

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

**Projet Recherche & Développement**  
**2018-2019**

**Ordonnancement de production dans  
un problème intégrant la distribution**

**Tuteur académique**  
**Hugo CHEVROTON**

**Étudiant**  
**Jean HILLERITEAU (DI5)**

5 avril 2019



# Liste des intervenants

Nom	Email	Qualité
Jean HILLERITEAU	<a href="mailto:jean.hilleriteau@etu.univ-tours.fr">jean.hilleriteau@etu.univ-tours.fr</a>	Étudiant DI5
Hugo CHEVROTON	<a href="mailto:hugo.chevroton@etu.univ-tours.fr">hugo.chevroton@etu.univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Jean Hilleriteau susnommé l'auteur.

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

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

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

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

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

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

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

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



## Pour citer ce document

Jean Hilleriteau, *Ordonnancement de production dans un problème intégrant la distribution*,  
Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais  
de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Hilleriteau, Jean},
  title={Ordonnancement de production dans un problème intégrant la distribution},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Liste des Algorithmes	iv
<b>1 Introduction</b>	<b>1</b>
1 Acteurs, enjeux et contexte .....	1
1.1 Présentation du sujet général .....	1
1.2 Acteurs .....	2
2 Objectifs .....	2
3 Hypothèses .....	2
4 Bases méthodologiques .....	2
4.1 outils de gestion de version .....	2
4.2 gestion de projet .....	2
4.3 déploiement continu.....	3
<b>2 Description générale</b>	<b>4</b>
1 Environnement du projet .....	4
2 Caractéristiques des utilisateurs .....	4
3 Fonctionnalités du système .....	4
4 Structure générale du système .....	6
4.1 partie commune .....	6

4.2	Méthode exacte .....	6
4.3	Recherche local .....	6
<b>3</b>	<b>Etat de l'art</b> .....	<b>7</b>
1	Problème de flow-shop avec permutation .....	7
2	Problème du voyageur de commerce .....	7
3	Recherche locale .....	8
<b>4</b>	<b>analyse et conception</b> .....	<b>9</b>
1	méthode exacte .....	9
1.1	Paramètres .....	9
1.2	Variables.....	10
1.3	Objectif.....	10
1.4	Contraintes.....	10
2	méthode heuristique.....	11
2.1	Opérateur de voisinage.....	12
2.2	Exploration du voisinage .....	12
2.3	Stratégie de sortie des minimum locaux .....	12
2.4	Condition d'arrêt .....	12
3	Modélisation logicielle .....	12
3.1	Bibliothèque.....	13
3.1.1	Structures de données .....	13
3.1.2	Services .....	14
<b>5</b>	<b>Mise en œuvre</b> .....	<b>15</b>
1	Outils .....	15
1.1	Git .....	15
1.2	Github .....	15
1.3	Visual studio .....	15
1.4	Tex Live .....	15
1.5	Visual studio code.....	16
1.6	Doxygen .....	16
2	Implémentations.....	16
2.1	Fonctionnalités communes.....	16
2.2	Utilisation du modèle mathématique .....	16
2.3	Recherche locale .....	16
2.3.1	Opérateurs de voisinage .....	17
2.3.2	Sélection du voisin.....	17
2.4	Limites .....	17
3	Performance .....	17

3.1	Analyse des résultats .....	17
3.2	Opérateur de voisinage.....	17
3.3	Sélection du voisin .....	18
<b>6</b>	<b>Bilan et conclusion</b>	<b>19</b>
1	Bilan semestre 9.....	19
2	Bilan semestre 10.....	19
<b>7</b>	<b>Bibliographie</b>	<b>20</b>
	<b>Annexes</b>	<b>21</b>
<b>A</b>	<b>modélisation du problème global</b>	<b>22</b>
1	Paramètres .....	22
2	Variables .....	22
3	Objectif .....	24
4	Contraintes .....	24
<b>B</b>	<b>Sous-problème initial</b>	<b>26</b>
1	Paramètres .....	26
2	Variables .....	27
3	Objectif .....	27
4	Contraintes .....	27
<b>C</b>	<b>Spécifications fonctionnelles</b>	<b>29</b>
1	Lecture des paramètre.....	29
2	Lecture de l'instance.....	29
3	Résolution de l'instance .....	29
4	Ecriture des résultats.....	29
<b>D</b>	<b>Planification</b>	<b>31</b>
<b>E</b>	<b>Installation</b>	<b>33</b>
0.1	Installation des livrables.....	33
0.1.1	Resolution d'instance .....	33
0.1.2	Bibliothèque de fonction commune .....	33
0.1.3	Visualisation des solutions .....	33
<b>F</b>	<b>Manuel d'utilisation</b>	<b>34</b>
1	recherche locale .....	34
2	bibliothèque.....	34
3	visionneuse de solution .....	34



# Liste des Algorithmes

1	Algorithme de recherche locale .....	8
---	--------------------------------------	---



# 1

## Introduction

### 1 Acteurs, enjeux et contexte

Ce projet reprend le travail de mon encadrant, Hugo Chevroton, qui travaille sur l'intégration de problèmes de routing dans des problèmes d'ateliers en flow-shop avec permutation. Le problème de flow-shop avec permutation implique que tous les produits passent par toutes les machines, dans un même ordre. L'objectif de cette approche est de fournir de meilleures solutions. Dans ce projet, je travaille sur une sous-partie de ce problème pour laquelle les produits sont envoyés en livraison par lots déjà définis.

#### 1.1 Présentation du sujet général

Dans une entreprise qui réalise des commandes et les fait livrer chez ces clients, il faut déterminer dans quel ordre réaliser les commandes et planifier les itinéraires des livreurs. Ces deux problèmes sont généralement résolus séparément. Le travail de mon encadrant porte sur l'intégration de ces problèmes en un seul, de façon à trouver de meilleures solutions.

Les commandes sont réalisées dans un atelier, elles passent toutes successivement par les mêmes étapes de production, dans le même ordre. Dans le cadre de ce projet, on nomme les commandes jobs et les étapes de productions des machines. Le travail que doit réaliser une machine sur un job est appelé une tâche.

On considère qu'une machine ne peut travailler que sur un job à la fois et qu'un job ne peut être que sur une machine à la fois. La durée des tâches peut varier selon les machines et selon les jobs.

Lorsqu'un job est en attente entre deux machines, il engendre des coûts d'inventaires. On distingue les coûts d'inventaire des jobs pendant la production et les coûts d'inventaire des jobs terminés en attente de livraison.

Les jobs sont répartis en lots, une fois que tous les jobs d'un lot sont terminés, ils partent en livraison. Chaque lot est livré par un même véhicule en une tournée.

Pour résoudre ces problèmes, il y a quatre degrés de liberté que l'on peut utiliser :

- Ordre de réalisation des jobs.
- Mise en attente de la production pour réduire les coûts d'inventaires.

- Constitution des lots.
- Ordre de distribution des jobs pour chaque lot.

La modélisation du problème général est présentée dans la partie [Annexe A](#).

## 1.2 Acteurs

Ce projet est un sujet de recherche, mon encadrant est le seul client.

## 2 Objectifs

Mon encadrant a déjà résolu le sous-problème où l'ordre de production des jobs et la constitution des lots sont fixés. L'objectif de ce projet est d'ajouter la possibilité de changer l'ordre des jobs.

Il faut donc résoudre simultanément les problèmes de routing et d'atelier pour des problèmes où les lots sont fixés à l'avance.

Je vais réaliser un programme en C++ pour résoudre le problème, de façon exacte pour commencer puis avec une méthode heuristique.

## 3 Hypothèses

Dans ce projet, on considère que l'atelier de production est constitué de plusieurs machines et que tous les jobs doivent passer successivement sur chaque machine dans le même ordre.

Les durées des tâches sont connues à l'avance. Les durées et les coûts des trajets entre les lieux de livraisons sont connus à l'avance et fixes (le trajet ne coûte pas plus si le véhicule est remplie que s'il est vide).

Dans un premier temps, je vais utiliser un solveur pour résoudre le problème, s'il ne permet pas de le résoudre suffisamment rapidement, je vais faire un programme en C++ qui intégrera des heuristiques.

## 4 Bases méthodologiques

### 4.1 outils de gestion de version

L'ensemble du projet est hébergé sur Github ([https://github.com/JHilleri/projet\\_recherche\\_developpement](https://github.com/JHilleri/projet_recherche_developpement)).

### 4.2 gestion de projet

La gestion de projet suit une méthode agile et s'appuie sur les outils de gestion de projet fournis par Github.

Chaque semaine, les tâches à réaliser sont définies avec mon encadrant. Ces tâches sont ajoutées sur github dans l'onglet Issues et leurs avancements est suivi sur un tableau kanban.

Lorsque je termine une tâche, j'ouvre un "pull request" pour soumettre mon travail à mon encadrant. Une fois mon travail validé, j'intègre mes modifications à la branche principale du projet.

### 4.3 déploiement continu

Sur Github, seul les code sources du programme de résolution du problème et le rapport sont disponibles. Les livrables sont disponibles à l'adresse <http://prd.jhilleri.ovh>. Sur ce site, on trouve les livrables pour les différentes branches du projet. J'utilise l'outil CircleCI (<https://circleci.com>) pour l'intégration continue et le déploiement automatique des livrables. Le rapport est généré avec l'image docker aergus/latex qui fournit une installation complète de texlive.

# 2

## Description générale

### 1 Environnement du projet

Pour ce projet, je me base sur le code de mon encadrant, qui résout le sous-problème où l'ordre de réalisation des jobs et la constitution des lots sont fixés. Le code est écrit en C++ avec Visual studio 2017 et utilise la bibliothèque Cplex.

Le projet existant utilise des fonctionnalités de Visual c++ et n'est pas compatible avec le C++ standard, une partie de mon travail consiste à l'adapter.

### 2 Caractéristiques des utilisateurs

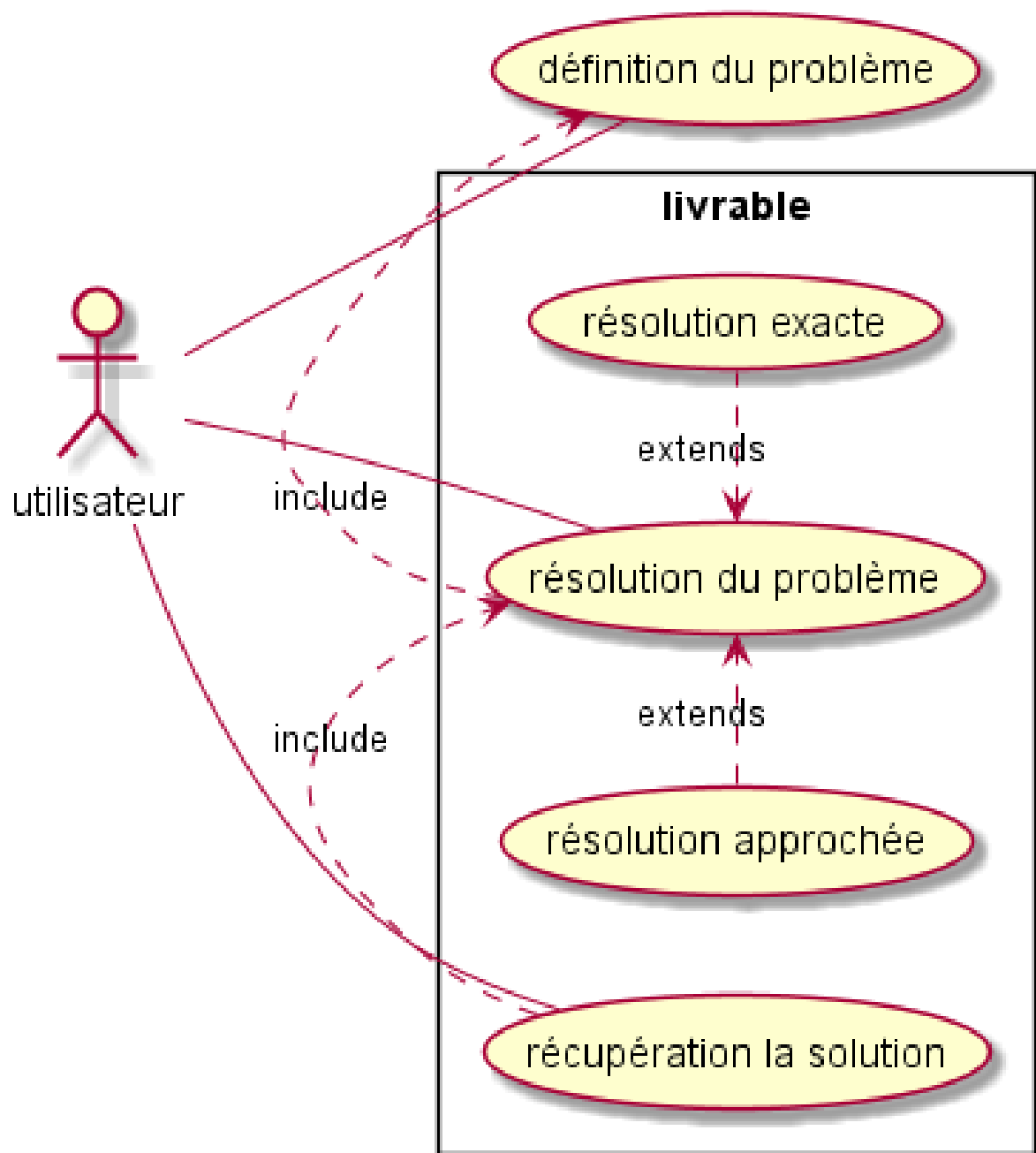
Il s'agit d'un projet de recherche, les utilisateurs seront des chercheurs souhaitant tester les algorithmes développés dans ce projet.

Pour utiliser les méthodes de résolution d'instance, l'utilisateur doit pouvoir utiliser une invite de commande.

Pour ajouter de nouvelles méthodes de résolutions, l'utilisateur doit disposé de Visual Studio 2017 et connaître le C++.

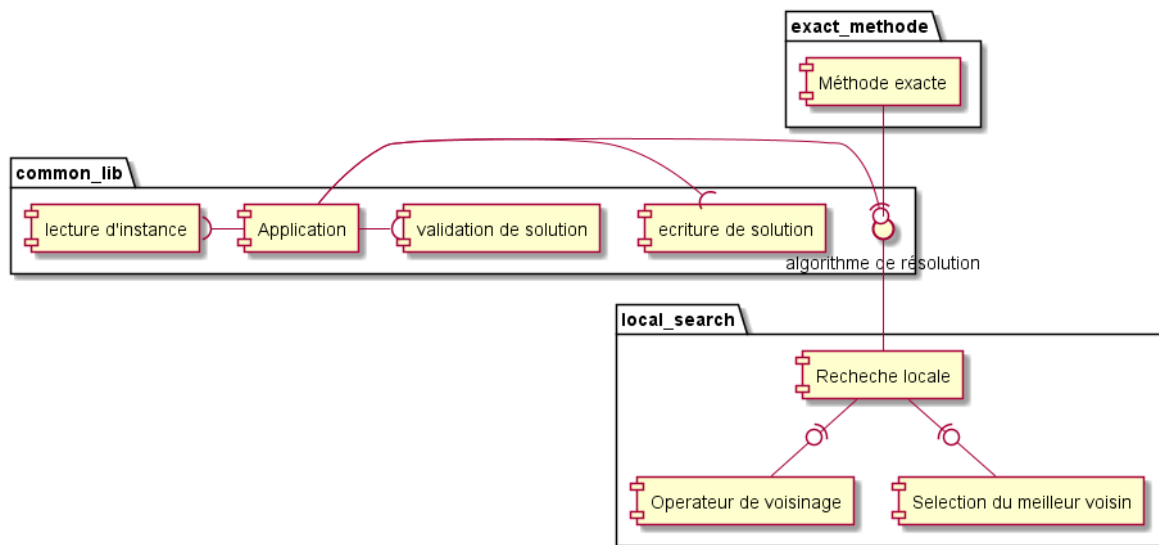
### 3 Fonctionnalités du système

Les applications développées dans ce projet ont pour objectif de comparer plusieurs méthodes de résolution du problème défini précédemment. Pour cela, deux méthodes de résolutions sont testées, une méthode exacte utilisant un solveur et une recherche local. Pour permettre de comparer les méthodes, elles utilisent la même structure de données pour les instances et les solutions.



Les implémentations des méthodes donnent deux applications distinctes, utilisant une même bibliothèque qui définit les structures de données du problème. Les exécutables s'exécutent en console et prennent en paramètre le chemin vers le fichier d'instance et le chemin vers le fichier où écrire les résultats. Les résultats sont écrits en JSON et incluent des données telles que la durée de la résolution. L'utilisation du format JSON permet d'exploiter facilement les résultats avec de nombreux outils.

## 4 Structure générale du système



### 4.1 partie commune

Le code pour la lecture d'instance et les validations et écritures de solutions sont regroupés dans la bibliothèque `common_lib`. Cette bibliothèque définit également un composant qui peut appeler et mesurer une méthode de résolution.

La bibliothèque est liée de façon statique, il n'y a pas de dll à fournir.

### 4.2 Méthode exacte

L'exécutable `exact_methode` donne une implémentation d'une méthode de résolution utilisant le solveur CPLEX et un modèle mathématique.

### 4.3 Recherche local

L'exécutable `local_search` implémente une méthode de résolution basée sur une recherche locale. Les opérateurs de la recherche local peuvent avoir plusieurs implémentations.

# 3

## Etat de l'art

### 1 Problème de flow-shop avec permutation

Le problème de flow-shop avec permutation est un problème où l'on cherche à trouver l'ordre dans lequel réaliser des jobs. Dans ce problème, chaque job doit passer par différentes machines, toujours dans le même ordre.

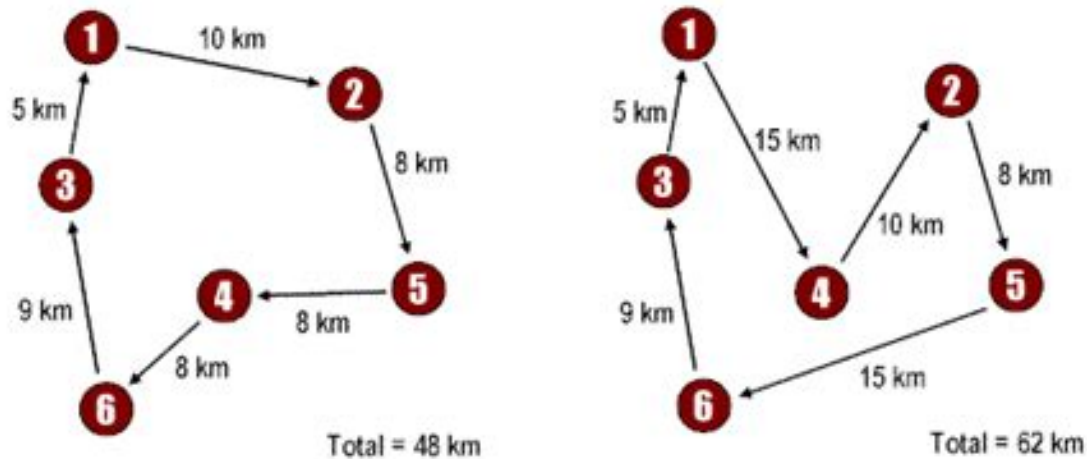
Pour ce problème, les solutions sont généralement représentées sous la forme de diagrammes de Gantt.



Ce problème est NP-difficile, le résoudre de façon exacte prend au moins un temps exponentiel.

### 2 Problème du voyageur de commerce

Le problème du voyageur de commerce est un problème où l'on doit définir le trajet le plus court pour passer par un ensemble de lieux.



Ce problème est NP-complet, sa résolution prend un temps exponentiel.

### 3 Recherche locale

Les algorithmes de recherche locale permettent de chercher une bonne solution en apportant des améliorations successives à une solution.

A partir d'une solution, on explore les solutions voisines (qui sont obtenus en appliquant un opérateur de voisinage sur la solution). Parmi ces solutions, on en prend une qui donne un meilleur score.

```

/* solution_actuelle : une solution générée aléatoirement */
/* F(s) : la fonction objectif à minimiser */
/* Operateur_de_voisinnage(s) : une fonction qui donne la liste des
   solutions voisines à la solution s. */
meilleure_voisine ← solution_actuelle
répéter
  solution_actuelle ← meilleure_voisine
  pour toutes solution_voisine de Operateur_de_voisinnage(solution_actuelle) faire
    si F(solution_voisine) < F(meilleure_voisine) alors
      | meilleure_voisine ← solution_voisine
    fin
  fin
tant que F(solution_actuelle) < F(meilleure_voisine);

```

**Algorithme 1 :** Algorithme de recherche locale



# 4

## analyse et conception

Pour ce problème, les méthodes exactes ne sont pas suffisamment performantes pour résoudre de grandes instances dans des délais raisonnables. La qualité d'une méthode exacte est estimée à partir de plusieurs critères :

- Le temps de calcul nécessaire pour trouver la solution en fonction de la taille de l'instance.
- L'espace mémoire requis pour résoudre l'instance en fonction de la taille de l'instance.
- L'efficacité de la parallélisation de la méthode.

Pour des résultats plus rapides, il faut passer par des méthodes heuristiques. Ces méthodes permettent de trouver plus rapidement une solution correcte, mais pas forcément la meilleure. Pour évaluer une méthode heuristique, il faut prendre en compte les critères suivants :

- La qualité de la solution en fonction du temps et de la taille de l'instance.
- L'espace mémoire utilisé en fonction de la taille de l'instance.
- L'écart de qualité de la solution par rapport à celle de la méthode exacte en fonction du temps.
- Le temps qu'il faut pour obtenir une solution correcte par rapport à la méthode exacte.

Pour évaluer une méthode heuristique, il faut une méthode exacte pour faire la comparaison.

### 1 méthode exacte

Hugo Chevroton a proposé une méthode pour résoudre le sous-problème pour lequel les ordres des jobs et les lots sont fixés, cette méthode utilise un solveur et est présentée dans la partie [Annexe B](#).

Je me base sur sa modélisation pour ajouter la liberté sur les ordres des jobs dans les lots.

#### 1.1 Paramètres

$m$	Nombre de machines
$n$	Nombre de jobs.
$V$	Nombre de lots.
$N_k$	Nombre de job du lot $k$ .

$p_{i,j}$	Durée de travail sur la machine $i$ pour le job $j$ .
$h_j^{\text{WIP}}$	Cout d'inventaire pendant la production du job $j$ .
$h_j^{\text{FIN}}$	Cout d'inventaire après la production du job $j$ .
$a_k$	Date de départ au plus tôt du lot $k$ .
$b_k$	Date de départ du lot $k$ à partir de laquelle les couts d'inventaires sont minimisés.
$\lambda_k$	Nombre de segments de la fonction $F_k$
$\alpha_{k,s}$	Cout de chaque unité de temps dans le segment $s$ de la fonction $F_k$ .
$c_{k,s}$	Cout des pénalités de retard pour le $s^{\text{e}}$ segment de la fonction $F_k$ .
$t_{k,s}$	Date de début du $s^{\text{e}}$ segment de la fonction $F_k$ , $t_{k,\lambda_k+1}$ vaut $b_k$ .

## 1.2 Variables

$x_{k,s}$	Vaut 0 ou 1 si la date de départ du lot $k$ est dans le segment $s$ de la fonction $F_k$ .
$d_{k,s}$	Représente le temps écoulé dans le segment $s$ avant le départ du lot $k$ .
$C_{i,j}$	Date de fin du travail de la machine $i$ sur le job $j$ .
$y_{j1,j2}$	Variable de précédences en production, vaut 1 si le job $j1$ est effectué avant le job $j2$ .
$j_k$	Dernier job du lot $k$ .
$F_k$	Date de fin de production du lot $k$ .
$\text{IC}^{\text{WIP}}$	Cout d'inventaire pendant la production.

$$\text{IC}^{\text{WIP}} = \sum_{j=1}^n \sum_{i=1}^{m-1} (C_{i+1,j} - p_{i+1,j} - C_{i,j}) h_j^{\text{WIP}}$$

$\text{IC}^{\text{FIN}}$	Cout d'inventaire entre la fin de la production et la livraison des jobs.
--------------------------	---

$$\text{IC}^{\text{FIN}} = \sum_{k=1}^V \sum_{j \in j^k} (F_k - C_{m,j}) h_j^{\text{FIN}}$$

$\text{IC}$	Couts d'inventaire total.
-------------	---------------------------

$$\text{IC} = \text{IC}^{\text{WIP}} + \text{IC}^{\text{FIN}}$$

## 1.3 Objectif

Somme des coûts à minimiser

$$\text{IC} + \sum_{k=1}^V \sum_{s=1}^{\lambda_k} (x_{k,s} c_{k,s} + d_{k,s} \alpha_{k,s})$$

## 1.4 Contraintes

— Lien entre la fin du dernier job d'un lot et le départ du lot

$$C_{j_k, m} \leq \sum_{s=1}^{\lambda_k} (x_{k,s} t_{k,s} + d_{k,s}) \quad \forall k \in \{1, \dots, V\} \quad (1)$$

— La date de départ en livraison ne peut faire partie que d'un segment de la fonction  $F_k$ .

$$\sum_{s=1}^{\lambda_k} x_{k,s} = 1 \quad \forall k \in \{i, \dots, V\} \quad (2)$$

— Chaque segment n'est utilisé que s'il contient la date de départ du lot.

$$d_{k,s} \leq x_{k,s} (t_{k,s+1} - t_{k,s} - 1) \quad \forall k \in \{i, \dots, V\}, \forall s \in \{1, \dots, \lambda_k\} \quad (3)$$

$$0 \leq d_{k,\lambda_k} \quad \forall k \in \{i, \dots, V\} \quad (4)$$

— Chaque job est soit le successeur soit le prédécesseur de chacun des autres jobs.

$$y_{j1,j2} + y_{j2,j1} = 1 \quad \forall j1, j2 \in \{1, \dots, n\}, j1 < j2 \quad (5)$$

— Les jobs ne peuvent pas être leurs propres successeurs.

$$y_{j,j} = 0 \quad \forall j \in \{1, \dots, n\} \quad (6)$$

— Contrainte de gamme, les machines ne peuvent pas travailler sur plusieurs jobs simultanément. Cette contrainte assure également que les jobs soient produits dans l'ordre prescrit.

$$C_{i,j2} \geq C_{i,j1} + p_{i,j2} - M y_{j1,j2} \quad \forall i \in \{1, \dots, m\}, \forall (j1, j2) \in \{1, \dots, n\}^2 \quad (7)$$

— Contrainte de précédence, une machine ne peut pas travailler sur un job tant que la machine précédente n'a pas terminé de travailler dessus. Cette contrainte assure également que les tâches des jobs soient produit dans l'ordre prescrit.

$$C_{i,j} \geq C_{i-1,j} + p_{i,j} \quad \forall j \in \{1, \dots, n\}, \forall i \in \{2, \dots, m\} \quad (8)$$

— Un job est terminé quand toutes ces tâches sont terminées.

$$f_j \geq C_{m,j} \quad \forall j \in \{0, \dots, n\} \quad (9)$$

— Un lot ne peut être livré que si tous ces jobs sont terminés.

$$F_k \geq C_{m,j} \quad \forall k \in \{1, \dots, n\}, \forall j \in J^k \quad (10)$$

Les contraintes suivantes servent à accélérer la résolution du problème.

— La date de fin optimale du lot  $k$  est entre les dates  $a_k$  et  $b_k$  calculée au préalable.

$$a_k \leq C_{j_k,m} \leq b_k \quad \forall k \in \{1, \dots, V\} \quad (11)$$

— Le temps écoulé dans un segment de  $F_k$  avant le départ d'un lot ne doit pas faire partir le lot après  $b_k$ .

$$d_{k,\Lambda_k} \leq t_{k,s+1} - t_{k,s} \quad s \in \{1, \dots, \lambda_k\}, \forall k \in \{1, \dots, V\} \quad (12)$$

— La date du départ d'un lot ne peut pas précéder le début du segment auquel il appartient.

$$x_{k,s} t_{k,s} \leq C_{j_k,m} \quad s \in \{1, \dots, \lambda_k\}, \forall k \in \{1, \dots, V\} \quad (13)$$

## 2 méthode heuristique

Pour la méthode heuristique, j'ai décidé avec mon encadrant d'utiliser un algorithme de recherche locale. Cet algorithme est présenté dans la partie [Section 3](#) (Chapitre 3).

Les solutions sont représentées par une liste de jobs classés dans l'ordre de leurs réalisation.

## 2.1 Operateur de voisinage

L'opérateur a pour rôle de définir les solutions voisines de la solution principale.

Pour ce projet, deux opérateurs ont été envisagés :

- Un opérateur qui donne des solutions où les emplacements de deux jobs dans l'ordre de production ont été échangés.
- Un opérateur qui donne des solutions où l'emplacement d'un job dans l'ordre de réalisation a été changé. Cet opérateur permet de conserver des séquences de jobs qui peuvent être bon.

## 2.2 Exploration du voisinage

Il existe plusieurs façons d'explorer le voisinage fourni par un opérateur, dans ce projet, deux stratégies ont été testées :

- Meilleure amélioration :  
Cette stratégie consiste à évaluer tout les voisins de la solution actuelle et de prendre le meilleur. Le défaut avec cette méthode est qu'il y a un grand nombre de solutions évaluées, ce qui fait que chaque itération de la recherche prend du temps.
- Première amélioration :  
Cette stratégie consiste à sélectionner le premier voisin qui est meilleur que la solution actuelle. Cette stratégie permet de ne pas évaluer tout les voisins et d'itérer plus rapidement, ce qui permet d'explorer plus de minimum locaux dans le temps imparti.

## 2.3 Stratégie de sortie des minimum locaux

Lorsque la solution étudiée n'a pas de meilleur voisin, on est dans un minimum local. Il existe plusieurs façons de sortir des minimum locaux, dans ce projet, on a choisi de relancer une recherche locale à partir d'une nouvelle solution aléatoire.

## 2.4 Condition d'arrêt

Une fois dans un minimum local, l'algorithme repart sur une solution aléatoire, il n'y a donc pas de fin. Pour arrêter la recherche, il faut définir une condition d'arrêt. Pour ce projet, on a choisi de limiter le temps accordé à la recherche locale.

Lorsque la recherche locale atteint un minimum local, si le temps n'est pas écoulé, la stratégie de sortie des minimums locaux est utilisée. Lorsque le temps est écoulé, la recherche locale termine la recherche du minimum local en cours.

Cette durée minimum pour la recherche locale est définie à l'exécution par l'utilisateur de l'application.

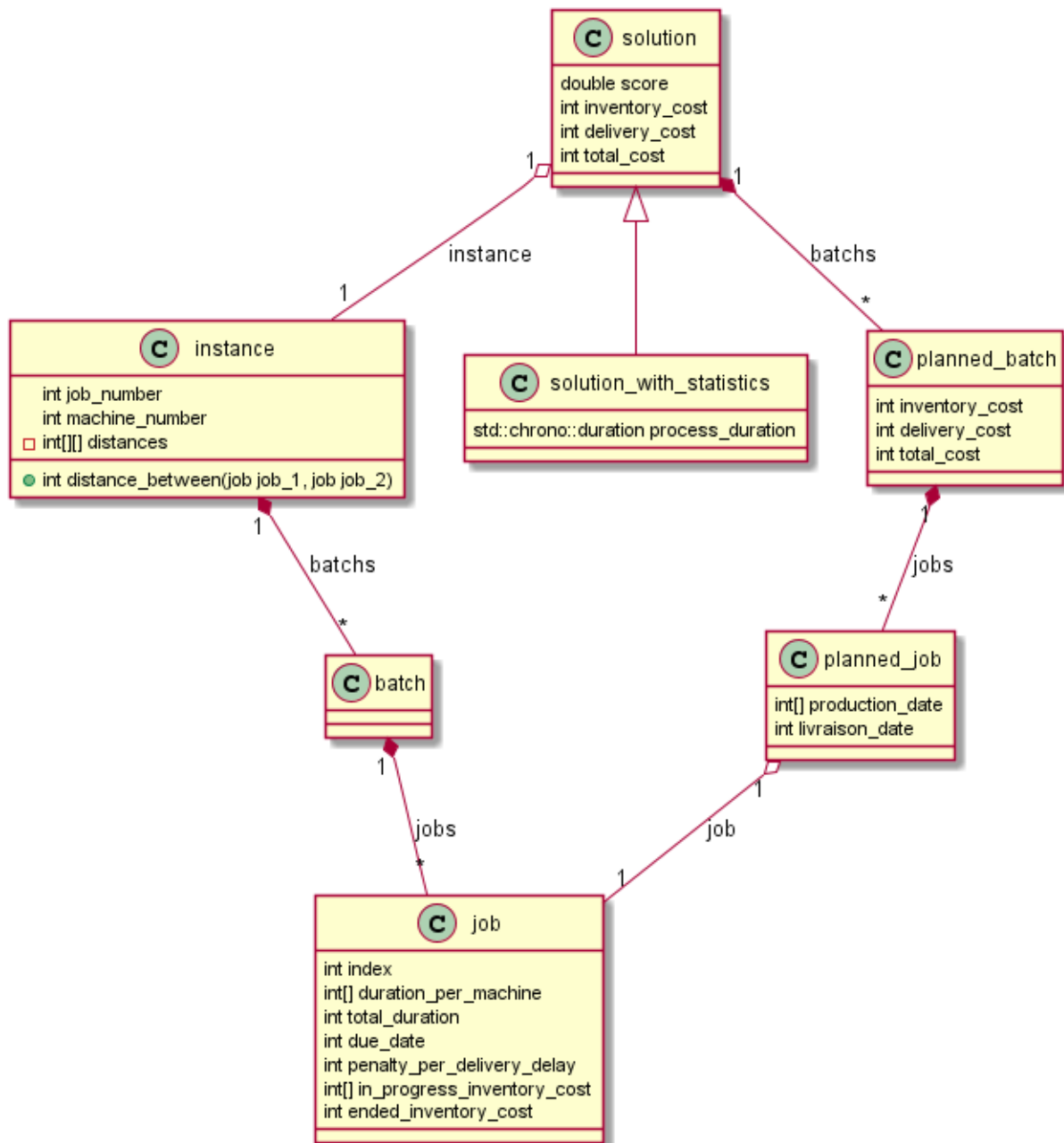
# 3 Modélisation logicielle

D'un point de vue logiciel, le projet est séparé en plusieurs parties, chaque méthode de résolution est implémentée dans un projet Visual Studio distinct et le code en commun est placé dans une bibliothèque.

### 3.1 Bibliothèque

La bibliothèque définit les structures de données des instances et solutions. Elle contient également des classes pour lire les instances, valider les solutions et écrire les solutions.

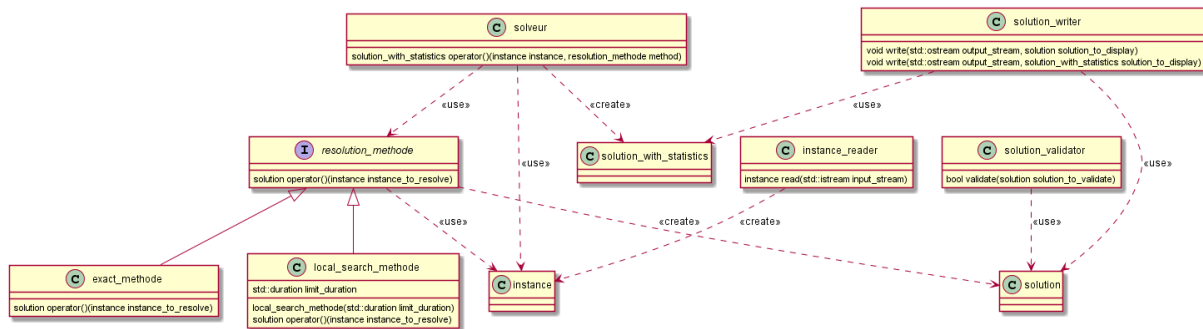
#### 3.1.1 Structures de données



Les classes **planned\_batch** et **planned\_job** représentent les lots et jobs qui ont été planifiés. La durée de la résolution ne fait pas partie de la solution, cette donnée ne concerne que la méthode qui génère la solution.

La classe **solution\_with\_statistics** étend la classe **solution**, elle ajoute la durée de la résolution.

## 3.1.2 Services



# 5

## Mise en œuvre

Le projet reprend un existant en C++, j'ai donc utilisé ce langage pour le développement. Pour visualiser les solutions, j'ai également utilisé les langages HTML, JavaScript.

J'ai choisi d'écrire les solutions en JSON pour permettre les lises avec d'autres outils.

### 1 Outils

#### 1.1 Git

Git est un outil de gestion de versions décentralisé et largement utilisé, je l'ai utilisé pour versionner mon code. Il permet de garder l'historique des versions du projet et de comparer les versions.

#### 1.2 Github

GitHub est une plateforme d'hébergement de dépôt Git qui propose également des outils pour gérer et suivre les tâche à réaliser et les demandes des utilisateurs.

#### 1.3 Visual studio

Visual studio est un environnement de développement qui intègre les outils nécessaires au développement d'application en C++ et C#.

#### 1.4 Tex Live

Tex Live est une distribution latex qui est utilisé dans ce projet pour générer les rapports.

## 1.5 Visual studio code

Visual Studio Code est un éditeur de code modulaire et open-source créé par Microsoft. Je l'ai utilisé pour :

- La rédaction des documents en Latex.
- Pour le développement d'un outil de visualisation de solution en HTML et Javascript.
- La lecture et mise en page des fichiers de solutions.
- La création de scripts PowerShell pour automatiser l'exécution de la recherche local avec différents paramètres.

## 1.6 Doxygen

Doxygen est l'outil qui a été utilisé pour générer la documentation du code du projet à partir de commentaires présents dans le code.

# 2 Implémentations

Le code du projet a été séparé en 4 parties : - implémentation de la recherche locale - implémentation de la méthode exacte avec CPLEX - bibliothèque de fonctionnalités communes - tests unitaires avec le Framework GTest

Durant le projet, j'ai également écrit un petit outil pour visualiser les données des solutions sous forme de diagramme de Gantt en HTML/javascript.

## 2.1 Fonctionnalités communes

Cette partie est la première que j'ai réalisé, car elle est nécessaire au développement des autres parties.

Je n'ai pas rencontré de problèmes particuliers lors de cette étapes mais elle a été plus longue que ce que j'avais prévue initialement. Lors du développement de cette partie, j'ai également écrit les tests unitaires pour valider le fonctionnement de la bibliothèque.

## 2.2 Utilisation du modèle mathématique

Pour cette partie, j'ai implémenté le modèle mathématique que j'ai présenté précédemment. J'ai rencontré successivement de nombreuses difficultés avec le solveurs CPLEX. Après avoir résolu les nombreux problèmes de compilations et d'édits des liens, j'ai obtenu une application qui donne des solutions invalides. Je n'ai pas réussi à corriger ce problème, j'ai finalement dû passer à l'implémentation de la recherche locale.

## 2.3 Recherche locale

Pour permettre de tester différents opérateurs de voisinage et méthode de sélection du meilleur voisin, ces opérateurs sont donnés à l'instanciation de la recherche locale.



### 2.3.1 Opérateurs de voisinage

J'ai commencé par implémenter l'opérateur de voisinage par permutation des jobs dans l'ordre de production. Par la suite j'ai implémenté un deuxième opérateur qui change la position des jobs dans l'ordre de production.

### 2.3.2 Sélection du voisin

Pour sélectionner le voisin à choisir pour améliorer la solution, j'ai commencé par prendre le premier voisin qui est meilleur que la solution actuelle. Par la suite, j'ai ajouté la stratégie qui consiste à sélectionner le meilleur voisin qui améliore la solution.

## 2.4 Limites

La résolution avec méthode exacte ne donne pas de solutions valides, je n'ai pas pu comparer les résultats des deux méthodes. De plus sans la méthode exacte, je n'ai pas les meilleures solutions pour les instances testés, j'ai donc comparé les résultats avec le meilleur résultat.

La génération de la fonction de coût par date de départ en livraison prend beaucoup de temps pour des lots de plus de 7 jobs, lors des tests, il fallait 1 heure 30 pour résoudre un problème de 100 jobs avec des lots de 8 jobs. Cette lenteur a rendu difficile l'évaluation de la recherche locale pour de grands lots.

## 3 Performance

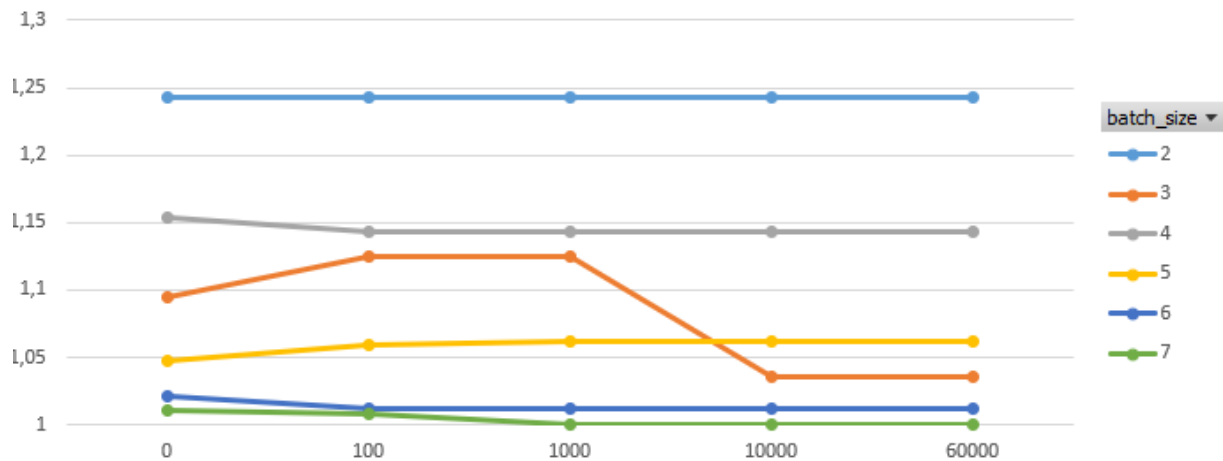
La recherche locale s'exécute sur un seul thread, il serait possible de paralléliser une partie de la résolution. La parallélisation n'a pas été mise en place dans ce projet dans le but de se concentrer sur les opérateurs.

### 3.1 Analyse des résultats

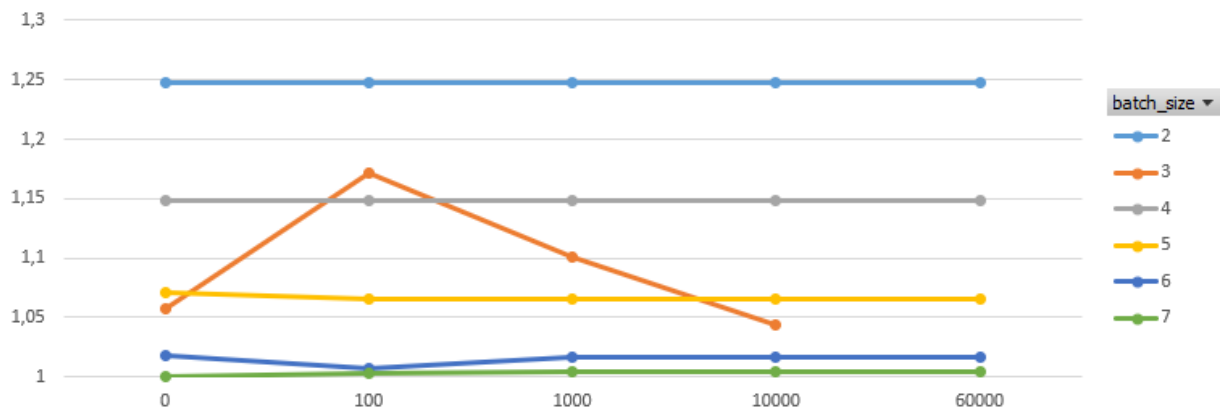
Pour la recherche local, j'ai comparé les résultats pour différents paramètres et opérateurs. Pour faire les comparaisons, j'ai utilisé un script PowerShell pour tester automatiquement un grand nombre de combinaisons. Le script extrait les données des solutions et les rassemble dans un fichier csv. Les données du fichier CSV sont ensuite lues dans Excel et utilisées pour faire des graphiques.

### 3.2 Operateur de voisinage

Pour comparer les résultats sur plusieurs instances, j'ai calculé pour chaque solution, le rapport entre son score et le meilleur score obtenu pour l'instance.



Le graphique ci-dessus concerne l'opérateur de voisinage par permutations. On peut voir l'évolution de la qualité de la solution en fonction de la durée de recherche locale (en millisecondes). On remarque que le score n'évolue pas beaucoup avec l'augmentation de la durée, cela peut s'expliquer par le fait que les batchs soient de petites tailles, il y a donc un nombre limité de solutions à tester.



Ce graphique concerne l'opérateur qui change les positions des jobs dans l'ordre de production. De la même manière que pour l'opérateur précédent, le résultat ne varie pas beaucoup avec la durée d'exécution.

Sur les 336 solutions regardées, le deuxième opérateur donne des résultats légèrement meilleurs.

### 3.3 Sélection du voisin

La méthode de sélection qui prend le meilleur voisin donne de meilleurs résultats pour des temps de recherche très courts (moins d'une seconde). Pour le reste, la méthode qui prend le premier voisin qui est meilleur donne de meilleurs résultats. Ce résultat peut s'expliquer par le fait que les lots ne soient pas très grands et que la deuxième méthode soit très rapide. La recherche locale explore donc plus de minimums locaux et peut plus facilement explorer toutes les solutions possibles.

# 6

## Bilan et conclusion

### 1 Bilan semestre 9

Durant ce premier semestre, j'ai étudié les modélisations et le code existant, j'ai également défini les algorithmes à développer. J'ai pris du retard au début, car j'ai rencontré des difficultés à comprendre les modèles mathématiques utilisés par mon encadrant et le code existant. Il me reste à implémenter les algorithmes et analyser leurs résultats.

### 2 Bilan semestre 10

La partie résolution exacte n'est pas terminée et la recherche locale peut être améliorée. La recherche locale est très lente pour de grands lots, la méthode de calcul des coûts de livraisons en fonction des dates de départ des lots doit être changée.

# 7

## Bibliographie

## Annexes

# A

## modélisation du problème global

### 1 Paramètres

$m$	Nombre de machines
$n$	Nombre de jobs. Il est possible de faire des lots ne contenant qu'un seul job, $n$ désigne donc également le nombre maximal de lot.
$h_{i,j}^{WIP}$	Coût d'inventaire unitaire pour le job $j$ entre la machine $i$ et la machine $i + 1$
$h_j^{FIN}$	Coût d'inventaire unitaire pour le job $j$ après la fin de la production
$p_{i,j}$	Durée du travail pour le job $j$ sur la machine $i$
$C^V$	Coût par véhicule (aussi le coût par lot), représente le coût de la mobilisation d'un véhicule.
$\pi_j$	Coût du retard de livraison par unité de temps pour le job $j$
$d_j$	Date de livraison demandée pour le job $j$
$t_{j_1,j_2}$	Durée du trajet entre les lieux de livraison des jobs $j_1$ et $j_2$
$c_{j_1,j_2}$	Coût du trajet entre les lieux de livraison des jobs $j_1$ et $j_2$

### 2 Variables

Pour représenter l'ordre des jobs lors de la production, on utilise une matrice qui indique pour chaque job s'il doit être réalisé avant un autre.

Par exemple, pour la matrice suivante.

$$y = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Dans le lot 1, le job 1 est précédé des jobs 2 et 3, il est donc le dernier à être effectué. Les jobs seront effectués dans l'ordre  $\{3, 2, 1\}$ .

Pour représenter l'ordre de distribution des jobs, on utilise pour chaque lot, une matrice qui indique quel est le prédécesseur direct de chaque job.

Par exemple, pour deux lots  $\{j1, j2\}$  et  $\{j3, j4\}$  :

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ et } \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Ce qui signifie dans le lot 1,  $j1$  est livré avant  $j2$  et dans le lot 2,  $j3$  est livré avant  $j4$ .

$y_{j1,j2}$	Variable de précédences en production, vaut 1 si le job $j1$ est effectué avant le job $j2$ .
$z_k$	Vaut 1 si le véhicule $k$ est utilisé.
$Z_{j,k}$	Vaut 1 si le job $j$ fait partie du lot $k$
$C_{i,j}$	Date de fin de la tâche $j$ pour la machine $i$
$F_k$	Date de départ du lot/véhicule $k$ .
$f_j$	Date de départ du job $j$ .
$IC^{WIP}$	Somme des coûts d'inventaire sur les jobs en cours

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

$IC^{FIN}$	Somme des coûts d'inventaire sur les jobs terminés.
------------	---

$$IC^{FIN} = \sum_{j=1}^n (f_j - C_{m,j}) h_j^{FIN}$$

VC	Coût des véhicules.
----	---------------------

$$VC = c^V \sum_{k=1}^n z_k$$

$x_{j1,j2,k}$	Variable de précédence pour la livraison, vaut 1 si $j1$ est livré juste avant $j2$ et que $j1$ et $j2$ font partie du lot $k$ .
$D_j$	Date de livraison réalisé pour le job $j$
$T_j$	Retard de la livraison du job $j$
RC	Cout des trajets de la livraison, sans compter le retour au dépôt après la tournée

$$RC = \sum_{k=1}^n \sum_{j_1=0}^n \sum_{j_2=1}^{n+1} x_{j_1,j_2,k} c_{j_1,j_2}$$

PC	Cout total des pénalités de retard
----	------------------------------------

$$PC = \sum_{j=1}^n T_j$$

### 3 Objectif

Somme des coûts à minimiser

$$IC = IC^{WIP} + IC^{FIN} + PC + VC + RC$$

### 4 Contraintes

— Chaque job est soit le successeur soit le prédécesseur de chacun des autres jobs.

$$y_{j1,j2} + y_{j2,j1} = 1 \quad \forall j1, j2 \in \{1, \dots, n\}, j1 < j2 \quad (1)$$

— Les jobs ne peuvent pas être leurs propres successeurs.

$$y_{j,j} = 0 \quad \forall j \in \{1, \dots, n\} \quad (2)$$

— Contrainte de gamme, les machines ne peuvent pas travailler sur plusieurs jobs simultanément. Cette contrainte assure également que les jobs soient produits dans l'ordre prescrit.

$$C_{i,j2} \geq C_{i,j1} + p_{i,j2} - M y_{j1,j2} \quad \forall i \in \{1, \dots, m\}, \forall (j1, j2) \in \{1, \dots, n\}^2 \quad (3)$$

— Contrainte de précédence, une machine ne peut pas travailler sur un job tant que la machine précédente n'a pas terminé de travailler dessus. Cette contrainte assure également que les tâches des jobs soient produit dans l'ordre prescrit.

$$C_{i,j} \geq C_{i-1,j} + p_{i,j} \quad \forall j \in \{1, \dots, n\}, \forall i \in \{2, \dots, m\} \quad (4)$$

— Chaque job appartient à exactement un lot.

$$\sum_{k=1}^n Z_{j,k} = 1 \quad \forall j \in \{1, \dots, n\} \quad (5)$$

— Un job est terminé quand toutes ces tâches sont terminées.

$$f_j \geq C_{m,j} \quad \forall j \in \{0, \dots, n\} \quad (6)$$

— Un lot ne peut être livré que si tous ces jobs sont terminés.

$$F_k \geq f_j - M(1 - Z_{j,k}) \quad \begin{array}{l} \forall j \in \{1, \dots, n\}, \\ \forall k \in \{1, \dots, n\} \end{array} \quad (7)$$

— Un lot doit exister à partir du moment où il a un job.

$$nz_k \geq \sum_{j=1}^n Z_{j,k} \quad \forall k \in \{1, \dots, n\} \quad (8)$$

— Si un job fait partie d'un lot, il a un successeur direct dans ce lot.

$$\sum_{j2=1}^{n+1} x_{j1,j2,k} = Z_{j1,k} \quad \forall j1 \in \{1, \dots, n\}, \forall k \in \{1, \dots, n\} \quad (9)$$



— Si un job fait partie d'un lot, pour la livraison, il a un prédécesseur direct dans ce lot.

$$\sum_{j1=0}^n x_{j1,j2,k} = Z_{j2,k} \quad \forall j2 \in \{1, \dots, n\}, \forall k \in \{1, \dots, n\} \quad (10)$$

— Si un job est livré avant un autre, il arrive plus tôt chez le client.

$$D_{j2} \geq D_{j1} + t_{j1,j2} - M(1 - x_{j1,j2,k}) \quad \forall (j1, j2) \in \{1, \dots, n\}^2 \quad (11)$$

— Les retards de livraisons ne peuvent pas être négatifs, les livraisons en avance n'ont pas d'impacts.

$$T_j \geq 0 \quad \forall j \in \{1, \dots, n\} \quad (12)$$

$$T_j \geq T_j \pi_j \quad \forall j \in \{1, \dots, n\} \quad (13)$$

— Un job ne peut pas être livré avant son départ en livraison.

$$D_j \geq f_j + t_{0,j} \quad \forall j \in \{1, \dots, n\} \quad (14)$$

# B

## Sous-problème initial

Dans ce sous-problème où les lots et les ordres des jobs sont fixés.

Il est possible de faire partir les lots au plus tôt pour minimiser les pénalités de retard ou les retarder pour réduire les couts d'inventaire. Pour connaître l'impact de ces décisions sans résoudre le problème de livraison pour chaque combinaison, mon encadrant a trouvé une solution, construire une fonction  $F_k$  qui donne pour un lot  $k$  les pénalités de retard en fonction de la date de départ du lot. Cette fonction n'est calculée qu'une seule fois pour chaque lot car les compositions des lots sont fixes. Une fois cette fonction calculée, il est possible de résoudre la date de départ de chaque lot séparément.

### 1 Paramètres

$m$	Nombre de machines
$n$	Nombre de jobs.
$V$	Nombre de lots.
$j_k$	Dernier job du lot $k$ .
$J^k$	Liste ordonnée des jobs du lot $k$ .
$p_{i,j}$	Durée de travail sur la machine $i$ pour le job $j$
$a_k$	Date de départ au plus tôt du lot $k$ .
$b_k$	Date de départ du lot $k$ à partir de laquelle les couts d'inventaires sont minimisés.
$\lambda_k$	Nombre de segment de la fonction $F_k$
$\alpha_{k,s}$	Couts de chaque unité de temps dans le segment $s$ de la fonction $F_k$ .
$c_{k,s}$	Couts des pénalités de retard pour le $s^{\text{e}}$ segment de la fonction $F_k$ .
$t_{k,s}$	Date de début du $s^{\text{e}}$ segment de la fonction $F_k$ , $t_{k,\lambda_k+1}$ vaut $b_k$ .
$h_j^{\text{WIP}}$	Cout d'inventaire pendant la production du job $j$ .
$h_j^{\text{FIN}}$	Cout d'inventaire après la production du job $j$ .

## 2 Variables

$C_{i,j}$	Date de fin du travail de la machine $i$ sur le job $j$ .
$x_{k,s}$	Vaut 0 ou 1 si la date de départ du lot $k$ est dans le segment $s$ de la fonction $F_k$ .
$d_{k,s}$	Représente le temps écoulé dans le segment $s$ avant le départ du lot $k$ .
$IC^{WIP}$	Cout d'inventaire pendant la production.

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

$IC^{FIN}$	Cout d'inventaire entre la fin de la production et la livraison des jobs.
------------	---

$$IC^{FIN} = \sum_{k=1}^V \sum_{j \in j^k} (C_{m,j_k} - p_{m,j_k^k} - C_{m,j}) h_j^{FIN}$$

$IC$	Couts d'inventaire total.
------	---------------------------

$$IC = IC^{WIP} + IC^{FIN}$$

## 3 Objectif

Somme des coûts à minimiser

$$IC + \sum_{k=1}^V \sum_{s=1}^{\lambda_k} (x_{k,s} c_{k,s} + d_{k,s} \alpha_{k,s})$$

## 4 Contraintes

$$C_{j_k,m} \leq \sum_{s=1}^{\lambda_k} (x_{k,s} t_{k,s} + d_{k,s}) \quad \forall k \in \{i, \dots, V\} \quad (1)$$

$$\sum_{s=1}^{\lambda_k} x_{k,s} = 1 \quad \forall k \in \{i, \dots, V\} \quad (2)$$

$$d_{k,s} \leq x_{k,s} (t_{k,s+1} - t_{k,s} - 1) \quad \forall k \in \{i, \dots, V\}, \forall s \in \{1, \dots, \lambda_k\} \quad (3)$$

$$0 \leq d_{k,\lambda_k} \quad \forall k \in \{i, \dots, V\} \quad (4)$$

— Lien entre la fin du dernier job d'un lot et le départ du lot

$$C_{j_k,m} \leq \sum_{s=1}^{\lambda_k} (x_{k,s} t_{k,s} + d_{k,s}) \quad \forall k \in \{i, \dots, V\} \quad (5)$$

— La date de départ en livraison ne peut faire partie que d'un segment de la fonction  $F_k$ .

$$\sum_{s=1}^{\lambda_k} x_{k,s} = 1 \quad \forall k \in \{i, \dots, V\} \quad (6)$$

— Chaque segment n'est utilisé que s'il contient la date de départ du lot.

$$d_{k,s} \leq x_{k,s} \left( t_{k,s+1} - t_{k,s} - 1 \right) \quad \forall k \in \{1, \dots, V\}, \forall s \in \{1, \dots, \lambda_k\} \quad (7)$$

$$0 \leq d_{k,\lambda_k} \quad \forall k \in \{1, \dots, V\} \quad (8)$$

Les contraintes suivantes servent à accélérer la résolution du problème.

— La date de fin optimale du lot  $k$  est entre les dates  $a_k$  et  $b_k$  calculé au préalable.

$$a_k \leq C_{j_k,m} \leq b_k \quad \forall k \in \{1, \dots, V\} \quad (9)$$

— Le temps écoulé dans un segment de  $F_k$  avant le départ d'un lot ne doit pas faire partir le lot après  $b_k$ .

$$d_{k,\Lambda_k} \leq t_{k,s+1} - t_{k,s} \quad s \in \{1, \dots, \lambda_k\}, \forall k \in \{1, \dots, V\} \quad (10)$$

— La date du départ d'un lot ne peut pas précéder le début du segment auquel il appartient.

$$x_{k,s} t_{k,s} \leq C_{j_k,m} \quad s \in \{1, \dots, \lambda_k\}, \forall k \in \{1, \dots, V\} \quad (11)$$

# C

# Spécifications fonctionnelles

## 1 Lecture des paramètre

La première étape du programme est de lire des paramètres donnés au lancement du programme. Le programme attend les paramètres suivant :

- le chemin vers le fichier qui décrit l'instance à résoudre.
- l'algorithme à utiliser.
- Pour la recherche locale, le temps accordé à l'algorithme.

## 2 Lecture de l'instance

Cette fonction permet de lire le fichier de l'instance à résoudre, elle a déjà été réalisée dans le programme existant fournit par mon encadrant.

## 3 Résolution de l'instance

Cette fonction est la partie principale du projet. Elle permet de résoudre l'instance avec l'algorithme choisie par l'utilisateur. Les algorithmes à implémenter sont :

- La méthode exacte présentée dans la partie [Section 1](#) (Chapitre 4).
- La recherche locale présenté dans la partie [Section 2](#) (Chapitre 4).

La résolution prend une instance en paramètre. Les données retournées sont :

- la solution.
- le score de la solution.
- le temps écoulé.
- la mémoire utilisée.

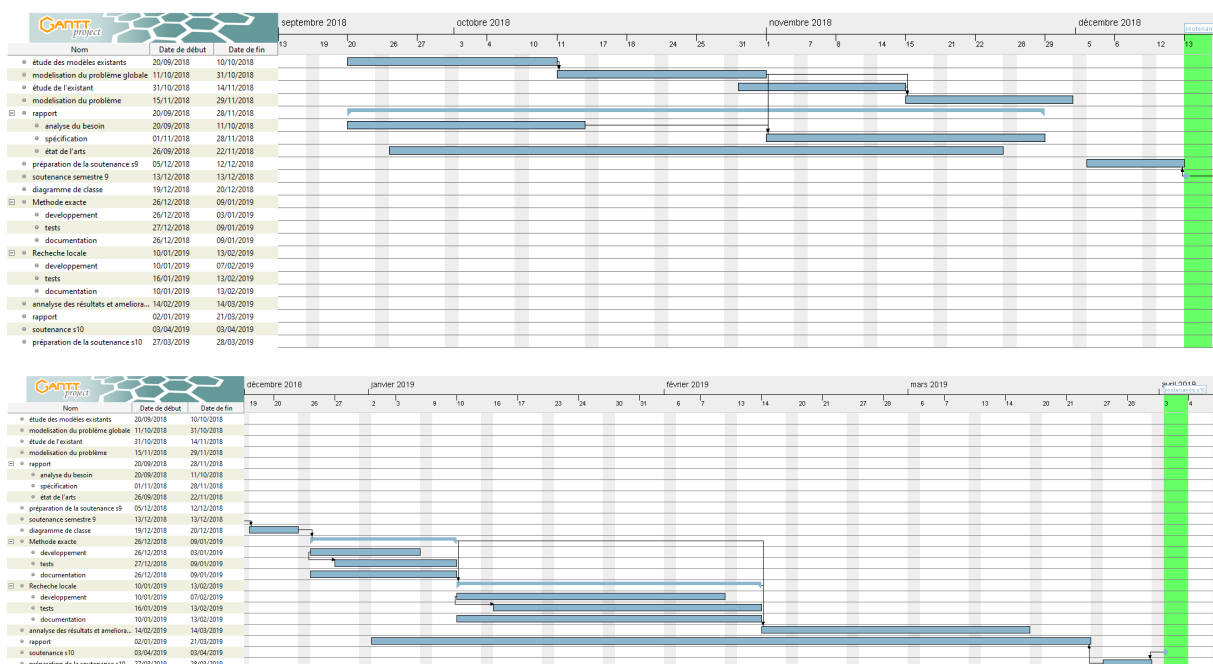
## 4 Ecriture des résultats

Cette fonction à pour rôle d'écrire les résultats de l'algorithme dans la console. La fonction prend en paramètre les données fournies par la résolution de l'instance et les écrit dans la sortie standard. Les données sont :

- le score de la solution.
- la solution.
- la durée de la résolution.
- la mémoire utilisée pour la résolution.

# D

# Planification



Nom	Date de début	Date de fin
• étude des modèles existants	20/09/2018	10/10/2018
• modelisation du problème globale	11/10/2018	31/10/2018
• étude de l'existant	31/10/2018	14/11/2018
• modelisation du problème	15/11/2018	29/11/2018
☐ • rapport	20/09/2018	28/11/2018
• analyse du besoin	20/09/2018	11/10/2018
• spécification	01/11/2018	28/11/2018
• état de l'arts	26/09/2018	22/11/2018
• préparation de la soutenance s9	05/12/2018	12/12/2018
• soutenance semestre 9	13/12/2018	13/12/2018
• diagramme de classe	19/12/2018	20/12/2018
☐ • Methode exacte	26/12/2018	09/01/2019
• developpement	26/12/2018	03/01/2019
• tests	27/12/2018	09/01/2019
• documentation	26/12/2018	09/01/2019
☐ • Recherche locale	10/01/2019	13/02/2019
• developpement	10/01/2019	07/02/2019
• tests	16/01/2019	13/02/2019
• documentation	10/01/2019	13/02/2019
• annalyse des résultats et ameliora...	14/02/2019	14/03/2019
• rapport	02/01/2019	21/03/2019
• soutenance s10	03/04/2019	03/04/2019
• préparation de la soutenance s10	27/03/2019	28/03/2019



# E

## Installation

### 0.1 Installation des livrables

#### 0.1.1 Resolution d'instance

Les programmes de résolutions sont des exécutables compilé pour Windows, ils ne nécessite pas d'installation particulière.

#### 0.1.2 Bibliothèque de fonction commune

La bibliothèque est composé d'un fichier lib et de fichiers de définitions h.

#### 0.1.3 Visualisation des solutions

L'outil de visualisation des solutions tient en un fichier HTML, il s'ouvre dans un navigateur web.

Les livrables sont des exécutables qui s'utilisent en invite de commande et ne nécessite pas d'étape d'installation particulière. Pour visualiser les solutions, il faut que la solution soit dans le même dossier que le fichier view.html.

# F

# Manuel d'utilisation

## 1 recherche locale

La recherche locale s'exécute en invite de commande et nécessite les paramètres suivants :

- Chemin vers le fichier d'instance.
- Taille maximum des lots.
- Chemin vers le fichier de solution.
- Durée minimum de la recherche locale.

Si le fichier de la solution existe déjà, son contenu sera écrasé.

## 2 bibliothèque

Pour l'utiliser dans un projet, il faut : - ajouter le répertoire include de la bibliothèque. - ajouter le fichier lib

## 3 visionneuse de solution

Pour visionner une solution, il faut ouvrir le fichier view.html, celui-ci doit être dans le même fichier que la solution. Le nom du fichier de solution peut être donné à la page web avec le paramètre filename. Par exemple, si le fichier de la solution est solution.json et se trouve à la racine de l'ordinateur, il faut entrer l'URL suivante dans le navigateur.

file:///C:/view.html?filename=res\_first.json

Pour visionner une solution, il faut un accès à internet pour charger aux bibliothèques utilisées.

# Ordonnancement de production dans un problème intégrant la distribution

Jean Hilleriteau

Encadrement : Hugo Chevrotton

## Objectifs

L'objectif de ce projet est de comparer des méthodes pour résoudre un problème d'ordonnancement d'atelier et un problème de livraison.



POLYTECH<sup>®</sup>  
TOURS

## Mise en œuvre

Le projet prend la forme d'une application codée en C++ qui implémente une méthode de résolution exacte et une recherche locale.



POLYTECH<sup>®</sup>  
TOURS

## Résultats attendus

Le résultat attendu est un programme permettant de résoudre le problème d'atelier en prenant en compte son impact sur le problème de livraison.



POLYTECH<sup>®</sup>  
TOURS



# Ordonnancement de production dans un problème intégrant la distribution

Jean Hilleriteau

Encadrement : Hugo Chevroton

## Objectifs

L'objectif de ce projet est de comparer des méthodes pour résoudre un problème d'ordonnancement d'atelier et un problème de livraison.

## Mise en œuvre

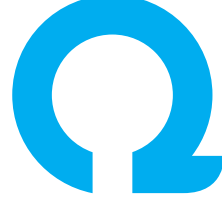
Le projet prend la forme d'une application codée en C++ qui implémente une méthode de résolution exacte et une recherche locale.

## Résultats attendus

Le résultat attendu est un programme permettant de résoudre le problème d'atelier en prenant en compte son impact sur le problème de livraison.



POLYTECH<sup>®</sup>  
TOURS



POLYTECH<sup>®</sup>  
TOURS



# Ordonnancement de production dans un problème intégrant la distribution

## Résumé

Ce projet vise à développer un algorithme pour optimiser la planification de jobs dans un problème d'atelier qui s'intègre avec un problème de livraison.

## Mots-clés

flow-shop avec permutation, C++, recherche locale, resolution exacte

## Abstract

This project is about to optimise a jobs scheduling in a flow-shop problem integrated with a salesman problem.

## Keywords

permutation flow-shop, C++, local search, exact resolution