Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche & Développement
2015-2016

Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur"

Tuteurs académiques
Jean Charles Billaut

Étudiants
Laura Bellego (DI5)

26 septembre 2016

Liste des intervenants

Nom	Mail	Qualité
Laura Bellego	laura.bellego@etu.univ-tours.fr	Étudiant DI5
Jean Charles Billaut	billaut@univ-tours.fr	Tuteur académique, Département infomatique

Avertissement

Ce document a été rédigé par Laura Bellego susnommé l'auteur.

L'école polytechnique de l'université François Rabelais 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 reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

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

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

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

L'auteur reconnait 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.

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 :
Laura Bellego, *Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur":* , Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={Bellego, Laura},
    title={Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur": },
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2016/02/01)

Table des matières

Remerciements 1

Introduction 2

I Partie 1 : Recherche 3

1 Cahier des charges 4

1 Contexte 4

2 Objectifs 5

3 Travail à réaliser 5

3.1 Modèle mathématique 5

3.2 Générateur de données 6

3.3 Algorithme de résolution 6

4 Contraintes 7

2 Etat de l'art 8

1 Rappel du modèle mathématique 8

1.1 Formules 8

1.2 Solution 10

1.3 Scénarios 11

2 Générateur de données 11

2.1 Cas d'application 11

2.1.1 Application 1 11

2.1.2 Application 2 12

2.2 Recherche des paramètres 12

3 Modélisation et solution **15**
1 Générateur ... 15
1.1 Entrée ... 15
1.1.1 Application 1 ... 15
1.1.2 Application 2 ... 17
1.2 Sortie ... 19
1.3 Fonctionnement général ... 19
2 Algorithme de résolution ... 20
2.1 Choix de la méthode utilisée ... 21
2.1.1 Méthode optimale ... 21
2.1.2 Méthode approchée ... 21
2.2 Fonctionnement général ... 22
2.2.1 Codage de la solution ... 22
2.2.2 Évaluation de cette solution ... 23
2.2.3 Exploration des voisinages ... 27
2.2.4 Choix de la meilleure solution ... 29
2.2.5 Fonction supplémentaire : La tournée du distributeur ... 30

II Partie 2 : Développement **31**

4 Mise en oeuvre **32**
1 Implémentation ... 32
1.1 Générateur de données ... 32
1.2 Algorithme de résolution ... 34
2 Détail sur un point difficile ... 36
2.1 SWAP ... 37
2.2 EBSR et EFSR ... 37
3 Complexité ... 37

5 Campagne de tests et résultats d'application des tests **38**
1 Conditions d'expérimentation ... 38
2 Génération des données de test ... 38
3 Déroulement des tests ... 38
4 Lancement des tests ... 39
5 Résultats des tests de performance ... 39
6 Résultats des tests fonctionnels ... 40

6 Reproductibilité **44**
1 Générateur de données ... 44
1.1 Fichier d'entrée et de sortie ... 44
1.2 Lancement du programme et résultat ... 46
1.3 Exemple ... 48
2 Algorithme de résolution ... 49
2.1 Configuration de l'IDE Eclipse ... 49
2.2 Fichier d'entrée et de sortie ... 51
2.3 Lancement du programme et résultat ... 51
2.4 Problème rencontré ... 52
2.5 Exemple ... 52

Gestion de projet 54

3 Planning 54

4 Livrables 56

5 Méthodologies et outils utilisés 56

Conclusion 57

Table des figures

1 Cahier des charges

1 Organisation du producteur 4
2 Organisation du distributeur 5
3 Analyse des coûts pour les 3 scénarios 6

2 Etat de l'art

1 Organisation du producteur 9
2 Explication de la formule du coût de stock 10

3 Modélisation et solution

1 Explication de la répartition des dates dues - Application 1 16
2 Explication de la répartition des dates dues - Application 2 19
3 Codage d'une solution 22
4 Exemple d'une solution codée 22
5 SWAP 27
6 EBSR 28
7 EFSR 28

4 Mise en oeuvre

1 Diagramme de classe du générateur 32
2 Diagramme de classe de l'algorithme de résolution 34

5 Campagne de tests et résultats d'application des tests

1 Fichier .bat 39
2 Test de performance - Mémoire 40
3 Test de performance - CPU 40
4 Solution finale codée 41
5 Ordonnancement de la solution 42
6 Distribution de la solution 42

6 Reproductibilité

1 Runnable JAR File 47
2 Choix du projet à exporter 47
3 Exemple de fichier de configuration 48
4 Exemple de fichier de résultats - 1 48
5 Exemple de fichier de résultats - 2 49
6 Ajout du .jar 50
7 Ajout de la dll 50
8 Exemple de fichier de résultats de la solution 52
9 Exemple du deuxième fichier de résultats de la solution 53
10 Exemple du deuxième fichier de résultats de la solution - 2 53

Gestion de projet

11 Planning du semestre 1 54
12 Planning prévisionnel du semestre 2 55
13 Planning réel du semestre 2 55

Liste des tableaux

2 Etat de l'art

1 Ensemble des variables présentes dans le modèle 8

2 Suite de l'ensemble des variables présentes dans le modèle 9

3 Modélisation et solution

1 Paramètres nécessaires dans l'application 1 15

2 Paramètres nécessaires dans l'application 1 16

3 Paramètres nécessaires dans l'application 2 17

4 Paramètres nécessaires dans l'application 2 18

5 Campagne de tests et résultats d'application des tests

1 Tableau pj 41

2 Tableau tj 41

3 Analyse de la meilleure méthode 43

6 Reproductibilité

1 Eléments du fichier de configuration 45

2 Eléments du fichier de résultats 46

3 Eléments du fichier de résultats 51

Remerciements

Je tenais à remercier Jean-Charles Billaut pour avoir proposé ce sujet et pour son entière disponibilité afin de m'aider face aux divers problèmes et interrogations que j'ai pu rencontrer tout au long du projet.

Introduction

Le projet que je vais vous présenter a été réalisé dans le cadre du projet de Recherche et Développement. Ce sujet a déjà fait l'objet d'un projet ainsi que de la publication d'un article.
L'encadrant de ce projet est Jean Charles Billaut, qui avait aussi participé aux précédents travaux.

Il s'agit de réaliser un générateur d'instances pour les entrer dans un algorithme afin de résoudre un problème d'heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur.

Le projet de recherche et développement se divise en deux parties, une première concerne la "recherche" et l'autre "le développement".
Dans la première partie, je vais donc présenter le cahier des charges du projet, ainsi que le récapitulatif de mes recherches et de mes analyses au sujet du modèle mathématique associé au sujet, du générateur de données et de l'algorithme de résolution.
Dans la seconde partie, j'aborderai la réalisation de l'implémentation de l'algorithme de résolution ainsi que son exécution sur les données créées par le générateur précédemment développé.

Première partie

Partie 1 : Recherche

1 Cahier des charges

1 Contexte

En juillet 2015, ce sujet a fait l'objet d'un précédent projet [4] ainsi que de la publication d'un article [7]. L'objectif principal du précédent travail a été de modéliser le problème. Un modèle mathématique a donc été proposé et publié dans un article.

Il s'agit d'un projet de recherche, proposé et encadré par Jean-Charles Billaut. Le projet concerne un système composé d'un producteur et d'un distributeur. Le but étant de planifier les différentes tâches (production et distribution) afin de minimiser les coûts. Il y a donc différentes contraintes à prendre en compte (type du produit, type du transport,...).

Nous sommes dans le cas d'un producteur et d'un distributeur. Pour le producteur il organise sa production sur m machines. Le producteur a des tâches à réaliser pour sa production. Celles-ci se déroulent sur 1 ou plusieurs machines, en flowshop ou en parallèle. Il faut donc organiser ces jobs pour minimiser certains coûts.

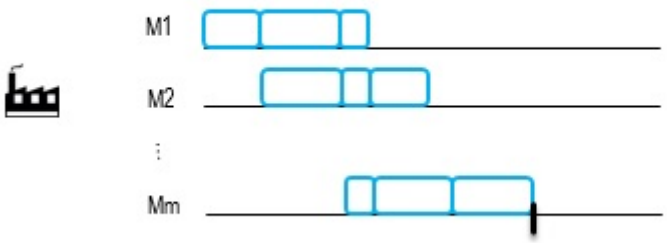


Figure 1 – *Organisation du producteur*

Pour ce qui est du distributeur, il effectue des tournées afin de livrer les produits entre les différents sites. Le distributeur possède un certain nombre de véhicules et ils peuvent partir d'un dépôt. Il y a donc des départs du dépôt qui doivent passer par l'usine de production pour ensuite livrer les différents sites. Il est donc nécessaire d'organiser la livraison entre les différents clients pour minimiser les coûts également.

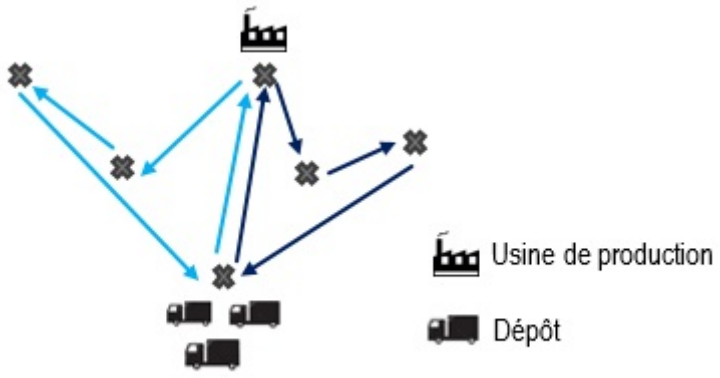


Figure 2 – *Organisation du distributeur*

Je dois donc prendre la suite de ce projet et travailler sur une méthode de résolution du problème respectant le modèle mathématique créé lors du dernier projet.

2 Objectifs

Ce projet se compose de deux objectifs majeurs.

Dans un premier temps, il est nécessaire de développer un générateur de données. En effet, pour le moment les données utilisées pour réaliser les tests sont des valeurs aléatoires, sans aucun lien avec les données que l'on trouve dans un système de production et de distribution.
Pour réaliser cela, le générateur prendra en compte différents paramètres que je vais détailler dans une des sections suivantes.

Le domaine de production et de distribution est très large. Il existe un nombre très important de produits différents qui sont fabriqués. Pour la distribution, c'est la même chose : les types de transport, les périmètres de livraison sont très variés en fonction du produit ou de l'entreprise. Pour cela, il est nécessaire de limiter notre génération à certains cas.

On a donc décidé de considérer deux types d'applications :

— Application 1 : la production de produits pas chers (par exemple du papier, du lait,...).

— Application 2 : la production de produits chers (par exemple les produits pour la chimiothérapie).

Le deuxième objectif du projet est de réaliser un programme pour trouver une solution en fonction des données entrées (celles produites par le générateur créé juste avant) en paramètre. Pour accomplir cette partie, je vais donc devoir trouver un algorithme de résolution et l'implémenter par la suite. Cet algorithme va devoir trouver une solution proche de celle proposée par le modèle mathématique. Cette heuristique permettra aussi de comparer les résultats obtenus par le modèle mathématique et ceux obtenus par cet algorithme.

3 Travail à réaliser

Dans ce projet, il y a trois tâches principales à réaliser.

3.1 Modèle mathématique

Dans un premier temps, il est nécessaire d'étudier le travail réalisé précédemment. Il faut que je m'imprègne de l'idée globale, que je définisse les différents coûts, les liens entre les formules ainsi que les différents scénarios. Il est nécessaire que je réalise une synthèse (2, 3 pages) de ce que j'ai compris ainsi qu'une présentation à mon encadrant fixée le mercredi 30 septembre 2015.

Le rapport et l'article publié sont composés d'un nombre important d'éléments pour comprendre le sujet. Le modèle mathématique nécessite donc une étude approfondie afin de comprendre toutes les variables prises en compte dans le système.

3.2 Générateur de données

Dans un second temps, le travail réalisé concerne le générateur de données. Cette tâche se décompose en plusieurs parties.

Avec l'étude des travaux précédents dans la première partie ainsi qu'avec des recherches supplémentaires dans l'article et dans le rapport, je dois énumérer les éléments passés en entrée dans le modèle mathématique.

Puis il faut rechercher les formules et les variables nécessaires pour le calcul de ces éléments.Il est donc nécessaire de faire des recherches sur le fonctionnement de la production et du transport pour savoir comment on peut générer des données proches du réel.

Dans un troisième temps, je dois faire des recherches sur internet, ainsi que dans des articles et rapports précédemment publiés, pour recueillir les informations me permettant de connaître la valeur des constantes présentes dans les formules ou alors directement la valeur des paramètres passés en entrée du système.

Enfin, je développerai le générateur de données, admettant dans l'application un fichier de configuration et qui engendrera un fichier de sortie contenant tous les éléments nécessaires du modèle mathématique.

3.3 Algorithme de résolution

Dans un dernier temps, le travail sera sur un algorithme de résolution. Pour cette partie je vais travailler avec mon encadrant Jean Charles Billaut afin de trouver un algorithme qui permet de résoudre le problème.

Dans le rapport du précédent projet, trois scénarios sont expliqués :

- Le producteur (M) domine c'est à dire que c'est lui qui prend les décisions.
- Le distributeur (3PL) domine.
- Le distributeur et le producteur coopèrent, il prennent donc les décisions ensemble.

Soit on va réaliser les 3 scénarios détaillés, ce qui permettra de faire remarquer les avantages de la coopération, soit un des scénarios.

Pour trouver cet algorithme, on pourra utiliser des méthodes heuristiques simples (méthode de voisinage, méthode tabou). Il sera éventuellement nécessaire de faire des recherches sur des méthodes déjà existantes.

Ensuite il faudra l'implémenter et le tester avec les données générées dans la partie précédente.

Il pourrait être intéressant de réaliser plusieurs générations de solution pour le cas de la coopération pour voir la fréquence à laquelle le coût est très intéressant (comme la croix en bas à gauche) ou bien peu ou pas intéressant.

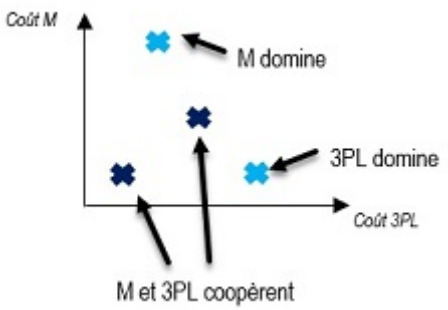


Figure 3 – *Analyse des coûts pour les 3 scénarios*

4 Contraintes

Dans ce projet, il n'y a pas de contrainte technologique particulière. Mon encadrant, Jean Charles Billaut m'a laissé la liberté de choisir le langage, le format des fichiers d'entrée et de sortie ainsi que de l'environnement de développement qui vont être utilisés pendant le projet.

Il faudra juste que je réalise une documentation expliquant la structure de mon fichier d'entrée et de sortie.

Il pourra juste être obligatoire d'utiliser un solver existant pour le langage choisi dans l'implémentation de l'algorithme de résolution qui utilisera le modèle mathématique décrit dans le précédent projet.

2 Etat de l'art

1 Rappel du modèle mathématique

1.1 Formules

Dans la première partie de mon projet, j'ai étudié les travaux réalisés précédemment. Il s'agit d'un rapport faisant suite à un projet et d'un article publié cette année.

Le modèle mathématique correspondant à mon sujet était présenté dans le rapport de Sonja Rohmer encadré par Jean-Charles Billaut [4]. Puis cette source d'information est complété par un article publié par les deux mêmes personnes [7].

Dans cette section, je vais vous expliquer le modèle mathématique qui a été présenté.

Pour commencer, figure ci-dessous un tableau récapitulant l'ensemble des variables utilisées dans le modèle mathématique. Cela simplifie la compréhension des formules qui suivent.

Table 1 – *Ensemble des variables présentes dans le modèle*

Nom de la variable	Signification
VC	Coût des véhicules
c^{V}	Coût d'un véhicule
V	Nombre de véhicules
PC_{M} ou PC_{3PL}	Pénalité du producteur ou du distributeur
π_j^{M} ou π_j^{3PL}	Pondération du retard du distributeur ou du producteur du job j
T_j^{M} ou T_j^{3PL}	Retard du distributeur ou du producteur du job j
PPC_{M} ou PPC_{3PL}	Pseudo coût de pénalité du distributeur ou du producteur
PT_j	Pseudo retard pour le job j

Table 2 – *Suite de l'ensemble des variables présentes dans le modèle*

d_j	Date de livraison due pour le job j
T	Délai de livraison
IC	Coût du stock
$C_{m,j}$	Date de fin de production du job j sur la machine m
C_j	Date de fin de production du job j
q_j	Quantité produite du job j
h_j^{WIP}	Coût de stockage du job j pour le produit en cours de production
h_j^{FIN}	Coût de stockage du job j pour le produit fini
F_k	Date de passage du véhicule k
RC	Coût du transport
$t_{i,j}$	Matrice de distance entre les points i et j

Nous sommes dans le cas d'un producteur et d'un distributeur.

Pour le producteur, il organise ses tâches sur m machines. Le schéma suivant permet de visualiser l'ordonnancement et donc de comprendre les différentes variables utilisées.

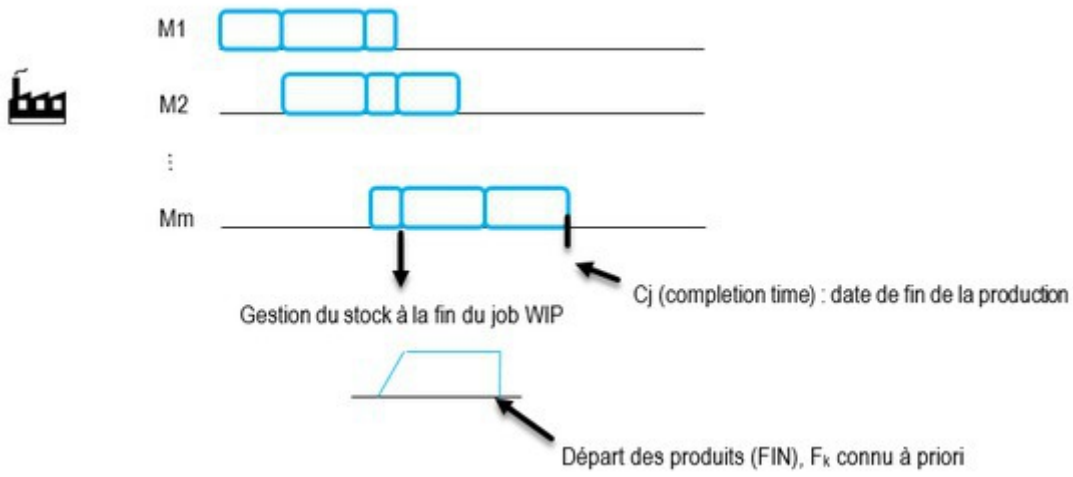


Figure 1 – *Organisation du producteur*

Le producteur cherche à minimiser les coûts suivants :

— Coût des véhicules : VC = coût d'un véhicule * le nombre de véhicule = $c^V * V$

— Coût de pénalité : PC_M = somme pondérée du retard = $\sum_{j=1}^{n} \pi_j^M x T_j^M$, où T_j^M = max (0, retard de livraison).

Comme le producteur ne connaît pas les dates réelles de livraison (et donc le retard) au moment de l'ordonnancement de sa production, on utilise un « pseudo » coût de pénalité PPC_M qui utilise un « pseudo » retard PT_j = max (0, (C_j + T) – d_j). Pour résoudre ce problème, il y a des méthodes optimistes ou pessimistes, ici on utilise C_j + T.

— Coût du stock :

$$IC = \sum_{j=1}^{n} (C_{m,j} - C_{1,j}) q_j h_j^{WIP} + \sum_{j=1}^{n} (F_k(j) - C_{m,j}) q_j h_j^{FIN}$$

La dernière formule qui concerne le coût du stock est assez complexe ; elle nécessite donc d'être approfondie.

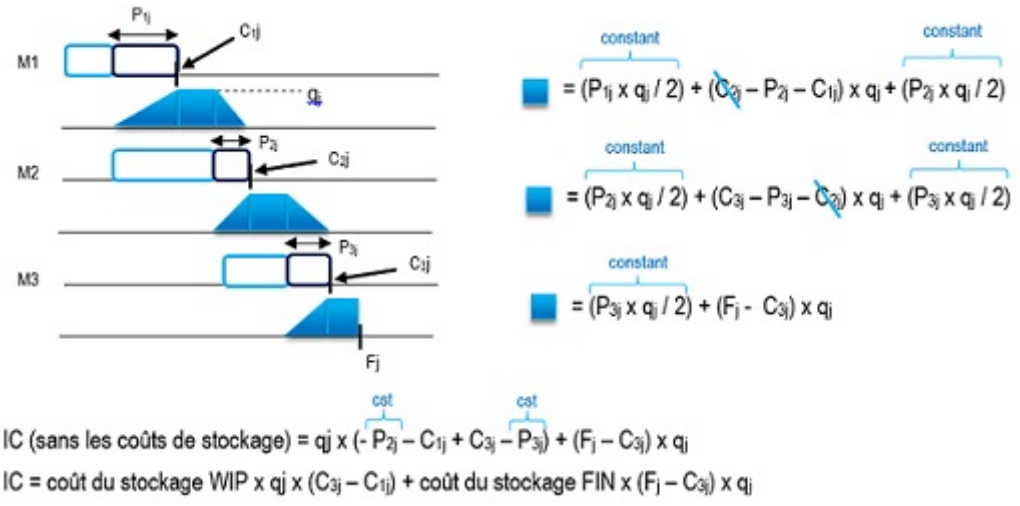


Figure 2 – *Explication de la formule du coût de stock*

Le coût total pour le producteur est donc : PTC_M : $IC + PPC_M + VC - PC_{3PL}$ (payé par le distributeur)

Quant au distributeur, il cherche à minimiser :

— Coût de transport : RC = somme des coûts de chaque véhicule : c'est lié à la distance parcourue par les véhicules.

— Coût de pénalité : S'il livre en retard, il paye une pénalité au producteur, tous les jobs doivent être livrés dans le délai de livraison [0, T] car il ne connaît pas d_j. PC_{3PL} = somme pondérée des retards = $\sum_{j=1}^{n} \pi_j^M x T_j^{3PL}$, où T_j^{3PL} = max (0,D_j - (t_0 + T)).

Le coût total du distributeur est donné par TC_{3PL} = RC + PC_{3PL} – VC (payé par la production).

Un job j correspond à la production pour un site customer j, c'est-à-dire qu'il y a autant de job que de site à livrer. Dans la matrice $t_{i,j}$, on a n + 2 colonnes, les n premières correspondent à la distance entre les n jobs, n + 1 représente la distance avec le site du fabricant, quant à n + 2 il s'agit de la distance avec le dépôt du distributeur.

1.2 Solution

On considère que le producteur et le distributeur sont deux agents indépendants, qui réalisent leur optimisation chacun de leur côté.

Dans un premier temps, le producteur réalise son ordonnancement, en sachant qu'il a convenu avec le distributeur de la valeur des variables V (le nombre de véhicules), T (le délai maximum de livraison pour le distributeur), Fk (date à laquelle les véhicules passent).

Ensuite, c'est au distributeur de résoudre son problème en minimisant son coût. On peut calculer le coût du distributeur.

Après cela, Dj est alors connue, il est donc possible de calculer le coût réel du producteur.
Il s'agit de l'algorithme présent à la page 12 du rapport [4].

1.3 Scénarios

Dans le rapport plusieurs scénarios sont envisagés. Cela influe sur la définition du modèle mathématique. Comme indiqué dans le travail à réaliser (3 (Chapitre 1))

Le distributeur et le producteur travaillent pour la même entreprise :

Dans ce cas, certaines variables sont amenées à disparaître. En effet, comme il s'agit d'une même entreprise si l'on gardait le coût du véhicule et les pénalités, cela reviendrait à retirer et remettre le même montant dans l'entreprise.
De plus, en étant dans la même entreprise, la communication est facilitée et la date due est alors connue (disparition du T).

L'objectif est de minimiser : $TC = \alpha TC_M + (1-\alpha)TC_{3PL} + VC$

$TC = \alpha(IC^{WIP} + IC^{FIN} + PC_M) + (1-\alpha)RC_{3PL} + VC$

Il y a en plus la présence d'une nouvelle donnée : α qui correspond au coefficient de la combinaison linéaire du coût total du producteur et donc 1- α pour le distributeur.

En dehors des coûts totaux du producteur et du distributeur pondérés par α, on doit ajouter à l'expression le coût du véhicule. En effet, il s'agit d'une variable importante à prendre en compte lors de la recherche de solution.

Le producteur domine sur le distributeur :

C'est dans ce scénario que le producteur impose au distributeur un ensemble de données. C'est lui qui décide comment cela va se dérouler. Il choisit le nombre de véhicules et le temps maximal de livraison T.

On cherche à minimiser : $PTC_M = IC + PPC_M + V \times C^V - PC_{3PL}$

Le distributeur domine sur le producteur :

C'est au distributeur, dans ce cas, d'imposer au producteur le nombre de véhicules et sa date de passage. En fonction de ça, c'est au producteur de s'organiser.

On cherche à minimiser : $RC + PC_{3PL}$ - VC

2 Générateur de données

2.1 Cas d'application

Il existe actuellement une multitude de produits réalisés dans les usines de fabrication. Cela peut aller de la production du papier à celle de produits chimiques. Les prix du stockage, de la production et de la distribution sont donc évidemment très variables.

Pour générer des données proches de la réalité, il est donc nécessaire de nous limiter à certains cas d'application. Les applications choisies sont les suivantes.

2.1.1 Application 1

Le but de cette première application est d'étudier le cas où une entreprise fabrique des produits qui ont peu de valeur, c'est à dire qui ont un coût unitaire bas.

Cela implique que le stockage des éléments ne coûte pas cher. En effet, le stockage prenant en compte la valeur du produit, celle-ci étant très faible, rend donc le coût de stockage faible aussi.

Cependant le transport quant à lui coûte très cher. Le plus souvent, les livraisons sont réalisés sur un périmètre important, le type de transport peut nécessiter des spécifications particulières (taille, réfrigérant,...) et tout cela a un coût.

Les produits concernés par cette application peuvent être le papier, les mouchoirs, le lait, les yaourts,...

Pour la suite nous avons décidé de nous concentrer sur le cas de la production de lait.

2.1.2 Application 2

Dans le cas de la deuxième application, l'objectif est de générer des données lorsque l'entreprise fabrique des produits qui ont une grande valeur.

Comme expliqué précédemment, cela entraîne un coût de stockage très important au vu de la valeur des produits.

Mais le transport, lui, devient faible. En effet, la plupart du temps, il s'agit de livraison restreinte à un périmètre plus petit, et le transport en lui-même n'a souvent pas besoin de fonctionnalité particulière.

Les produits concernés par cette application sont les produits chimiques, la nano technologie, des semi-conducteurs ou bien encore la chimiothérapie,...

Pour la suite nous avons décidé de nous concentrer sur le cas de la production de produits de chimiothérapie.

Cependant pour ce cas, nous avons décidé de le diviser en deux sous-applications. Cette division a été fixée pour répondre à une particularité de la distribution de ce type de produit. En effet celle-ci peut varier quand il s'agit du domaine de la chimiothérapie.

Application 2.1

Dans cette sous-application, il s'agit du cas où la distribution se réalise à pied.

Une personne dédiée parcourt le quartier pour effectuer les livraisons. Les coûts de transport sont donc totalement différents du cas d'application n°1.

Le coût de transport n'est plus le coût du véhicule mais il s'agit plutôt du salaire de la personne chargée de la distribution.

Application 2.2

Quant à la deuxième sous-application, c'est lorsque la livraison des produits de chimiothérapie nécessite un transport plus éloigné, par exemple dans le périmètre d'une ville.

Il s'agit d'un cas différent de la première sous-application car on retrouve un "vrai" coût de transport.

Cependant, il reste différent du cas de la première application. En effet, le véhicule utilisé dans cette situation est plus petit (une petite camionnette au maximum) et le périmètre est différent.

Les coûts de transport sont donc totalement différents.

2.2 Recherche des paramètres

Les données nécessaires en entrée de l'algorithme de résolution seront produites par le générateur. Cependant, afin que celles-ci soit cohérentes aux cas d'applications vus dans la partie précédente, il est donc nécessaire de réaliser quelques recherches.

L'objectif est de trouver comment fonctionne les systèmes de production dans les différents cas et quelles sont les valeurs des constantes utiles au générateur de données.

Dans un premier temps, une partie des réponses a été trouvée dans le rapport [4]. En effet, une partie aborde les débuts de recherches effectuées sur la valeur possible des constantes.

En moyenne dans le cadre de l'application 1, la distribution de lait s'effectue dans un périmètre de 150 km. Afin de simplifier les calculs, la vitesse de déplacement est fixé à 60 km/h.

Afin d'organiser la distribution, il est nécessaire de posséder les coordonnées des différents sites. Pour cela, le rapport nous propose de générer des coordonnées (x, y) du fabricant et des n clients de la manière suivante : x et y sont compris dans l'intervalle [1 ; $perimetre/\sqrt{2}$] et la distance se calcule avec la distance euclidienne.

De plus, il a été décidé dans le rapport [4] d'utiliser des quantités moyennes de produits qj ; elles peuvent être situées dans l'intervalle [10 ; 200] kg pour l'application 1.

Quant au type de transport, il se réalisera dans des véhicules moyens (moins de 3 tonnes, dans l'intervalle [1000 ; 3000]) également pour l'application 1.

Maintenant il est proposé d'aborder la définition des différents coûts.

Pour le coût de stockage pour un produit, dans le rapport, il est composé du coût du produit et du coût de détérioration et de l'obsolescence et formulé de la façon suivante. Pour le coût FIN :

$$h_j^{\mathrm{FIN}} = \delta_j \times t$$

δ_j est le prix unitaire du produit j, et t correspond à un taux pour l'autre composante du coût de stockage, en moyenne t est égal à 20 %. Et pour le coût WIP comme étant un pourcentage de celui de FIN :

$$h_j^{\mathrm{WIP}} = \beta \times h_j^{\mathrm{FIN}}$$

La plupart du temps β est fixé à 10%.

Quant à la thèse de doctorat de l'Ecole Nationale des Ponts et Chaussées [2], dans le cadre du modèle Sandoma, pour le coût de stockage, l'auteur propose la formule suivante (coût intrinsèque des stocks) :

$$Cstock = p \times ix\mathrm{T}/2$$

Pour p, il correspond à la valeur du produit, i correspond à un taux annuel ou mensuel définissant le coût du capital immobilisé (le plus souvent estimé à 20, 30%) et T/2 à la quantité stockée en moyenne.

Si l'on souhaite obtenir le coût de stockage unitaire, il suffit de remplacer pi par pi/Q où Q est la quantité annuelle ou mensuelle reçue.

D'après la thèse [2], le coût intrinsèque est lié à la quantité stockée, alors que le coût unitaire est lié au taux de rotation (c'est-à-dire que pour deux entrepôts les stocks mensuels et le coût total de stock sont identiques alors que la quantité circulée est différente).

Si nous faisons abstraction de la quantité dans la formule de la deuxième ressource, on retrouve le coût unitaire comme étant un pourcentage de la valeur du produit. De plus, ce pourcentage est presque similaire dans les deux documents (20%).

Pour le coût de pénalité du producteur et le coût unitaire d'un produit, seul le rapport [4] les aborde.

Le coût de pénalité correspond à un ratio du prix unitaire du produit par unité de temps de retard.

$$\pi_j^{\mathrm{M}} = \epsilon \times \delta_j \times q_j$$

Où ϵ appartient à l'intervalle suivant 3% ; 5% ; 7%.

Quant au coût unitaire d'un produit il est décrit comme étant dépendant de son type :

— Produits chers (application 2) : $\delta_j \in 50; 200$ €.

— Produits peu coûteux (application 1) : $\delta_j \in 0,05; 0,20$ €.

Pour le coût du transport, plusieurs documents présentent des propositions sur le sujet.

Dans la thèse de doctorat de l'Ecole Nationale des Ponts et Chaussées [2], le prix du transport ne dépend pas seulement de la distance mais aussi de la taille de l'envoi, du type de service (délai de transport, matériel spécifique (par exemple pour les véhicules citernes),...).

De plus, cette thèse de doctorat [2] (modèle EOQ), le rapport du précédent projet [4], le fichier excel [WWW1] ainsi que la thèse de Master - qui concerne l'étude du cas d'une entreprise laitière Algérienne (produit périssable) - [6] décomposent le coût d'un transport comme ayant une partie variable (carburant, pneu, péage, entretien...) et une partie fixe (prix du véhicule - location, amortissement des véhicules - et main d'œuvre).

D'autre part, le rapport de Sonja Rohmer [4] et la thèse de V. Gacogne définissent ce coût comme étant fonction de la quantité transportée et de la distance parcourue.

Dans le rapport [4] :

$$\mathrm{C}^{\mathrm{V}}(tt, q) = (c1 \times tt + c2) \times c3 + q^{c4}$$

Dans cette formule c1, c2, c3, c4 sont des constantes avec respectivement les valeurs suivantes 0,06, 10, 3, -0.05. La variable tt correspond à la distance parcourue et q la quantité transportée.

Et dans la thèse [2], dans le cadre du modèle Sandoma, il est indiqué que lorsque la taille de l'envoi augmente et que la fréquence d'envoi diminue, le coût de transport diminue, le stock moyen augmente

et le coût de stockage augmente. Dans la plupart des modèles, on ne prend pas en compte la taille de l'envoi mais seulement la distance, ou alors lorsque ce paramètre est pris en compte, il est fixé a priori.

Le tarif du transport est donc défini de la manière suivante :

$$Px(D,T) = (0,063D + 11,029) \times (3,7037T)^{-0,4431}$$

D : distance, T : taille de l'envoi, les deux premières constantes peuvent évoluer. Les constantes sont fixées à partir du guide des coûts publié par le CNR (Comité National Routier).

On remarque donc que les expressions sont très similaires, seule la précision des constantes varie.

Enfin, le fichier excel [WWW1] permet d'obtenir le coût d'un véhicule sous forme d'une constante. En effet, il n'utilise pas la distance parcourue et la quantité transportée comme indiqué précédemment mais utilise la distance parcourue en moyenne dans l'année ainsi que la capacité du transport. De plus, les coûts de type prix du carburant sont issus du document publié par le CNR (Comité National Routier) [5]

On obtient cette valeur après la saisie de différentes valeurs comme par exemple les termes suivants :

- Terme kilométrique variable (carburant, pneu, péage, entretien,...)
- Terme horaire conducteur : coûts liés au conducteur (rémunération, frais de déplacements,...)
- Terme journalier fixe de véhicule (coût de détention, assurance, coût de structure,...)

En plus de ces informations, le rapport du projet précédent [4] précise le coût pour un véhicule du 3PL provider, comme étant un ratio de ce coût du transport.

$$C_{j1,j2}(tt,q) = \lambda \times C^{V}(tt,q)$$

Où $\lambda = 20\%$.

Après ces recherches, il rese un élément important qui nous manquait : le temps de production pour les produits de l'application 1 et 2.

Pour celui du lait, les recherches ont été longues et difficiles. En effet, l'information était complexe à trouver. La plupart du temps, j'obtenais le cycle de production du lait avec les différentes étapes, mais jamais d'indication sur le temps nécessaire à chaque étape.

Après avoir lu plusieurs documents, j'en ai sélectionné un qui m'a paru le plus adapté et le plus cohérent avec mes recherches précédentes. Il s'agit d'un livre qui parle de la transformation du lait de ferme vers les produits finis tels que les yaourts, le lait après certains traitements,... Il s'agit d'un livre de GRET nommé Transformer les produits laitiers frais à la ferme [3].

Dans ce document, il annonce qu'en moyenne, pour produire 125 litres de lait, le temps est de 6h15 ; cela revient donc à un temps de 3 minutes pour un litre.
De plus, ce document et d'autres sources internet m'indiquent qu'en moyenne, la production de lait journalier est de 5 000 l par jour.

Nous avons donc :

$$5000l/jour \times 3min/l = 15000min$$

Ce qui donne 250 h, cela ne rentre évidemment pas dans une journée de 24 h. Pour que cela soit correct, ça implique qu'il y ait plusieurs lignes de production.

Dans le cadre de mes recherches, j'ai aussi appris que la production de lait se déroulait sur une journée entière de 24 h de type 3 x 8.

Pour la deuxième application, il a été nécessaire d'effectuer des recherches dans un rapport de PFE écrit par Thibault DREVON : Prise en compte des reliquats dans la fabrication de chimiothérapies [1]. Dans ce document, il y a une explication très précise sur le déroulement de la production des médicaments chimiothérapiques. Il y a donc l'information que je recherche au sujet du temps de production.

La production se déroule donc sur 2 machines, un premier traitement de 15 à 30 minutes est réalisé et ensuite un passage sur une deuxième machine pendant 5 minutes est nécessaire.

Dans ce document, il est question aussi des dates dues pour les produits chimiothérapiques. La journée se décompose en deux parties, les fabrications du premier bloc sont dues le 1er jour, quant à celles du deuxième bloc, elles sont dues le 2ème jour.

3 Modélisation et solution

1 Générateur

Maintenant que nous avons les principales informations dont nous avions besoin, il est désormais nécessaire de modéliser le générateur de données.

1.1 Entrée

L'algorithme qui concerne le générateur de données, possède des paramètres en entrée et génère des données en sortie. Comme les différentes informations varient en fonction de l'application, cette partie sera donc divisée en deux.

1.1.1 Application 1

Afin de faciliter la compréhension, les différents paramètres ainsi que leur valeur seront présentés dans le tableau suivant :

Table 1 – *Paramètres nécessaires dans l'application 1*

Nom du paramètre	Valeur	Détails
Nombre de machine	1	Il y a 10 lignes de production en parallèle mais 1 seule par ligne
Temps de production total	15000	$3min/l \times 5000l/j$
ϵ^{3PL}	20	20%, constante du calcul de la pénalité 3PL
Taux d'obsolescence t	20	20%, constante du calcul du coût de stockage FIN
β	10	10%, constante du calcul du coût de stockage WIP
Périmètre de livraison	150	Périmètre en km

Table 2 – *Paramètres nécessaires dans l'application 1*

Nom du paramètre	Valeur	Détails
Borne inférieure coordonnées	1	Pour le calcul des coordonnées des sites
Coût d'un véhicule Cv	570	Calculé à partir du fichier [WWW1] avec 140 000 km/an, 2000kg de capacité, 1300€pour le livreur
λ	20	20%, constante pour le calcul du coût d'un trajet
Constante 1	0,06	Constante pour le véhicule dans le coût d'un trajet
Constante 2	10	Constante pour le véhicule dans le coût d'un trajet
Constante 3	3	Constante pour le véhicule dans le coût d'un trajet
Constante 4	-0,05	Constante pour le véhicule dans le coût d'un trajet
Capacité	[1000 et 3000]	Capacité du véhicule en kg
ϵ^M	3 ou 5 ou 7	En %, constante du calcul pénalité M
Quantité qj	[10 et 500]	Quantité produite en kg
Prix du produit δ_j	[0,05 et 0,20]	Prix en €

Dans cette partie, un dernier paramètre entre en ligne de compte : la date due. En effet, lorsque l'on produit, il y a une date à laquelle la fabrication doit être livrée.

Pour calculer la valeur de ces dates, on a décidé que la simulation de cette application se déroule sur 5 jours. De plus, il est nécessaire d'ajouter une variable r_j qui représente la date de début au plus tôt. Ensuite, la répartition se déroule de la manière suivante :

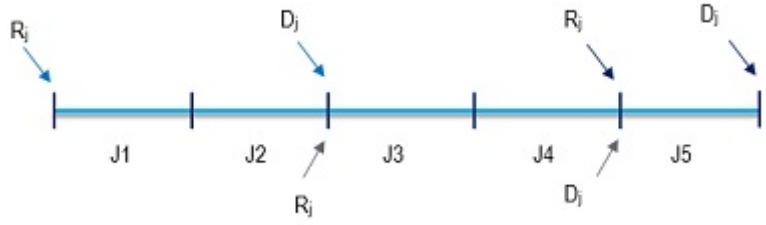


Figure 1 – *Explication de la répartition des dates dues - Application 1*

Nous avons donc :

— Pour les jobs de 0 à $\frac{2\times nbjobs}{5} - 1$: $r_j = 0$ et $d_j = 48h$

— Pour les jobs de $\frac{2\times nbjobs}{5}$ à $\frac{2\times nbjobs}{5} + \frac{2\times nbjobs}{5} - 1$: $r_j = 48h$ et $d_j = 96h$

— Pour les jobs de $\frac{4\times nbjobs}{5}$ à nbjobs : $r_j = 96h$ et $d_j = 120h$

Dans les entrées de cet algorithme, on distingue deux types de paramètres : ceux entrés par l'intermédiaire d'un fichier et ceux entrés directement lors de l'exécution du programme.
Ces derniers représentent les paramètres qui sont amenés à être modifiés à chaque lancement du générateur de données. Contrairement à ceux du fichier qui peuvent être changés mais cela restera plus rare car ce sont des constantes ou des valeurs lors des recherches.

Dans le cadre de l'application 1, on peut considérer ces trois paramètres avec les valeurs suivantes :

— Nombre de jobs : 50

— Délai de livraison T : 24 h

— Nombre de véhicules : 1

1.1.2 Application 2

Afin de faciliter la compréhension, les différents paramètres ainsi que leur valeur seront présentés dans le tableau suivant :

Table 3 – *Paramètres nécessaires dans l'application 2*

Nom du paramètre	Valeur	Détails
Nombre de machine	1	Pour simplifier la modélisation on a décidé de regrouper les deux machines en une machine. Il y a trois lignes de production mais une machine par ligne
ϵ^{3PL}	20	20%, constante pour le calcul de la pénalité 3PL
Taux d'obsolescence t	20	20%, constante pour le calcul du coût de stockage FIN
β	10	10%, constante pour le calcul du coût de stockage WIP
Périmètre de livraison	0,5 / 15	0,5km pour la livraison à pied et 15 km en voiture
Borne inférieure coordonnées	0,1 / 1	0,1 pour la livraison à pied et 1 en voiture Pour le calcul des coordonnées des sites

Table 4 – *Paramètres nécessaires dans l'application 2*

Nom du paramètre	Valeur	Détails
Coût d'un véhicule Cv	50 / 200	50€pour la livraison à pied et 200€en voiture Calculé à partir du fichier [WWW1] avec 10 000 km/an, 15kg de capacité, 1300€pour le livreur
λ	20	20%, constante pour le calcul du coût d'un trajet
Constante 1	0,06	Constante pour le véhicule dans le coût d'un trajet
Constante 2	10	Constante pour le véhicule dans le coût d'un trajet
Constante 3	3	Constante pour le véhicule dans le coût d'un trajet
Constante 4	-0,05	Constante pour le véhicule dans le coût d'un trajet
Capacité	[1 et 3] / [10 et 30]	Capacité du livreur en kg Capacité du véhicule en kg
ϵ^M	3 ou 5 ou 7	En %, constante du calcul pénalité M
Quantité qj	[0,1 et 0,5]	Quantité produite en kg
Prix du produit δ_j	[500 et 2500]	Prix en €

Comme il a été vu précédemment, dans cette partie, un dernier paramètre entre en ligne de compte : la date due.

Pour calculer la valeur de ces dates, on a décidé que la simulation de cette application se déroule sur 1 journée. De plus, il est nécessaire d'ajouter une variable r_j qui représente la date de début au plus tôt. Ensuite, la répartition se déroule de la manière suivante :

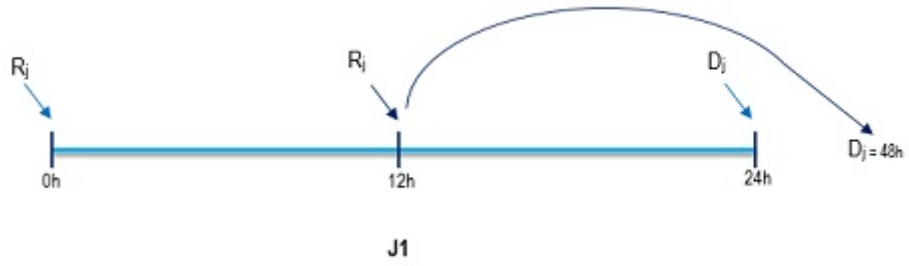


Figure 2 – *Explication de la répartition des dates dues - Application 2*

Nous avons donc :

— Pour la première moitié des jobs : $r_j = 0$ et $d_j = 24h$

— Pour l'autre moitié des jobs : $r_j = 12h$ et $d_j = 48h$

Dans le cadre de l'application 2, on peut considérer les trois paramètres entrés directement dans le programme, avec les valeurs suivantes :

— Nombre de jobs : 150

— Délai de livraison T : 24h

— Nombre de véhicules : 1

1.2 Sortie

En sortie, l'algorithme génère et transmet les données nécessaires dans le modèle mathématique, ce sont donc toutes les variables décrites dans le tableau de la partie 1.1 (Chapitre 2)

1.3 Fonctionnement général

Dans un premier temps, je vais présenter l'idée générale de fonctionnement, puis dans un second temps, j'écrirai l'algorithme correspondant.

Au début du lancement de l'algorithme, les données passées en paramètres (ceux du fichier et ceux passés directement) sont lues et stockées dans un tableau .

Ensuite lorsqu'il s'agit d'intervalles ou d'un choix entre plusieurs valeurs, l'algorithme tire aléatoirement le nombre.

Une fois cette étape terminée, le système réalise le calcul de toutes les formules avec les données entrées et génère les différentes matrices dont on a besoin.

Enfin, il génère la sortie, en écrivant dans un fichier les différentes données créées.

L'algorithme résultant de la modélisation est le suivant :

```
//récupération des paramètres entrés directement;
nbJob=50;
T=24*60;
nbVehicule=1;
//lecture du fichier de paramètres;
while !fichierTermine do
    if vartype = "simple valeur" then
        tabvaleur[lignefichier[clé]] = lignefichier[donnee];
    end
    if vartype = "intervalle" then
        tabvaleur[lignefichier[clé]] = tirageAleatoire(lignefichier[donnee]);
    end
    if vartype = "borne" then
        getvaleurBorneDateDue();
    end
end
calculCoordonnees();
calculPi();
calculHj();
calculPj();
calculMatDistance();
calculMatCoutDistance();
calculRjDj();
ecritureSortie();
```

Algorithme 1 : Générateur de données

Comme on peut le constater, l'algorithme précédent est composé de plusieurs fonctions. Il est donc nécessaire que j'explique brièvement maintenant ce que réalise chacune de ces fonctions.

Pour la fonction *calculCoordonnees()*, pour chaque site plus deux (en ajoutant le site de production et le dépôt), il s'agit de générer les coordonnées x et y. Pour faire cela, il suffit de tirer de manière aléatoire entre la borne inférieure et le $\frac{perimetre}{\sqrt{2}}$.

Les fonctions *calculPi()*, *calculHj()*, *calculPj()* calculent la valeur de ces différentes variables avec les formules précédemment définies.

calculMatDistance() est une fonction qui permet, à partir des coordonnées calculées plus tôt, de déterminer la distance entre les différents points. Pour cette distance, j'utilise la distance euclidienne. La fonction *calculMatCoutDistance()* est liée à cette dernière, car elle calcule pour chaque distance présente dans la matrice, le coût de celle-ci en utilisant la formule déjà définie.

Quant à la fonction *calculRjDj()*, elle affecte aux jobs leur date de début et de fin espérée, en fonction du découpage de tâches effectué dans le fichier de paramètres.
Enfin, la fonction *ecritureSortie()* génère le fichier de sortie du générateur de données, avec tous les éléments nécessaires à l'algorithme de résolution.

Une fois cette partie terminée, nous pouvons commencer à réfléchir à la modélisation de l'algorithme de résolution.

2 Algorithme de résolution

Cet algorithme va permettre de générer une solution à partir des données entrées, qui ont été précédemment créées grâce au générateur.

2.1 Choix de la méthode utilisée

Il est nécessaire de faire un choix entre le fait d'utiliser une méthode approchée ou optimale.

2.1.1 Méthode optimale

La première idée qui nous vient à l'esprit est d'utiliser une méthode optimale. En effet, le fait d'utiliser ce type de méthode permet d'obtenir la solution optimale. C'est à dire que ce résultat est le meilleur que l'on puisse obtenir.

Cependant au vu des nombreuses contraintes présentes dans le problème, il nous est impossible d'utiliser ce type de méthode. Le temps nécessaire pour trouver une solution serait trop important.
Dans le cadre de ce projet, cela pose donc un problème. Nous devons alors nous orienter vers des méthodes approchées.

2.1.2 Méthode approchée

Cette méthode nous donne une solution non optimale. En effet, celle-ci s'approche de la solution optimale mais ne l'atteint pas. Il s'agit d'une solution réalisable (elle respecte les différentes contraintes) mais elle n'est pas forcément la meilleure.

Dans les méthodes approchées, plusieurs existent déjà. Dans le paragraphe suivant, voici quelques exemples de ces méthodes.

La première est la méthode de voisinage. Le principe général est de réaliser plusieurs transformations mineures à notre solution afin d'obtenir d'autres solutions et donc d'obtenir son voisinage.

Ensuite, en fonction de plusieurs critères définis, on choisit la meilleure solution générée.

Cependant, derrière cette méthode, il existe de multiples façons de coder cela : cela varie notamment en fonction du contexte. Il existe différentes opérations. Celles que l'on utilise le plus souvent sont les suivantes :

- Complétion
- Echange
- Insertion-décalage
- Inversion

Il y a aussi la méthode tabou : elle appartient au groupe de recherche de voisinage, tout en y intégrant la mémoire. C'est à dire qu'elle s'interdit de regénérer une solution déjà trouvée précédemment.

Il s'agit donc bien de trouver une meilleure solution dans le voisinage de celle de départ, tout en évitant de faire machine arrière en retombant sur une configuration déjà testée.

Dans ce projet, nous avons décidé d'utiliser les méthodes de voisinage. Cependant, ces méthodes peuvent poser le problème suivant : la complexité des différentes contraintes implique la nécessité d'insérer des temps morts dans la planification des différents jobs. De plus, le problème est très complexe, ce qui rend l'utilisation directe de ces méthodes de voisinage difficiles à appliquer.
Il sera donc nécessaire d'intégrer à ces méthodes un solver.

Cette idée nous permettrait d'entrer les contraintes présentes dans le modèle mathématique déjà modélisé ainsi que les données utiles, puis de générer la solution correspondant au problème.

2.2 Fonctionnement général

Tout d'abord, le système prendra en entrée, un fichier contenant toutes les données nécessaires pour résoudre le problème.

Ensuite, l'algorithme doit prendre en compte toutes les contraintes présentes dans le modèle mathématique du rapport précédent [4]. Il faut planifier les différents jobs sur le site de production et ceux qui sont concernés par la livraison des différents sites, tout en minimisant tous les coûts.

Enfin en sortie, l'algorithme retourne la planification effectuée avec les différents coûts générés.

Les étapes principales pour cette application seront donc les suivantes :

- Récupération des différentes données en entrée
- Codage de la solution
- Évaluation de cette solution
- Utilisation de méthodes approchées pour explorer les voisinages
- Choix de la meilleure solution
- Affichage de cette solution

2.2.1 Codage de la solution

Dans un premier temps, on réalise le codage d'une solution et pour cela on peut utiliser la représentation suivante :

Figure 3 – *Codage d'une solution*

Le premier élément est la séquence. Il s'agit de l'ensemble des jobs à planifier écrits dans l'ordre de passage sur les machines.

Ensuite, on indique le nombre de véhicules qui sera utilisé pour la planification.

Enfin, pour les n véhicules du système, on écrit les deux éléments suivants. Le premier est le nombre de jobs qui sera présent dans le véhicule en question, puis on indique les jobs concernés. On répète cette opération pour tous les véhicules. Sur cette partie, la répartition des jobs dans les véhicules suit l'ordre de la séquence.

Afin de mieux comprendre cette représentation, voici un exemple de codage d'une solution.

Figure 4 – *Exemple d'une solution codée*

Dans cet exemple, nous avons trois jobs. La séquence choisie est 3, 1 puis 2. Il y a deux véhicules prévus pour la distribution. Dans le premier véhicule, il y a un job (le 3) et dans le second véhicule, il y a deux

jobs (le 1 et le 2).

Une fois la représentation choisie, il est nécessaire de trouver comment définir la solution initiale.

Pour cela nous allons utiliser la méthode suivante.

Pour la partie qui concerne la séquence, nous allons utiliser le tri EDD (Earliest Due Date) c'est à dire que le tri se fera de la plus petite date due à la plus grande.

Ensuite, pour le choix du nombre des véhicules et la répartition des jobs dans ces derniers, il est nécessaire de réaliser une première exploration des possibilités.

On va donc prendre pour commencer n véhicules en répartissant un job dans chaque transport. Puis diviser le nombre de véhicules par deux et répartir deux jobs par véhicule et ainsi de suite jusqu'à ce que l'on n'ait plus que deux transports avec n/2 des jobs dans chacun.

Chaque solution créée sera évaluée pour choisir la bonne répartition des transports. Ensuite, il faudra explorer les différentes séquences possibles.

2.2.2 Évaluation de cette solution

L'indicateur qui va nous servir de comparaison est le coût de la solution. En effet, c'est l'objectif du projet : réaliser une planification de bonne qualité, minimisant les coûts.

C'est dans cette section que le solver va intervenir. Pour évaluer une solution, il faut ordonnancer au mieux la séquence en respectant toutes les contraintes et les données, puis calculer les différents coûts pour obtenir l'indicateur nécessaire pour son évaluation.

Le problème possède de nombreuses contraintes. L'ordonnancement est donc complexe et long à réaliser "à la main". De plus, les méthodes "classiques" pour coder cette évaluation auraient ordonnancé les différentes tâches le plus à gauche possible, ce que nous ne souhaitions pas. En effet, comme précisé précédemment, le but est de minimiser les coûts. On peut donc être amené à laisser des temps de non utilisation sur la machine afin de convenir à l'objectif. C'est pour cette raison que nous allons nous servir d'un solver pour cet élément, afin qu'il trouve l'ordonnancement idéal en fonction de nos contraintes et de nos données.

Pour cela, nous avons dû réaliser le modèle mathématique à transmettre au solver. Dans le cadre de la réflexion, deux modèles ont été réalisés.

Le premier utilise la position du job alors que le deuxième utilise la précédence des jobs.

Premier modèle :

Variables :

- $t_{i,k}$: matrice contenant la date de début du job en position k sur la machine i
- $C_{i,k}$: matrice contenant la date de fin du job en position k sur la machine i
- IC_k^{FIN} : coût de stockage FIN du job en position k
- IC^{FIN} : coût de stockage FIN total
- IC^{WIP} : coût de stockage WIP total
- D_k : date due du job en position k

Données :

— n : nombre de jobs
— V : nombre de véhicules
— m : nombre de machines
— h_j^{WIP} : coût du stockage WIP du job j
— h_j^{FIN} : coût du stockage FIN du job j
— q_j : quantité produite pour la tâche j
— $x_{j,k}$: égale à 1 si le job j est à la position k sinon 0
— $y_{j,v}$: égale à 1 si le job j est dans le véhicule v sinon 0
— p_j : durée de la tâche j
— F_v : date de départ du véhicule v
— T : délai de livraison
— π_{jm} : coût de pénalité du job j sur la machine m

Fonction objectif :

L'objectif étant de minimiser les coûts.

$$(\text{M}inimiser) \qquad \text{Z} = \text{IC}^{\text{FIN}} + \text{VC} + \text{IC}^{\text{WIP}} + \text{PPC} \tag{1}$$

Contraintes :

La date de début du job en position 1 sur la machine 1 doit être 0 :

$$t_{11} = 0 \tag{2}$$

L'ordre des jobs sur la machine 1 doit être respecté :

$$t_{1k} \geq t_{1k-1} + \sum_{j=1}^{n} p_j x_{jk} \qquad \forall k \in \{2,...,n\} \tag{3}$$

Les jobs doivent s'enchaîner correctement (dans l'ordre) sur les différentes machines :

$$t_{1k} \geq t_{i-1k} + \sum_{j=1}^{n} p_j x_{jk} \qquad \forall i \in \{2,...,m\}, \quad \forall k \in \{2,...,n\} \tag{4}$$

L'ordre des tâches sur la machine i doit être respecté :

$$t_{1k} \geq t_{ik-1} + \sum_{j=1}^{n} p_j x_{jk-1} \qquad \forall i \in \{2,...,m\}, \quad \forall k \in \{2,...,n\} \tag{5}$$

Le coût de stockage prend en compte la quantité stockée, le coût de stockage et le temps passé en production :

$$\text{IC}^{\text{WIP}} = \sum_{k=1}^{n} (t_{mk} - t_{1k}) \times \sum_{j=1}^{n} (h_j^{\text{WIP}} x_{jk}) \times \sum_{j=1}^{n} (q_j x_{jk}) \tag{6}$$

Pour trouver la date de fin du job en position k, il faut ajouter sa date de début et le temps de production :

$$\text{C}_{ik} = t_{ik} + \sum_{j=1}^{n} p_j x_{jk} \qquad \forall i \in \{1,...,m\}, \quad \forall k \in \{1,...,n\} \tag{7}$$

Le véhicule ne doit pas partir avant que les jobs qu'il transporte ne soient terminés :

$$C_{mk} \leq F_v + HV(2 - y_{jv} - x_{jv}) \qquad \forall v \in \{1,...,V\}, \quad \forall k \in \{1,...,n\} \tag{8}$$

Le départ du job j correspond à la date de départ du véhicule qui le transporte :

$$Dep_j = \sum_{v=1}^{V} F_v y_{jv} \forall j \in \{1,...,n\} \tag{9}$$

Le calcul du coût de stockage FIN pour un job k est donné par la formule vue précédemment :

$$IC_k^{FIN} \leq \sum_{j=1}^{n} (Dep_j x_{jk} - C_{mk}) \times \sum_{j=1}^{n} q_j h_j^{FIN} x_{jk} \forall k \in \{1,...,n\} \tag{10}$$

Le coût de stock FIN revient donc à :

$$IC^{FIN} = \sum_{k=1}^{n} IC_k^{FIN} \tag{11}$$

Le coût des véhicules correspond au nombre de véhicules multiplié par le coût d'un véhicule :

$$VC = V \times c^V \tag{12}$$

La date due du job à la position k est inférieure à la date de départ du véhicule qui le transporte plus le délai T :

$$D_k \leq F_v + T + HV(2 - y_{jv} - x_{jk}) \tag{13}$$

Le pseudo retard du job en position k correspond à la différence entre sa date due et sa date de livraison multipliée par le coût d'une pénalité :

$$PT_k = D_k - \sum_{j=1}^{n} d_j x_{jk} \times \sum_{j=1}^{n} \pi_{jm} x_{jk} \tag{14}$$

Par conséquent le retard total est égal à :

$$PPC^M = \sum_{k=1}^{n} PT_k \tag{15}$$

Après l'élaboration de ce modèle, on s'est rendu compte que le fait d'utiliser les positions des jobs au lieu des jobs directement compliquait fortement les formules (sommes dans quasiment toutes les contraintes). Nous avons donc décidé de refaire notre modèle en utilisant une matrice de précédence des jobs à la place de la matrice contenant les positions des jobs.

Deuxième modèle :

Variables :

- $t_{i,k}$: matrice contenant la date de début du job en position k sur la machine i
- $C_{i,k}$: matrice contenant la date de fin du job en position k sur la machine i
- IC_k^{FIN} : coût de stockage FIN du job en position k
- IC^{FIN} : coût de stockage FIN total
- IC^{WIP} : coût de stockage WIP total
- D_k : date due du job en position k

Données :

- n : nombre de jobs
- V : nombre de véhicules
- m : nombre de machines
- h_j^{WIP} : coût du stockage WIP du job j
- h_j^{FIN} : coût du stockage FIN du job j
- q_j : quantité produite pour la tâche j
- $x_{j1,j2}$: égale à 1 si le job j1 est avant le job j2 sinon 0
- $y_{j,v}$: égale à 1 si le job j est dans le véhicule v sinon 0
- p_j : durée de la tâche j
- F_v : date de départ du véhicule v
- T : délai de livraison
- π_{jm} : coût de pénalité du job j sur la machine m

Fonction objectif :

L'objectif étant de minimiser les coûts.

$$(\mathit{Minimiser}) \qquad \mathrm{Z} = \mathrm{IC}^{\mathrm{FIN}} + \mathrm{V}c^{\mathrm{V}} + \mathrm{IC}^{\mathrm{WIP}} \sum_{j=1}^{n} \pi_j \mathrm{PT}_j \tag{16}$$

Contraintes :

L'ordre des tâches sur la machine i doit être respecté :

$$\mathrm{C}_{ij2} \geq \mathrm{C}_{ij1} + p_{ij2} \times x_{j1j2} \qquad \forall j1, \ \forall j2, \ j1 \neq j2 \tag{17}$$

La fin du job sur la première machine doit être supérieure à la durée du job :

$$\mathrm{C}_{1j} \geq p_{1j} \qquad \forall j1, \qquad \forall j \in \{1, ..., n\} \tag{18}$$

Les jobs doivent s'enchaîner correctement (dans l'ordre) sur les différentes machines :

$$\mathrm{C}_{i+1j} \geq \mathrm{C}_{ij} + p_{i+1j} \qquad \forall j \in \{1, ..., n\}, \ \forall i \in \{1, ..., m-1\} \tag{19}$$

Le coût de stock FIN pour le job j correspond à :

$$\mathrm{IC}_j^{\mathrm{FIN}} = \sum_{v=1}^{\mathrm{V}} (\mathrm{F}_v y_{jv} - \mathrm{C}_{mj}) \times q_j h_j^{\mathrm{FIN}} \tag{20}$$

Le coût de stock FIN revient donc à :

$$\mathrm{IC}^{\mathrm{FIN}} = \sum_{j=1}^{n} \mathrm{IC}_j^{\mathrm{FIN}} \tag{21}$$

Le coût de stockage WIP pour un job j prend en compte la quantité stockée, le coût de stockage et le temps passé en production :

$$\mathrm{IC}_j^{\mathrm{WIP}} = (\mathrm{C}_{mj} - \mathrm{C}_{1j}) \times h_j^{\mathrm{WIP}} \times q_j \tag{22}$$

Le coût de stockage WIP total correspond à :

$$\mathrm{IC}^{\mathrm{WIP}} = \sum_{j=1}^{n} (\mathrm{IC}_j^{\mathrm{WIP}}) \tag{23}$$

Le véhicule ne doit pas partir avant que les jobs qu'il transporte ne soient terminés :

$$C_{mj} \leq \sum_{v=1}^{V} F_v + y_{jv} \tag{24}$$

Le pseudo retard du job j est égal ou supérieur à 0 :

$$PT_j^M \geq 0 \tag{25}$$

Le pseudo retard du job en position k correspond à la différence entre sa date due et sa date de livraison :

$$PT_j \geq \sum_{v=1}^{V} F_v \times y_{jv} + T - d_j \qquad \forall j \in \{1, ..., n\} \tag{26}$$

2.2.3 Exploration des voisinages

Après avoir trouvé la bonne répartition pour les véhicules et avoir initialisé une séquence des tâches à ordonnancer, il est nécessaire de rechercher les voisins de cette solution afin de trouver la meilleure.

Pour réaliser cela, nous utilisons trois méthodes de voisinage différentes.

La première est la **SWAP**, en paramètre on lui transmet deux jobs. C'est une méthode qui prend un job et le change de position avec un autre job.

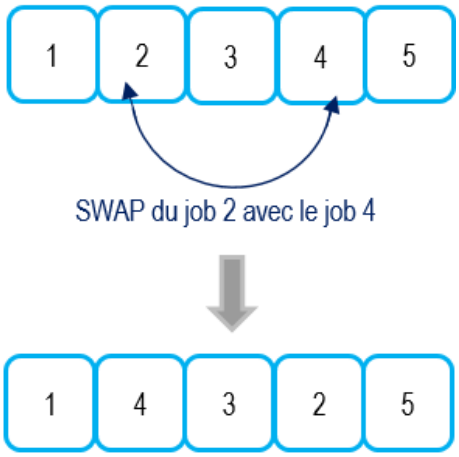


Figure 5 – *SWAP*

Ensuite il y a le **EBSR** : Extraction and Backward Shift Reinsertion, celle-ci prend un job et une position en paramètre. Elle consiste à introduire le job à la position i, vers l'arrière, c'est à dire que la position transmise doit être inférieure à celle du job en question.

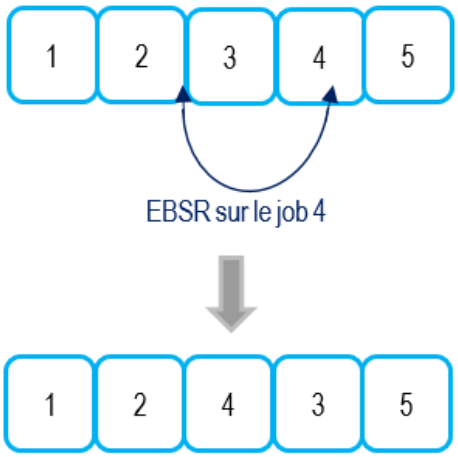


Figure 6 – *EBSR*

La dernière méthode est très semblable à la précédente, c'est le **EFSR** : Extraction and Forward Shift Reinsertion. Comme son nom l'indique, il s'agit du même principe que le EBSR sauf que le changement de position se réalise vers l'avant (forward) au lieu de l'arrière (backward).

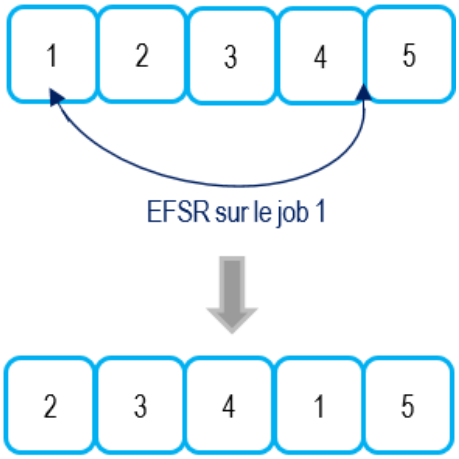


Figure 7 – *EFSR*

Après avoir choisi les méthodes que nous allions utiliser, il a fallu choisir de quelle manière nous allons réaliser la descente locale.

Pour cela, deux techniques différentes seront utilisées.

La première se nomme **Best Improve** : on teste toutes les solutions possibles ; à chaque fois qu'une solution meilleure est trouvée on la stocke et on continue. Et à la fin du parcours, la solution devient la meilleure stockée. Puis on recommence jusqu'au moment où la solution initiale et la meilleure stockée sont identiques.

La seconde qui s'appelle **First Improve**, consiste à parcourir les voisinages de la solution initiale, et dès qu'une meilleure solution est trouvée, on remplace l'initiale par celle-ci et on continue le parcours jusqu'à ce que la solution se stabilise.

L'algorithme pour le parcours du voisinage est donc le suivant :

```
Solution S;
Best Z = HV ;
//SWAP;
for i = 1 à n - 1 do
    for j = i+1 à n do
        Z = SWAP(i,j);
        //qui donne une solution S';
        if Z < Best Z then
            MAJ Best Z;
            if FirstImprove then
                S = S';
            end
        end
    end
end
//EBSR;
for i = 1 à n - 1 do
    for j = position(S) à 1 do
        Z = EBSR(i,j);
        //qui donne une solution S';
        if Z < Best Z then
            MAJ Best Z;
            if FirstImprove then
                S = S';
            end
        end
    end
end
//EFSR;
for i = 1 à n - 1 do
    for j = position(S) à n do
        Z = EFSR(i,j);
        //qui donne une solution S';
        if Z < Best Z then
            MAJ Best Z;
            if FirstImprove then
                S = S';
            end
        end
    end
end
S = S'
```

Algorithme 2 : Parcours du voisinage

On réalise cet algorithme tant qu'il existe une meilleure solution.

2.2.4 Choix de la meilleure solution

Pour choisir la solution parmi celles générées, c'est assez simple. Il suffira de prendre celle qui a le coût total le moins élevé. En effet l'objectif principal est de minimiser les coûts.

2.2.5 Fonction supplémentaire : La tournée du distributeur

Comme le développement des parties précédentes avaient bien avancé, mon encadrant m'a demandé d'ajouter la génération de la tournée de distributeur afin de compléter la modélisation et le développement du problème.

Il s'agit d'un travail en plus, et non pas la modélisation principale. C'est pour cela que nous utilisons une méthode simple. Il s'agit de la méthode du plus proche voisin.

Nous avons donc les différents véhicules de livraison chargés avec nos tâches. On sait que le véhicule part d'un dépôt, passe par l'usine de production et doit revenir au dépôt à la fin de sa tournée.

Initialement, on commence donc pour chaque véhicule, par ajouter à sa tournée, le trajet dépôt - usine.

Pour tous les trajets, trois éléments sont calculés et donc mis à jour : le coût du trajet, la distance parcourue et la date de livraison pour chaque job.

Ensuite pour savoir vers quel site de livraison aller, on choisit de regarder les distances usine - job i et de prendre la plus petite. Puis on réalise cette étape jusqu'à ce qu'il n'y ait plus de tâche à livrer. On termine donc par ajouter le trajet job n - dépôt.

Une fois cette étape terminée, on peut donc maintenant mettre à jour les différents coûts (grâce aux éléments mis à jour à chaque trajet). En effet, on peut désormais calculer les éléments manquants, comme le coût de transport, le retard réel et donc le coût du producteur réel et celui du distributeur.

Le problème est désormais complet.

Deuxième partie

Partie 2 : Développement

4 Mise en oeuvre

1 Implémentation

Une fois la modélisation terminée, il est nécessaire de commencer l'implémentation du générateur de données et de l'algorithme de résolution.

1.1 Générateur de données

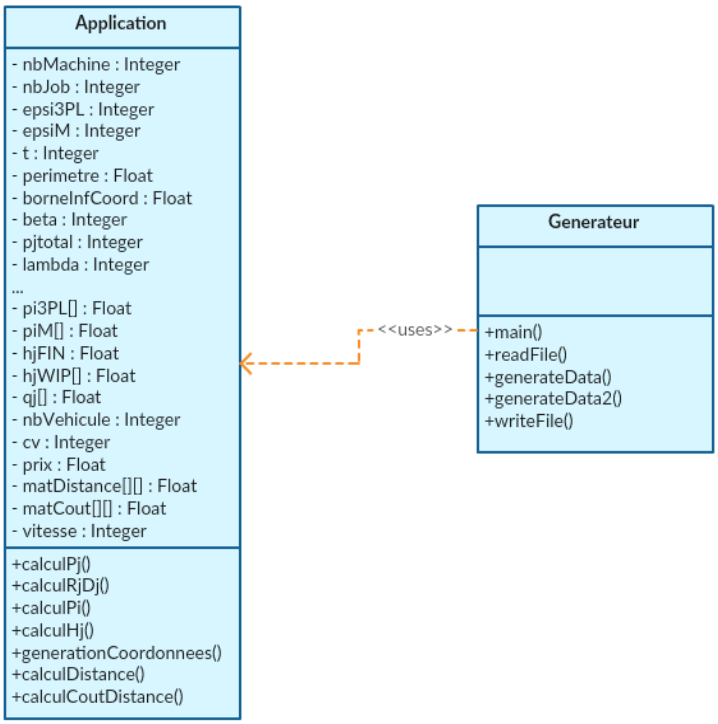


Figure 1 – *Diagramme de classe du générateur*

Cette première implémentation contient deux classes : Generateur et Application.

Pour "Generateur", elle est composée du main, de la méthode chargée de la lecture du fichier (readFile) ainsi que d'une méthode pour la création de l'Application et de l'appel à ses méthodes (generateData).

Pour stocker les données d'entrée et celles générées, j'utilise donc la classe Application qui possède en attributs les différents éléments : nombre de machines, nombre de jobs, coût de pénalité du producteur et du distributeur, le périmètre de livraison, la matrice de distance, la matrice des coordonnées des sites à livrer, ... Ses méthodes sont toutes celles nécessaires pour les calculs des paramètres : génération des matrices de distances, calcul des dates dues à partir des bornes,...

Le générateur de données prend plusieurs paramètres, tout d'abord on lui donne le chemin du fichier de configuration, puis on lui indique deux variables : le nombre de jobs à créer et la valeur de T (le délai de livraison).

Dans la méthode main de la classe Generateur, je récupère les arguments passés à l'application. A partir de ces éléments, il est possible d'initialiser certaines dimensions de tableaux c'est donc ce que je fais ensuite.

Maintenant, il faut récupérer les différentes informations présentes dans le fichier de configuration. Pour cela j'ai créé une méthode *readFile*.

Comme vu précédemment dans ce rapport, plusieurs applications existent (trois différentes), la tâche que réalise cette méthode est donc d'appeler la méthode de génération en fonction de l'application concernée (cette information est écrite en première ligne du fichier).

Dans la méthode de génération (*generationData*), la première chose qu'elle fait consiste à lire les informations du fichier de configuration et de les stocker dans l'objet Application.

Certains éléments de la configuration comme la date due nécessite un traitement supplémentaire pour séparer les données écrites dans le fichier de configuration : les bornes, et la valeur pour les jobs présents dans ces bornes.

Une fois tous les éléments récupérés, il est temps de passer aux différentes générations. Les méthodes concernées sont dans la classe Application.

Premièrement, il y a *calculPj* : elle permet de calculer les durées de chaque job. Pour cela on tire de manière aléatoire la valeur entre deux bornes (issues du fichier de configuration).

Ensuite, il y a le calcul des dates dues des jobs. Pour cela nous utilisons des dates de début rj et en fonction de cela, on lui affecte une certaine date due. Pour connaître le rj, c'est précisé dans le fichier de configuration avec la proportion d'éléments à fixer à telle valeur pour rj et telle valeur pour dj.

On trouve par la suite la méthode qui concerne le calcul des coûts de pénalité pour le producteur et pour le distributeur. Cela s'effectue grâce aux formules trouvées précédemment et en utilisant les paramètres du fichier de configuration.

La méthode *calculHj*, comme celle des pénalités utilise les données du fichier et la formule vue précédemment pour calculer le coût de stockage en cours et en fin de production pour chaque tâche.

Enfin, il reste les trois matrices à calculer : les coordonnées, la matrice de distances, la matrice de coûts de transport

Pour cela, la première méthode *generationCoordonnees* sert à générer les coordonnées x et y pour chaque job. On ajoute à cela deux sites supplémentaires pour représenter l'usine et le dépôt. Des coordonnées sont aussi générées pour ces deux éléments.

Ensuite pour la matrice de distances c'est *calculDistance* qui s'en occupe. Pour chaque "couple" de sites à livrer, elle calcule la distance euclidienne et la saisie dans la matrice correspondante.

Pour finir, il ne reste plus que la matrice des coûts de transport. La méthode *calculCoutDistance* consiste à calculer pour chaque distance de la matrice précédemment trouvée le coût avec la formule que l'on a conçue plus tôt.

Maintenant toutes les données nécessaires à l'algorithme de résolution sont calculées, il faut alors les écrire dans un fichier de sortie pour pouvoir l'utiliser dans l'algorithme.

Pour cela j'ai créé une méthode *writeFile*,qui se trouve dans la classe générateur. Elle utilise la classe PrinterWriter pour écrire dans un fichier de sortie dont le nom et le chemin ont été passés dans les paramètres du programme. Chaque élément nécessaire en entrée de l'algorithme de résolution est donc écrit ligne par ligne dans le fichier.

1.2 Algorithme de résolution

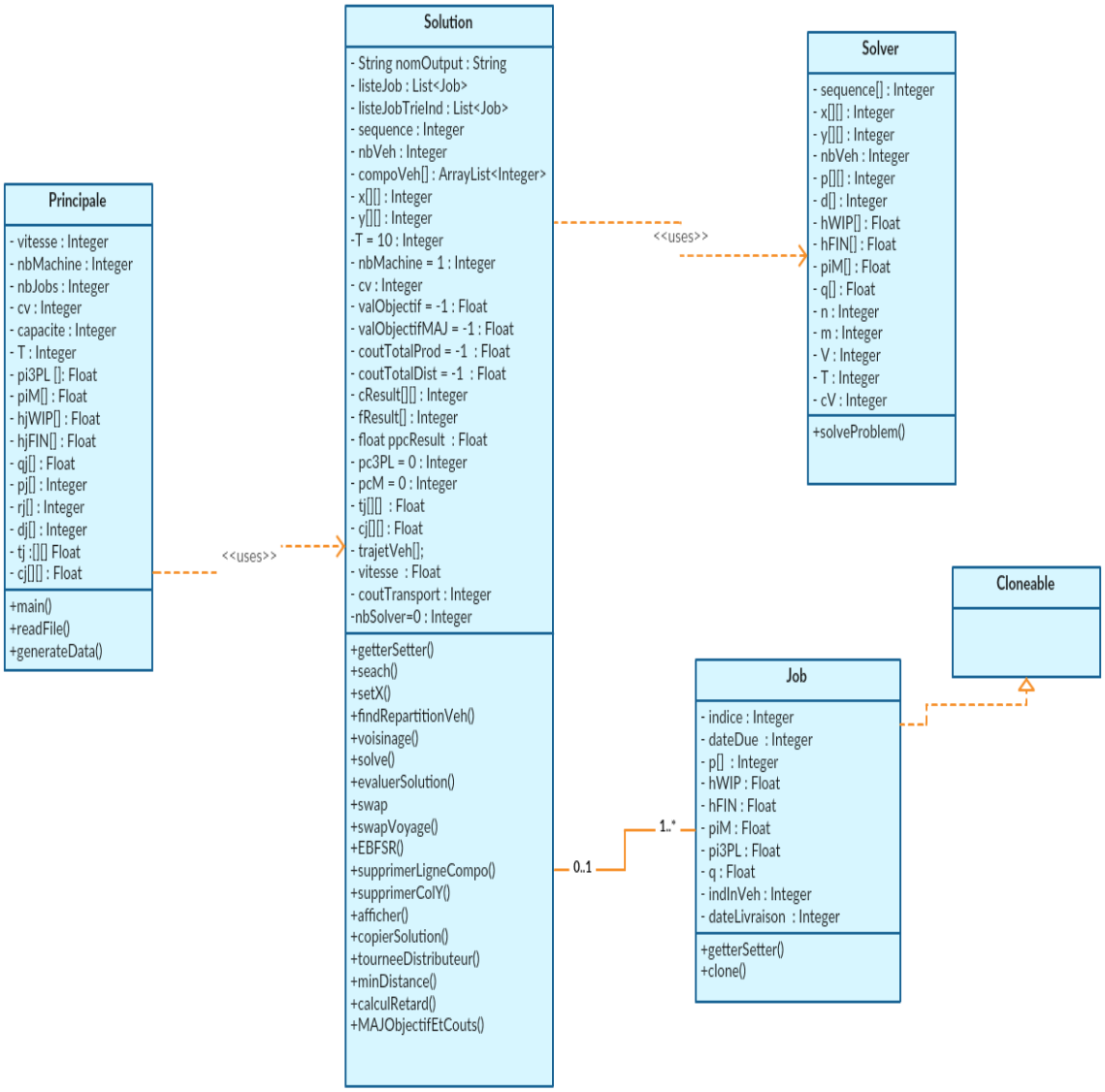


Figure 2 – *Diagramme de classe de l'algorithme de résolution*

Une fois les données générées et écrites dans un fichier, il faut donc développer l'algorithme de résolution.

Pour cela j'ai créé quatre classes : Principale, Solution, Solver et Job.

La première est Principale : c'est dans cette classe que se trouve la fonction *main* qui lance le programme, la fonctionnalité pour lire le fichier de données et remplir l'objet solution.

Tout d'abord, la fonction *main* récupère les deux paramètres. Il y a le chemin et le nom du fichier de données (créé par le générateur), le temps en minutes pour borner l'exécution de l'algorithme. Puis il initialise un objet Solution.

Ensuite il est nécessaire de lire le fichier de données et d'initialiser l'objet Solution avec les différentes informations récupérées. Cette partie est réalisée par les fonctions *readFile* et *generateData*, qui s'occupent

de lire le fichier ligne par ligne et de compléter l'instance de la classe Solution.

Une fois l'objet complété, il faut maintenant résoudre le problème.

Dans un premier temps, afin de mesurer le temps d'exécution, la fonction *main* démarre un timer avec la méthode *System.nanoTime()* et lance ensuite la méthode search de la classe Solution sur l'instance qu'il vient de créer.

Une fois la résolution terminée, dans la fonction main, les différentes valeurs sont récupérées (valeur objectif, coût total du distributeur, coût total du producteur, nombre de voisins visités), puis écrites dans un fichier de sortie.

Pour la classe Job, c'est dans celle-ci que l'on décrit ce qu'est une tâche. Elle implémente la classe Cloneable.

C'est donc une classe assez simple, regroupant les attributs d'un job (coût de pénalité, quantité produite, prix, date due, date de livraison,...) et les méthodes getter et setter de chacun des éléments.

Il y a en plus de cela une méthode *clone* qui permet de gérer la duplication d'une instance Job.

Comme expliqué précédemment, l'évaluation de la solution s'effectue à l'aide d'un solver, c'est donc dans la classe Solver qu'il est défini. Pour l'implémenter, j'utilise la bibliothèque cplex qui met à disposition une classe IloCplex afin de représenter notre modèle et de le résoudre.

Dans le constructeur de la classe Solver, je récupère les différentes informations de la solution à évaluer (qui est passée en paramètre) puis j'initialise les matrices de résultats.

Ensuite la classe est composée d'une méthode Solver qui prend en paramètre les différents matrices et tableaux où vont être stockés les résultats (matrice des dates de début des jobs sur chaque machine, date de départ des véhicules,...).

On crée donc une instance de la classe IloCplex (cplex), cet objet représente notre solver. Puis il y a une succession d'étapes.

La première partie consiste à décrire le modèle. Au départ, nous déclarons les différentes variables avec l'objet IloNumVar que l'on affecte à notre objet cplex.

Ensuite il reste à créer les constantes et la fonction objectif. Dans la classe IloCplex, ces éléments représentent des expressions. Donc on crée un objet IloLinearNumExpr avec la méthode d'IloCplex : *linearNumExpr()*. Puis on lui ajoute les différents composantes de notre expression (somme, multiplication) avec la méthode *addTerm* de la classe IloLinearNumExpr. Enfin il suffit de l'affecter à notre objet cplex en lui précisant si c'est la fonction objectif (*addMinimize* ou *addMaximize*) ou une contrainte avec un signe inférieur (*addLe*), supérieur (*addGe*) ou égal (*addEq*)

Une fois le modèle créé, il ne reste plus qu'à lancer sa résolution et donc l'évaluation de notre solution. C'est la méthode *solve()* qui sert à cela.

Une fois l'instance résolue, il est désormais possible de récupérer la valeur objectif obtenue (*getObjValue*) et les valeurs des différentes variables (*getValue*).

Pour finir la dernière classe est Solution. C'est cette classe qui représente le coeur du programme. En effet, c'est ici que l'on implémente une solution et qu'on la résout.

Cette classe est composée de beaucoup d'attributs, chacun représente soit les données d'entrée (liste des jobs, nombre de machines, ...) soit ceux où seront stockés les résultats (coût du producteur, valeur objectif, trajet de livraison,...). Ensuite, nous avons de multiples méthodes en plus des getter et setter.

Le fonctionnement de cette classe est le suivant. Une fois une instance de Solution créée. On appelle la fonction *search*, c'est ici que toute la recherche se passe.

Tout d'abord, la liste des jobs est triée par date due croissante. On initialise la séquence de départ (avec les dates dues triées). On peut désormais initialiser la matrice X qui décrit la succession des tâches avec la méthode *setX*.

Ensuite, il faut initialiser la répartition des tâches dans les véhicules : pour cela on lance la méthode *findRepartitionVeh* qui étudie toutes les répartitions des véhicules possibles (de 1 job par véhicule à n/2 jobs par véhicule) afin de trouver la meilleure proposition. A chaque fois, le solver est appelé pour l'évaluer.

Une fois cette étape réalisée, nous avons initialisé notre première solution. Maintenant il faut étudier son voisinage. C'est la méthode voisinage qui réalise cette partie. Elle sera expliquée en détail dans la partie suivante.

Nous avons donc désormais la meilleure solution trouvée. Nous pouvons donc calculer le pseudo coût du producteur mais il manque le coût réel du producteur et du distributeur. Pour cela, il est nécessaire d'ordonnancer la livraison des produits pour connaître entre autres les dates de livraison et donc le retard. J'ai donc développé une fonction tourneeDistributeur, qui, pour chaque véhicule planifie sa tournée. Pour chaque tâche, la fonction parcourt la matrice des distances et prend le site le plus proche, l'ajoute à son trajet et met à jour la somme de la distance parcourue et du coût de transport.

Désormais, il ne reste plus qu'à appeler les méthodes *calculRetard* et *MAJObjectifEtCout*, qui comme leur nom l'indique permettent d'obtenir la valeur objectif et le coût du producteur et du distributeur mis à jour.

En plus de cela, il y a des méthodes "outils" que j'ai créées pour ne pas surcharger les méthodes principales. Nous avons par exemple *swapVoyage* qui est utilisée dans le voisinage, *solve* pour lancer le solver, copierSolution pour faire une copie correcte de l'objet Solution, etc.

2 Détail sur un point difficile

Pour cette partie, j'ai décidé d'aborder la méthode voisinage. En effet, cet élément est assez complexe.

Cette méthode prend plusieurs paramètres : l'heure de début de la recherche, et un booléen qui précise si l'on veut lancer l'exploration du voisinage en first improve ou en best improve.

Je commence le code de cette méthode par une boucle while pour lancer la recherche des voisins tant que l'on trouve une solution meilleure.

Ensuite pour chaque passage dans la boucle, je conserve tout d'abord la solution dans une variable avant qu'elle ne soit modifiée. Puis j'initialise la "bestSolution" avec la solution initiale.

Maintenant, on peut commencer le parcours des voisinages.

Tout d'abord, pour chaque méthode de voisinage (Swap, EBSR, EFSR). On a deux boucles for imbriquées :

- Pour le swap, chaque job est swappé avec tous les autres jobs.
- Pour EBSR, on récupère la position du job et on l'essaye à toutes les positions précédentes.
- Pour EFSR, on récupère la position du job et on l'essaye à tous les positions suivantes.

Ensuite pour toutes les méthodes, à chaque changement, la solution est évaluée. Si la valeur objectif est meilleure, on copie cette solution dans "bestSolution" . Dans le cas du best improve, on réinitialise la solution avec celle sauvegardée au début de la boucle while, sinon pour le cas du first improve, dès que la solution est meilleure, elle devient la solution sur laquelle on continue la recherche des voisins.

Après le parcours des trois méthodes différentes, on regarde si la meilleure solution trouvée est la même que celle du départ. Sinon la boucle while continue. Si oui, on a trouvé la meilleure solution ou bien un minimum local, on met donc le booléen d'arrêt de la boucle while à true et la méthode voisinage s'arrête.

Une fois le fonctionnement principal décrit, voici le fonctionnement de chacune des techniques de voisinage qui sont représentées par des méthodes distinctes.

2.1 SWAP

Nous avons dans cette méthode, deux jobs en paramètre qui sont les éléments à swapper.

Tout d'abord, on échange leur position dans la séquence, et on met à jour la matrice X grâce à la méthode *setX*.

Il faut désormais modifier la composition des véhicules de ces jobs pour les échanger aussi. Pour réaliser cela, j'ai créé une méthode *swapVoyage*.

Une fois la composition des véhicules modifiée, il est nécessaire de mettre à jour la matrice Y qui représente ces véhicules.

2.2 EBSR et EFSR

Pour ces deux techniques, j'ai écrit une méthode. Cette fonction prend deux paramètres : le job à déplacer et la position à laquelle on veut le déplacer.

La première étape est là aussi de modifier la séquence. Comme c'est une liste, il suffit de supprimer l'élément à la position et de l'insérer à la nouvelle position avec la méthode *add*. Puis la matrice X est mise à jour grâce à *setX*.

Il faut ensuite mettre à jour la composition des véhicules. On récupère le job qui est avant la nouvelle position de la tâche ; si la position est 0, on récupère donc celui après cette nouvelle position. A partir du véhicule du job récupéré, je modifie la composition des véhicules en fonction. Puis, il reste juste à mettre à jour la matrice Y.

Une dernière modification est nécessaire. En effet, dans le cas où le changement de véhicule d'un job engendre un véhicule vide, il faut donc le supprimer et mettre à jour la matrice Y en éliminant la colonne concernée.

3 Complexité

Tout d'abord pour le générateur de données, comme celui-ci ne fait que des tirages et des calculs simples, la complexité est assez simple.

Pour celle de l'algorithme de résolution, c'est différent.

En effet, nous avons un nombre polynomial de voisins. Cependant, l'exploration en first improve et en best improve revient à faire la descente d'une arborescence en profondeur ou en largeur selon la méthode. C'est pour cela que sa complexité est exponentielle.

Toutefois, vu justement que cette complexité exponentielle engendre des temps d'exécution trop élevés (voir le chapitre des tests), nous avons décidé de borner en temps cette exploration. La complexité devient donc polynomiale.

5 Campagne de tests et résultats d'application des tests

1 Conditions d'expérimentation

Les tests ont été réalisés sur une machine Windows 10, 64 bits possèdant un processeur Intel Core i5 et une RAM de 6Go.

2 Génération des données de test

La première étape dans les tests est de produire les instances de données avec le générateur.

Il a fallu tout d'abord énumérer les différents nombres de jobs à prendre. J'ai donc choisi de me baser sur l'estimation que nous avions faite lors de l'étude de la première partie afin de s'approcher de la réalité.

Dans le cas de l'application 1, c'était entre 5 et 10 jobs et celle de l'application 2 : environ 50 jobs.

Afin de tester plusieurs situations, j'ai décidé de lancer les tests avec 5 jobs, puis 10, ensuite 25, 50, 100 et 150. Le but est d'énumérer le plus de cas possible. Je commence donc avec une petite instance de 5 jobs, ensuite j'essaie des nombres de jobs intermédiaires, pour enfin terminer sur des grosses instances (100 et 150).

Je lance alors le générateur de données, et pour les trois applications, je génère 5 fichiers d'instances pour chaque nombre différent de jobs.

Pour le nom des fichiers, j'utilise le format suivant :

result-numeroApplication-nbJobs-delaiT-numeroInstance

Par exemple, pour le fichier result-1-5-24-1.txt, il s'agit d'une instance de données de l'application 1, avec 5 jobs, et un délai T 24h et c'est le premier fichier d'instance.

Une fois les différents fichiers créés, je les copie dans un dossier data.

3 Déroulement des tests

Pour lancer les tests, j'utilise un fichier .bat. Ce fichier se présente de la façon suivante :

```
@echo off
for /f %%f in ('dir /b data\') do java -jar solver_best.jar data\%%f 10

@echo off
for /f %%f in ('dir /b data\') do java -jar solver_first.jar data\%%f 10
```

Figure 1 – *Fichier .bat*

Il s'agit d'un fichier qui, lors de son lancement, parcourt le dossier data et pour chaque élément, lance mon programme de résolution en best improve dans un premier temps puis en first improve.

4 Lancement des tests

J'ai donc lancé une première série de tests. Ce premier lancement ne contenait pas les instances de données de 150 jobs.

Pour les petites instances de jobs, le temps d'exécution était relativement court : entre 1 et 6 secondes. Quant aux instances moyennes, pour 25 jobs l'exécution dure entre 150 et 200 secondes et pour 50 jobs, environ 8000 secondes. Par contre, le lancement des instances de 100 jobs prenait beaucoup de temps : l'instance a tourné plus de 16h.

J'ai donc fait le point avec mon encadrant et il m'a indiqué que le but de ces tests, c'est d'évaluer les méthodes développées, mais sans avoir une durée d'exécution trop longue.

Suite à cette remarque, nous avons donc décidé de borner l'exécution en temps. Pour cela, nous avons ajouté dans la boucle while du parcours de voisinage, une condition supplémentaire de temps. Cette limite est passée en paramètre du programme. Pour mes tests, la limite est fixée à 10 minutes.

J'ai donc relancé ma série de tests, et cette fois-ci en intégrant les instances de 150 jobs.

La recherche de la meilleure solution est donc désormais limitée à 10 minutes. Pour les instances de 5 à 50 jobs, le temps n'est pas dépassé. Cependant pour les plus grandes instances, avant d'atteindre la vérification du temps, il faut passer une première fois complètement dans cette boucle while, et dans le cas des grosses instances, pour ce premier passage, le temps est beaucoup trop élevé : on met ainsi entre 2 et 12h d'exécution.

Après un deuxième point avec mon encadrant au sujet de ces tests, nous avons donc décidé d'ajouter un test supplémentaire dans la boucle for de chaque méthode de voisinage. Cette condition consiste à limiter la distance entre les jobs à échanger. Pour les instances de 100 jobs, ce delta maximum est fixé à 4 et pour 150 jobs à 2.

Une fois les modifications effectuées, les tests sont lancés une dernière fois.

5 Résultats des tests de performance

En parallèle de l'exécution, j'ai lancé JVM afin de tester les performances de mon programme de résolution. Les tests ont été lancés sur une instance de 100 et 150 jobs car les autres duraient trop peu de temps pour les analyser.

Ci-dessous les résultats au niveau de la mémoire :

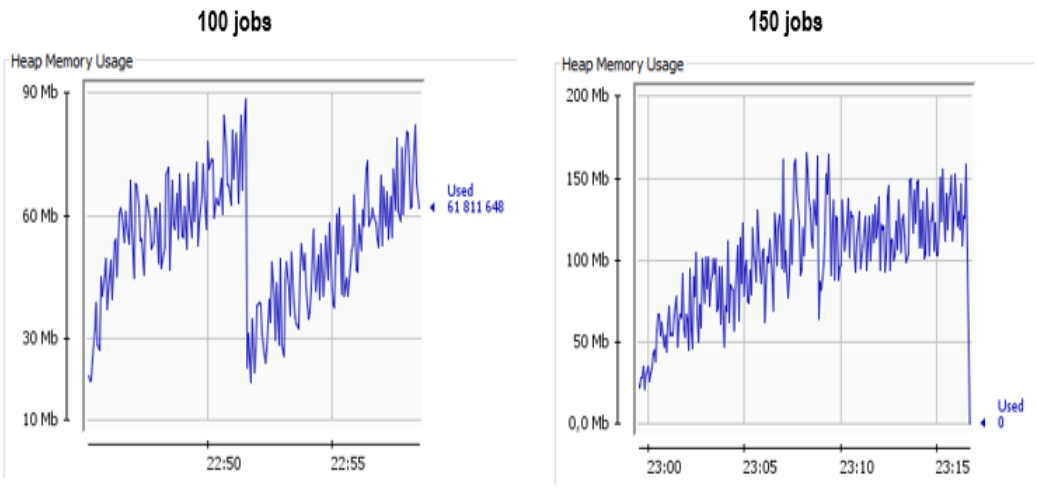


Figure 2 – *Test de performance - Mémoire*

Tout d'abord, on peut remarquer que le test sur 100 jobs est divisé en 2 parties alors que celui sur 150 jobs non. En effet cela représente chaque passage dans la boucle while de la recherche de voisinage : pour 100 jobs, en 10 minutes il y a deux passages, alors que pour 150 jobs, le passage dans la boucle est plus long donc il n'y en a eu qu'un seul.

Ensuite on remarque que plus on a de jobs et plus on avance dans la boucle while, plus la mémoire utilisée augmente.

Au niveau du CPU nous avons le résultat suivant :

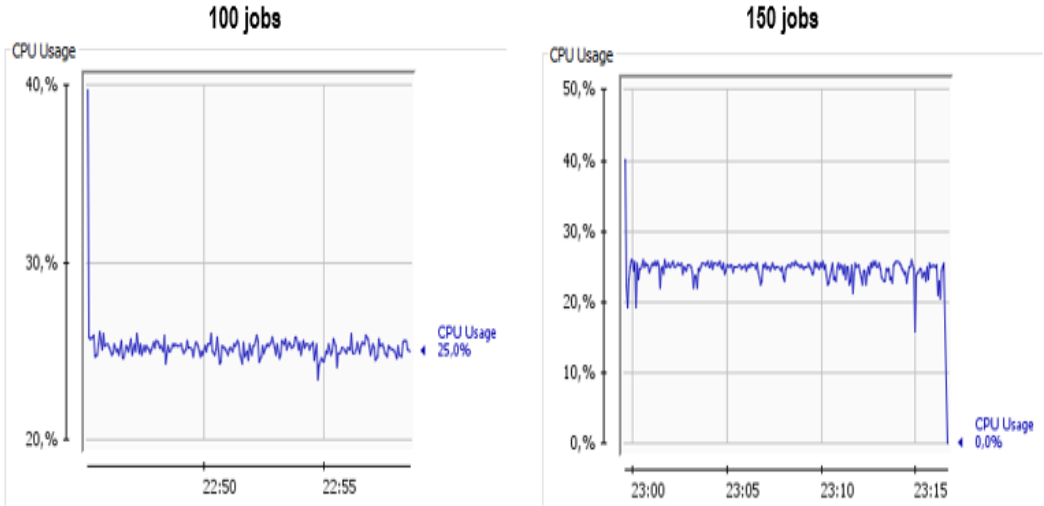


Figure 3 – *Test de performance - CPU*

En moyenne, pour 100 jobs et 150 jobs nous avons 25% d'utilisation CPU. C'est donc une utilisation assez faible. Pour améliorer cela, on pourrait paralléliser le travail pour que l'exécution soit plus rapide en utilisant plus de CPU.

6 Résultats des tests fonctionnels

Une fois tous les tests terminés, je réalise deux analyses.

Tout d'abord, je vais regarder si le résultat donné par mon programme est cohérent. Pour faire cela, je prends un fichier d'instance assez simple (5 jobs) et j'analyse si le résultat est possible.

Nous avons en données :

- 5 jobs
- 2 machines
- 24h de délai de livraison

La matrice pj est la suivante :

Table 1 – *Tableau pj*

	Job 1	Job 2	Job 3	Job 4	Job 5
Machine 1	34	33	30	30	31
Machine 2	35	27	26	29	28

La matrice tj :

Table 2 – *Tableau tj*

	Job 1	Job 2	Job 3	Job 4	Job 5	Usine	Dépôt
Job 1	0.0	0.08	0.13	0.16	0.13	0.06	0.17
Job 2	0.08	0.0	0.04	0.08	0.16	0.11	0.15
Job 3	0.13	0.04	0.0	0.05	0.2	0.15	0.17
Job 4	0.16	0.08	0.05	0.0	0.2	0.17	0.15
Job 5	0.13	0.16	0.2	0.2	0.0	0.07	0.09
Usine	0.06	0.11	0.15	0.17	0.07	0.0	0.12
Dépôt	0.17	0.15	0.17	0.15	0.09	0.12	0.0

Je lance donc le programme Best Improve sur cette instance de données.

Tout d'abord voici le codage de la solution finale :

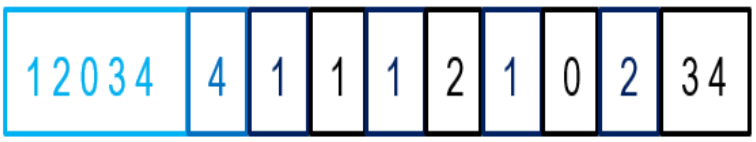


Figure 4 – *Solution finale codée*

Nous avons donc dans un premier temps la séquence, ensuite la composition des 4 véhicules : les trois premiers transportent 1 job quant au dernier il en a 2.

L'ordonnancement des tâches est le suivant :

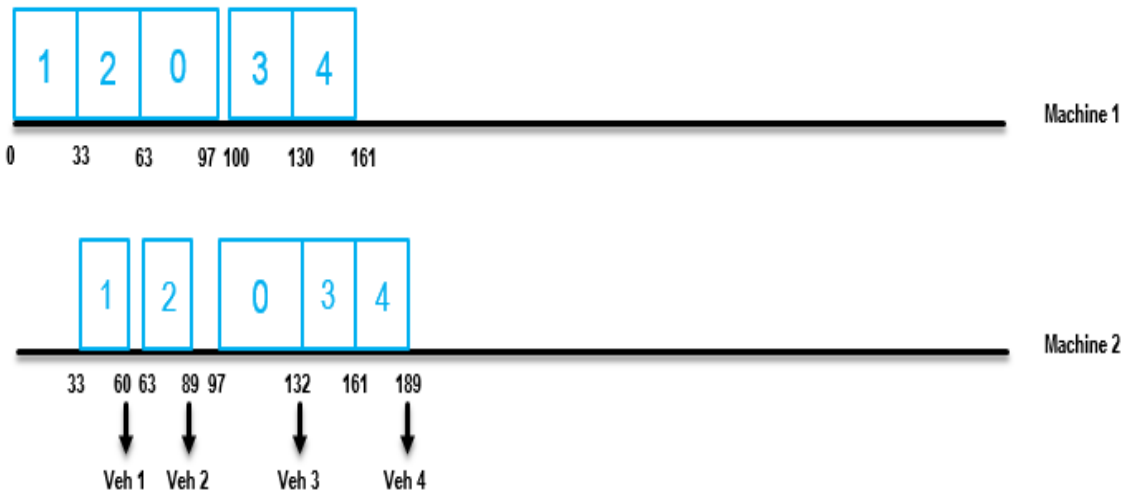


Figure 5 – *Ordonnancement de la solution*

Comme nous pouvons le constater, le résultat est cohérent. Maintenant on peut regarder la distribution des jobs.

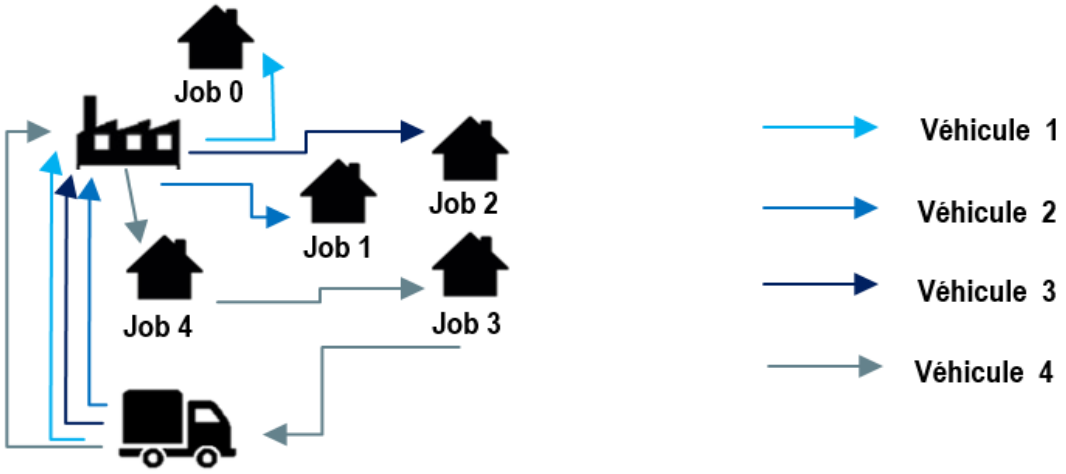


Figure 6 – *Distribution de la solution*

Là aussi, la distribution respecte les contraintes. En effet on peut voir qu'un trajet commence par l'usine et termine par le retour à l'entrepôt, de plus pour le dernier véhicule le job 4 est le plus proche de l'usine c'est donc le premier site livré.

Quant à la deuxième analyse, elle contiste à analyser le fichier texte des deux méthodes (resultBest et resultFirst) avec les différentes lignes écrites par chaque instance.

L'une des manières la plus simple pour effectuer l'analyse est d'utiliser un fichier Excel.

J'ouvre donc les fichiers resultFirst et resultBest dans Excel et je fusionne les chiffres dans un même fichier. Nous avons ainsi deux parties dans le fichier, celle concernant la première méthode et une deuxième pour la seconde méthode.

La composition du document est la suivante : nous avons une première colonne en commun qui contient le nom du fichier d'instances concerné par la ligne de résultat. Ensuite la constitution des deux parties est la même pour les deux méthodes. Nous avons tout d'abord une colonne avec le temps d'exécution en seconde, puis la valeur objectif trouvée, et celle mise à jour, ensuite il y a le coût du distributeur et le

coût du producteur, et enfin le nombre de voisins visités.

L'élément à analyser dans ce fichier est le fait de savoir si la méthode best improve est meilleure que celle en first improve ou si c'est l'inverse.

Pour cela, j'ai créé deux colonnes supplémentaires dans mon fichier excel : une colonne où la cellule contient 1 si la méthode best improve est meilleure et une autre colonne où la cellule contient 1 si c'est la méthode first improve qui est la meilleure. A la fin des colonnes, je réalise une somme sur les colonnes pour avoir une idée globale de la tendance et ensuite je réalise un tableau synthèse en regroupant ces résultats par instance avec le même nombre de jobs.

Pour les derniers tests que j'ai réalisés voici le tableau de synthèse correspondant, pour chaque nombre de jobs nous avons le nombre d'instance pour laquelle Best Improve ou First Improve était la meilleure méthode ou si elles sont à égalité :

Table 3 – *Analyse de la meilleure méthode*

	Best Improve	First Improve	Egalité
5 jobs	3	0	**12**
10 jobs	4	1	**10**
25 jobs	5	3	**7**
50 jobs	0	**15**	0
100 jobs	0	**13**	2
150 jobs	0	**10**	5
Total	12	**42**	36

Comme on peut le voir, globalement, c'est la méthode first improve qui donne le plus souvent la meilleure solution.

S'il l'on regarde la comparaison des résultats regroupés par nombre de jobs, on observe que pour les petites instances, soit c'est la best improve qui donne la meilleure solution soit les deux méthodes sont à égalité . Cependant, à partir de 50 éléments, c'est la méthode first improve qui devient la plus efficace.

Pour l'analyse d'un point de vue temporelle, celle-ci est un peu faussée vu que l'on borne la recherche de solution en temps.

6 Reproductibilité

1 Générateur de données

Ci-dessous les différents éléments à connaître pour reprendre le projet, en particulier le générateur de données.

1.1 Fichier d'entrée et de sortie

Comme nous l'avons vu précédemment, le générateur de données prend en entrée un fichier de configuration. Le format de ce fichier doit être impérativement respecté sinon le générateur n'arrivera pas à lire les différentes données.

Il y a trois types d'application, donc trois fichiers de configuration. Le nom du fichier n'a pas d'importance puisque c'est à vous de le renseigner lors du lancement du générateur.

La première ligne contient le nom de l'application concernée par le fichier : application1, application2 (il s'agit du même nom pour les application 2.1 et 2.2).

Pour chaque ligne à part les dates dues (que j'expliquerai plus loin), elle se compose du nom de la variable, d'un espace, puis de sa valeur. Lorsque c'est un intervalle, on écrit le nom de la variable, espace, puis deux points, espace, la première valeur, et après un espace la seconde valeur. Si c'est un choix entre différentes valeurs, c'est la même syntaxe sauf que c'est un point virgule au lieu des deux points.

Nous avons donc dans l'ordre :

Table 1 – *Eléments du fichier de configuration*

Nom de la variable	Type	Détails
vitesse	Km/h	Vitesse moyenne de déplacement du véhicule
nbmachine	Chiffre	Nombre de machine
pjtotal	Chiffre	(uniquement pour l'application 1) Temps total de production pour tous les jobs
epsi2	Pourcentage	Epsilon du distributeur
t	Pourcentage	Taux d'obsolescence
beta	Pourcentage	Pour le calcul du coût de stockage
perimetre	Km	Périmètre de livraison
borninfcoord	Chiffre	Borne inférieure pour les coordonnées des sites
cv	Euros	Coût d'un véhicule
lambda	Pourcentage	Pour le calcul du coût d'un trajet
cstcout1	Chiffre	Constance pour le calcul du côut d'un trajet
cstcout2	Chiffre	Constance pour le calcul du côut d'un trajet
cstcout3	Chiffre	Constance pour le calcul du côut d'un trajet
cstcout4	Chiffre	Constance pour le calcul du côut d'un trajet
datedue	Chiffre	Nombre de dates dues différentes
détail des dates dues	Chiffre	Voir explication en dessous
pj	Intervalle de chiffre	(uniquement pour l'application 2) Pj minimum et pj maximum
capacite	Intervalle en kg	Capacité min et max du véhicule
epsi1	Choix de pourcentage	Epsilon du producteur
quantite	Intervalle en kg	Quantité produite min et max
prix	Intervalle en euros	Prix min et max des produits

Pour la date due, l'écriture est un peu différente. Sur la première ligne, nous avons datedue, espace, le nombre de lignes dessous qui concerne les dates dues. Ensuite pour chaque ligne où la date due est différente, on écrit la proportion des jobs concernés espace, rj, espace la date due correspondante. Par exemple si on a la moitié des jobs avec rj à 12 et dj à 24 et l'autre partie 24 et 48 cela donne :

date due 2
1 2 12 24
1 2 24 48

Une fois le générateur lancé, celui crée et écrit ses résultats dans un fichier de sortie.

Pour son format, pour les variables où il n'y qu'un chiffre derrière, on suit la règle précédente, variable espace, la valeur. Par contre pour représenter les matrices, le générateur écrit le nom de la matrice, et à la ligne suivante la matrice correspondantes (après chaque ligne de la matrice il y a un retour à la ligne).

Nous avons alors :

Table 2 – *Eléments du fichier de résultats*

Nom de la variable	Type	Détails
vitesse	Km/h	Vitesse moyenne de déplacement du véhicule
nbmachine	Chiffre	Nombre de machine
nbjob	Chiffre	Nombre de jobs
cv	Euros	Coût d'un véhicule
capacite	Intervalle en kg	Capacité min et max du véhicule
T	Heure	Délai de livraison
pi3PL	Matrice 1 ligne	π 3PL pour chaque job
piM	Matrice 1 ligne	π M pour chaque job
hjFIN	Matrice 1 ligne	Coût de stockage FIN pour chaque job
hjWIP	Matrice 1 ligne	Coût de stockage WIP pour chaque job
qj	Matrice 1 ligne	Quantité produite pour chaque job
pj	Matrice	Temps de production pour chaque job sur chaque machine
rj	Matrice 1 ligne	Date de début de production pour chaque job
dj	Matrice 1 ligne	Date due pour chaque job
tj	Matrice	Distance entre tous les sites de livraison + dépôt et usine
cj	Matrice	Coût de trajet entre tous les sites de livraison + dépôt et usine

1.2 Lancement du programme et résultat

Pour lancer le générateur, j'utilise la console. Pour réaliser cela, j'ai donc besoin du .jar correspondant à mon projet.

Pour créer ce fichier avec Eclipse c'est assez simple :

Il faut cliquer sur File -> Export, ensuite dans le dossier Java sélectionner Runnable JAR File puis cliquer sur Next :

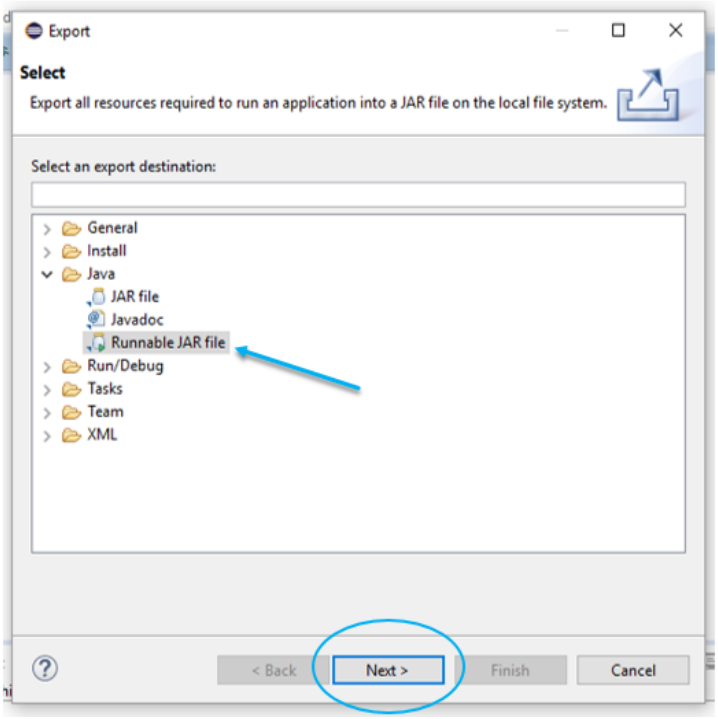


Figure 1 – *Runnable JAR File*

Sur l'écran suivant, il faut sélectionner le projet que l'on veut exporter : PRD - Générateur de données, puis saisir le chemin de destination et enfin cliquer sur Finish :

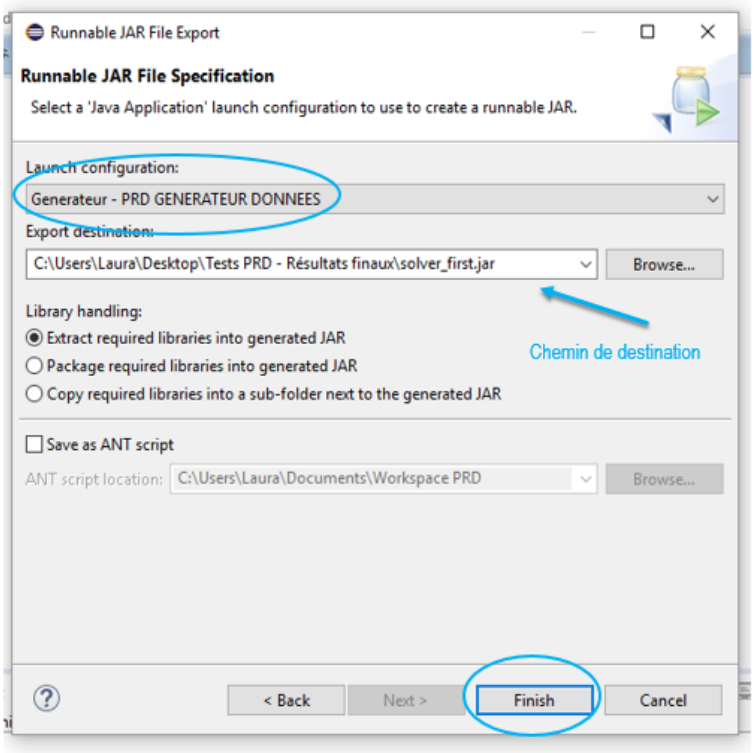


Figure 2 – *Choix du projet à exporter*

Le projet est maintenant exporté en .jar.

Une fois cela terminé, il est nécessaire de le lancer. On utilise alors la ligne de commande suivante :

java -jar generateur.jar "le chemin et le nom du fichier de configuration" "le chemin et le nom du fichier de résultats" nombredejob delai.

Comme on peut le voir, il s'agit de la commande classique pour lancer un fichier .jar, puis l'enchaînement de nos 4 paramètres. Ensuite, on appuie sur entrée et notre fichier de résultat est généré.

1.3 Exemple

Ci dessous un exemple de fichier de configuration pour l'application 1 :

```
application1
vitesse 60
nbmachine 1
pjtotal 1500
epsi2 20
t 20
beta 10
perimetre 150
borninfcoord 1
cv 570
lambda 20
cstcout1 0,06
cstcout2 10
cstcout3 3
cstcout4 -0,05
datedue 3
2 5 0 48
2 5 48 96
1 5 96 120
capacite : 1000 3000
epsi1 ; 3 5 7
quantite : 10 500
prix : 0,05 0,20
```

Figure 3 – *Exemple de fichier de configuration*

Et après le lancement de la commande sur le fichier de configuration "config1.txt", avec l'écriture des résulats dans le fichier "result-1-5-24-1.txt" et la génération de 5 jobs avec un délai de livraison de 24h.

java -jar generateur.jar "config1.txt" "result-1-5-24-1.txt" 5 24

Le résultat est le suivant :

```
vitesse 60
nbmachine 1
nbjob 5
cv 570
capacite 1574
T 1440
pi3PL
6.88 0.38 2.62 0.49 6.92
piM
1.72 0.09 0.66 0.12 1.73
```

Figure 4 – *Exemple de fichier de résultats - 1*

```
hjFIN
0.04 0.04 0.04 0.04 0.04
hjWIP
0.0 0.0 0.0 0.0 0.0
qj
181.0 10.0 69.0 13.0 182.0
pj
317
314
313
320
310
rj
0 0 2880 2880 5760
dj
2880 2880 5760 5760 7200
tj
0.0 53.49 74.41 94.41 36.35 80.53 39.0
53.49 0.0 37.01 64.2 57.57 27.46 21.95
74.41 37.01 0.0 27.29 57.58 32.56 56.89
94.41 64.2 27.29 0.0 68.62 56.44 83.22
36.35 57.57 57.58 68.62 0.0 78.03 56.85
80.53 27.46 32.56 56.44 78.03 0.0 48.08
39.0 21.95 56.89 83.22 56.85 48.08 0.0
cj
6.14 8.06 8.82 9.54 7.45 9.04 7.54
8.06 6.14 7.47 8.45 8.21 7.13 6.93
8.82 7.47 6.14 7.12 8.21 7.31 8.19
9.54 8.45 7.12 6.14 8.61 8.17 9.13
7.45 8.21 8.21 8.61 6.14 8.95 8.19
9.04 7.13 7.31 8.17 8.95 6.14 7.87
7.54 6.93 8.19 9.13 8.19 7.87 6.14
```

Figure 5 – *Exemple de fichier de résultats - 2*

2 Algorithme de résolution

Maintenant le fonctionnement du générateur de données expliqué, il est nécessaire de présenter les différents éléments de l'algorithme de résolution

2.1 Configuration de l'IDE Eclipse

Dans le cas où l'on souhaite modifier le code de l'algorithme de résolution, il faut réaliser quelques configurations pour que cela fonctionne.

L'IDE que j'utilise est Eclipse, ça sera donc sur cet IDE que les explications seront faites.

Comme cela a été détaillé plus tôt, l'algorithme de résolution utilise un solver pour évaluer sa solution. Dans le cadre du développement, j'ai choisi d'utiliser Cplex. Par conséquent, cela nécessite quelques éléments.

Tout d'abord, il faut avoir Cplex d'installer sur son ordinateur. C'est un logiciel payant mais gratuit pour les étudiants. Pour l'installer, il suffit de mettre le dossier contenant les fichiers dans le dossier Programme Files du disque C.

Une fois cette étape réalisée, il faut le configurer dans Eclipse.

Pour cela, il suffit de faire un clic droit sur le projet puis Properties, dans le volet de gauche il faut cliquer sur Java Build Path puis l'onglet Libraries et enfin il faut ajouter le Jar en cliquant sur Add External JAR.

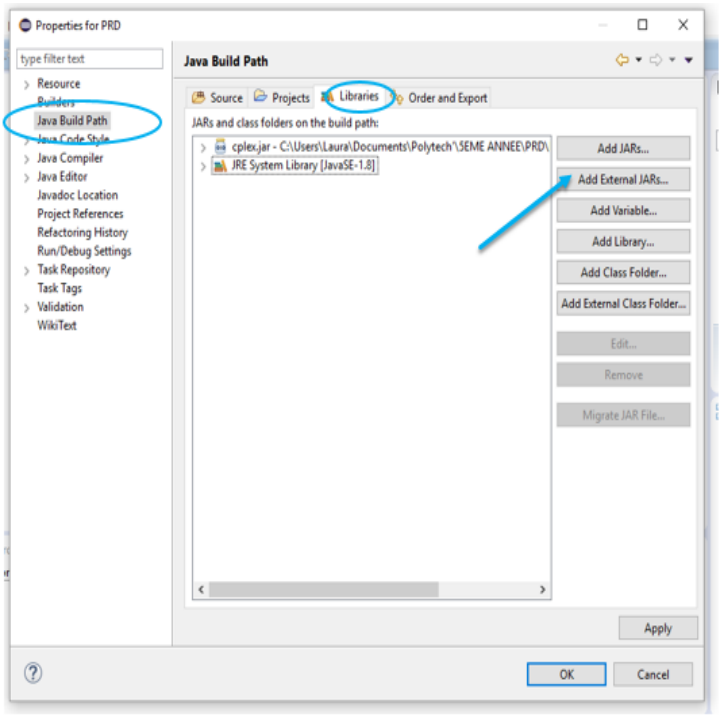


Figure 6 – *Ajout du .jar*

Il suffit ensuite d'aller le chercher dans vos dossiers et de valider l'ajout.

Cependant l'ajout du .jar ne suffit pas : comme Cplex est payant, il faut préciser où se trouve la dll de Cplex. Tout d'abord, récupérez la dll et copiez la dans un dossier lib de votre projet. Enfin pour configurer cette partie, il faut aller dans Run -> Run Configurations, puis dans l'onglet Arguments, on saisit la ligne suivante dans VM Arguments :

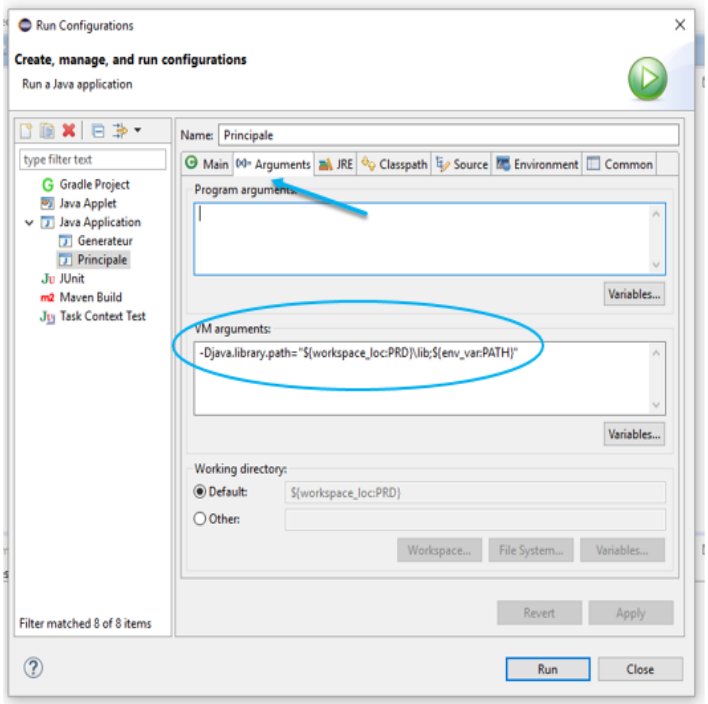


Figure 7 – *Ajout de la dll*

2.2 Fichier d'entrée et de sortie

Pour ce qui est du fichier d'entrée, il utilise le fichier que l'on a en sortie du générateur. Le format a donc été déjà décrit dans la partie précédente.

Quant au fichier de sortie, il y en a deux :

Le premier fichier est celui qui contient les résultats de la résolution de l'instance. Nous avons donc dans l'ordre :

Table 3 – *Eléments du fichier de résultats*

Nom de la variable	Type	Détails
Objectif	Décimal	Valeur objectif de la solution
X	Matrice	Séquence des jobs
Y	Matrice	Composition des véhicules
Sequence	Matrice 1 colonne	Séquence finale des jobs
Date fin de prod	Matrice 1 colonne	Fin de production des jobs
Compo	1 ligne pour chaque véhicule	Composition des véhicules
Trajet	1 ligne pour chaque véhicule	Parcours de chaque véhicule
Date due	Matrice 1 colonne	Date due de chaque véhicule
Date livraison	Matrice 1 colonne	Date de livraison

Pour ce qui est du deuxième fichier, à chaque résolution d'une instance, on écrit une ligne dans le fichier de la méthode correspondante (first improve ou best improve).

Cette ligne utilise le format suivant :

Le nom du fichier, le temps en seconde, la valeur objectif, la valeur Objectif MAJ, le coût du distributeur, le coût du producteur et le nombre de voisins visités

2.3 Lancement du programme et résultat

Comme pour le générateur, j'utilise la console pour lancer l'algorithme de résolution, il est donc aussi nécessaire de générer le .jar également.

Pour cela, il suffit de suivre la méthode utilisée pour le générateur de données. Un message d'avertissement concernant la non exportation des librairies peut s'afficher, ce qui est normal.

Une fois le jar créé, il faut le lancer via la ligne de commande. Pour cela on utilise la ligne de commande suivante :

java -jar nomdujar.jar nomdufichierdedonnée

Ensuite il ne reste plus qu'à attendre la fin de l'exécution du programme.

2.4 Problème rencontré

Lorsqu'on lance le programme en ligne de commande, une erreur en lien avec Cplex peut apparaître. En effet Cplex s'exécute sur un JDK 32 bits et si on l'exécute sur un système d'exploitation 64 bits cela peut poser un problème.

Pour cela, il faut télécharger sur le site d'Oracle un JDK 32 bits et modifier la configuration de Windows pour qu'il prenne ce JDK par défaut.

Pour cela il suffit d'aller dans la configuration java et entrer le chemin du JDK 32 bits, en désactivant celui en 64 bits.

2.5 Exemple

Le fichier d'entrée de ce programme pris pour l'exemple est celui présenté en résultat du générateur plus tôt dans le rapport.

On lance donc la ligne de commande suivante :

java -jar solverbest.jar "data.txt"

Le programme solverbest est lancé (méthode best improve) avec comme fichier de données en entrée "data.txt".

Le premier fichier de résultat donne les informations suivantes :

```
Objectif : 1996.7599935904145
X :
01000
00001
00000
00100
00010
Y :
100
010
001
001
010
Sequence :
0
1
4
3
2
Date de fin de production :
317
631
1574
1261
941

Compo :
[0]
[1, 4]
[3, 2]
Trajet :
[5, 0, 6]
[5, 1, 4, 6]
[5, 2, 3, 6]
```

```
Date due :
2880
2880
5760
5760
7200
Date livraison :
485
1074
1738
1765
1131
Date de fin de production :
317
631
1574
1261
941
```

Figure 8 – *Exemple de fichier de résultats de la solution*

De plus une ligne suivante est écrite dans le fichier resultBest.txt :

```
1-5-24-1.txt 1.5982566 1996.7599935904145 1996.7599935904145 -487.0 1996.7599935904145 90
```

Figure 9 – *Exemple du deuxième fichier de résultats de la solution*

On peut remarquer que la valeur objectif, la valeur objectif MAJ et le coût du producteur sont identiques pour cet exemple. Cela s'explique parce qu'il s'agit d'une instance relativement simple et après la livraison des différentes tâches il n'y a pas eu de retard, le coût du producteur n'est donc pas modifié.

Ci-dessous un exemple où l'on a eu du retard après la livraison, on remarque ici que les résultats sont différents.

```
2_1-100-24-2.txt 9248.470616376 1141495.8884783026 1330008.8884783026 1258204.0 73230.88847830263 16645
```

Figure 10 – *Exemple du deuxième fichier de résultats de la solution - 2*

Gestion de projet

3 Planning

Dans un premier temps, le planning de la première partie qui concerne la recherche est le suivant :

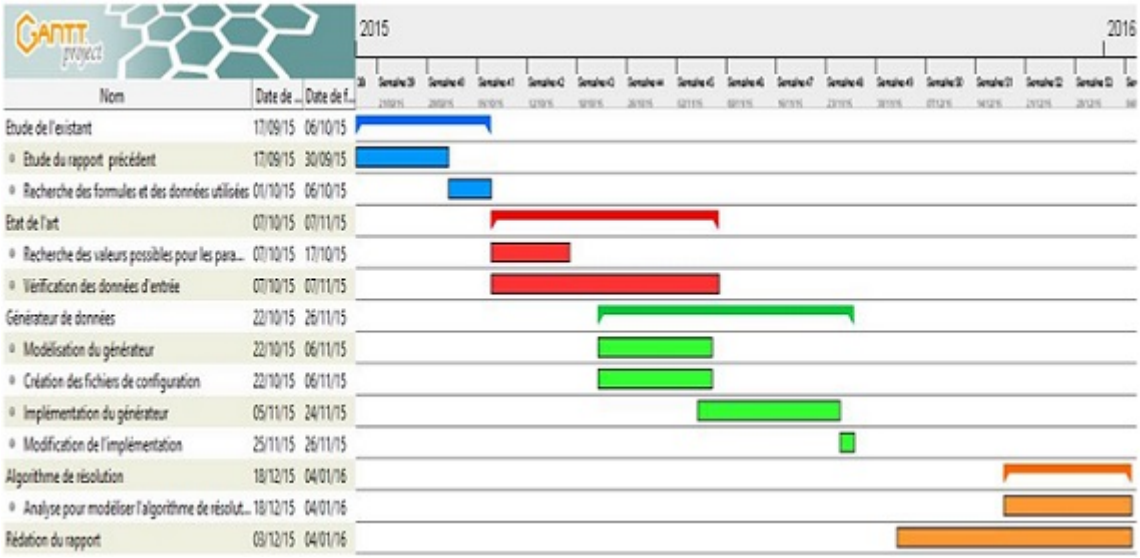


Figure 11 – *Planning du semestre 1*

Pendant le 1er mois du projet, l'étude de l'existant a été réalisée. Cette étude est composée de l'analyse du rapport du projet précédent et l'inventaire des formules et des paramètres que nous allons utiliser.

Pendant le mois qui suit, les recherches ont été effectuées pour trouver les valeurs possibles de chaque constante. Ensuite, il a fallu vérifier et faire valider les différentes données.

Au milieu du mois d'octobre, en parallèle de la partie précédente, le travail sur le générateur de données est réalisé. Cette section comprend sa modélisation, la création des fichiers de configuration, l'implémentation du générateur.

Enfin pour finir, la dernière partie du semestre a été partagée entre la modélisation de l'algorithme de résolution et la rédaction du rapport.

Dans un second temps, le planning prévisionnel concernant la partie développement projet est le suivant :

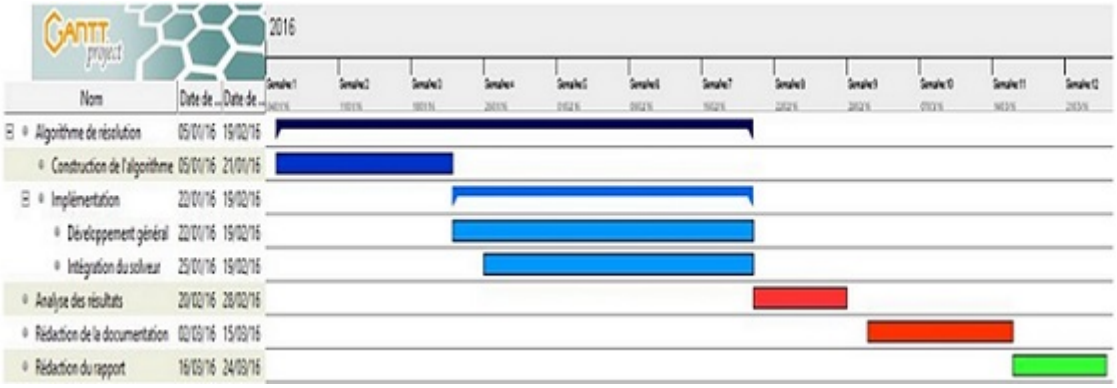


Figure 12 – *Planning prévisionnel du semestre 2*

Il est donc prévu de continuer à modéliser l'algorithme de résolution dans les premières semaines du premier semestre.

Puis pendant un mois, il sera temps d'implémenter cet algorithme, en intégrant le solver choisi.

Une troisième partie représente l'analyse des résultats donnés par l'implémentation et la comparaison avec ceux du modèle mathématique.

Enfin, le reste du temps sera consacré à la rédaction des différentes documentations et du rapport.

Maintenant le deuxième semestre terminé, vous pouvez trouver ci-dessous le planning réel de la partie développement.

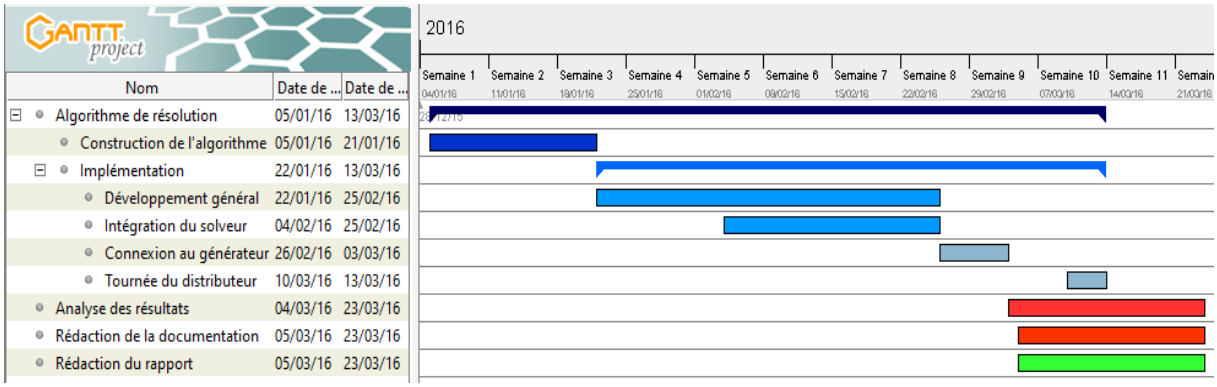


Figure 13 – *Planning réel du semestre 2*

Le planning se divise toujours en 3 parties.

La première concerne la construction de l'algorithme de résolution : comme on peut le voir les dates prévues et celles réelles sont identiques.

Ensuite nous avons débuté la partie développement, elle a duré plus longtemps que prévu initialement, mais il y a plusieurs explications à cela. Tout d'abord, des tâches ont été rajoutées, comme la connexion au générateur (je l'avais incluse dans le développement général alors qu'elle représente une étape en elle-même). Quant à la tournée de distributeur, comme expliqué précédemment dans le rapport, cette tâche a été rajoutée afin de compléter mon travail, mais elle n'était pas prévue au départ. Cependant comme j'avais bien avancé dans le développement, on a pu réaliser cette partie en plus.

Pour les tests et l'analyse des résultats, ces tâches ont commencé plus tard et ont duré plus longtemps. En effet, à la fin du premier semestre, je n'avais qu'une idée vague de l'algorithme à implémenter et donc aucune idée de sa complexité et donc de son temps d'exécution. Finalement, les tests de plusieurs instances ont demandé plusieurs jours d'exécution. Ainsi, dans le cas où il y avait des modifications à apporter, cela a demandé encore quelques jours pour réaliser les nouveaux tests.

Enfin la dernière étape était la rédaction du rapport, pour cela la date réelle est presque identique à celle prévue.

4 Livrables

Le projet possède deux livrables. Le premier concerne le générateur de données avec ses fichiers de configuration. Quant au deuxième, il s'agit de l'implémentation d'un algorithme de solution concernant le problème.

Ces derniers sont attendus à la fin du projet.

5 Méthodologies et outils utilisés

Avec mon encadrant, nous avons mis en place des réunions chaque semaine afin de faire le point régulièrement. Cela nous permettait de voir l'avancement de mon travail et donc de le valider.

Pour générer les différents diagrammes de Gantt, j'ai utilisé le logiciel Gantt Project.

De plus pour réaliser mon rapport, j'ai utilisé Sharelatex, qui est un outil en ligne. Cette fonctionnalité m'a permis de garder une sauvegarde en ligne en plus de celle en local.

Conclusion

Ce projet m'a permis de travailler sur un projet de recherche, lui-même issu d'un ancien projet. Il concerne le travail sur le générateur de données et sur un algorithme de résolution correspondant au modèle mathématique du projet précédent.

Pendant le premier semestre de ce projet, j'ai réalisé l'état de l'art et l'analyse de l'ancien projet, puis j'ai effectué des recherches sur les valeurs des constantes nécessaires pour le générateur de données. Ensuite, j'ai réalisé la modélisation et l'implémentation du générateur. Pour finir, j'ai commencé la modélisation de l'algorithme de résolution.

Durant le seconde semestre, j'ai fini la modélisation de l'algorithme de résolution puis j'ai réalisé son implémentation. Ensuite j'ai connecté mon générateur de données à cette nouvelle partie et aussi développé une fonction supplémentaire pour déterminer la tournée du distributeur. Enfin j'ai lancé mes tests et j'ai analysé les résultats obtenus.

J'ai trouvé ce projet très intéressant. Il m'a permis de développer mes compétences dans la recherche bibliographique ainsi que dans l'étude d'un projet existant (modèle mathématique). De plus, j'ai du assimiler le contexte du sujet et l'analyser. J'ai également amélioré mes compétences de développement et de modélisation pour le générateur de données et en particulier pour l'algorithme de résolution, car il a demandé plus de réflexion et de recherches notamment avec le modèle mathématique. Enfin j'ai pu apprendre à lancer des tests sur un programme avec plusieurs fichiers de données en entrée (fichier .bat). J'ai trouvé cette expérience très gratifiante et instructive car j'ai pu réaliser le projet de A à Z, en participant à la modélisation jusqu'aux tests finaux sur le programme développé.

D'un point de vue organisationnel, j'ai beaucoup apprécié le fait que mon encadrant se soit rendu disponible pour que l'on fasse le point sur mon projet chaque semaine. En effet, cela m'a permis de veiller à rester dans le sujet et de faire valider chaque étape de mon travail.

Il pourrait être maintenant intéressant de réaliser le même travail dans le cas où le distributeur domine pour voir les différences dans les résultats et enfin on pourrait étudier le cas où le producteur et le distributeur coopèrent afin de comparer les résultats aux deux autres situations.

Comptes rendus hebdomadaires

Compte rendu n°1 du 17/09/2015

Cette semaine, j'ai lu : - Sonja Rohmer. A two-agent model for production and outbound distribution scheduling. Rapport, pp. 1-44, 2 juillet 2015 - Sonja Rohmer, Jean-Charles Billaut. Production and outbound distribution scheduling : a two-agent approach. 6th IESM Conference, pp. 1-10, octobre 2015

La semaine prochaine, j'ai prévu de fixer un rendez-vous avec vous, afin de discuter du sujet et de connaitre les caractéristiques et les objectifs du projet.

Compte rendu n°2 du 24/09/2015

Cette semaine, j'ai une réunion avec M. Billaut pour discuter du sujet et des tâches à réaliser pendant ce projet. Ensuite j'ai approfondi l'étude de : Sonja Rohmer. A two-agent model for production and outbound distribution scheduling. Rapport, pp. 1-44, 2 juillet 2015

La semaine prochaine, il est convenu que je réalise une présentation à l'oral de ce que j'ai compris du sujet, mercredi 30 septembre. Après cela, je commencerai les recherches pour la génération de données proche de la réalité.

Compte rendu n°3 du 01/10/2015

Cette semaine, j'ai une réunion avec M. Billaut pour lui présenter ce que j'avais réalisé la semaine précédente. Ensuite j'ai étudié plusieurs documents afin de travailler sur la génération de données réelles : - Comité National Routier. http ://www.cnr.fr/Outils-simulation2/Simulateurs. Last access July 2015. - Gacogne, V. (2003). Impact des coûts de transport sur les systèmes logistiques par une modélisation en dynamique des systèmes. PhD thesis, Ecole Nationale des Ponts et Chaussées, 2003. - Yamna Salhi. La logistique du transport au sein de la filière agroalimentaire, cas de l'entreprise Trèfle produits laitiers. Thèse Master of Science du CIHEAM – IAMM n°IAMM n°108 – 2010 - Comité National Routier. Les coûts du TRM – Quelles perspectives pour 2015 ?. Novembre 2014

La semaine prochaine, il est convenu que je présente mon avancement, mercredi 7 octobre. Après cela, je continuerai les recherches pour la génération de données proche de la réalité

Compte rendu n°4 du 08/10/2015

Cette semaine, j'ai eu une réunion avec M. Billaut pour lui présenter ce que j'avais réalisé la semaine précédente. Ensuite j'ai réalisé des recherches sur la génération de données pour le cas des produits

« pas chers", j'ai passé beaucoup de temps sur cette partie car il était difficile de trouver des chiffres sur la production de lait ou de papier. Enfin j'ai répertorié toutes les variables qui seront entrées dans l'algorithme et j'ai vérifié si l'on avait toutes les informations nécessaires pour les générer. J'ai constaté que l'on avait la plupart des données sauf quelques-unes pour lesquelles j'ai des questions à poser à M. Billaut. Pour les recherches j'ai utilisé internet et un livre : GRET. Transformer les produits laitiers frais à la ferme. Livre, pp. 216-220, 2010.

La semaine prochaine, le rendez-vous avec M. Billaut est le jeudi 15 octobre à 15h. Avant et après cet entretien je continuerai mes recherches pour proposer une solution au sujet des données manquantes et je commencerai à créer les différents fichiers de configuration.

Compte rendu n°5 du 15/10/2015

Cette semaine, j'ai été absente (justifiée) le mercredi 14 octobre. Le jeudi, j'ai eu une réunion avec M. Billaut où l'on a complété les informations nécessaires à la génération de données. Puis j'ai lu : - Thibault DREVON. Prise en compte des reliquats dans la fabrication de chimiothérapies. Rapport de PFE, 13 mai 2014

La semaine prochaine, je finirai de compléter les informations pour la génération de données et je vais créer les différents fichiers de configuration ainsi que le générateur de données. Le prochain rendez-vous avec M. Billaut est fixé le 4 novembre 2015.

Compte rendu n°6 du 22/10/2015

Cette semaine, j'ai travaillé sur la génération de données. J'ai relu : - Thibault DREVON. Prise en compte des reliquats dans la fabrication de chimiothérapies. Rapport de PFE, 13 mai 2014

Puis j'ai créé le fichier de configuration pour l'application 1, et commencé celui de l'application 2 ainsi que le générateur. Cependant j'attends la réponse aux questions que j'ai envoyées par email à Jean Charles Billaut afin de compléter les valeurs manquantes du fichier de configuration. A la fin de la séance du jeudi, nous avons eu une petite réunion pour faire le point sur notre projet PR&D avec Nicolas Monmarché et Sébastien Aupetit.

Pendant la semaine de pause, je compte terminer le générateur de données. La semaine prochaine, le rendez-vous avec M. Billaut est fixé le 4 novembre 2015. Je réaliserai les éventuelles modifications abordées lors de la réunion ainsi que la prochaine tâche du projet.

Compte rendu n°7 du 05/11/2015

Cette semaine, j'ai travaillé sur la génération de données. J'ai eu une petite réunion avec M. Billaut pour répondre à mes questions et lui montrer l'avancement de mon travail. J'ai passé du temps à réfléchir à comment organiser le générateur puis j'ai commencé l'implémentation pour l'application 1.

La semaine prochaine, je continuerai l'implémentation du générateur. Le prochain rendez-vous avec M. Billaut est fixé le 18 novembre 2015.

Compte rendu n°8 du 12/11/2015

Cette semaine, j'ai commencé l'élaboration du rapport. J'ai été absente aux séances de PR&D de cette semaine car j'étais en déplacement sur Nantes pour passer deux entretiens de stage.

La semaine prochaine, je continuerai l'implémentation du générateur. Le prochain rendez-vous avec M. Billaut est fixé le 18 novembre 2015.

Compte rendu n°9 du 19/11/2015

Cette semaine, j'ai travaillé sur la génération de données. J'ai terminé l'implémentation du générateur pour les applications 1 et 2. Cependant j'attends le rendez-vous avec M. Billaut la semaine prochaine, pour valider les paramètres écrits dans le fichier de configuration.

La semaine prochaine, je réaliserai la documentation concernant le format du fichier entrant et sortant du générateur. Le prochain rendez-vous avec M. Billaut est fixé le 25 novembre 2015 à 9h.

Compte rendu n°10 du 26/11/15

J'ai eu un rendez-vous avec M. Billaut le 25 novembre, je lui ai montré le fonctionnement et le résultat de mon générateur de données et il a validé mon travail. Maintenant je travaille sur mon rapport de projet. Le jeudi 26 novembre il y avait le forum d'entreprise.

La semaine prochaine, je continuerai la rédaction de mon rapport. Le prochain rendez-vous avec M. Billaut est fixé le 9 décembre 2015 à 9h.

Compte rendu n°11 du 03/12/15

Le mercredi 2 décembre, j'ai participé à la présentation de la spécialité informatique au Peip avec M. Raveaux, et jeudi 3 décembre à partir de 16h c'était la nuit de l'info, je n'ai donc pas pu travailler mon PR&D pendant ce temps-là. Dans les créneaux restants, j'ai continué la rédaction de mon rapport (la partie cahier des charges).

La semaine prochaine, je continuerai la rédaction de mon rapport et je vais faire valider le plan par mon encadrant. Le prochain rendez-vous avec M. Billaut est fixé le 9 décembre 2015 à 9h.

Compte rendu n°12 du 10/12/15

J'ai eu un rendez-vous avec M. Billaut le 9 décembre, je lui ai montré le plan que j'avais prévu d'utiliser dans mon rapport, il l'a validé. Dans les créneaux restants j'ai continué la rédaction de mon rapport (fin de la partie cahier des charges, début de l'étude).

La semaine prochaine, je continuerai la rédaction de mon rapport. Le prochain rendez-vous avec M. Billaut est fixé le 6 janvier 2015 à 9h.

Compte rendu n°13 du 17/12/15

J'ai continué la rédaction de mon rapport (fin de l'état de l'art et modélisation).

Pendant les vacances, je continuerai la rédaction de mon rapport. Le prochain rendez-vous avec M. Billaut est fixé le 6 janvier 2015 à 9h.

Compte rendu n°14 du 07/01/16

Pendant la réunion avec M. Billaut, nous avons discuté de ma soutenance prévue cette semaine ainsi que de mon rapport. J'ai travaillé sur un début de modélisation pour l'algorithme de résolution. Ensuite pendant le temps restant, j'ai préparé ma soutenance, en réalisant les slides et en m'entraînant.

La semaine prochaine, il est prévu de réaliser la modélisation de l'algorithme de résolution. Le prochain rendez-vous avec M. Billaut est fixé le 12 janvier 2016 à 9h.

Compte rendu n°15 du 14/01/16

J'ai eu un rendez-vous avec Jean Charles Billaut le 12 janvier, nous avons pendant ce créneau récapitulé le système et nous avons travaillé sur la modélisation de l'algorithme de résolution dans le cas où le producteur domine. Un premier modèle mathématique a été élaboré. Dans les créneaux restant, j'ai réalisé et mis au propre le modèle et les différents éléments de modélisation vu lors du rendez-vous.

La semaine prochaine, je continuerai de travailler sur la modélisation. Le prochain rendez-vous avec M. Billaut est fixé le 19 janvier 2016 à 14h.

Compte rendu n°16 du 21/01/16

J'ai eu un rendez-vous avec Jean Charles Billaut le 19 janvier. Après avoir réfléchi, nous avons réalisé un deuxième modèle mathématique pour le problème avec de meilleures variables. Ce modèle a été testé sur une machine. Nous avons aussi décidé le fonctionnement général de l'algorithme. Dans les créneaux restants, j'ai mis au propre le nouveau modèle mathématique. Puis j'ai implémenté la première partie de l'algorithme qui concernait le choix du nombre de véhicules et l'implémentation du modèle sous Cplex.

La semaine prochaine, je continuerai de travailler sur l'implémentation. Le prochain rendez-vous avec M. Billaut est fixé le 3 février 2016 à 9h.

Compte rendu n°17 du 28/01/16

J'ai continué à travailler et j'ai terminé l'implémentation de la première partie de l'algorithme.

La semaine prochaine, je vais travailler sur la suite de l'algorithme. Le prochain rendez-vous avec M. Billaut est fixé le 3 février 2016 à 9h.

Compte rendu n°18 du 04/02/16

J'ai eu un rendez-vous avec Jean Charles Billaut le 4 février. Nous avons fait le point sur le travail que j'avais réalisé, puis nous avons modélisé la deuxième partie de l'algorithme concernant l'ordre de la séquence. Dans les créneaux restants, j'ai mis au propre les méthodes vues lors de la réunion avec M. Billaut. Puis j'ai réalisé les modifications demandées et j'ai commencé à réfléchir comment réaliser la seconde partie.

La semaine prochaine, je continuerai de travailler sur l'implémentation de la seconde partie. Le prochain rendez-vous avec M. Billaut est fixé le 2 mars 2016 à 9h.

Compte rendu n°19 du 11/02/16

J'ai réalisé les différents algorithmes nécessaires à la seconde partie afin de visualiser comment gérer les différentes contraintes. Il a aussi été nécessaire de réfléchir à la manière de regrouper les fonctions pour qu'elles fonctionnent ensemble.

La semaine prochaine, je continuerai de travailler sur l'implémentation de la seconde partie. Le prochain rendez-vous avec M. Billaut est fixé le 2 mars 2016 à 9h.

Compte rendu n°20 du 25/02/16

Pendant ces séances j'ai commencé et terminé l'implémentation de la seconde partie, puis j'ai réalisé 2 tests avec des instances simples.

La semaine prochaine, je vais réaliser la partie concernant les tests et continuer la rédaction de mon rapport. Le prochain rendez-vous avec M. Billaut est fixé le 2 mars 2016 à 9h.

Compte rendu n°21 du 03/03/16

J'ai eu un rendez-vous avec Jean Charles Billaut le 2 mars, nous avons fait le point sur le travail que j'avais réalisé et celui restant à faire. Dans les créneaux restants, j'ai réalisé les modifications demandées sur les méthodes de voisinage. Puis j'ai connecté mon algorithme de résolution à mon générateur de données et j'ai développé la génération des fichiers résultats.

La semaine prochaine, je vais lancer les tests de mon algorithme de résolution. Le prochain rendez-vous avec M. Billaut est fixé le 16 mars 2016 à 9h.

Compte rendu n°22 du 10/03/16

Pendant ces séances, j'ai lancé les tests et réaliser les modifications en fonction des premiers résultats. Puis j'ai développé l'algorithme concernant la tournée du distributeur afin de calculer les coûts réels.

La semaine prochaine, je vais faire valider mon travail par mon encadrant et proposer mon plan de rapport. Le prochain rendez-vous avec M. Billaut est fixé le 16 mars 2016 à 9h.

Compte rendu n°23 du 17/03/16

Pendant ces séances, j'ai réalisé les modifications dans les formules et complété le fichier de sortie de mon programme de résolution, après la réunion avec M. Billaut. Puis j'ai lancé de nouveau les tests. En parallèle j'ai continué la rédaction de mon rapport.

La semaine prochaine, je vais faire valider mon travail par mon encadrant et proposer mon plan de soutenance. Le prochain rendez-vous avec M. Billaut est fixé le 23 mars 2016 à 9h.

Webographie

[WWW1] Comité National Routier. *Simulateur de coût de revient*. 2015. URL : http://www.cnr.fr/Outils-simulation2/Simulateurs.

Bibliographie

[1] Thibault DREVON. *Prise en compte des reliquats dans la fabrication de chimiothérapies*. Rapport de PFE. 2014.

[2] V. Gacogne. « Impact des couts de transport sur les systèmes logistiques par une modélisation en dynamique des systèmes. » Thèse de doct. PhD thesis, Ecole Nationale des Ponts et Chaussées, 2003.

[3] GRET. « Transformer les produits laitiers frais à la ferme ». In : 2010.

[4] Sonja Rohmer. *A two-agent model for production and outbound distribution scheduling*. Rapport. 2015.

[5] Comité National Routier. *Les coûts du TRM – Quelles perspectives pour 2015 ?* 2014.

[6] Yamna Salhi. « La logistique du transport au sein de la filière agroalimentaire, cas de l'entreprise Trèfle produits laitiers ». Thèse Master of Science du CIHEAM, 2010.

[7] Jean-Charles Billaut Sonja Rohmer. « Production and outbound distribution scheduling : a two-agent approach ». In : *6th IESM Conference* (2015).

Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur" :

Laura Bellego

Encadrement : Jean Charles Billaut

Objectifs

Le but est de planifier les différentes tâches de production et de distribution de produits pour minimiser les coûts.
Les deux objectifs majeurs sont :

- Générer des données proches du réel.
- Réaliser un programme pour trouver une planification des tâches de bonne qualité

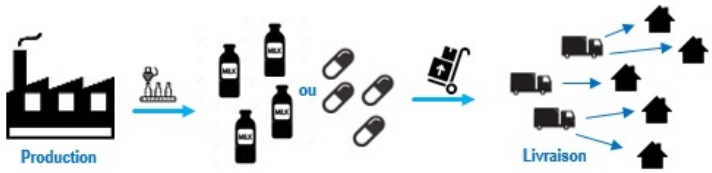
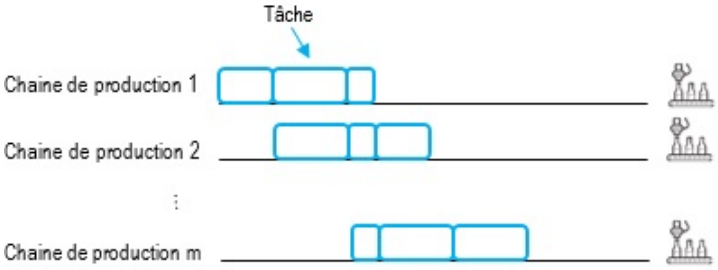


Schéma général

Mise en œuvre

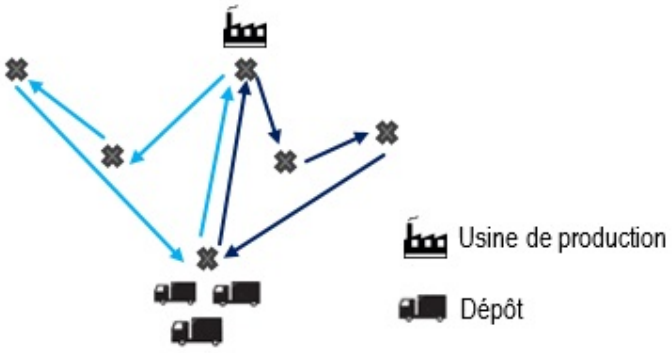
- Etudier le travail réalisé précédemment.
- Rechercher les formules et les variables nécessaires
- Rechercher les valeurs possibles des constantes
- Développer le générateur
- Modéliser et développer l'algorithme de résolution



Usine de production

Résultats attendus

- Un générateur de données proches du réel
- Un programme qui planifie les tâches en minimisant les coûts tout en respectant les paramètres et les contraintes du problème



Distribution des produits

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
www.polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur"

Résumé

Le projet Heuristiques pour un scénario de collaboration entre un "producteur" et un "distributeur" est la suite d'un ancien projet qui a donné lieu à un rapport et un article.

Nous nous situons dans un contexte producteur-distributeur. Chacun a des tâches à réaliser, et il faut les planifier. Pour le faire, plusieurs contraintes et constantes sont à prendre en compte. Un modèle mathématique a déjà été modélisé.

Le projet possède deux objectifs principaux. Le premier concerne la génération de données : il faut générer des données proches de la réalité afin de tester l'algorithme. Dans une seconde partie, il y a la modélisation et l'implémentation de l'algorithme correspondant au modèle mathématique du projet précédent.

Mots-clés

PRD, Ordonnancement, Producteur Distributeur, Algorithme de résolution, Production, Distribution, Solveur

Abstract

The project about an Heuristic for collaboration between Production and outbound distribution is based on an old project which resulted in an article and a report.

The context is a manufacturer and a provider. Each part has several tasks to be done. To do this, many constrains have to be taken in the project. Mathematical model has already been developped.

The project have two main objectives. The first is data generation, it has to generate data close to the reality to test the algorithm. Then, there is modelling and implemetation about the algorithm which corresponds to the mathematical model of previous project.

Keywords

PRD, Scheduling, Manufacturer Provider, Resolution Algorithm, Distribution Production, Solver

Tuteurs académiques
Jean Charles Billaut

Étudiants
Laura Bellego (DI5)