des librairies standards et sont installées lors de l'installation de python.

### 2 Éléments d'implémentation, choix techniques

La première heuristique a pour but principal de tester le fonctionnement général de la méthode de résolution envisagée c'est-à-dire la diminution du temps de calcul en définissant des variables comme paramètres. Pour pallier la perte de qualité du résultat, il est envisagé de relancer plusieurs fois le nouveau modèle afin de ne garder que la meilleure solution.

L'algorithme de cette heuristique est le suivant :

```
1 resultats = []
2 solution_min = valeur négative très petite
3
4 maximum = solution_min
5 indice_max = -1
6
7 itemax = le nombre d'itérations voulues
8 temps_max = le temps à ne pas dépasser
9 chemin_fichier_dat = le chemin du fichier .dat
```

```

10 chemin_fichier_etendu = le chemin du fichier .dat étendu
11
12 n, Q1, Q2, p1a, p1c, ra, rb, p2max, aj, ddej, T, QVh <- analyse du
    modèle
13
14 nom_modele = "modele_partiel_v1.mod"
15 nom_stockage = "tmp_resultat.txt"
16
17 POUR iteration ALLANT DE 0 A itemax FAIRE
18     Y1j, p1j, Y2j, qj, p2j <- generation solution partielle du routing
19     création d'un nouveau fichier .dat avec les données étendues
20
21     lancement de la résolution
22
23     resolution_termine <- True
24
25     compteur <- 0
26     TANT QUE LA RESOLUTION N'EST PAS TERMINEE FAIRE
27         attendre 5 secondes
28         compteur <- compteur + 5
29         SI compteur >= temps_max ET temps_max != -1 FAIRE
30             resolution_termine = False
31             FIN TANT QUE
32
33     SI resolution_termine VAUT vrai FAIRE
34         valide, chemin, tb, LesBenefits <- analyse resultat(Solveur.
            CHEMIN_SIMULATION+nom_stockage,n)
35     SINON
36         valide <- False
37
38     SI valide VAUT Faux FAIRE
39         ajouter à resultats la solution vide
40     SINON
41         ajouter à resultat ((Y1j, p1j, Y2j, chemin, tb), LesBenefits)
42         SI LesBenefits > maximum FAIRE
43             indice_max <- iteration
44             maximum <- LesBenefits
45
46     SI maximum DIFFERENT solution_min FAIRE
47         la solution est : resultats[indice_max]
48     SINON
49         aucune solution n'a été trouvée

```

Dans cette méthode de résolution, il est possible de modifier deux variables pour changer les performances. Tout d'abord, on peut augmenter le nombre de solutions générées en modifiant la variable itemax. Ensuite, on peut également déterminer un temps de résolution maximal par le solveur grâce à la variable temps\_max. Le temps doit alors être spécifié en secondes.

La deuxième heuristique reprend le début de la première mais ne résout que l'ordonnancement grâce au solveur. Pour faire cela, la contrainte de la fonction objectif a dû être modifiée. Une fois que l'ordonnancement est terminé, on génère aléatoirement plusieurs tournées et on vérifie à chaque fois la qualité de la solution retournée. L'algorithme de la seconde heuristique est repris ci-dessous.

```

1  resultats = []
2  solution_min = valeur négative très petite
3
4  maximum = solution_min
5  indice_max = -1
6
7  itemax_ordo = le nombre d'itérations voulues pour l'ordonnancement
8  itemax_routing = le nombre d'itérations voulues pour la distribution
9  temps_max = le temps à ne pas dépasser
10 chemin_fichier_dat = le chemin du fichier .dat
11 chemin_fichier_etendu = le chemin du fichier .dat étendu
12
13 n, Q1, Q2, pla, plc, ra, rb, p2max, aj, ddej, T, QVh <- analyse du
    modèle
14
15 nom_modele_tmj = "modele_partiel_tmj_v1.mod"
16 nom_stockage_tmj = "tmp_resultat_tmj.txt"
17
18 nom_modele_routing = "modele_partiel_routing_v1.mod"
19 nom_stockage_routing = "tmp_resultat_routing.txt"
20
21 POUR iteration ALLANT DE 0 A itemax FAIRE
22     Y1j, p1j, Y2j, qj, p2j <- generation solution partielle du routing
23     création d'un nouveau fichier .dat avec les données étendues
24
25     lancement de la résolution de l'ordonnancement uniquement
26
27     resolution_termine <- True
28
29     compteur <- 0
30     TANT QUE LA RESOLUTION N'EST PAS TERMINEE FAIRE
31         attendre 5 secondes
32         compteur <- compteur + 5
33         SI compteur >= temps_max ET temps_max != -1 FAIRE
34             resolution_termine = False
35             FIN TANT QUE
36
37     SI resolution_termine VAUT vrai FAIRE
38         valide, chemin, tb, LesBenefits <- analyse resultat
39     SINON
40         valide <- False
41
42     SI valide VAUT Faux FAIRE
43         ajouter à resultats la solution vide
44     SINON
45         z <- extrapolation des valeurs de z avec t1, p1j et T

```

```

46
47     POUR iteration_routing ALLANT DE 0 A itermx_routing FAIRE
48         indice_iteration += 1
49
50         solution_routing <- generation solution partielle de la
           distribution
51
52         xR <- extrapolation des valeurs de xR avec
           solution_routing
53
54         création d'un nouveau fichier .dat avec les données é
           tendues
55
56         lancement de la résolution de la distribution uniquement
57
58         resolution_termine <- True
59
60         compteur <- 0
61         TANT QUE LA RESOLUTION N'EST PAS TERMINEE FAIRE
62             attendre 5 secondes
63             compteur <- compteur + 5
64             SI compteur >= temps_max ET temps_max != -1 FAIRE
65                 resolution_termine = False
66             FIN TANT QUE
67
68         SI resolution_termine VAUT vrai FAIRE
69             valide, LesBenefts, tb <- analyse resultat
70         SINON
71             valide <- False
72
73         SI valide VAUT Faux FAIRE
74             ajouter à resultats la solution vide
75         SINON
76             ajouter à resultat ((Y1j, p1j, Y2j, chemin, tb), LesBenefts
              )
77             SI LesBenefts > maximum FAIRE
78                 indice_max <- iteration
79                 maximum <- LesBenefts
80
81     SI maximum DIFFERENT solution_min FAIRE
82         la solution est : resultats[indice_max]
83     SINON
84         aucune solution n'a été trouvée

```

Il est à nouveau possible de modifier le nombre d'itérations à réaliser avec les variables `itermx_ordo` qui fixe le nombre de solutions d'ordonnancement générées et `itermx_routing` qui fixe le nombre de solutions de distribution pour chaque solution d'ordonnancement réalisé. Le nombre de solutions total devient donc `itermx_ordo × itermx_routing` à condition que toutes les solutions d'ordonnancement soient réalisables. On retrouve également la possibilité de limiter le temps d'exécution du solveur avec la variable `temps_max`.

Il est à noter que les méthodes de résolutions proposées ci-dessus ont pour but d'enregistrer

l'intégralité des solutions proposées. Cela permet d'analyser les solutions explorées pour pouvoir améliorer l'heuristique. Cependant, dans le rendu, l'affichage de l'intégralité des solutions a été commenté pour ne pas polluer les informations affichées dans la console.

### 3 Analyse des résultats, évaluation, qualité

Les différentes méthodes de résolution ont été testées sur des instances de petites tailles (4 ou 5 tâches à ordonnancer), des instances de moyennes tailles (10 tâches à ordonnancer) et des instances de grandes tailles (20 tâches à ordonnancer). Étant donné que certaines résolutions prennent trop de temps à s'exécuter, un temps maximum d'une heure a été fixé à l'aide de la variable `temps_max` qui prend la valeur 3600. Les résultats obtenues pour l'obtention d'une solution sont les suivants :

		résolution utilisée		
		optimale	heuristique 1	heuristique 2
Nombre de tâches	4-5	de instantané à plus d'une heure	moins d'une minute	instantané
	10	temps dépassé	temps dépassé	quelques minutes
	20	temps dépassé	temps dépassé	temps dépassé

**Figure 5.1** – Tableau du temps d'exécution d'une solution en fonction des instances et de la méthode utilisée

Cependant, les solutions générées par les différentes méthodes ne sont pas toutes de même qualité. Il est cependant assez compliqué de quantifier la perte de qualité puisque dans la plupart des cas on ne connaît pas la solution optimale. Pour cette raison, on se contentera d'une estimation qualitative plutôt que quantitative. Les résultats peuvent être décrits de la manière suivante :

		résolution utilisée		
		optimale	heuristique 1	heuristique 2
Nombre de tâches	4-5	optimal	correct	moyen
	10			très mauvais
	20			

**Figure 5.2** – Tableau de la qualité d'une résolution en fonction des instances et de la méthode utilisée

Pour expliciter les résultats, on va se servir de l'instance `donnees_base.dat` (dont le temps d'exécution pour chaque méthode est respectivement de 40 minutes, 45 secondes et quelques secondes). La solution optimale est de 1302, l'heuristique 1 donne un résultat la plupart du temps compris entre -400 et 150 et l'heuristique 2 donne un résultat la plupart du temps compris entre -3000 et -200. Pour les instances de 10 tâches, même si on ne connaît pas la solution optimale, on peut noter des résultats autour de valeurs tel que -40 000 ou -30 000 et qui ont donc peu de chances d'être proches de la solution optimale.

# 6

## Bilan et conclusion

### 1 Bilan du semestre 9

Pendant ce semestre, j'ai effectué les recherches nécessaires à la compréhension du sujet. Cela m'a permis de rédiger un cahier de spécifications présent en annexe afin de préparer les consignes de réalisation du logiciel. Avec la **MOA**, nous avons pu commencer la modélisation du problème qu'il faudra mettre en œuvre au prochain semestre. Ainsi, toutes les tâches prévues ont été réalisées et la conception du modèle a pu être démarré en avance ce qui va permettre de commencer plus tôt la modélisation informatique du problème et le début de la conception du logiciel.

### 2 Bilan du semestre 10

L'objectif du projet étant de réaliser un modèle permettant de résoudre un problème d'ordonnement de tâches et de distribution a été rempli avec en plus des heuristiques pour permettre de réaliser des résolutions plus rapidement. Cependant, le modèle peut encore être amélioré pour pouvoir prendre en compte plus de possibilités organisationnelles. Il est également nécessaire de prévoir la création d'heuristiques plus perfectionnées pour permettre la résolution d'instances en moins de temps. Enfin, la création d'une interface pour permettre l'utilisation du programme par un utilisateur extérieur au projet sera également nécessaire lorsque les deux étapes précédentes auront été remplies.

### 3 Bilan sur la qualité

Tout au long du projet, un effort particulier a été porté à la qualité du programme dans le but de faciliter sa reprise dans un projet futur. Les conventions de nommage ont été respectées, de nombreux commentaires ont été écrits et des tests ont été effectués et mis en place pour vérifier la bonne implémentation de toutes les fonctions.

### 4 Bilan auto-critique

Avec ce projet, j'ai pu découvrir le domaine de la recherche. J'ai dû faire preuve de beaucoup de rigueur pour réussir à traiter les tâches en temps voulu. En revanche, la fin du projet a été compliquée avec en plus les différents rendus à effectuer ce qui a causé des retards et finalement l'impossibilité de terminer à temps une dernière fonction non primordiale.

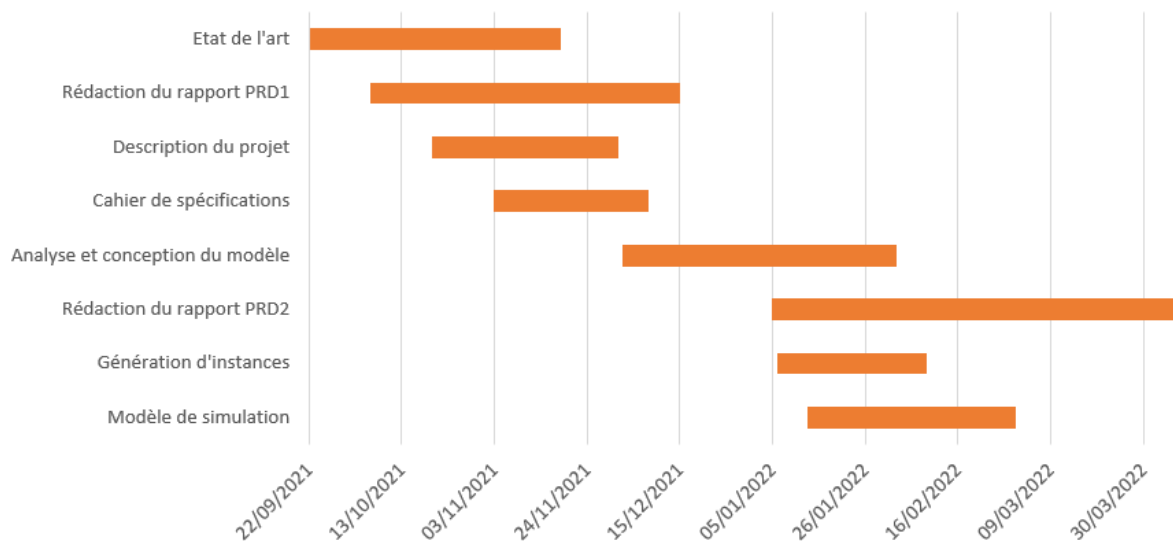
## Annexes

# A

## Planification, gestion de projet

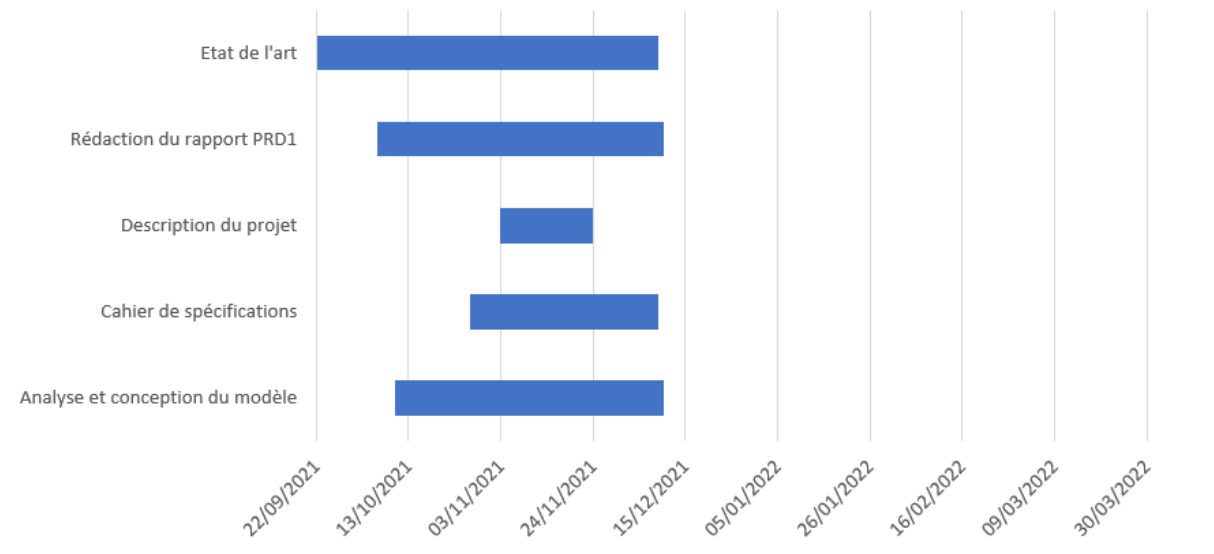
### 1 Evolution du projet

Un premier diagramme de Gantt a été réalisé dans le but de prévoir les grandes étapes du projet ainsi que leur temps de réalisation.



**Figure A.1** – Le diagramme de Gantt prévisionnel

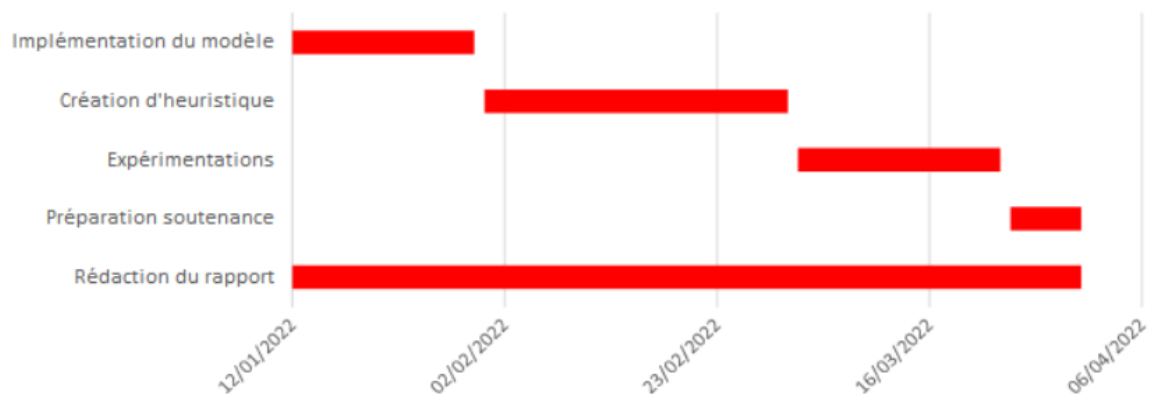
Cependant, la gestion du projet a amené à allonger certaines tâches et à en démarrer certaines plus tôt. Ainsi, à la fin du semestre 9, le diagramme de gantt correspondant au travail réalisé est le suivant.



**Figure A.2** – Le diagramme de Gantt à la fin du semestre 9

Les tâches n'ayant pas commencé ne sont pas affichées sur ce diagramme. De plus, les tâches toujours en cours et qui se poursuivent au semestre 10 sont arrêtées au 10 décembre, date de rendu du premier rapport.

Au début du semestre 10, un nouveau diagramme de gantt prévisionnel a été réalisé dans le but d'organiser le temps de travail restant, mais aussi pour développer plus précisément les différentes étapes de la résolution d'instances. Le diagramme de gantt est le suivant :



**Figure A.3** – Le diagramme de Gantt prévisionnel à la fin du semestre 10

Enfin, le diagramme de gantt final est affiché ci-dessous. On peut voir notamment que la plupart des tâches ont été réalisées globalement dans le temps alloué. Les tâches pour ce diagramme ont été arrêtées au 2 mars.

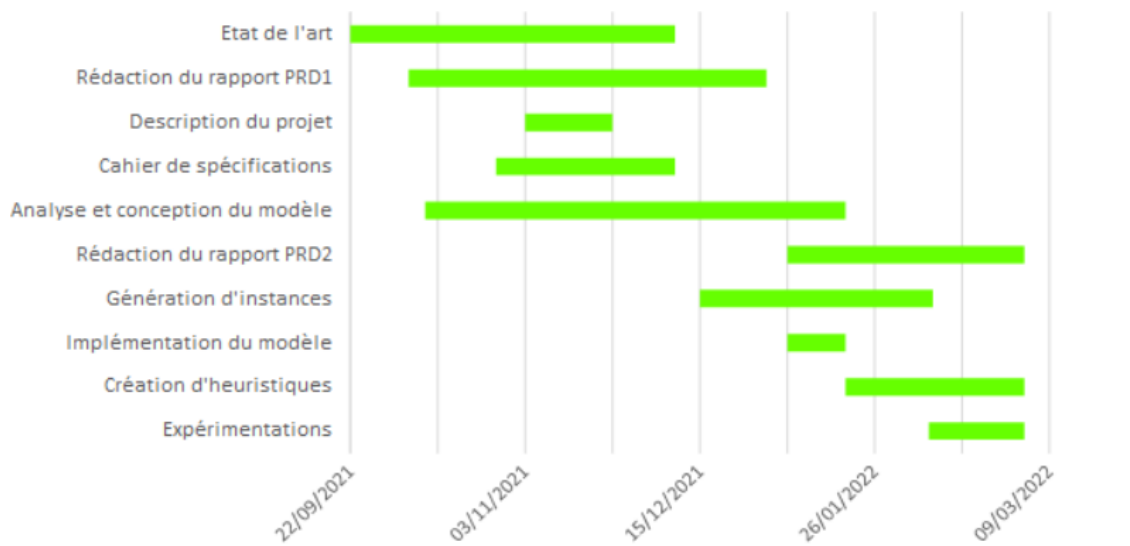


Figure A.4 – Le diagramme de Gantt final à la fin du projet

Pour mieux voir les durées exactes et les dates de début et fin de chaque tâche, on peut se référer aux tableaux suivants :

Tâche	date de début	durée	date de fin
Modèle de simulation	13/01/2022	47	01/03/2022
Génération d'instances	06/01/2022	34	09/02/2022
Rédaction du rapport PRD2	05/01/2022	91	06/04/2022
Analyse et conception du modèle	02/12/2021	62	02/02/2022
Cahier de spécifications	03/11/2021	35	08/12/2021
Description du projet	20/10/2021	42	01/12/2021
Rédaction du rapport PRD1	06/10/2021	70	15/12/2021
Etat de l'art	22/09/2021	57	18/11/2021

Figure A.5 – Tableau pour le diagramme de Gantt prévisionnel

Tâche	date de début	durée	date de fin
Analyse et conception du modèle	10/10/2021	61	10/12/2021
Cahier de spécifications	27/10/2021	43	09/12/2021
Description du projet	03/11/2021	21	24/11/2021
Rédaction du rapport PRD1	06/10/2021	65	10/12/2021
Etat de l'art	22/09/2021	78	09/12/2021

Figure A.6 – Tableau pour le diagramme de Gantt à la fin du semestre 9

Tâche	date de début	durée	date de fin
Rédaction du rapport	12/01/2022	78	31/03/2022
Préparation soutenance	24/03/2022	7	31/03/2022
Expérimentations	03/03/2022	20	23/03/2022
Création d'heuristique	31/01/2022	30	02/03/2022
Implémentation du modèle	12/01/2022	18	30/01/2022

Figure A.7 – Tableau pour le diagramme de Gantt prévisionnel pour le semestre 10

Tâche	date de début	durée	date de fin
Expérimentations	08/02/2022	23	03/03/2022
Création d'heuristiques	19/01/2022	43	03/03/2022
Implémentation du modèle	05/01/2022	14	19/01/2022
Génération d'instances	15/12/2021	56	09/02/2022
Rédaction du rapport PRD2	05/01/2022	57	03/03/2022
Analyse et conception du modèle	10/10/2021	101	19/01/2022
Cahier de spécifications	27/10/2021	43	09/12/2021
Description du projet	03/11/2021	21	24/11/2021
Rédaction du rapport PRD1	06/10/2021	86	31/12/2021
Etat de l'art	22/09/2021	78	09/12/2021

Figure A.8 – Tableau pour le diagramme de Gantt à la fin du projet

## 2 Description des tâches

### 2.1 Tâche 1 : Etat de l'art

- Date de début : 22/09/2021
- Date de fin : 09/12/2021
- Durée : 78 jours
- Description : Rédaction d'une synthèse des connaissances actuelles dans la littérature scientifique notamment avec des résumés de publications.

### 2.2 Tâche 2 : Rédaction du rapport PRD1

- Date de début : 06/10/2021
- Date de fin : 31/12/2021
- Durée : 86 jours
- Description : Première partie de la rédaction du rapport de PRD.

### 2.3 Tâche 3 : Description du projet

- Date de début : 03/11/2021
- Date de fin : 24/11/2021
- Durée : 21 jours
- Description : Ecriture d'une description du sujet spécifiant les éléments à traiter et les limites du projet ainsi que les hypothèses prises en comptes.

### 2.4 Tâche 4 : Cahier de spécifications

- Date de début : 27/10/2021
- Date de fin : 09/12/2021
- Durée : 43 jours
- Description : Ecriture d'un cahier de spécifications.

**2.5 Tâche 5 : Analyse et conception du modèle mathématique**

- Date de début : 10/10/2021
- Date de fin : 19/01/2022
- Durée : 101 jours
- Description : Création d'un modèle mathématique permettant de simuler le problème de production-distribution.

**2.6 Tâche 6 : Rédaction du rapport PRD2**

- Date de début : 05/01/2022
- Date de fin : 03/03/2022
- Durée : 57 jours
- Description : Rédaction de la deuxième partie du rapport de PRD.

**2.7 Tâche 7 : Génération d'instances**

- Date de début : 15/12/2021
- Date de fin : 09/02/2022
- Durée : 56 jours
- Description : Création d'instances permettant de lancer des simulations sur les données.

**2.8 Tâche 8 : Modèle de simulation**

- Date de début : 05/01/2022
- Date de fin : 03/03/2022
- Durée : 57 jours
- Description : Simulations du modèle à partir des instances générées et du modèle mathématique. Cette tâche peut être divisée en 2 sous-tâches respectivement l'implémentation du modèle (tâche 9) et la création d'heuristiques (tâche 10).

**2.9 Tâche 9 : Implémentation du modèle**

- Date de début : 05/01/2022
- Date de fin : 19/01/2022
- Durée : 14 jours
- Description : Implémentation du modèle pour permettre une résolution afin d'obtenir une solution optimale au problème de production-distribution conçu à la tâche 5.

**2.10 Tâche 10 : Création d'heuristiques**

- Date de début : 19/01/2022
- Date de fin : 03/03/2022
- Durée : 43 jours
- Description : Implémentation d'heuristiques pour permettre une résolution afin d'obtenir une solution non optimale mais plus rapide au problème de production-distribution conçu à la tâche 5.

### 2.11 Tâche 11 : Expérimentations

- Date de début : 08/02/2021
- Date de fin : 03/03/2021
- Durée : 23 jours
- Description : Manipulation des différentes méthodes de résolution avec divers jeux de données pour vérifier les performances de chaque méthode.

### 2.12 Tâche 12 : Préparation soutenance

- Date de début : -
- Date de fin : -
- Durée : -
- Description : Rédaction des différents supports et préparation à la soutenance.

# B

# Cahier de Spécifications

## 1 Spécifications Fonctionnelles

### 1.1 Fonctionnalités à développer

Le programme sera composé de plusieurs fonctionnalités. Les principales fonctionnalités sont définies ci-dessous.

### 1.2 Définition de la fonction 1 : Lecture d'instance

**Présentation de la fonction 1 :**

#### Élément 1

Entrée : un nom de fichier sous le format d'une chaîne de caractères

Sortie : une instance du problème de production-distribution

Préconditions : le fichier spécifié en entrée doit contenir les données d'une instance à résoudre et dans le bon format

Postconditions : aucune

**Description de la fonction 1 :**

Le but de cette fonction est de permettre d'importer des instances automatiquement depuis des fichiers textes. L'instance obtenue en sortie sera ensuite utilisée dans le solveur.

### 1.3 Définition de la fonction 2 : Ecriture d'instance

**Présentation de la fonction 2 :**

#### Élément 1

Entrée : un nom de fichier sous le format d'une chaîne de caractères

Sortie : un fichier contenant les données du problème dans un format compréhensible par la fonction 1

Préconditions : le nom de fichier ne doit pas correspondre à un fichier déjà existant

Postconditions : le fichier en sortie doit contenir les données d'une instance à résoudre et dans le bon format

### Description de la fonction 2 :

Le but de cette fonction est de permettre de créer des instances afin d'obtenir un fichier valide pour la précondition de la fonction 1 sans que l'utilisateur n'ait besoin de le modifier directement ce qui pourrait amener à des erreurs de lectures ou des malentendus sur les données.

## 1.4 Définition de la fonction 3 : Résolution d'instance

### Présentation de la fonction 3 :

#### Élément 1

Entrée : une instance du problème de production-distribution

Sortie : affichage du résultat de la résolution

Préconditions : aucune

Postconditions : la solution retournée par la fonction doit être la solution optimale du problème

### Description de la fonction 3 :

Cette fonction a pour but d'effectuer la résolution d'une instance. Pour cela, elle utilisera principalement la fonction de lecture d'instance afin d'obtenir les données d'une instance, puis utilisera le solveur pour obtenir le résultat de l'analyse.

## 1.5 Définition de la fonction 4 : Sauvegarde de résultat

### Présentation de la fonction 4 :

#### Élément 1

Entrée : la liste des informations à sauvegarder sous le format d'une liste de chaînes de caractères

Sortie : un fichier

Préconditions : aucune

Postconditions : le fichier doit contenir les informations à sauvegarder passées en entrée

### Description de la fonction 4 :

Cette fonction a pour but d'effectuer la sauvegarde du résultat obtenu par le solveur.

## 2 Spécifications non fonctionnelles

### 2.1 Contraintes de développement et conception

Je programmerai sur un système d'exploitation Windows. La solution sera développée en python 3.8 avec l'environnement spyder. J'utiliserai également le solveur GLPK utilisant le langage GLPM, pour résoudre des modélisations, qui sera appelé dans le code python à partir de l'exécutable présent dans le projet. Une interface graphique pourra être ajoutée mais n'est pas nécessaire. Dans ce cas, elle serait implémentée avec le package arena. Dans un premier temps, il n'est prévu qu'un seul modèle, cependant dans le cas où ce modèle serait terminé rapidement, d'autres modèles pourront être ajoutés pour permettre de prendre en compte de nouveaux paramètres.

## 2.2 Contraintes de fonctionnement et d'exploitation

### 2.2.1 Performances

Les performances du programme ne nécessitent pas un temps de réponse rapide. J'essaierai de réduire le temps d'exécution notamment sur la partie python, mais le plus gros du temps de calcul sera de toute façon lié à l'utilisation du solveur GLPK. Pour de petites instances, le but sera de trouver des solutions optimales, mais pour de plus grosses instances, il faudra se contenter de solutions approchées calculées avec des heuristiques sans quoi les temps de calculs pourraient être trop longs. L'ajout d'heuristiques se fera dans la partie python du programme.

### 2.2.2 Capacités

Le programme devra être en mesure de traiter un problème contenant un seul producteur avec un maximum de 10 livraisons différentes. Dans la mesure du possible, on essaiera de fournir en sortie un résultat clair sur la résolution du modèle.

### 2.2.3 Contrôlabilité

La résolution et la génération d'instance pourront être utilisées en lançant le programme python en ligne de commande par exemple. Elle permettra de lancer automatiquement la résolution avec le solveur GLPK.

Dans le cas où il serait impossible ou trop complexe de lancer GLPK directement à partir du programme python et d'en obtenir les résultats, il faudrait après avoir lancé le script python, permettant d'importer une instance, lancer manuellement la résolution à l'aide du logiciel GLPK. Dans ce cas, la fonction 2 aura pour rôle de créer un fichier .dat pouvant être exploité par le solveur.

L'utilisateur pourra suivre l'évolution du processus de résolution avec un affichage pendant la résolution et aura un aperçu de la solution obtenue dans la console de l'application.

### 2.2.4 Sécurité

Le logiciel ne sera accessible que pour la MOA. Il n'y a donc pas de sécurité particulière à mettre en place d'autant que le logiciel n'a pas besoin de connexion à internet.

## 2.3 Contraintes d'évolution

Pour permettre la reprise de la solution, un rapport de projet contenant des informations et pouvant servir de documentation sera fourni en fin de projet. Pour permettre une meilleure lisibilité du code, des commentaires seront ajoutés aux différentes parties du programme. Une convention de nommage sera également utilisée suivant les règles suivantes pour python :

- une constante d'utilisation sera écrite en screaming snake case, ex : `UNE_CONSTANTE`
- une variable sera écrite en snake case, ex : `une_variable`
- une fonction sera écrite en snake case, ex : `une_fonction()`
- une classe sera écrite en snake case avec une majuscule sur le premier mot, ex : `Une_classe`

Le code ainsi que les commentaires seront écrits en français.



# Cahier du développeur

## 1 Diagrammes architecturaux et UML

### 1.1 Structure du projet

Le projet est composé des classes suivantes :

1. Generateur : permet de créer des instances rapidement
2. Analyseur : permet d'analyser des fichiers pour en obtenir des informations qui sont exploitées dans la classe Solveur
3. Solveur : permet de résoudre une instance du problème de production-distribution
4. Aleatoire : permet d'obtenir des nombres aléatoires

L'agencement des classes ainsi que leur position dans des fichiers est résumé sur l'UML suivant :

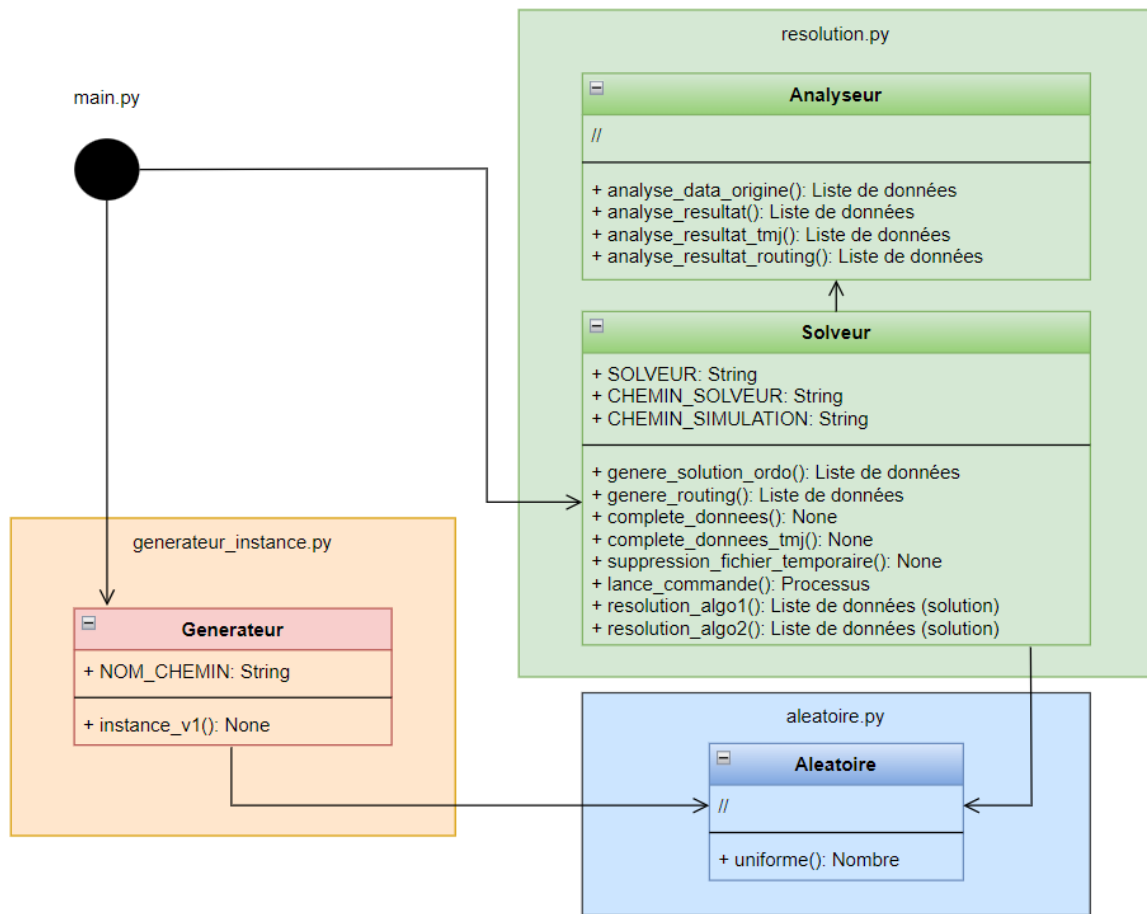


Figure C.1 – UML du programme avec les principales fonctions

## 1.2 Architecture de résolution d'une instance

Entre la résolution optimale et les heuristiques, il existe 3 moyens différents de résoudre une instance mettant en jeu 3 formes de jeux de données et 4 modèles de résolution. Le premier type d'instance comporte en paramètres tous les paramètres présentés dans l'analyse et la conception du modèle. Cette instance est faite pour être résolu de manière optimale par le modèle : `modele_ordonnancement_routing_v1.mod`

Le deuxième type d'instance comporte en plus de tous les paramètres de la première instance, les paramètres  $p1_j$ ,  $p2_j$ ,  $q_j$  et  $Y1_j$  et  $Y2_j$  réunis en un paramètre  $Y_{m,j}$  définis initialement en tant que variable. Cette instance peut alors être résolu par le modèle suivant pour obtenir une solution complète :

`modele_partiel_v1.mod` Cette résolution est pratiquement la même que pour la solution optimale mais en modifiant les variables fixées comme paramètres du modèle.

Mais ce type d'instance peut également être résolue pour déterminer uniquement une solution de production avec le modèle suivant :

`modele_partiel_tmj_v1.mod`

On va à nouveau réutiliser les paramètres, les variables et les contraintes restantes. Cependant, un des problèmes qui apparaît lorsqu'on ne réalise que la production est l'impossibilité de conserver la fonction objective initiale (56)(Chapitre 4) à cause du calcul des coûts liés aux retards définis lors de la distribution (1)(Chapitre 4). On va donc faire une estimation de ce temps de retard

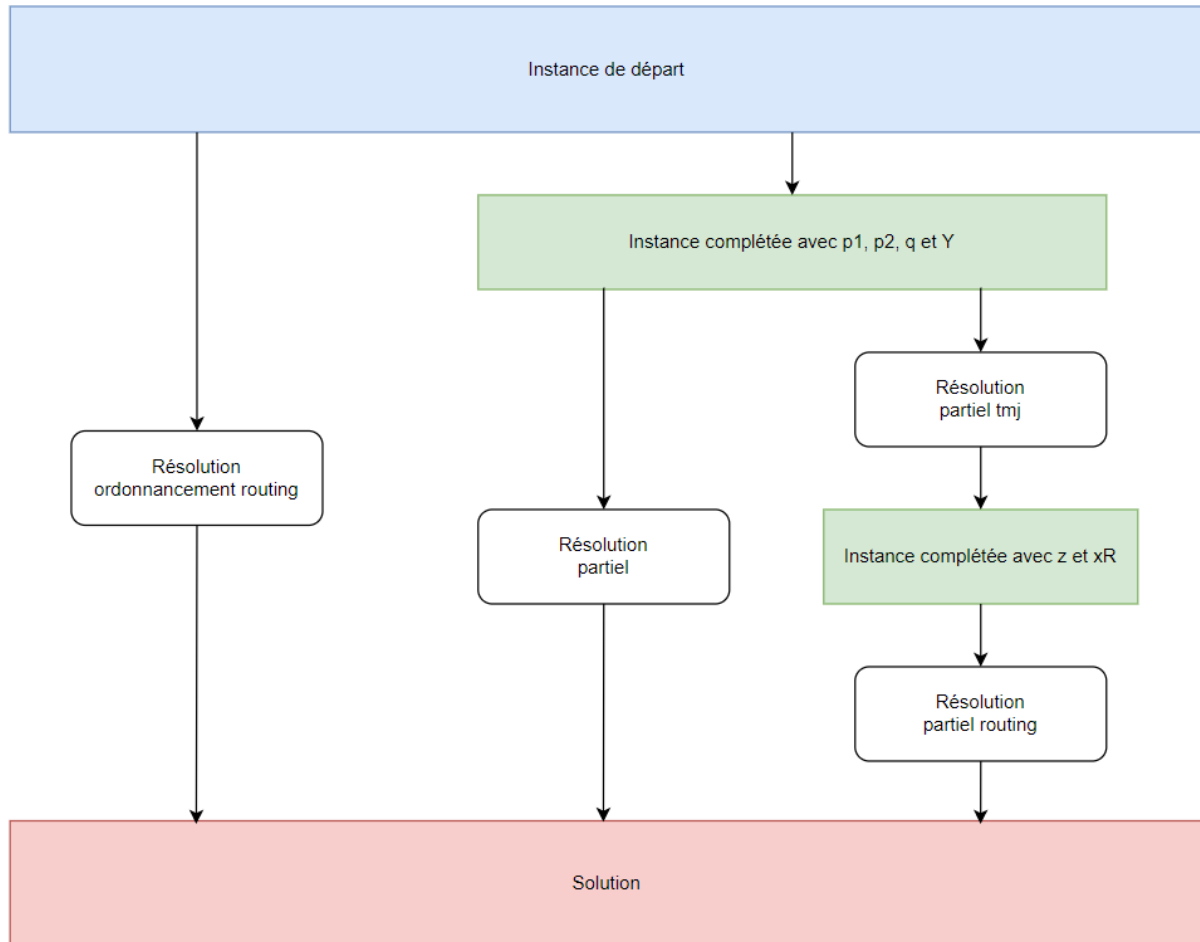
pour transformer la contrainte de calcul de  $TD_j$  de la manière suivante :

$$\forall j \in [1, n] \quad TD_j \geq \sum_{t=0}^T t \times x_{2,j,t} + p_{2j} - \text{deadline}_j \quad (1)$$

Le troisième type d'instance comporte en plus des paramètres de la deuxième instance les paramètres  $z_{i,j,t}$  et  $xR_{i,j,k}$  définis initialement en tant que variable. La résolution de cette instance se fait grâce au modèle :

modele\_partiel\_routing\_v1.mod Ce modèle est également extrapolé depuis le premier modèle auquel on a retiré les variables et les contraintes qui ne sont plus nécessaires.

L'utilisation des différents modèles et des 3 types d'instances est résumé dans le diagramme suivant :



**Figure C.2** – Le diagramme de résolution d'une instance

# D

## Document d'installation

Le projet est constitué d'un dossier contenant les scripts pythons. L'installation de ce dossier en plus d'avoir une version de python fonctionnelle est le seul pré-requis pour pouvoir lancer les différents scripts.

Toutes les bibliothèques utilisées dans les scripts font partie des bibliothèques par défaut de python et il n'y a donc pas d'installations supplémentaires à effectuer.

# E

## Document d'utilisation

Pour pouvoir utiliser les différentes réalisations, il faut importer les classes de la manière suivante (si le script python se trouve dans le même dossier du projet) :

- pour la génération d'instance : `from generateur_instance import Generateur`
- pour la résolution d'instance : `from resolution import Solveur`

Une fois le package importé, il est possible d'utiliser les commandes suivantes : Si on veut générer une instance, alors il faut utiliser : `Generateur.instance_v1()`

Cette fonction prend en paramètre le nom du fichier à créer, le nombre de tâches à générer (ce qui correspond au paramètre `n` du modèle) et l'horizon de la planification (ce qui correspond au paramètre `H` du modèle).

Si on veut résoudre une instance, alors il faut utiliser : `Solveur.resolution_algo1()` ou `Solveur.resolution_algo2()` selon l'heuristique que l'on veut utiliser.

Cette fonction prend un unique paramètre qui est le nom du fichier du modèle à résoudre.

# F

## Cahier de test

Cette partie a pour but de présenter les différents tests du projet.

### 1 Tests unitaires

Les tests unitaires sont effectués grâce à la librairie unittest de python, une librairie standard du langage.

#### 1.1 Tests du générateur d'instance

COMPOSANT
Génération d'instances
DESCRIPTION
Le but est de vérifier la présence du fichier créé après l'exécution de la commande. Pour cela, on vérifie dans un premier temps que l'emplacement prévu est vide, puis on lance le générateur et enfin on supprime le fichier.
RESULTATS ATTENDUS
Création d'un fichier.
RESULTATS OBTENUS
Création d'un fichier.

## 1.2 Tests du Solveur

COMPOSANT
Résolution d'instances
DESCRIPTION
Le but est de vérifier que le calcul pour retrouver la variable $q_j$ à partir de $a_j$ et $Y1_j$ se fait bien.
RESULTATS ATTENDUS
Liste de valeurs : [3,2,1].
RESULTATS OBTENUS
Liste de valeurs : [3,2,1].

COMPOSANT
Résolution d'instances
DESCRIPTION
Le but est de vérifier que le calcul pour retrouver la variable $p2_j$ à partir de $p2max_j$ , $q_j$ et $Y2_j$ se fait bien et notamment que la valeur de $p2_j$ est au moins égale à 1.
RESULTATS ATTENDUS
Liste de valeurs : [2,1].
RESULTATS OBTENUS
Liste de valeurs : [2,1].

COMPOSANT
Analyse d'instances
DESCRIPTION
Le but est de vérifier que l'analyseur est capable de lire des données dans un fichier. Pour cela, on crée un petit fichier définissant uniquement le nombre de tâche et la date de début au plus tôt de chacune.
RESULTATS ATTENDUS
Liste de valeurs avec $n = 2$ , $p1a = [8, 4]$ et les autres valeurs à None.
RESULTATS OBTENUS
Liste de valeurs avec $n = 2$ , $p1a = [8, 4]$ et les autres valeurs à None.

## 2 Scénario de test

La vérification de la possibilité de résolution d'une instance est plus complexe et ne peut pas être testée simplement de manière automatique. Pour cela, 3 scénarios de test ont été envisagés. Pour se préparer à ces scénarios, il faut dans un premier temps aller à l'emplacement suivant du projet : programme/glpk-4.65/w64 et lancer un invité de commande (écrire "cmd" dans la barre d'adresse et appuyer sur la touche entrer).

Dans les commandes qui suivent, il faut remplacer le signe [NOM\_FICHER\_INSTANCE] par le nom du fichier. Le chemin du fichier doit comporter l'extension .dat. Pour tester la résolution d'une instance original, il faut utiliser la commande :

```
glpsol -model ../../simulation/modele_ordonnancement_routing_v1.mod
-data ../../simulation/[NOM_FICHER_INSTANCE].dat
```

Pour tester la résolution d'une instance avec des données partielles d'ordonnancement, il faut utiliser la commande :

```
glpsol -model ../../simulation/modele_partiel_v1.mod
-data ../../simulation/[NOM_FICHER_INSTANCE].dat
```

Pour tester la résolution d'une instance avec les données complètes d'ordonnancement et des données partielles de routing, il faut utiliser la commande :

```
glpsol -model ../../simulation/modele_partiel_routing_v1.mod
-data ../../simulation/[NOM_FICHER_INSTANCE].dat
```

Dans le cas en revanche où on veut juste vérifier qu'une instance avec des données partielles est solvable, il est possible d'utiliser la commande suivante qui ne vérifie pas la phase de distribution :

```
glpsol -model ../../simulation/modele_partiel_tmj_v1.mod
-data ../../simulation/[NOM_FICHER_INSTANCE].dat
```

Pour savoir si le test est positif ou non, il faut regarder les informations renvoyées par l'invité de commande. Si le message OPTIMAL LP SOLUTION FOUND apparaît, alors le test est réussi. Ci-dessous, un exemple de l'affichage que l'on peut obtenir.

```
Solving LP relaxation...
GLPK Simplex Optimizer, v4.65
6925 rows, 4214 columns, 27035 non-zeros
    0: obj =  4.281000000e+03 inf =  1.031e+03 (102)
   1614: obj =  4.281000000e+03 inf =  5.638e-16 (0) 8
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Long-step dual simplex will be used
+ 1614: mip =      not found yet <=      +inf      (1; 0)
```

Figure F.1 – Exemple de résolution faisable

Il est important de noter que si ce message apparaît, alors la résolution de l'instance va commencer. Pour arrêter l'exécution, il est possible d'utiliser le raccourci CTRL + C.

Si le message PROBLEM HAS NO PRIMAL FEASIBLE SOLUTION apparaît, alors cela signifie que l'instance correspond bien au modèle mais qu'il n'existe aucune solution. Dans ce cas, il n'est pas nécessaire de stopper l'exécution. Un exemple de l'affichage est le suivant :

```
GLPK Integer Optimizer, v4.65
1028 rows, 583 columns, 4137 non-zeros
368 integer variables, all of which are binary
Preprocessing...
PROBLEM HAS NO PRIMAL FEASIBLE SOLUTION
Time used:  0.0 secs
Memory used: 1.5 Mb (1595022 bytes)
```

Figure F.2 – Exemple de résolution infaisable

Enfin, si le message MathProg model processing error apparaît, alors cela signifie que l'instance

ne correspond pas au modèle de résolution utilisé. Un exemple de ce genre d'erreur est affiché ci-dessous :

```
Reading data section from ../../simulation/donnees_base_accelere.dat...  
../../simulation/donnees_base_accelere.dat:17: one item missing in data group beginning with 1  
Context: ...am T := 15 ; param HV := 1000 ; param p1a := 1 2 2 3 4 4 1 ;  
MathProg model processing error
```

**Figure F.3** – *Exemple d'erreur de résolution*

Il est à noter comme on peut le voir dans l'exemple qu'en cas d'erreur, le message apporte des informations sur l'origine de l'erreur. Dans le cas présent, il s'agit d'une valeur non défini pour l'indice 1 d'un groupe de valeur (liste, tableau, etc...) dans le fichier .dat.

- [1] C B Basnet et L R FOULDS ET J M WILSON. « Scheduling Contractors' Farm-to-Farm Crop Harvesting Operations ». In : (2018).
- [2] Pierre-Antoine MORIN. « Planification et ordonnancement de projets sous contraintes de ressources complexes ». In : (2018). URL : <https://hal.laas.fr/tel-02053199v1>.
- [3] Maxime OGIER. « Contributions à la chaine logistique numérique : conception de circuits courts et planification décentralisée. » In : (2013). URL : <https://tel.archives-ouvertes.fr/tel-00981923>.

# H

## Acronymes

**B&C** Branche & Cut. 11, 12

**CRS** Conception de Réseau de Service. 9

**DB** Décomposition de Benders. 11, 12

**MIP** Mixed Integer Program ou Programme Linéaire à Variables Mixtes. 9, 18

**MOA** Maitrise d'Ouvrage. 1, 3, 5, 29

**MOE** Maitrise d'Oeuvre. 1

**PPD** Procédure de Pentes Dynamiques. 11, 12

**RCPSP** Resource-Constrained Project Scheduling Problem ou Problème d'ordonnancement de projet sous contraintes de ressources. 8

**TS** Tabu Search ou Recherche Tabou. 7, 8

**UML** Unified Modeling Language. 3

Thomas DUVAL

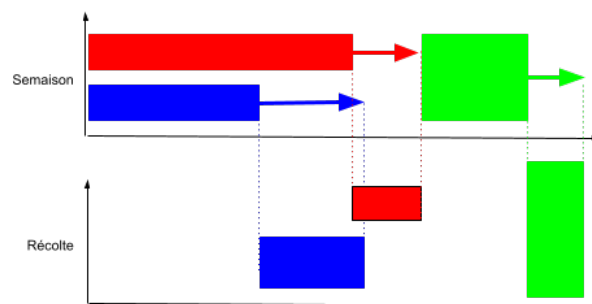
Encadrement : Jean Charles BILLAUT



En collaboration avec Polytech

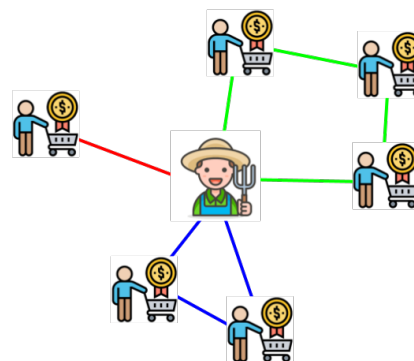
## Ordonnancement

L'ordonnancement est un problème qui consiste à organiser un emploi du temps pour pouvoir réaliser différentes tâches. Les différentes contraintes à respecter sont de planifier les tâches de récolte après la tâche de semaison, d'avoir un terrain où semer pour une tâche de semaison d'avoir de la main d'oeuvre pour une tâche de récolte.



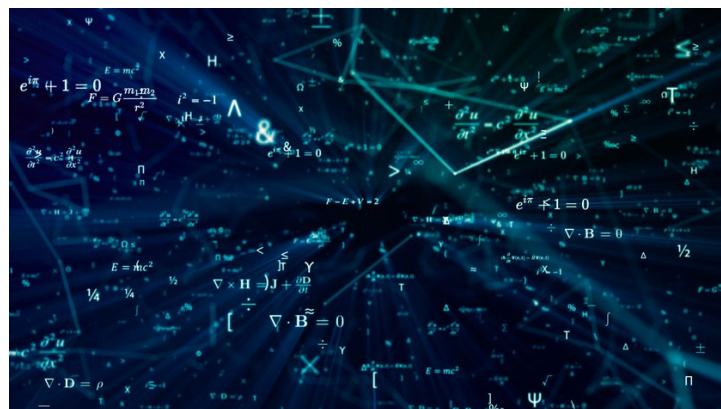
## Distribution

La distribution est un problème qui consiste à organiser des tournées pour permettre la livraison de produits demandés par le client. Il faut prendre en compte les distances à parcourir mais aussi la capacité du véhicule



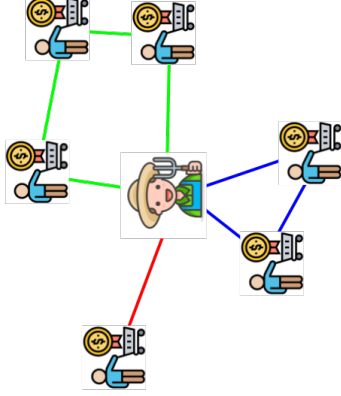
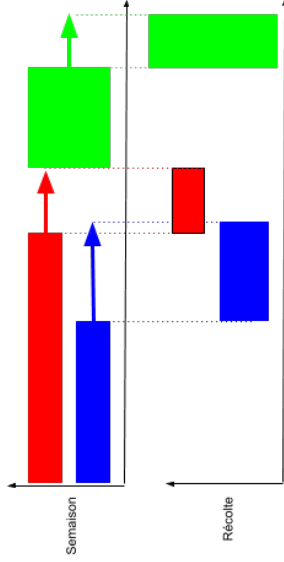
## Heuristique

Une heuristique est une méthode qui permet d'obtenir une solution à un problème de manière beaucoup plus rapide qu'une méthode exacte. L'inconvénient d'utiliser une heuristique est de diminuer la qualité de la solution obtenue ainsi. Le but recherché est de trouver l'heuristique qui permettra d'obtenir le meilleur équilibre entre solution satisfaisante et temps de résolution raisonnable.



# Ordonnancement

L'ordonnement est un problème qui consiste à organiser un emploi du temps pour pouvoir réaliser différentes tâches. Les différentes contraintes à respecter sont de planifier les tâches de récolte après la tâche de semaison, d'avoir un terrain où semer pour une tâche de semaison d'avoir de la main d'oeuvre pour une tâche de récolte.



# Heuristique

Une heuristique est une méthode qui permet d'obtenir une solution à un problème de manière beaucoup plus rapide qu'une méthode exacte. L'inconvénient d'utiliser une heuristique est de diminuer la qualité de la solution obtenue ainsi. Le but recherché est de trouver l'heuristique qui permettra d'obtenir le meilleur équilibre entre solution satisfaisante et temps de résolution raisonnable.

# Logistique des circuits courts / Production-Distribution

## Résumé

Ce projet est un projet de recherche et développement proposé par l'Ecole Polytechnique de l'Université de Tours dans le cadre de la dernière année de préparation au diplôme d'ingénieur. Il est réalisé par moi, Thomas Duval sous la supervision de M. Billaut. Le but de ce projet est de permettre la création d'un programme dans le but d'améliorer la logistique des circuits courts dans le milieu agricole afin de faciliter la production et la distribution. Ce projet s'inscrit dans la continuité de deux autres projets de recherche et développement cherchant eux à améliorer des thématiques proches toujours en lien avec les circuits courts.

## Mots-clés

circuit court, production, distribution, heuristique

## Abstract

This project is a research and development project proposed by the Polytechnic School of the Tours University as part of the last year of preparation to the engineering degree. It is realized by me, Thomas Duval under the supervision of Mr. Billaut. Main goal of this project is to allow the creation of a software in order to improve the local distribution network in the agricultural environment to facilitate the production and the distribution. This project is in the continuation of two other research and development projects seeking to improve similar themes linked to the local distribution network.

## Keywords

local distribution network, production, distribution, heuristic

Entreprise

Polytech



Tuteur entreprise

Jean Charles BILLAUT

Étudiant

Thomas DUVAL (DI5)

Tuteur académique

Jean Charles BILLAUT