

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 et affectation des ressources dans le cloud Computing

Cas multi critères et multi machines

**Tuteurs académiques**  
Boukhalfa ZAHOUT  
Ameur SOUKHAL  
Patrick MARTINEAU

**Étudiant**  
Abdoulaye KAKE (DI5)



## Liste des intervenants

Nom	Email	Qualité
Abdoulaye KAKE	<a href="mailto:abdoulaye.kake@etu.univ-tours.fr">abdoulaye.kake@etu.univ-tours.fr</a>	Étudiant DI5
Boukhalfa ZAHOUT	<a href="mailto:boukhalfa.zahout@univ-tours.fr">boukhalfa.zahout@univ-tours.fr</a>	Tuteur académique, Département Informatique
Ameur SOUKHAL	<a href="mailto:ameur.soukhal@univ-tours.fr">ameur.soukhal@univ-tours.fr</a>	Tuteur académique, Département Informatique
Patrick MARTINEAU	<a href="mailto:patrick.martineau@univ-tours.fr">patrick.martineau@univ-tours.fr</a>	Tuteur académique, Département Informatique



## Avertissement

Ce document a été rédigé par Abdoulaye Kake susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Boukhalifa Zahout, Ameer Soukhal et Patrick Martineau susnommés les tuteurs académiques.

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 des tuteurs académiques 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

Abdoulaye Kake, *Ordonnancement et affectation des ressources dans le cloud Computing: Cas multi critères et multi machines*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Kake, Abdoulaye},
  title={Ordonnancement et affectation des ressources dans le cloud Computing: Cas multi
    critères et multi machines},
  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
Table des figures	v
Liste des Algorithmes	vii
<b>1 Introduction</b>	<b>1</b>
1 Contexte de la réalisation .....	1
1.1 Acteur du projet .....	1
1.2 Contexte et Enjeux .....	2
1.3 Objectifs .....	3
1.4 Hypothèses.....	3
1.5 Base méthodologiques .....	4
<b>2 Description Générale</b>	<b>5</b>
1 Environnement du Projet .....	5
2 Caractéristique des utilisateurs.....	5
3 Fonctionnalité du système.....	6
<b>3 État de l'art/veille</b>	<b>8</b>
1 Étude de la documentation .....	8
1.1 Cas Mono critère - Mono machine .....	8

1.2	Cas Mono critère - Multi machine.....	10
1.3	Cas Multi critère - Mono machine.....	13
1.3.1	Définition d'un Front de Pareto .....	15
2	Étude de l'existant .....	16
<b>4</b>	<b>Analyse et Conception</b>	<b>18</b>
1	Méthode exacte.....	18
1.1	La Combinaison Lineaire.....	19
1.2	Epsilon - Contrainte.....	20
2	Heuristique - Méta heuristique .....	20
2.1	Heuristique .....	20
2.1.1	Méthode de Tries : .....	20
2.1.2	Méthode d'affectation :.....	21
2.1.3	Politique et Formation d'heuristique : .....	22
2.2	Méta heuristique.....	23
2.2.1	Initialisation de la Population .....	25
2.2.2	Sélection.....	25
2.2.3	Croisement et Mutation.....	25
3	Déroulement de la validation des algorithmes .....	28
4	Conception de l'application : .....	29
4.1	Model - Vue - Controleur (MVC).....	30
4.1.1	Package Vue .....	30
4.1.2	Package Contoleur .....	31
4.1.3	Package Modèle .....	32
<b>5</b>	<b>Mise en Oeuvre</b>	<b>34</b>
1	Implémentation de l'application .....	34
2	Expérimentation et des résultats.....	36
2.1	Les Performances de NSGA2 :.....	37
2.2	Les Performances globales :.....	38
<b>6</b>	<b>Bilan et conclusion</b>	<b>42</b>
1	Réalisé : .....	42
2	Reste à faire :.....	43
3	Bilan sur la qualité .....	43
4	Bilan auto-critique sur la gestion de projet .....	43
5	Conclusion .....	44

<b>7</b>	<b>Annexe</b>	<b>45</b>
1	Description des interfaces externes du logiciel .....	45
1.1	Interfaces matériel/logiciel .....	45
1.2	Interfaces homme/machine.....	45
2	Spécifications fonctionnelles.....	45
2.1	Description des fichiers d'entrée.....	45
2.2	Description des fichiers de sortie ou fichier de résultats.....	47
2.3	Fonctionnalité du système.....	48
2.3.1	Définition de la fonction 1 : Exécution méthode Exacte .....	48
2.3.2	Définition de la fonction 2 : Écriture du résultat dans Fichier.....	48
2.3.3	Définition de la fonction 3 : Exécution Heuristique.....	48
2.3.4	Définition de la fonction 4 : Obtention du Front Pareto optimale .....	49
2.4	Programme de comparaison de front de Pareto :.....	49
2.4.1	Définition de la fonction 1 : Pourcentage de la Solution Exacte ....	49
2.4.2	Définition de la fonction 2 : Distance Moyenne .....	50
2.5	Programme IHM et l'affichage des résultats.....	50
2.5.1	Définition de la fonction 1 : Mise en place d'une Application.....	50
2.5.2	Définition de la fonction 2 : Graphisme du Front de Pareto .....	50
2.5.3	Définition de la fonction 3 : Diagramme de Gantt des jobs ordonnées .....	51
3	Spécifications non fonctionnelles.....	51
3.1	Contraintes de développement et conception .....	51
3.2	Performances.....	51
3.3	Capacités.....	52
3.4	Modes de fonctionnement .....	52
3.5	Contrôlabilité.....	52
3.6	Sécurité .....	52
3.7	Intégrité.....	52
3.8	Maintenance et évolution du système.....	52
4	Gestion de projet .....	53
4.1	Découpage en tâche .....	53
4.1.1	Tâche 1 : Gestion de projet et version .....	53
4.1.2	Tâche 2 : Compréhension des objectifs et le contexte.....	53
4.1.3	Tâche 3 : Étude de l'existant.....	53
4.1.4	Tâche 4 : Étude de la documentation .....	54
4.1.5	Tâche 5 : Rédaction de l'état de l'art .....	54
4.1.6	Tâche 6 : Recherche, Analyse et conception du cas Multi critère...	54
4.1.7	Tâche 7 : Rédaction de Cahier de Spécification .....	54
4.1.8	Tâche 8 : Développement de l'épsilon-contrainte.....	54

4.1.9	Tâche 9 : Étudier l'algorithme génétique NSGA2 .....	54
4.1.10	Tâche 10 : Rédaction du rapport final mi-parcours .....	55
4.1.11	Tâche 11 : Préparation de la soutenance mi-parcours.....	55
4.1.12	Tâche 12 :Développement de l'épsilon contrainte .....	55
4.1.13	Tâche 13 :Écriture des résultats dans un fichier .....	55
4.1.14	Tâche 14 :Développement des Heuristiques .....	55
4.1.15	Tâche 15 :Obtention du front de Pareto .....	55
4.1.16	Tâche 16 :Développement de l'algorithme génétique NSGA2 .....	55
4.1.17	Tâche 17 :Interface graphique JAVA.....	56
4.1.18	Tâche 18 :Validation des résultats.....	56
4.1.19	Tâche 19 :Analyse des différents algorithmes .....	56
4.1.20	Tâche 20 Rédaction du Rapport final .....	56
4.1.21	Tâche 21 :Préparation de la soutenance finale .....	56
4.2	Analyse de faisabilité.....	56
4.3	Analyse de risque.....	57
4.4	Planning.....	57
4.4.1	Le planning de la partie recherche .....	57
4.4.2	Le planning de la partie Développement .....	57
<b>8</b>	<b>Document d'installation</b>	<b>59</b>
<b>9</b>	<b>Document d'utilisation</b>	<b>61</b>
<b>10</b>	<b>Document de Test</b>	<b>64</b>

# Table des figures

## 2 Description Générale

1	Diagramme de cas d'utilisation du programme .....	6
2	Diagramme de séquence lors du programme .....	7
3	Diagramme de cas d'utilisation de la comparaison.....	7

## 3 État de l'art/veille

1	Algorithme pour obtenir les sous ensembles maximaux.....	9
2	Exemple de Solution de Pareto .....	16
3	Architecture du programme du PRD de Boyang Wang.....	17

## 4 Analyse et Conception

1	Algorithme d'affectation des jobs machine par machine .....	21
2	Algorithme d'affectation des jobs par machine moins chargée .....	22
3	L'algorithme génétique .....	24
4	La représentation d'un individu .....	25
5	Etape 1 du croisement .....	27
6	Etape 2 du croisement .....	27
7	Etape 3 du croisement .....	29
8	Exemple de mutation .....	29
9	Architecture de l'application JAVA - MVC.....	30
10	les classes du Package Vue.....	31
11	les classes du Package Contrôleur .....	32
12	Les sous Packages du Package Model.....	33

**5 Mise en Oeuvre**

1	Exemple de front de Pareto obtenue par epsilon contrainte d'une instance de 100 jobs et 7 machines.....	35
2	Exemple de front de Pareto obtenue par NSGA2 d'une instance de 50 jobs et 4 machines .....	36
3	Exemple de front de Pareto obtenue par Make Span MachineParMachine d'une instance de 50 jobs et 4 machines .....	36
4	Tableau de données obtenu par NSGA2 .....	38
5	Temps CPU par approches .....	39
6	Gap Pourcentage Moyen de %S.....	40
7	Gap Pourcentage Moyen de %S weak .....	40
8	Hypervolume entre Méthode exacte et NSGA2 avec les heuristiques .....	41
9	la distance euclidienne entre Méthode exacte et NSGA2 avec les heuristiques .....	41

**7 Annexe**

1	Structure pour les fichiers d'instances.....	46
2	Exemple de fichier d'instances.....	46
3	Structure pour les fichiers de résultat d'une instance.....	47
4	Exemple de fichier de résultat d'une d'instances .....	47
5	diagramme de Gantt du planning de la recherche .....	57
6	diagramme de Gantt du planning de la partie developpement .....	58

**8 Document d'installation**

1	Version de l'éditeur IntelliJ .....	59
---	-------------------------------------	----

**9 Document d'utilisation**

1	Fenêtre d'accueil de l'application.....	61
2	Les différents menu de l'application .....	62
3	Fonctionnalité de résolution d'instance.....	62
4	Fonctionnalité de résolution d'instance de l'application .....	63

**10 Document de Test**

1	Architecture de tests avec JUnit du programme Java .....	64
2	Exemple de fichier de test. ....	65
3	le résultat des test JUnit .....	66



# Liste des Algorithmes

1	Algorithme d'initialisation de la population Initiale .....	26
2	Algorithme de vérification de redondances.....	26
3	Algorithme de sélection des individus .....	27
4	L'algorithme de croisement .....	28
5	Algorithme de mutation.....	29

# 1

## Introduction

Dans le but d'obtenir le diplôme d'ingénieur et d'acquérir des compétences en conduite de projet et le développement d'application, nous réalisons en 5ème année le Projet de Recherche et Développement (**PRD**) qui s'inscrit dans la formation dispensée à l'École Polytechnique de Tours et vise à consolider nos compétences.

D'où mon intérêt de réaliser mon **PRD** qui porte sur l'Ordonnancement et affectation dans des systèmes distribués, ce projet a été proposé par l'équipe Recherche Opérationnelle, Ordonnancement et Transport (ROOT) du Laboratoire Informatique de Polytech Tours. Il a été encadré par Boukhalfa ZAHOUT, Ameer SOUHKAL et Patrick MARTINEAU qui représente la MOA. Abdoulaye kake, élève ingénieur en 5ème année de l'école Polytech Tours qui représente la MOE.

Ce document constitue les spécifications concernant le projet de recherche et développement "Ordonnancement et affectation dans des systèmes distribués" (cas multicritère et multi machine). Il contient les différentes caractéristiques du projet, les besoins, l'environnement du projet et les objectifs à réaliser. Il précise également les différentes réflexions réalisées afin de mettre en place le développement. Le **PRD** se déroule du 19 septembre 2018 jusqu'au début du mois d'avril 2019.

## 1 Contexte de la réalisation

### 1.1 Acteur du projet

- **La maîtrise d'œuvre (MOE)** : Kake Abdoulaye, étudiant en 5ème année à l'école d'ingénieur Polytech Tours, spécialité Système Informatique au département Informatique.
- **La maîtrise d'ouvrage (MOA)** : Boukhalfa Zahout doctorant, Patrick Martineau enseignant-chercheur et Ameer Souhkal enseignant-chercheur. Tous membres de l'équipe de recherche ROOT du laboratoire d'informatique de Polytech Tours. Ils sont spécialisés dans l'étude, la recherche et l'essai de méthode de résolution des problèmes d'ordonnancement et de planification.

## 1.2 Contexte et Enjeux

De nos jours, les systèmes distribués ont une place clé dans la vie quotidienne d'où l'avènement du Cloud comme service car nos besoins en performances ne cesse d'augmenter et que nos ordinateurs portables n'arrivent pas à suivre le rythme. C'est pourquoi, l'utilisation du cloud comme nouvelle alternative a été évidente, il permet de stocker des données et faire des tâches (calcul scientifiques, aide à la décision sur des volumes de données, ...), des tâches qu'un ordinateur ordinaire ne saurait faire.

La demande est tellement grande maintenant qu'il est nécessaire de trouver des méthodes d'affectation ou d'ordonnancement des tâches (jobs) sur chaque machine du cloud afin de satisfaire le maximum de demandes en respectant les capacités de la machine qui doit réaliser les tâches. Un job, peut être l'exécution d'un calcul ou d'une demande de machine virtuelle.

En prenant exemple sur les Cloud Data Center composés de clusters (des machines physiques) qui ont des ressources qui peuvent être partagées entre les machines virtuelles (ici représentent des jobs) soumises par les utilisateurs (agents), afin de répondre aux besoins de ces utilisateurs on s'occupe d'affecter ou d'ordonner les jobs sur les clusters avec pour but d'utiliser à bon usage les ressources des clusters et satisfaire les demandes des utilisateurs.

A la différence d'un problème d'ordonnancement classique qui consiste à organiser un nombres de jobs sur un ou plusieurs machines, en respectant des contraintes et les ressources disponible dans le but d'optimiser une fonction objectif. Un problème d'ordonnancement multi critère diffère d'un mono critère en optimisant plusieurs fonctions objectives au lieu d'une seule. La qualité de la solution est donc mesurée par plusieurs critères.

Nous nous intéressons dans ce projet au cas multi critère et multi machine qui fait allusion à plusieurs agents et plusieurs machines qui possèdent chacun des ressources (**CPU, RAM, DISQUE, ...**). Particulièrement, nous allons trouver une politique d'ordonner le nombre maximal des tâches (minimiser le nombre de tâches rejetées) dans un intervalle de temps donné. On vérifie que, pour chaque moment d'exécution et pour chaque type de ressource, la somme de ce type de ressource utilisée est inférieure ou égale à la quantité totale disponible sur une machine.

Pour la suite, nous considérons qu'il y n'a pas plus de deux agents (donc pas plus de deux critères différents). Pour chaque agent, il possède ses propres tâches à exécuter. Et puis chaque agent veut maximiser le nombre de ses tâches exécutées.

Plus généralement, le problème d'ordonnancement et d'affectation des jobs cas multi critère et multi machine se présente comme suit :

- Des machines  $m$  :
  1. Une machine est toujours disponible, avec un intervalle de temps :  $[0, +\infty[$
  2. Une machine a  $Q_{m,j}$  une quantité de ressource renouvelable de type  $R_{m,j}$ , avec  $j = 1, \dots, K$  et  $m$  le numéro de la machine
  3. Une machine peut exécuter plus d'un job/tâche à un moment donné tant que la consommation de ressources n'excède pas  $Q_{m,j}$
- $n$  jobs/tâches :
  1. un job  $J_i$  est exécuté durant un intervalle  $[s_i, c_i]$  où  $s_i$  et  $c_i$  sont la date de début et de fin. la durée d'un job sera :
 
$$p_i = c_i - s_i$$
  2.  $r_{i,j}$  est la quantité de ressource nécessaire à l'exécution d'un job  $J_i$  d'une ressource  $R_j$
  3. Tous les jobs sont indépendants et devrait être ordonnancé sans préemption.
  4.  $w_i$  est le poids d'un job  $J_i$ .
- Critères :

1. Nous avons deux agents A et B, chacun associé à un ensemble de job dénoté par  $N^A = J_1, \dots, J_{na}$  et  $N^B = J_{na+1}, \dots, J_n$  où  $n = n_A + n_B$
2. la fonction objective de chaque agent : de minimiser le nombre de job/tâche rejeté durant un temps donné.

$$\sum_{i \in N} x_i$$

où  $x_i$  est 1 si le job  $i$  est rejeté, 0 sinon.

### 1.3 Objectifs

Ce projet a pour objectif de proposer un algorithme permettant l'ordonnancement des tâches/jobs fixés dans le temps appartenant à deux agents (A et B), avec  $j$  type de ressources sur plusieurs machines.

On ne considère que deux agents A et B. Chaque agent possède ses propres tâches à exécuter. Nous allons séparer l'ensemble des tâches en deux sous-ensembles **A** et **B**. **A** est l'ensemble des tâches fournies par l'agent A. **B** est donc l'ensemble des tâches fournies par l'agent B. Ils ont le même but celui de minimiser le nombre de leurs tâches rejetés, donc nous avons deux fonctions objectives  $f_A$  et  $f_B$ .

Il est donc ici important de répartir les différents jobs de chaque agent sur différentes machines tout en minimisant le nombre de jobs rejetés (ou en maximisant le nombre de jobs exécutés). La minimisation des deux critères à la fois est NP-difficile. Une méthode exacte serait nécessaire et serait coûteuse en temps machine. Dans ce travail nous allons chercher les méthodes heuristiques. Ces méthodes ont un meilleur rapport qualité/temps de calcul que les méthodes exactes, même si elles ne garantissent pas l'optimalité, néanmoins elles fournissent rapidement une solution dite "approchée".

On est dans un cas multicritère. Nous allons chercher l'ensemble des solutions non dominées ou un ensemble représentatif de celui-ci : le **front de Pareto**. Ce front peut être obtenu à partir de la méthode exacte et des heuristiques.

Pour obtenir ce **front de Pareto** nous considérons l'approche **epsilon-contrainte**. Par cette approche, nous chercherons une solution non-dominée dite solution optimale de Pareto. Il s'agit de chercher une solution minimisant le critère de l'agent A tout en bornant supérieurement la valeur de la fonction objectif de l'agent B.

La liste des objectifs à réaliser est la suivante :

- Obtenir un modèle mathématique indexé temps permettant de trouver des solutions optimales (méthode exacte).
- Trouver différentes heuristiques permettant d'obtenir des résultats proches de la méthode exacte à notre problème avec un temps d'exécution bien plus faible.
- Implémenter les modèles mathématiques ainsi que les différentes heuristiques afin de les valider et mesurer les performances de résolution.
- Afficher le **front de Pareto** afin de voir les solutions permettant de choisir et prendre une décision.
- Implémenter un module de comparaison de solution afin de comparer les performances et les résultats obtenus en utilisant les heuristiques par rapport à la méthode exacte.

### 1.4 Hypothèses

Ce PRD est un projet principalement porté sur la recherche de solution à notre problème d'ordonnancement. De ce fait la partie de recherche demandera plus de temps que la partie

développement. La recherche de méthode exacte et d'heuristique par exemple est essentielle à notre projet. Le développement ne sera possible que si les recherches sont acceptés par le MOA et il est utile d'arrêter la phase de recherche suffisamment tôt afin d'avoir le temps de développer.

## 1.5 Base méthodologiques

Au cours de ce projet, la méthode de gestion de projet est la méthode agile ce qui implique un échange hebdomadaire avec l'encadrant pour valider les choix de conception et de développement. Concernant la programmation, une convention de nommage doit être respectée comme il existe déjà un projet initiale. Cette convention est de donner chaque variable, classe ou fonction un sens à partir de son nom et suivre la structure de code déjà existante. Les outils de projet utilisés seront la plateforme de gestion Trello, le diagramme de Gantt et l'outil de versionnage Git pour le développement.

# 2

## Description Générale

### 1 Environnement du Projet

Ce projet proposé par **Boukhalfa ZAHOUT, Ameer SOUHKAL et Patrick MARTINEAU** sur l'ordonnancement multicritère et multi machine est une suite logique liée au problème d'ordonnancement. De ce fait, le développement de ce projet fera suite à ce qui a déjà été produit et utilisera les mêmes technologies que pour le cas mono critère et multi machine.

Le développement se fera sur le système d'exploitation Linux, distribution Ubuntu et en deux applications :

- La première application qui contient la majorité des objectifs (la méthode exacte , quelques heuristiques, l'interface graphique) fait avec le langage de programmation C++ qui a été privilégié. Il est en effet plus logique d'utiliser le langage C++ dans un projet de ce type, car c'est un langage de bas niveau relativement performant. Enfin, un solveur est également utilisé pour rechercher les solutions optimales des différents problèmes à résoudre. Le solveur utilisé est Cplex216, c'est une bibliothèque permettant de réaliser une approche exacte du problème avec de la programmation linéaire. En plus il est facile de configuration et utilisable avec le langage c++. Le programme s'exécute en mode interactif avec une interface graphique qui sera programmé dans le langage QT qui est l'un des modules interfaces graphiques du langage C++.
- la deuxième applications qui contient le programme qui implémente l'algorithme génétique NSGA2 (la méta heuristique) . Cette application est faite en JAVA du fait de sa rapidité et garbage Collector qui s'occupe de la gestion de mémoire. Ce choix a été fait car cet algorithme consomme ou utilise beaucoup la mémoire et le c++ nécessite une bonne connaissance de la gestion de la mémoire.

### 2 Caractéristique des utilisateurs

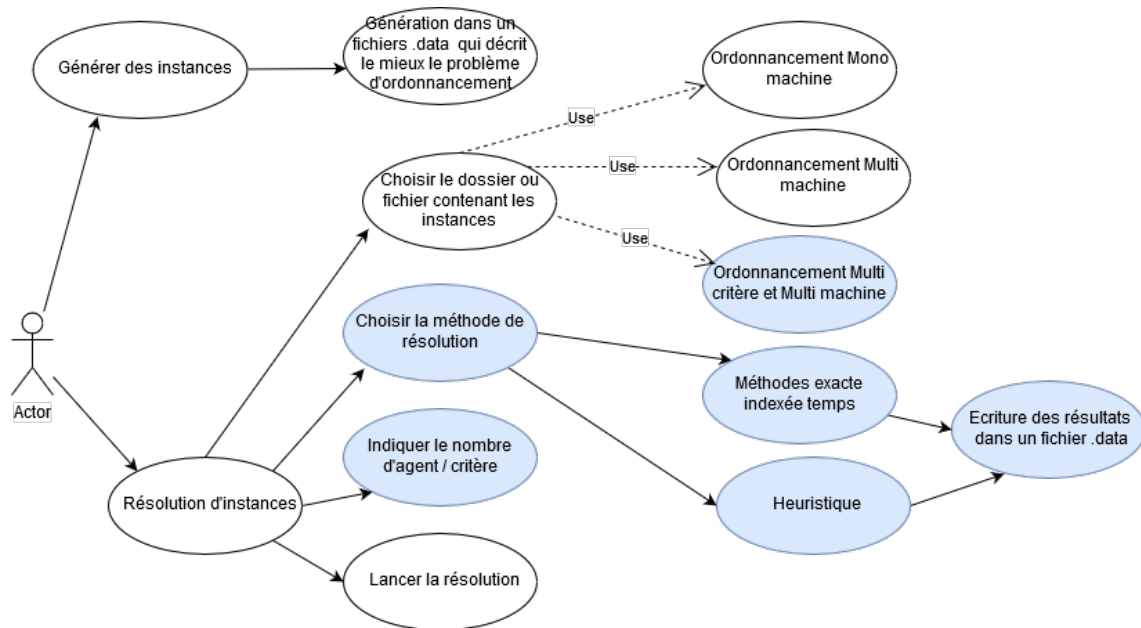
L'utilisateur principale de l'application sera les MOA qui possèdent de bonnes connaissances en informatique et en recherche opérationnelle. Ce système est développé pour donner les différents résultats des méthodes et permettre aux MOA de faire des tests sur un ensemble d'instance. Le but est d'analyser ces résultats et les comparer avec d'autre résultat. Donc l'utilisateur doit être

une personne qui connaît le domaine informatique et la théorie du problème d'ordonnancement et d'affectation des ressources de type multicritères.

### 3 Fonctionnalité du système

Ce système permet de résoudre un problème multi critère et multi machine, en choisissant une instance à résoudre contenant toutes les informations comme le nombre de ressources par machine, les machines avec la capacité de chaque ressources, les jobs avec la quantité de ressources à consommer pour chaque ressource et leur date de début et de fin.

Le diagramme de cas d'utilisation, montre les fonctionnalités qui existent déjà avec le programme précédent et celle que nous allons ajouté ou modifié avec la couleur bleu **Figure 1**.



**Figure 1** – Diagramme de cas d'utilisation du programme

L'utilisateur passe par une interface graphique, dans la quelle il a un menu pour générer une instance, un autre pour résoudre une instance qu'il a crée. Pour cela, il doit choisir le chemin du fichier, entrer le nombre d'agent, puis sélectionner une ou des méthodes de résolution, et enfin la lancer le traitement qui va créer un fichier de sortie contenant les résultats et le front de Pareto. Comme le montre le diagramme de séquence **Figure 2**

Après le traitement, on obtient pour chaque méthode un fichier de sortie contenant le front de Pareto. Puis nous lisons, les fichiers de sorties avec les coordonnées du front de Pareto x et y qui des solutions non-dominées pour comparer les différents fronts de Pareto et évaluer la performance des différents algorithmes, comme le montre ce diagramme de cas d'utilisation **Figure 3**.

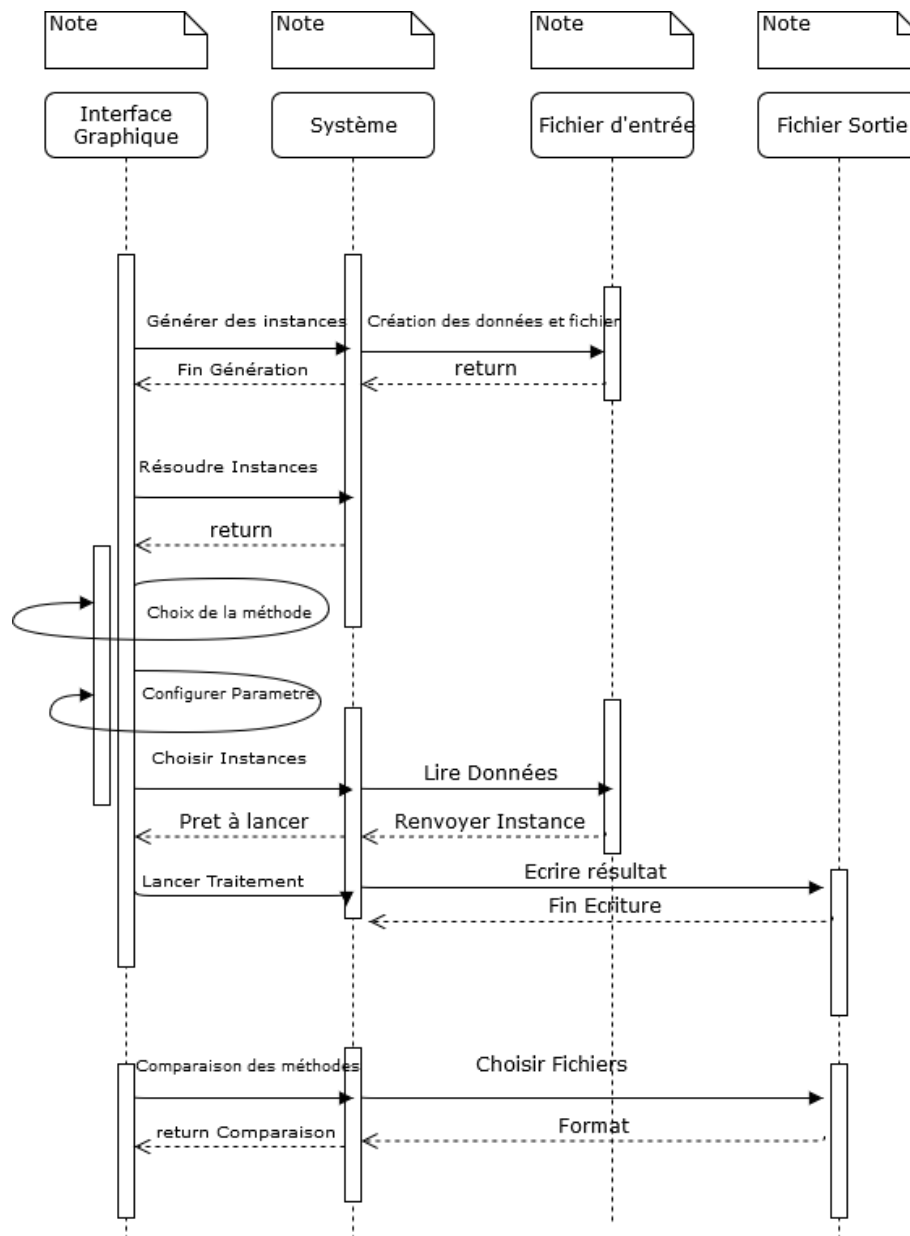


Figure 2 – Diagramme de séquence lors du programme

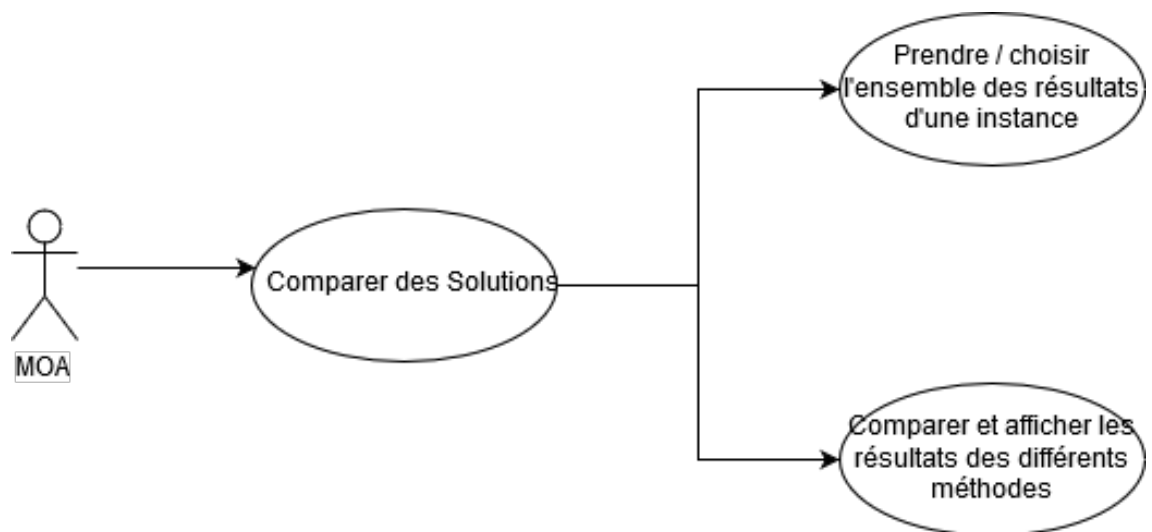


Figure 3 – Diagramme de cas d'utilisation de la comparaison

# 3

## État de l'art/veille

Il est question dans tous les cas d'un problème d'ordonnancement, le problème réside sur le fait d'affecter des jobs ne pouvant pas être déplacés et qui nécessitent une certaines quantités de ressources pour fonctionner.

### 1 Étude de la documentation

#### 1.1 Cas Mono critère - Mono machine

La lecture de la publication de **Boukhalfa ZAHOUT, Ameer SOUKHAL et Patrick MARTINEAU** à propos du cas mono machine, montre qu'il est possible de trouver une solution optimale pour le cas mono machine avec une résolution exacte NP-Difficile ce qui veut dire qu'elle demande du temps de calcul qui croît de façon exponentielle en fonction de la taille des données d'entrées.

Afin de trouver une solution optimale au problème, il y a deux approches : Une première approche, celle temporelle permet de trouver une solution exacte :

- $x_i$  une variable binaire égale à 1 si le job  $J_i$  est rejeté, 0 sinon.
- $y_{i,t}$  une variable binaire égale à 1 si le job  $J_i$  est exécuté au temps  $t$ , autrement elle vaut 0.

La Formulation mathématique du problème est la suivante :

$$\text{Minimiser : } \sum_{i \in N} x_i$$

$$\text{Contrainte : } \sum_{t=s_i}^{c_i} y_{it} = (c_i - s_i) * (1 - x_i); \forall i \in N$$

$$\sum_{i=0}^N y_{it} * r_{i,j} \leq Q_j; \forall j \in R; \forall t \in T$$

$$x_i \in \{0, 1\} \text{ et } y_{it} \in \{0, 1\}; \forall i \in N; \forall t \in T$$

\* La fonction objective permet de minimiser le nombre de jobs rejetés.

- \* La première contrainte permet de vérifier que si un job  $J_i$  n'est pas rejeté alors il est affecté durant sa période d'exécution  $[s_i, c_i]$  ce qui implique que  $y_{it}$  vaut 1 pour tout  $t$  appartenant à  $[s_i, c_i]$
- \* La seconde contrainte cherche à vérifier qu'il n'y a pas plus de ressources  $r_{ij}$  consommés sur la machine à l'instant  $t$  que la valeur de la ressource  $Q_j$  pour toutes les ressources et pour tous les jobs devant être exécuté à l'instant  $t$ .

Comme la première approche (méthode exacte : [Section 1.1](#)) peut prendre du temps en fonction de la taille des données, il a été judicieux de trouver une deuxième approche qui donne des résultats qui se rapprochent de la solution exacte. La deuxième approche (une heuristique) est basé sur la création de sous ensemble de jobs possédant un maximum de jobs se chevauchant. Chaque sous ensemble ne pouvant pas être contenu dans un autre.

L'algorithme de création des sous ensembles :

```

Data: an instance I
Result: Maximal Subset
begin
  - Sort the events  $e_h$  from the earliest to the latest (in
    case of ties, finish times come first)
  - Set  $J = \emptyset$  (current subset of overlapping jobs)
  - Set  $k = 0$  (counter of maximal subsets)
  - Set  $nb_{start.event} = 0$  (counter of number start
    event)
  for ( $h = 1$  to  $2n$ ) do
    if ( $e_h$  is a start event) then
       $nb_{start.event} = nb_{start.event} + 1$ 
       $J_k = J_k \cup \{job_h\}$ 
    else
      if ( $nb_{start.event} = n$ ) then
         $J_{k+1} = J_k - \{job_h\}$ 
        return ( $J_h = J_1 \dots J_k$ )
      else
         $J_{k+1} = J_k - \{job_h\}$ 
         $k = k + 1$ 
      end
    end
  end
end

```

Figure 1 – Algorithme pour obtenir les sous ensembles maximaux

Ce qui donne une expressions mathématique indexée job :

- $x_i$  est une variable binaire égale à 1 si le job  $J_i$  est rejeté, 0 sinon.
- $J_h$  est le sous-ensemble maximal  $h$  de jobs se chevauchant.

$$\text{Minimiser : } \sum_{i \in N} x_i$$

$$\text{Contrainte : } \sum_{l \in j_h} x_l * r_{l,j} \leq Q_j; \forall j \in R; h = 1, \dots, k$$

$$x_i \in \{0, 1\}; \forall i \in N;$$

- \* La fonction objective est de minimiser le nombre de jobs rejetés.
- \* La contrainte permet de vérifier que pour chaque sous ensemble  $h$  de jobs on ne dépasse pas la quantité disponible de ressource  $Q_j$  où  $r_l, j$  est la quantité de ressource  $R_j$  nécessaire pour exécuter un job  $i$  dans le sous ensemble  $L$ .

Il y a eu aussi d'autres algorithmes qui ont été développés afin de trier les jobs (mettre un ordre de passage dans la liste de jobs) avant leur affectation sur une machine.

Les Algorithmes utilisés et énoncés :

**Shortest Processing Time (SPT) :** les jobs sont triés par ordre croissant de temps d'exécution  $p_i = c_i - s_i$ . En cas d'égalité, le job se terminant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique (ordre défini sur des suites finies d'éléments d'un ensemble ordonné, ici le numéro du job).

**Capacity-makespan (CC max) :** les jobs sont triés par ordre croissant d'espace occupé. En cas d'égalité, le job se terminant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique. L'idée de CC max est de minimiser l'espace pris par les jobs.

**Average Ressources Consummed (ARC) :** les jobs sont triés par ordre décroissant des ressources nécessaires par unités de temps entre  $[s_i, c_i]$ . En cas d'égalité, le job se terminant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique. ARC minimise la consommation moyenne de ressource par jobs par unités de temps.

Il est ressorti que l'algorithme **CC max** est la plus efficace grâce aux résultats obtenus sur des jeux de données avec le nombre de job variant de 20 à 100. Pour des raisons d'amélioration c'est possible d'appliquer chaque algorithme à chaque sous ensemble de job avant les affectations sur machine.

## 1.2 Cas Mono critère - Multi machine

Cet problème a été fait par mon prédécesseur **Pierre Fervault "PRD Ordonnancement et affectation des ressources dans le cloud Computing"** lors de son PRD. Dans son rapport, il s'attaque à la résolution du cas de l'ordonnancement des jobs sur plusieurs machines en partant du principe d'avoir une liste de jobs avec  $n$  le nombre de jobs ainsi qu'une liste de machines parallèle avec  $m$  le nombre de machines parallèles. Chaque machine possède  $R_j$  unité de ressource renouvelable, nécessaire pour l'exécution des jobs. Un job à un instant de départ  $s_i$  et un instant de fin  $f_i$  et avec une valeur  $v_i$  qui est la priorité du job  $J_i$ .

Et chaque machine peut exécuter plusieurs jobs tant que leur exécution ne consomme pas ou ne dépasse pas les ressources disponibles sur la machine. Avant d'arriver à une modélisation mathématique du problème, sa lecture du document écrit par **Enrico Angelelli, Nicola Bianchessi, Carlo Filippin** en 2014 qui se base sur l'ordonnancement des jobs multi machine avec chaque machine ayant une seule ressource renouvelable l'aide à ce mettre sur une voie.

Afin de résoudre le problème d'ordonnancement plus rapidement, des heuristiques ont été essayées :

**Stand alone job-sorting :** Il s'agit de classer les jobs selon des critères spécifiques de tri et de les insérer dans une liste. Ensuite, les jobs sont affectés sur la première machine (en vérifiant bien sûr que la valeur de ressources n'est pas dépassée) et chaque job affecté sur la première machine est enlevé de la liste. La même procédure est réalisée sur les autres machines afin d'ordonnancer un maximum de jobs. Quatre algorithmes ont été définis :

- JS1 classe les jobs par ordre décroissant selon  $v_i$  (priorité du job) , puis par ordre croissant selon  $r_j$  et enfin par ordre croissant selon  $d_i$  (durée du job)
- JS2 classe les jobs par ordre décroissant selon  $v_i/d_i$ , puis par ordre croissant selon  $r_j$  et enfin par ordre croissant selon  $d_i$  (durée du job).
- JS3 classe les jobs par ordre décroissant selon  $v_i/r_j$  , puis par ordre croissant selon  $r_j$  et enfin par ordre croissant selon  $d_i$  (durée du job).
- JS4 classe les jobs par ordre décroissant selon  $v_i/(r_j.d_i)$ , puis par ordre croissant selon  $r_j$  et enfin par ordre croissant selon  $d_i$ (durée du job).

**Maximal-set based job sorting :** Il s'agit ici de trier les jobs dans des sous-ensembles maximaux  $J_h$  de manière lexicographique en prenant en compte un ensemble de valeur fixé. Ensuite, les jobs sont classés à l'intérieur de chaque sous-ensemble maximal en prenant compte des critères en particulier. On obtient une liste de jobs en prenant les jobs de manière séquentielle suivant l'ordre de tri précédemment choisi. Les occurrences multiples d'un même job sont ignorées. Dans le cas où plusieurs critères de tris sont utilisés, l'algorithme les réalise de manière itérative et retient la meilleure solution.

Trois critères de tris sont utilisés ici :

- Tri suivant  $v_h$  (somme des  $v$  des jobs de  $J_h$  ) dans l'ordre décroissant.
- Tri suivant  $v_h/|J_h|$  dans l'ordre décroissant.
- Tri suivant  $v_h/R_h$  dans l'ordre décroissant.

Finalement on prend le meilleur des trois algorithmes : BestGreddy MS Au final, le meilleur algorithme est défini comme étant  $\max(\text{BestGreddy JS Stand alone}, \text{BestGreddy MS Maximal-set})$ .

Comme pour le cas mono machine, une formulation mathématique a été fait afin de trouver une solution exacte.

- $x_{im}$  une variable binaire égale à 0 si le job  $J_i$  est rejeté sur la machine  $m$  , 1 sinon.
- $y_{imt}$  une variable binaire égale à 1 si le job  $J_i$  est exécuté au temps  $t$  sur la machine  $m$  , autrement elle vaut 0.

$$\text{Minimiser : } \sum_{i \in N} \sum_{m \in M} x_{im}$$

$$\text{Contrainte : } \sum_{t=s_i}^{f_i} y_{imt} = (f_i - s_i) * (1 - x_{im}); \forall i \in N; m \in M$$

$$\sum_{i=0}^N y_{imt} * r_{ijm} \leq Q_{jm}; \forall j \in R; \forall m \in M; \forall t \in T$$

$$\sum_{m=0}^M x_{im} \leq 1; \forall i \in N$$

$$x_{im} \in \{0, 1\} \text{ et } y_{imt} \in \{0, 1\}; \forall i \in N; m \in M; \forall t \in T$$

- \* La fonction objectif : permet de minimiser le nombre de jobs rejetés, c'est à dire maximiser le nombre de jobs ordonnancés sur l'ensemble des machines.
- \* La première contrainte permet de vérifier que si un job  $J_i$  n'est pas rejeté alors il est affecté durant sa période d'exécution  $[s_i, f_i]$  sur une machine  $m$  ce qui implique que  $y_{imt}$  vaut 1 pour tout  $t$  appartenant à  $[s_i, f_i]$ .

- \* La seconde contrainte cherche à vérifier qu'il n'y a pas plus de ressources  $r_{ijm}$  (quantité de ressource dont à besoin un job  $i$  sur une ressource  $j$  dans une machine  $m$ ) consommés sur la machine  $m$  à l'instant  $t$  que la valeur de la ressource  $Q_{jm}$ .
- \* La troisième contrainte permet de vérifier qu'un job n'est exécuté que sur une seule machine et unique une machine.

Après avoir une méthode de résolution temporelle, il utilise le même algorithme de création de sous ensemble maximal de jobs que le cas mono critère et mono machine [Section 1.1](#) mais la formulation mathématique change :

$$\text{Minimiser : } \sum_{i \in N} \sum_{m \in M} x_i$$

$$\text{Contrainte : } \sum_{l \in j_h} x_{lmt} * r_{ljm} \leq Q_{jm}; \forall j \in R; \forall m \in M$$

$$\sum_{m=0}^M x_{im} \leq 1; \forall i \in N$$

$$x_{im} \in \{0, 1\}; \forall i \in N; m \in M$$

- \* Tout comme le modèle temporel, la fonction objective est de minimiser le nombre de jobs non ordonnancés sur la machine  $m$  alors  $x_{im}$  vaut 0 si le job  $i$  est exécuté sur la machine  $m$ , sinon 1.
- \* La première contrainte permet pour chaque sous-ensemble maximal de jobs, il vérifie la consommation  $r_{ljm}$  ne dépasse la capacité totale  $Q_{jm}$  pour toutes les ressources  $j$  appartenant à  $R$ , toutes les machines  $m$  appartenant à  $M$  et pour tous les sous-ensembles maximaux de jobs  $J_h$ , avec  $h$  entre 1 et  $k$ .
- \* La deuxième contrainte permet de vérifier qu'un job n'est exécuté que sur une seule machine.

Après ces formulations mathématiques du problème, nous voyons que l'ordonnancement multimachine peut être résolue avec des méthodes exactes mais le problème est NP-Difficile, c'est pourquoi nous utilisons des heuristiques pour s'approcher de la solution optimale avec un temps de calcul plus faible. D'où l'importance des algorithmes de tri pour obtenir une liste de jobs ordonné d'une instance à résoudre. La liste des méthodes de tries :

**CC max :** L'ensemble des jobs de l'instance sont triés par ordre croissant d'espace occupé  $\sum_{j \in R} r_{ij} * (f_i - s_i)$ . En cas d'égalité, le job se terminant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique. L'idée de CC max est de minimiser l'espace pris par les jobs. Cette méthode de trie a été testée dans la publication mono-machine et ils'avère que sur l'ensemble des méthodes testées dans la publication, c'est elle qui nous permet d'obtenir les résultats les plus proches des solutions optimales. Cela est sans doute lié au fait que cette méthode de tri prend en compte à la fois le temps pendant lequel les jobs s'exécutent ainsi que la quantité de ressources que chaque job utilise.

**Quantité de ressources instantanées (Somme des valeurs de ressources) :** Ici, nous trions les jobs par ordre croissant suivant la valeur totale de ressources utilisées  $\sum_{j \in R} r_{ij}$ , en cas d'égalité, les jobs sont triés par ordre croissant suivant leurs durées d'exécutions  $(f_i - s_i)$ , sinon, on se réfère à l'ordre lexicographique. L'idée ici est de réduire la possibilité qu'un job utilisant une valeur très importante de ressources soit ordonnancé. Contrairement au CC max, cette méthode ne prend en compte que les valeurs de ressources des jobs sans tenir compte du temps d'exécution mais cela permet d'éviter qu'un job demandant une quantité de ressources très importantes sur une période courte soit ordonnancé et favorise l'ordonnancement de jobs moins coûteux.

**Quantité moyenne de ressources par sous-ensembles maximaux :** Nous recherchons les sous-ensembles maximaux à partir de l'algorithme que nous avons vu précédemment, puis nous les trions par ordre croissant suivant la quantité moyenne de ressources : La somme des ressources consommées par sous ensemble  $J_h$  / (nombre de jobs de  $J_h$ ). La liste des jobs triés est obtenue en prenant les jobs de chaque sous-ensemble maximal et en les ajoutant à la liste des jobs à ordonnancer. Si un job contenu dans un sous-ensemble maximal fait déjà partie de la liste à ordonnancer, il n'est pas à ajouté. Cette méthode est inspirée des heuristiques de la publication d'ordonnancement multi-machine à ressource unique. Il s'agit de la partie maximal-set based job sorting . Ici, nous cherchons à privilégier les sous-ensembles maximaux ayant une quantité de ressources moyenne la plus faible possible.

Ces méthodes permettent d'avoir une liste de jobs ordonné en premier lieu avant leur affectation à une machine.

En ce qui concerne la méthode d'affectation il y a deux méthodes :

- On affecte le maximum de jobs sur une première machine et on renouvelle l'opération sur la machine suivante.
- On affecte chaque job sur la machine étant la moins chargée, en cas d'égalité , l'ordre lexicographique est utilisé.

### 1.3 Cas Multi critère - Mono machine

Toujours dans le problème d'ordonnancement mono machine avec un paramètre de plus celui de plusieurs critères à prendre en compte. Avant nous résolvons le cas avec un seul critère. Et en générale un critère peut être un facteur ou un utilisateur avec une liste de tâches à réaliser ce qui explique le cas suivant le fait d'avoir plusieurs utilisateurs ayant des tâches à affecter sur une machine. Le but essentiel est de diminuer le nombre de job rejeter par utilisateur tout en satisfaisant tous les autres utilisateurs.

Le rapport de **Boyang Wang** "Politique d'accès concurrentiel aux ressources partagées pour la minimisation du nombre de jobs rejetés" qui a fait son PRD sur ce problème , m'a permis de mieux comprendre le cas multicritère sur une machine et la combinaison avec la cas multi-machine de Pierre me permettra de résoudre le problème lié à mon projet de recherche et de développement.

Revenons sur le cas multi critère mono machine , comme dit précédemment un critère est un agent (un utilisateur) qui peut soumettre plusieurs tâches à exécuter. Pour chaque tâche, elle nécessite plusieurs types de ressources partagés pour son traitement. Elle aussi possède une date de début fixée et une date de fin fixée. Si la tâche ne peut pas être exécutée dès le moment même que son date de début, elle va être rejetée.

Un problème d'ordonnancement multi critère diffère d'un mono critère car elle optimise plusieurs fonctions objectifs au lieu d'une seule. La qualité de la solution est donc mesurée par plusieurs critères.

Plus généralement, le problème multi-critères peut se présenter comme  $\alpha|\beta|f_k$  :

- On note  $\alpha$  l'environnement (nombre de machines).
- On note  $\beta$  l'ensemble des contraintes.
- On note  $f_k (1 \leq k \leq K \text{ avec } K \geq 2 \text{ une fonction objectif à partir d'un agent } )$ .

S'il n'y a que deux agents A et B ( $K=2$ ) , on note  $f_A$  la critère de agent A et  $f_B$  le critère de l'agent B. Dans ce cas, la résolution du problème est sur une seule machine. Donc  $\alpha$  est égale à 1.

La Formulation mathématique du problème est la suivante :

- type de booléen avec 0 si la tâche  $J_i$  est rejetée, sinon 1
- type de booléen avec 1 si la tâche  $J_i$  est exécutée en moment de  $t$ , sinon 0

$$\text{Minimiser : } \sum_{i \in N} x_i$$

$$\text{Contrainte : } \sum_{t=s_i}^{f_i} y_{it} = (f_i - s_i) * (1 - x_i); \forall i \in N$$

$$\sum_{i=0}^N y_{it} * r_{i,j} \leq Q_j; \forall j \in R; \forall t \in T$$

$$x_i \in \{0, 1\} \text{ et } y_{it} \in \{0, 1\}; \forall i \in N; \forall t \in T$$

- \* La fonction objective permet de minimiser le nombre de jobs rejetés.
- \* La première contrainte permet de vérifier que si un job  $J_i$  n'est pas rejeté alors il est affecté durant sa période d'exécution  $[s_i, f_i]$  ce qui implique que  $y_{it}$  vaut 1 pour tout  $t$  appartenant à  $[s_i, f_i]$
- \* La seconde contrainte cherche à vérifier qu'il n'y a pas plus de ressources  $r_{i,j}$  consommés sur la machine à l'instant  $t$  que la valeur de la ressource  $Q_j$  pour toutes les ressources et pour tous les jobs devant être exécuté à l'instant  $t$ .

Cette formulation n'est pas différente de celle du cas mono machine, comme la suite l'indique, l'objectif est de calculer la fonction objective selon chaque critère puis trouver des points de compromis afin de satisfaire tous les critères.

Nous cherchons l'ensemble des solutions non-dominés ou un ensemble représentatif de celui-ci : le **front de pareto**. La vision générale est de construire un **front de pareto** optimale pour résoudre le problème multicritère. Afin d'atteindre, il y a des méthodes qui se sont distinguées et qui sont facile de mettre en pratique :

**Combinaison linéaire :** L'idée de cette méthode est de construire une combinaison linéaire avec les fonctions objectives provenant de chaque critère. On note une fonction objective comme  $f_k$  avec  $k \in K$ , pour  $K$  critères. La formule se généralise comme :

$$\sum_{k=1}^K a_k f^k$$

Ici  $a_k$  est un coefficient pour la fonction objectif  $f^k$ . De là nous avons une seule fonction objective et on peut varier les valeurs de  $a_k$  pour trouver un front de pareto optimale.

**l'algorithme L'epsilon Contrainte :** Pour déterminer une solution non-dominée, on utilise la formulation PLNE précédente où nous devons minimiser une fonction objective :

$$f^k = \sum_{i=0}^N x_i$$

Plus les autres contraintes (1 et 2), on ajoute une nouvelle contrainte :

$$f^k \leq Q_a$$

Par exemple pour le cas de 2 agents : on peut calculer la fonction objective de l'agent B  $f_b$  puis calculer la fonction objective de A en ajoutant aux contraintes déjà existante, une nouvelle  $f_b \leq Q_b$  qui veut dire le nombre total de jobs rejetés de l'agent B ne doit pas dépasser la valeur donnée  $Q_b$ . La manipulation de cette epsilon valeur permet de construire un front de pareto optimale.

Puis nous avons une méthode approché pour résoudre le problème multicritère sur une machine :

**l'algorithme de Greedy :** D'après M.Boukhalfa Zahout dans le papier "Fixed interval multi-agent scheduling problem with rejected costs", l'algorithme Greedy avec le cas multicritères se déroule dans l'ordre suivante :

- Si la méthode exacte  $\epsilon$  – contrainte approche est utilisée, les tâches de chaque agent vont être triées selon une règle de tri. Et on peut obtenir, pour chaque agent, une séquence de ses propres tâches trié. Maintenant on considère qu'il y a deux agents A et B. Tout d'abord, on va essayer d'ordonnancer les tâches d'agent A en respectant la contrainte  $f_A \leq \epsilon$ . Le choix de tâche suit l'ordre dans la séquence des tâches d'agent A trié. Si une tâche ne peut pas être ordonnancée, elle va être rejetée. Après avec une solution temporaire, on va essayer d'ordonnancer les tâches d'agent B en minimisant la fonction objectif  $f_B$ .
- Si la méthode exacte Combinaison Linéaire est utilisée, on va trier toutes les tâches de tous les agents selon la règle de tri pré-définie. On peut obtenir une séquence de toutes les tâches triées. Après on va essayer d'ordonnancer les tâches selon leur priorité par ordre. Si on associe un poids de rejet  $w_i$  pour chaque tâche  $J_i$ , on propose ces deux règles de tri :

**Weighted Shortest Processing Time First (WSPT) :** Les tâches sont triées selon le calcul  $(f_i - s_i)/w_i$  dans l'ordre non décroissant. En cas d'égalité, la tâche avec la plus petite date de fin va être exécutée en priorité.

**Weighted Capacity-Makespan (WCM) :** les tâches sont triées selon leur espace occupé divisé par le poids de tâche dans l'ordre non décroissant. L'espace occupé peut se produire avec cette formule de calcul :

$$\sum_{j=0}^R r_{ij} * ((f_i - s_i)/w_i) \quad \forall i \in N$$

En cas d'égalité, la tâche avec la plus petite date de fin va être exécutée en priorité.

### 1.3.1 Définition d'un Front de Pareto

Pour un problème multi critère ou multi Objectif nous avons plusieurs solutions obtenus car nous avons plusieurs fonctions objective qui nous donne des résultats. Delà vient la difficulté de choisir une solution optimale pour satisfaire tous les critères.

D'où la définition d'un front de Pareto, il est toujours utilisé dans les problèmes de type multi critère, c'est une allocation des ressources pour laquelle il n'existe pas une alternative dans la quelle tous les acteurs seraient dans une meilleur solution. Il permet d'avoir un plan dont les axes sont les différents critères  $A_1, \dots, A_k$  pour K critère et les points sont formés à partir des solutions obtenus de chaque fonction objective , un point T aura des coordonnées comme ceci :  $T : \{x_{A_1}^1, x_{A_2}^2, \dots, x_{A_k}^k\}$ .

A travers ce plan une courbe peut être formé et permettre la prise décision. Au sein d'une solution de Pareto, nous avons trois types :

- la solution de Pareto optimale : Une solution  $(x, y)$  est dite Pareto optimale ou non dominée s'il n'existe pas de solution  $(x', y')$  tel que :  $x \geq x'$  et  $y > y'$  ||  $x > x'$  et  $y \geq y'$ .
- La solution de Pareto faible.
- la solution de Pareto dominée : pour tout  $i \in I$ ,  $x \geq y$  et pour un  $i \in I$ ,  $x$  est strictement supérieur à  $y$ .

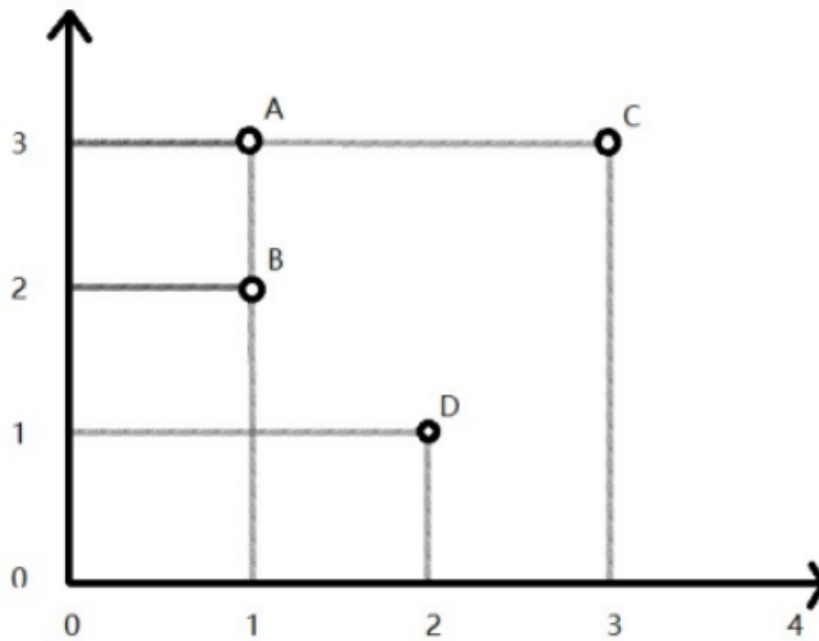


Figure 2 – Exemple de Solution de Pareto

Le graphe **Figure 2**, nous pouvons mieux cerner le front de Pareto. Dans ce graphe, on a les Points A, B, C et D qui représente quatre solutions de Pareto.

- la solution A est faible
- la solution B est optimale
- la solution C est dominée
- la solution D est optimale

D'après ce graphe, on peut constater que pour les points B et D, on a  $f_1(B) < f_1(D)$ , mais  $f_2(B) > f_2(D)$ . C'est-à-dire que pour la première fonction objectif, la solution B est plus performante mais pour la deuxième fonction objectif, la solution D est plus performante. Plus précisément, B ne domine pas D et D ne domine pas B. Donc B et D sont les solutions de Pareto optimale.

La solution A est une solution de Pareto faible car il existe une solution B avec  $f_1(B) = f_1(A)$  et  $f_2(B) < f_2(A)$ .

Après, si on fait la liaison de toutes les solutions de Pareto optimales, on peut obtenir une courbe dans le graphe qui s'appelle le front de Pareto optimale.

Sur le front de Pareto optimal, toutes les solutions sont des solutions de Pareto optimales. Et donc une solution sur le front de Pareto optimal ne domine pas une autre solution qui est aussi sur le front de Pareto optimal. Toutes les solutions sur le front de Pareto sont faisables. Ce front de pareto est toujours utilisé pour donner un ensemble de résultats. Après, le décideur peut faire son décision à partir de ces solutions de Pareto.

## 2 Étude de l'existant

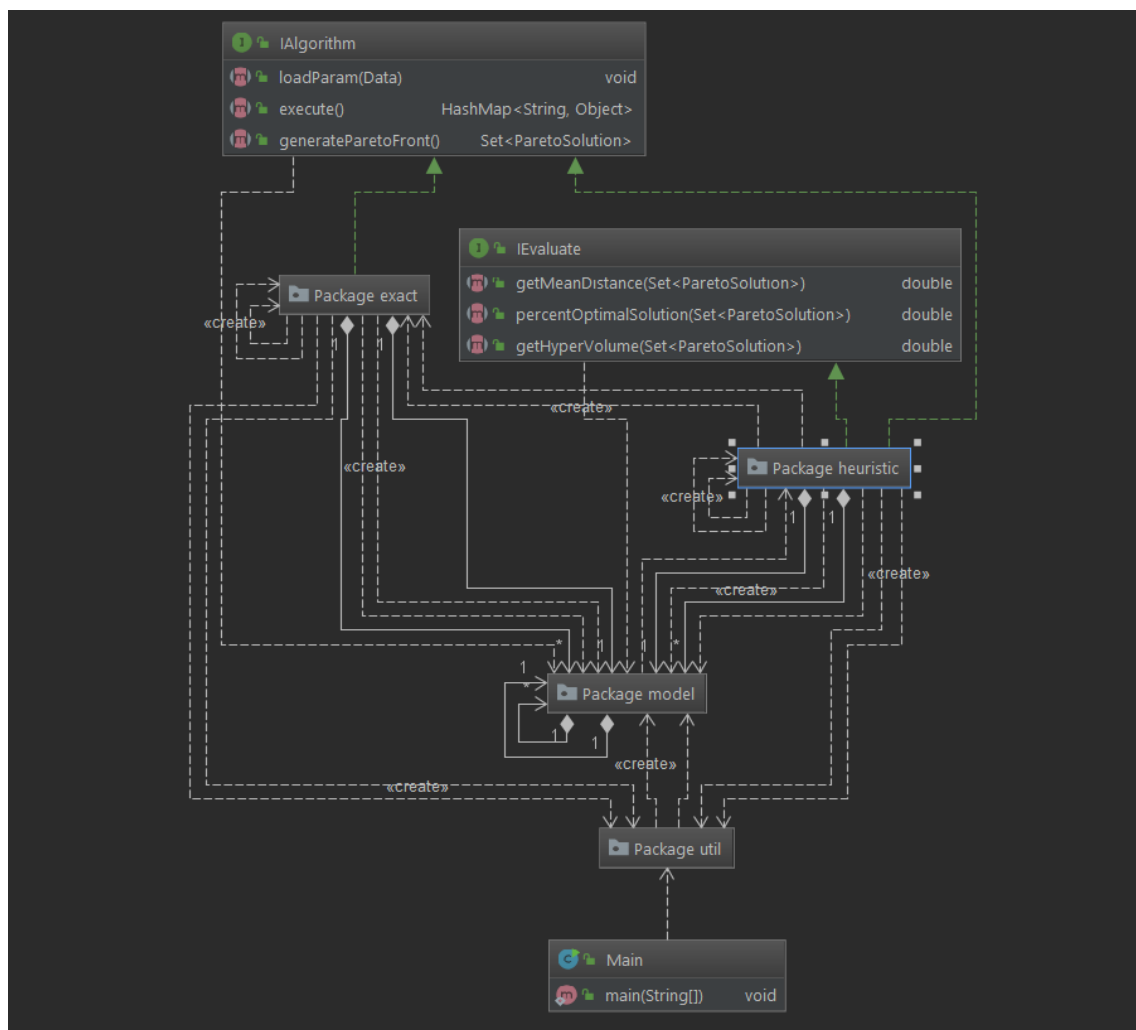
Pour réaliser mon PRD, je pars d'un existant de programme fourni d'un précédent PRD réalisé par Boyang Wang sur le problème d'ordonnancement et affectation cas mono machine et multi critère. Donc il y a une structuration de code qui existe dans lequel Boyang répond à son problème, tout en laissant une possibilité d'évolution du programme afin de répondre à d'autre problème d'ordonnancement et affectation.

Dans la suite, je vais utiliser son programme qui bien que , est bien structuré ne me permet pas d'incorporer facilement mes solutions. Le programme sera modifié afin qu'il réponde à mes solutions.

Cette amélioration se base sur l'ajout des méthodes , classes et la création d'une interface graphique qui prendra en compte l'ordonnancement et affectation cas multicritère et multi machine. la description de cette amélioration est faite dans la suite de ce document.

Le rapport “Politique d'accès concurrentiel aux ressources partagées pour la minimisation du nombre de jobs rejetés” de Boyang Wang permettra de mieux comprendre le programme existant.

l'architecture du programme **Figure 3** existante sur le quelle nous devons faire évoluer.



**Figure 3** – *Architecture du programme du PRD de Boyang Wang.*

Où le package :

- Exact : contient l'implémentation de la méthode exacte ( l'épsilon contrainte)
- Heuristic : contient l'implémentation des heuristiques (Greedy) et l'algorithme génétique NSGA2 pour le cas monomachine et multi critère.
- Model : contient l'implémentation de la modelisation du problème, ce sont les classes Jobs, instances, Machine , Pareto Solution.
- Util : contient le parseur du fichier d'instance.

# 4

## Analyse et Conception

Afin de résoudre ce problème, nous allons prendre en compte la formulation du cas monocritère - multi machine et cas multicritère - mono machine, la combinaison de ces deux cas et mes recherches. Dans un premier temps, je me suis basé sur la résolution exacte du problème d'ordonnancement multicritère - multi machine avec plusieurs ressources.

Pour cela deux genres de méthodes existent :

- Méthodes exactes : on peut obtenir le front de Pareto exact en utilisant une formulation mathématique, mais le temps d'exécution est très coûteux en fonction des données d'entrées.
- Méthodes approchées (Heuristique et Méta-Heuristique) : on peut obtenir le front de Pareto avec une méthode approchée, elle est plus rapide mais la qualité du résultat n'est pas assurée. Ces méthodes se basent sur les algorithmes heuristiques et génétiques.

### 1 Méthode exacte

Afin de résoudre ce problème, nous allons prendre en compte la formulation qui se trouve dans l'état de l'art du cas mono critère - multi machine et cas multicritère - mono machine, la combinaison de ces deux cas et mes recherches me permettent de définir une formulation de mon problème. Dans un premier temps, je me suis basé sur la résolution exacte du problème d'ordonnancement multicritère - multi machine avec plusieurs ressources.

Avant cela, la description du contexte reste primordiale :

- Plusieurs machines  $m$  :
  1. Une machine est toujours disponible, avec un intervalle de temps :  $[0, \infty[$
  2. Une machine a  $Q_{m,j}$  une quantité de ressource renouvelable de type  $R_{m,j}$ , avec  $j = 1, \dots, K$  et  $m$  le numéro de la machine.
  3. Une machine peut exécuter plus d'un job/tâche à un moment donné tant que la consommation de ressources n'excède pas  $Q_{m,j}$
- $n$  jobs/tâches :
  1. un job  $J_i$  est exécuté durant un intervalle  $[s_i, c_i]$  où  $s_i$  et  $c_i$  sont la date de début et de fin. la durée d'un job sera :  $p_i = c_i - s_i$ .
  2.  $r_{i,j}$  est la quantité de ressource nécessaire à l'exécution d'un job  $J_i$  d'une ressource  $R_j$ .
  3. Tous les jobs sont indépendants et devrait être ordonnancé sans préemption.

— Critères :

1. Nous avons deux agents A et B, chacun associé à un ensemble de job dénoté par  $N^A = \{J_1, \dots, J_{na}\}$  et  $N^B = \{J_{na+1}, \dots, J_n\}$  où  $n = n_A + n_B$
2. la fonction objective de chaque agent : de minimiser le nombre de job/tâche rejeté durant un temps donné.

$$\text{Min}Z = \sum_{i \in N} x_i$$

où  $x_i$  est 1 si le job  $i$  est rejeté, 0 Sinon.

A partir de là, je me suis fait une formulation mathématique du problème (un PLNE) :

- $x_{im}$  une variable binaire égale à 0 si le job  $J_i$  est rejeté, 1 si le job  $i$  est ordonnancé sur la machine  $m$ .
- $y_{imt}$  une variable binaire égale à 1 si le job  $J_i$  est exécuté au temps  $t$  sur la machine  $m$ , autrement elle vaut 0.

$$\text{Minimiser : } \sum_{i \in N} \sum_{m \in M} x_{im}$$

$$\text{Contrainte : } \sum_{t=s_i}^{f_i} y_{imt} = (f_i - s_i) * (1 - x_{im}); \quad \forall i \in N; \quad m \in M$$

$$\sum_{i=0}^N y_{imt} * r_{ijm} \leq Q_{jm}; \quad \forall j \in R; \quad \forall m \in M; \quad \forall t \in T$$

$$\sum_{m=0}^M x_{im} \leq 1; \quad \forall i \in N$$

$$x_{im} \in \{0, 1\} \text{ et } y_{imt} \in \{0, 1\}; \quad \forall i \in N; \quad m \in M; \quad \forall t \in T$$

- \* La fonction objectif : permet de minimiser le nombre de jobs rejetés, c'est à dire maximiser le nombre de jobs ordonnancés sur l'ensemble des machines.
- \* La première contrainte permet de vérifier que si un job  $J_i$  n'est pas rejeté alors il est affecté durant sa période d'exécution  $[s_i, c_i]$  sur une machine  $m$  ce qui implique que  $y_{imt}$  vaut 1 pour tout  $t$  appartenant à  $[s_i, c_i]$ .
- \* la seconde contrainte cherche à vérifier qu'il n'y a pas plus de ressources  $r_{ijm}$  (quantité de ressource dont à besoin un job  $i$  sur une ressource  $j$  dans une machine  $m$ ) consommés sur la machine  $m$  à l'instant  $t$  que la valeur de la ressource  $Q_{jm}$ .
- \* La troisième contrainte permet de vérifier qu'un job n'est exécuté que sur une seule machine et unique machine.

La formulation ci-dessus, permet d'avoir la fonction objective ou la solution pour un seul critère ou un seul agent donc chaque agent devra. Pour avoir les fonctions objectives de tous les critères, nous utiliserons les méthodes suivantes : La Combinaison Linéaire et Epsilon - Contrainte afin d'obtenir un front de Pareto.

## 1.1 La Combinaison Linéaire

Le but de ce principe est d'avoir une seule fonction objective obtenue à partir de toutes les autres. Dans notre cas, en mode bi critère, nous aurons une combinaison de la fonction objective de l'agent A et de B.

$$F = \alpha f^A + (1 - \alpha) f^B; \text{ avec } \alpha_i \in [0, 1]$$

Les fonctions objectives de A et de B sont obtenues à partir du modèle mathématique ci-dessus. Nous discutons de la solution optimale en fonction de la variation de  $\alpha$  et obtenons une solution de Pareto.

## 1.2 Epsilon - Contrainte

Dans cette approche, l'un des objectifs est minimisé, tandis que les autres sont utilisés comme des contraintes liées par certains niveaux admissibles.

Afin de trouver plusieurs solutions optimales de Pareto, nous devons résoudre le problème  $\epsilon$ -Contrainte en utilisant plusieurs valeurs de  $\epsilon_i$ . Cette méthode est un processus d'optimisation itératif dans lequel l'utilisateur doit fournir l'intervalle de l'objectif référence. Cet incrément détermine le nombre de solutions Pareto optimales générées.

En ce qui concerne l'approche Epsilon - Contrainte pour le cas bi critère, nous devons calculer la fonction objective d'un des deux agents A ou B qui se caractérise comme suit :

$$f^A = \sum_{i \in N^A} x_i$$

$$f^B = \sum_{i \in N^B} x_i$$

Cette fonction objective se base sur les mêmes contrainte que le modèle mathématique ci-dessus. D'après cette approche si nous conservons la fonction objective de l'agent B  $f^B$  à minimiser, nous ajoutons une nouvelle contrainte aux autres qui existe déjà tel que  $f^A \leq \epsilon$ .

Avec la variation de  $\epsilon$  nous pouvons formé de solution de Pareto de façon symétrique ce qui nous conduira à un Front de Pareto représentant les solutions réalisable.

## 2 Heuristique - Méta heuristique

Le problème d'ordonnancement multi critère et multi machine peut être résolue efficacement avec la méthode exacte, avec des solutions optimales néanmoins le problème est NP-Difficile, cela signifie que le temps de calcul croit de manière exponentielle en fonction de la taille des données d'entrées.

D'où l'importance d'avoir des méta heuristique et heuristique permettant de trouver des solutions proches de celle de la méthode exacte mais un temps de calcul beaucoup plus faible.

### 2.1 Heuristique

Dans cette partie, nous nous basons sur comment organiser ou trier les jobs afin de les affectés sur les machines :

#### 2.1.1 Méthode de Tries :

D'après ma lecture des documents et mes recherches, nous avons retenu les méthodes de tries suivantes :

**CC max :** L'ensemble des jobs de l'instance sont triés par ordre croissant d'espace occupé  $\sum_{j \in R} r_{ij} * (f_i - s_i)$ . En cas d'égalité, le job se commençant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique. L'idée de CC max est de minimiser l'espace pris par les jobs lors de leur exécution. Cette méthode de trie a été testée dans la publication mono-machine et ils'avère que sur l'ensemble des méthodes testées dans la publication, c'est elle qui nous permet d'obtenir les résultats les plus proches des solutions

optimales. Cela est sans doute lié au fait que cette méthode de tri prend en compte à la fois le temps pendant lequel les jobs s'exécutent ainsi que la quantité de ressources que chaque job utilise.

**le maximum des ressources consommés :** Ici, nous trions les jobs par ordre croissant suivant le maximum entre ces ressources qu'il consomme  $\forall i \in N; \forall j \in R; \text{Max}(r_{ij} * (f_i - s_i))$ , en cas d'égalité, les jobs sont triés par ordre croissant suivant leurs durées d'exécutions  $(f_i - s_i)$ , sinon, on se réfère à l'ordre lexicographique. Issue de la résolution du problème de bin-packing avec des dimensions, le but est de trier les jobs selon leur dimension maximale.

**Durée :** Cette méthode permet trier les jobs en fonctions de leur temps (durée) d'exécution. Elle nous permet de d'exécuter le plus rapidement les jobs qui prennent moins de temps. Pour chaque job  $i$ ,  $p_i = (f_i - s_i)$

### 2.1.2 Méthode d'affectation :

Une fois que nous avons trié nos jobs et que nous avons obtenu une liste à ordonnancer, trois types d'approches sont possibles :

- Nous affectons le maximum de jobs sur la première machine, puis nous essayons d'ordonnancer le maximum de jobs restants sur la deuxième machine et ainsi de suite jusqu'à la dernière machine.

L'algorithme (issue du PRD de Pierre Fervault ) est expliqué ici [Figure 1](#)

```

- Soit  $M$  l'ensemble des machines disponibles
- Soit  $L$  l'ensemble des jobs triés selon la règle  $\pi$ 
- Soit  $S_m^*$  l'ensemble des jobs ordonnancés sur la machine  $m$ , Initialement,  $S_m^* = \emptyset$ 
for  $m \in M$  do
    Soit  $L_m \subseteq L$ , le sous-ensemble des jobs restants à ordonnancer;
    while  $L_m \neq \emptyset$  do
        Soit  $J_k$  le prochain job disponible;
        Soit  $S_m \subseteq S_m^*$ , le sous-ensemble des jobs se chevauchant sur la machine  $m$  durant
         $(sk, fk)$ ;
        if  $\sum_{i \in S_m} q_{ijm} + q_{kjm} \leq Q_{jm}; \forall j \in R$  then
            Ordonnancer  $k$  sur la machine  $m$ ;
             $S_m^* = S_m^* \cup \{k\}$ ;
             $L = L / \{k\}$  ( $J_k$  est rejeté);
        end
         $L_m = L_m / \{k\}$  ( $J_k$  est rejeté);
    end
end
end

```

**Figure 1** – Algorithme d'affectation des jobs machine par machine

- Nous affectons chaque job sur la machine la moins chargée (la charge est ici définie par la formule suivante  $\sum_{t \in T} y_{imt} * r_{ijm} \quad \forall i \in N; \forall j \in R$  o  $y_{imt}$  vaut 1 si le job  $i$  est exécuté sur la machine  $m$  à l'instant  $t$ , sinon  $y_{imt}$  vaut 0). S'il n'est pas possible d'ordonnancer le job sur la machine la moins chargée, alors nous essayons de l'ordonnancer sur la deuxième machine la moins chargée et ainsi de suite jusqu'à la dernière machine (la machine la plus chargée). S'il n'est pas possible de l'affecter sur aucune des machines, le job est rejeté. l'algorithme (issue du PRD de Pierre Fervault ) expliquant la procédure à suivre se trouve ici [Figure 2](#)
- Nous affectons un ensemble de job qui ne se chevauche pas à une machine (Cette méthode est optionnelle) , si la première machine n'arrive pas on passe à la deuxième ainsi de suite

```

- Soit M l'ensemble des machines disponibles
- Soit L l'ensemble des jobs triés selon la règle  $\pi$ 
- Soit  $S_m^*$  l'ensemble des jobs ordonnancés sur la machine m, Initialement,  $S_m^* = \emptyset$ 
while L  $\neq \emptyset$  do
    Soit  $J_k$  le prochain job disponible;
    Soit G la liste des machines triés par ordre croissant selon les charges de chaque machine;
    for m  $\in$  G do
        if  $\{k\} \in L$  then
            Soit  $S_m \subseteq S_m^*$ , le sous-ensemble des jobs se chevauchant durant  $(s_k, f_k)$ ;
            if  $\sum_{i \in S_m} q_{ijm} + q_{kjm} \leq Q_{jm}; \forall j \in R$  then
                Ordonnancer k sur la machine m;
                 $S_m^* = S_m^* \cup \{k\}$ ;
                 $L = L / \{k\}$  (Jk est rejeté);
            end
        end
    end
    if  $\{k\} \in L$  then
         $L = L / \{k\}$  (Jk est rejeté);
    end
end
end

```

Figure 2 – Algorithme d'affectation des jobs par machine moins chargée

jusqu'à la dernière machine.

### 2.1.3 Politique et Formation d'heuristique :

Après avoir vu dans les sections précédentes où il y a l'explication des méthodes de trie des jobs et d'affectations des jobs triés sur les machines. Nous devons savoir combiné tous cela afin de former nos heuristiques qui nous permettront d'obtenir nos front de Pareto car il faut pas oublié que nous sommes dans le cas multicritère et multi machine.

Le principe de mise en place d'heuristique répondant à notre problématique , ici nous suivons la politique d'épsilon contrainte afin d'obtenir les fronts de Pareto. :

- Avoir une première liste des jobs  $l_A$  de l'agent A puis une deuxième liste de jobs  $l_B$  de l'agent B.
- On trie d'abord la liste des jobs  $l_A$  de A , en choisissant une des méthodes de trie , ce qui nous donne  $T_{l_A} = \text{methodeTrie}(l_A)$  et puis on trie la liste des jobs  $l_B$  de l'agent B, nous aurons donc  $T_{l_B} = \text{methodeTrie}(l_B)$ . A noter que les deux méthodes de trie choisit peuvent être identique ou différente.
- Puis nous utilisons une des méthodes d'affectation parmi celle décrit ci-dessus à la quelle nous passons la liste des jobs triés  $T_{l_A}$  de l'agent A afin d'obtenir le nombre maximale de jobs qui peut être ordonnancé pour l'agent A et la liste de jobs *listeAOrdon* de l'agent A qui ont pu être ordonnancé.

$\text{NombreMaxOrdonA} = \text{methodeAffectation}(T_{l_A})$  et  $\text{listeAOrdon} = \text{methodeAffectation1}(T_{l_A})$  ■

- De là, nous partons de 0 à  $\text{NombreMaxOrdonA}$ , comme le principe de l'épsilon contrainte le demande, nous allons faire varié la valeur de  $\epsilon$  de 0 à  $\text{NombreMaxOrdonA}$  ce qui veut dire qu'on prend 0 à  $\text{NombreMaxOrdonA}$  jobs de l'agent A dans la liste de jobs *listeAOrdon* de l'agent A qui ont pu être ordonnancé. Ainsi nous pourrons créer une nouvelle liste de jobs *Alljobs* qui contiendra tous les jobs trié  $T_{l_B}$  de l'agent B et en fonction

de la valeur de  $\epsilon$  qui varie de 0 à  $\text{NombreMaxOrdonA}$ , des jobs de l'agent A provenant de  $\text{listeAOrdon}$ .

- Avec cette nouvelle liste  $\text{Alljobs}$  créée à chaque itération ou à chaque changement de la valeur de  $\epsilon$ , nous passons cette liste à une des méthodes d'affectations des jobs sur machines :  $\text{listeABOrdon} = \text{methodeAffectation2}(\text{Alljobs})$ , ainsi nous avons la liste des jobs des deux agents A et B qui ont pu être ordonnancé et nous calculons le nombre de jobs ordonnancés sur les machines de chaque agents puis nous déduisons le nombre de jobs rejeter pour chacun des agents. A noter que nous pouvons utiliser différente méthode d'affectation de jobs sur les machines donc  $\text{methodeAffectation1}$  et  $\text{methodeAffectation2}$  peuvent être identique ou différente.
- A la fin de chaque itération, nous aurons la liste des jobs ordonnancés  $\text{listeABOrdon}$  des deux agents A et B puis la Solution de Pareto lié à cette liste, c'est à dire le nombre de jobs rejeté pour l'agent A et pour l'agent B ce qui correspond à leur  $\text{valeurObjA}$  et  $\text{valeurObjB}$ .

Avec la combinaison des méthodes de trie et d'affectation nous pouvons obtenir plusieurs heuristique, car nous pouvons utiliser la même ou différente méthode de trie des jobs de chaque agents et utiliser la même ou différente méthode d'affectation sur machine des jobs de chaque agent.

Par exemple :

- $\text{trieParDurée}(\text{Liste de jobs de l'agent A})$  et  $\text{trieParDurée}(\text{Liste de jobs de l'agent B})$  puis  $\text{affectationParMachine}(\text{Liste de jobs trié de l'agent A})$  et  $\text{affectationParMachine}$  (la grande liste contenant tous les jobs de l'agent B et certain de l'agent A).
- $\text{trieParDurée}(\text{Liste de jobs de l'agent A})$  et  $\text{trieParCCmax}(\text{Liste de jobs de l'agent B})$  puis  $\text{affectationParMachine}(\text{Liste de jobs trié de l'agent A})$  et  $\text{affectationParMachineMoinsChargé}$  (la grande liste contenant tous les jobs de l'agent B et certain de l'agent A).

Ainsi avec 3 méthodes méthodes de trie et deux méthode d'affection on peut avoir jusqu'à 28 différents heuristiques en suivant bien l'ensemble des combinaison possible des méthodes de tri et d'affectation des jobs de chaque agent sur différents secteur de l'algorithme.

## 2.2 Méta heuristique

Dans cette section nous avons choisie un algorithme génétique NSGA2 suite à sa performance dans la littérature et la possibilité de l'implémenter à notre problème.

Cet algorithme génétique se base sur une simulation du processus de sélection naturelle : les individus les plus forts ont plus de chance de suivre que ceux qui sont faibles. Il utilise un ensemble de solutions candidates appelées "Population d'individus" où un individu est une solution de notre problème.

Dans notre population d'individus, on va sélectionner ceux qui nous paraissent les plus intéressants. Ensuite, ils vont créer des enfants qui pourront subir des mutations et croisements afin de donner une nouvelle population d'individus. On continue, ainsi de suite, jusqu'à atteindre un certain nombre de générations, un individu avec un fitness en dessous d'une valeur mise en paramètre ou un autre critère d'arrêt.

Le but espéré est de trouver à chaque nouvelle génération, un individu meilleur pour obtenir finalement un individu proche de la perfection souhaitée.

Cet algorithme suit les principales étapes suivantes : **Figure 3** :

Le premier point avant de se lancer dans l'implémentations de l'algorithme génétique NSGA2 est de penser à comment représenter un individu (une solution candidate) et les gènes de cet individu qui correspondent aux jobs.

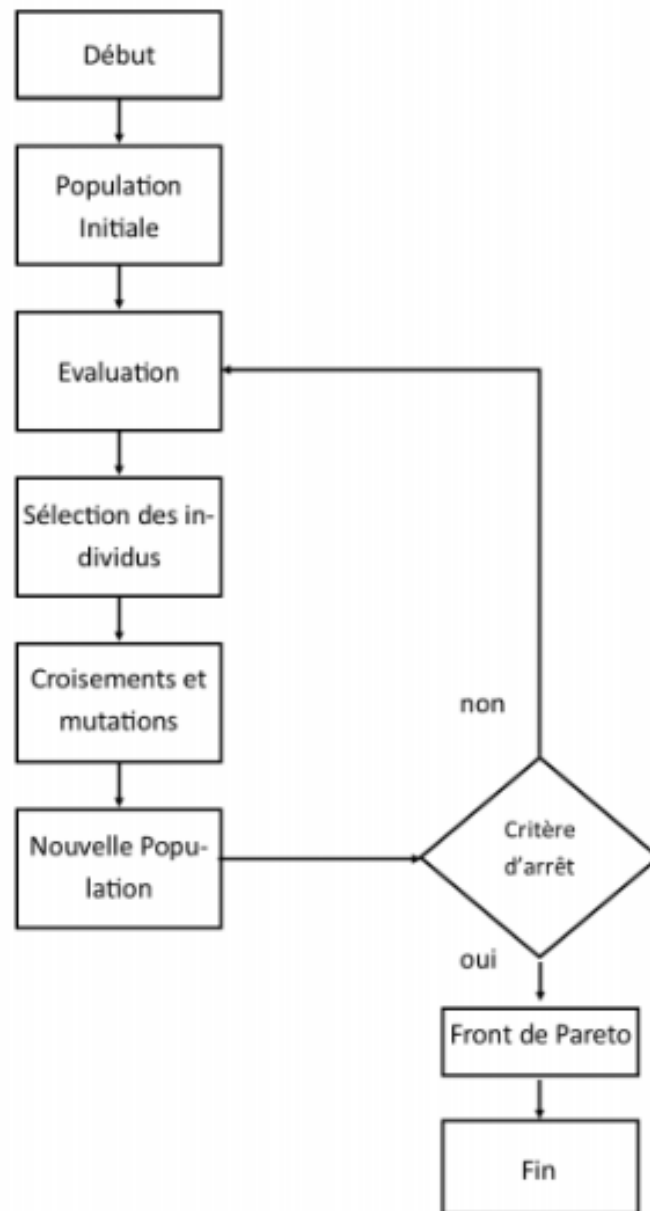


Figure 3 – L'algorithme génétique

Pour que notre représentation soit pertinente, nous coderons un Individu  $I$  comme un tableau d'entiers où une gène  $i$  correspondant à un job qui aura pour valeur le numéro de machine sur la quelle, elle est ordonnancée ce qui nous donne  $I[i] = \text{NumeroMachine}$  : Ce qui veut dire que le job  $i$  est ordonnancée sur la machine  $\text{NumeroMachine}$  où  $\text{NumeroMachine}$  est comprise entre  $\{0, M - 1\}$  avec  $M$  : nombre de machines. Dans le cas où le job  $i$  n'est pas ordonnancée sur aucune des machines alors  $I[i] = -1$ .

Cette méthode de codage d'un individu nous permet d'avoir différents individu, de réduire la redondance des individus car contrairement à la représentation binaire nous avons plusieurs possibilités. Elle permet aussi de savoir si un job est ordonnancé ou pas et sur quelle machine.

Exemple : Pour un ensemble de 8 jobs  $\{0, 1, 2, 3, 4, 5, 6, 7\}$  et 2 machines  $\{0, 1\}$ .

D'après cette représentation, nous voyons que le job 2 est ordonnancé sur le machine 1  $I[1] = 0$  et le job 4 n'est ordonnancé sur aucune machine  $I[3] = -1$ .

Jobs	0	1	2	3	4	5	6	7
Machines	0	1						
	Représentation :							
	-1	0	1	-1	-1	0	1	1

**Figure 4 – La représentation d'un individu**

En suivant le principe de l'algorithme génétique, nous aurons une population d'individu  $P_1$  de taille  $N$  à définir, dans la quelle nous aurons des Individus respectant la représentation d'un Individu expliqué ci-dessus, puis nous avons choisi comme critère d'arrêt de l'algorithme un nombre d'itération  $N_{iter}$  à définir .

### 2.2.1 Initialisation de la Population

C'est la première étape, elle consiste à créer un ensemble d'Individu donc un ensemble de solution possible, cet ensemble aura une taille  $N$  défini au préalable.

la stratégie utiliser pour diversifier notre population d'individus, est d'initialiser les individus avec les gènes comprise entre 0 et M (nombre de machine), où chaque gène est initialisée aléatoirement puis nous calculons la faisabilité afin de vérifier que la consommation de ressources de l'individu respecte la quantité disponible de ressources de chacune des machines et nous calculons les valeurs objectifs ( nombre de jobs rejeté de l'agent A et de l'agent B).

Dans le cas où l'individu est faisable, nous l'ajouter dans la population d'individu sinon nous continuons l'itération jusqu'à atteindre la taille de la population définit.

L'algorithme est décrit ici **Algorithme 1**.

Pendant l'initialisation de la population initiale, nous pouvons rencontrer des individus identiques, ils ont les mêmes gènes. Du coup pour diversifier notre population initiale, nous allons vérifier qu'un individu n'existe pas déjà avant de l'ajouter dans la population.

Du coup nous pouvons combiner cet algorithme **Algorithme 2** avec celui de **Algorithme 1** de la population initiale.

## 2.2.2 Sélection

Dans cette partie, nous mettons en place une politique de choix des parents : Papa et Maman qui sont des individus.

Pour cela, nous tirons au sort ou choisissons aléatoirement *nombreI* individus et on retourne le meilleur individu parmi les *nombreI* individus où *nombreI* est un paramètre à définir. Nous pourrions à chaque fois obtenir des parents de qualités.

La qualité de l'individu est basée sur la dominance de l'individu par rapport aux autres individus *nombreI* choisit aléatoirement, ce qui conduit à dire que le meilleur Individu est celui qui domine tous les *nombreI* individus et qui n'est pas dominée par aucun autres individus.

L'algorithme est expliqué ici **Algorithme 3**.

### 2.2.3 Croisement et Mutation

Le but du croisement et de la mutation permet de quitter l'optimale locale, quitter une zone et aller vers une autre zone afin de trouver de meilleure solution.

```

Result : Population avec les individus  $P_i$ 
ListeNuméroMachine : contenant les numéros de machines initialization;
while taille de la population <  $N$  do
    Créer un Individu indi;
    for index de 0 à NombreJobs do
        | Mettre -1 pour la position index d'un gène de indi;
    end
    Choisir de façon random le nombre de gène à modifier ou nombre de job à
    ordonnancer comprise entre 0 et NombreJobs;
    nombreJobOrd  $\leftarrow$  random(NombreJobs);
    while nombreJobOrd  $\neq$  0 do
        | Choisir de façon random les indices entre 0 et NombreJobs, puis modifier à chaque
        | indice par un numéro de machine de la ListeNuméroMachine;
        indice  $\leftarrow$  random(NombreJobs);
        Vérifier que l'indice n'était pas utiliser au préalable;
        indi[indice] = random numéro machine dans ListeNuméroMachine;
        nombreJobOrd - ;
    end
    Calcul de la faisabilité;
    Calcul des fonctions objectives;
    if indi est faisable then
        | ajouter indi à la population;
    else
        | retirer indi;
    end
end

```

**Algorithme 1 :** Algorithme d'initialisation de la population Initiale

```

Result : Vrai s'il existe un individu identique , faux sinon
Individu indi à verifier;
exist = faux;
for indice de 0 à taille de la population initiale do
    if indi.genes est egale à population[indice].genes then
        | exist = Vrai;
        | sortir de la boucle;
    else
        | continuer;
    end
end

```

**Algorithme 2 :** Algorithme de vérification de redondances

Après la sélection des parents parmi les individus de la population, nous utilisons ces parents pour obtenir deux fils ou filles comprenant les gènes des deux parents, cela est fait à travers le croisement et toujours dans l'optique de la diversité, le point de croisement est pris de façon aléatoire. Nous aurons le premier fils avec les gènes non croisé du père et des gènes croisé de la mère et l'inverse pour le deuxième fils. Puis nous calculons les valeurs des fonctions objectifs des deux fils et leurs faisabilités.

L'algorithme de croisement est ici **Algorithme 4 :**

les étapes du croisement en images :

**Result :** Indice du meilleur Individu

Entrée : population d'individus et nombreI (nombre d'individu à choisir);

Création d'une liste d'individu de taille nombreI , contenant nombreI individus choisis aléatoirement;

Individu1  $\leftarrow$  listeIndividus.get(0);

index = 0;

**for** indice de 1 à nombreI **do**

    Individu2  $\leftarrow$  listeIndividus.get(indice);

**if** Individu1 domine Individu2 **then**

        index = indice de Individu1;

**else**

        index = indice de Individu2;

**end**

**end**

retourne index;

**Algorithme 3 :** Algorithme de sélection des individus

- On sélectionne les deux parents, puis on choisit aléatoirement le nombre de gène à croiser et une liste d'indice de gène où se passera le croisement : **Figure 5**.

Nombre Gene à croiser : 4								
Indice de Croisement								
1	3	4	7					
Individu Papa								
-1	0	1	-1	-1	0	1	1	
Individu Maman								
0	0	-1	1	-1	0	1	-1	

**Figure 5 – Etape 1 du croisement**

- Nous initialisons les deux enfants dont l'enfant 1 qui prendra les gènes du parent 1 et l'enfant ceux du parents 2 : **Figure 6**.

Nombre Gene à croiser : 4								
Indice de Croisement								
1	3	4	7					
Individu Papa								
-1	0	1	-1	-1	0	1	1	
Individu Maman								
0	0	-1	1	-1	0	1	-1	
Enfant 1								
-1	0	1	-1	-1	0	1	1	
Enfant 2								
0	0	-1	1	-1	0	1	-1	

**Figure 6 – Etape 2 du croisement**

- Nous utilisons la liste des indices de croisement pour croiser, l'enfant 1 à ces indices prendra les gènes du parent 2 et l'enfant 2 à ces indices prendra les gènes du parent 1 : **Figure 7**.

Dans un deuxième temps , nous mettons en place la mutation des enfants en remplaçant la

```

Result : Les deux fils après le croisement
if valeur aleatoire < Probalité de croisement then
    Individu papa , maman;
    Choisir nombre de position à croiser NombreCrois aleatoirement;
    count = 0 ;
    listPosition = vide ;
    while count < NombreCrois do
        Choisir la position de croisement aléatoirement;
        while position exists do
            Choisir la position de croisement aléatoirement;
        end
        Ajouter la position dans la listPosition;
        count ++ ;
        Individu fils1 , fils2 ;
        for indice de 0 à nombre de gene do
            for indice2 de 0 à taille de la liste de Position do
                if listPosition[indice2] == indice then
                    fils1.gene[indice] = maman.gene[indice];
                    fils2.gene[indice] = papa.gene[indice];
                    break ;
                else
                end
            end
            if indice2 == taille de listPosition then
                fils1.gene[indice] = papa.gene[indice];
                fils2.gene[indice] = maman.gene[indice];
            else
            end
        end
    end
else
end

```

Algorithme 4 : L'algorithme de croisement

valeur du gène  $i$  choisit aléatoirement par un numéro de machine aléatoire différente de celle qui y était avant.

L'algorithme est le suivant **Algorithme 5**.

Exemple de mutation **Figure 8**, nous avons un individu Ind, nous choisissons aléatoirement un indice et changeons la valeur du gène à cet indice par un numéro de machine si la gène était égale à -1 ou à un autre numéro de machine excepté celui qui y était avant.

### 3 Déroulement de la validation des algorithmes

En ce qui concerne les tests et la validation des algorithmes, nous mettons en place des tests unitaires et surtout un travail régulier est fait avec le MOA qui me fournit des jeux de données qui sont des exemples avec des résultats connu qui sont déjà optimale, cela me permettra devoir si une méthode exacte donne la bonne réponse et si une méthode approchée me donne des

Nombre Gene à croiser : 4									
Indice de Croisement									
1	3	4	7						
Individu Papa									
-1	0	1	-1	-1	0	1	1		
Individu Maman									
0	0	-1	1	-1	0	1	-1		
Enfant 1									
-1	0	1	1	-1	0	1	-1		
Enfant 2									
0	0	-1	-1	-1	0	1	1		

Figure 7 – Etape 3 du croisement

**Result :** Individu après la mutation

```

if valeur aléatoire < Probabilité de mutation then
    Choisir une position p de façon aléatoire;
    idMachine ← Individu.gene[p];
    if NouveauIdMachine != idMachine then
        Individu.gene[p] ← NouveauIdMachine;
    else
        Individu.gene[p] ← -1;
    end
    Calcul de la valeur objective;
    Vérifier la faisabilité;
else
    Pas de mutation;
end

```

Algorithme 5 : Algorithme de mutation

Individu Ind									
-1	0	1	-1	-1	0	1	1		
Mutation :									
Indice : 1									
-1	1	1	-1	-1	0	1	1		
Indice : 4									
-1	0	1	-1	0	0	1	1		

Figure 8 – Exemple de mutation

résultats proche ou juste. En cas de non de validation alors une autre méthode serait prise en compte ou une correction serait envisagée.

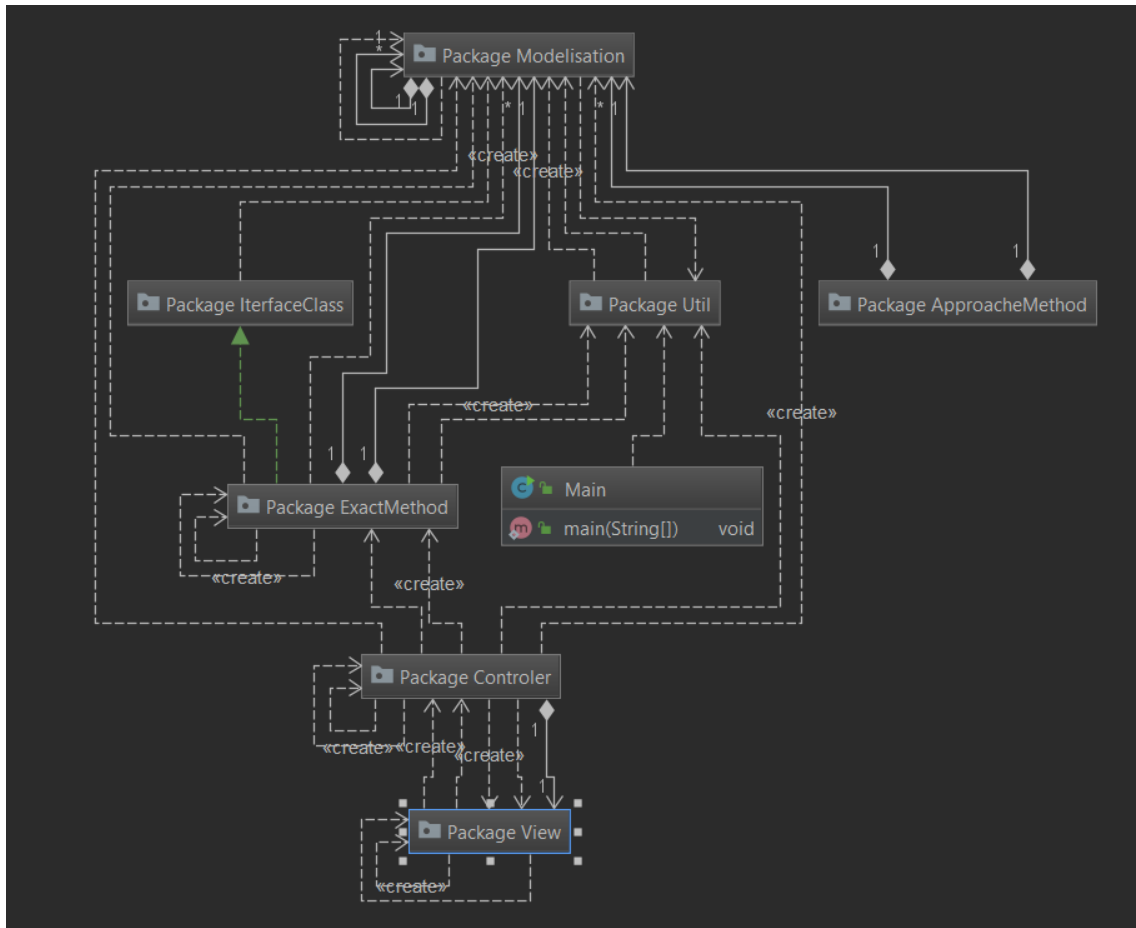
## 4 Conception de l'application :

En ce qui concerne, le developpement de l'application, j'ai utilisé différent approche de qualité logiciel qui permet la séparation du code, l'évolution du programme et la lisibilité du code.

## 4.1 Model - Vue - Contrôleur (MVC)

Dans un premier temps, l'application suit le design pattern MVC pour Modèle - Vue - Contrôleur afin de mieux séparer le modèle qui contient les heuristiques, méta heuristique, la méthode exacte et toutes les classes qui sont liés à ces dernières puis la vue qui contient les classes de l'interface graphique de l'application et enfin le contrôleur qui permet de relier le modèle à la vue ou inversement.

Nous aurons comme architecture celle ci **Figure 9** :



**Figure 9** – Architecture de l'application JAVA - MVC

Comme nous pouvons remarquer, nous avons un Package contrôleur, un Package View et les autres Packages restant font partie du modèle, pour un souci de lisibilité j'ai décomposé le modèle en sous Package relier les uns aux autres.

### 4.1.1 Package Vue

Ce Package contient les classes de création des différentes vues de l'application, chaque vue permet de répondre à une des fonctionnalités expliqués dans le cahier de spécification décrit dans l'annexe.

les différentes classes **Figure 10** ont chacune leur utilité :

- ResolutionInstance : est la classe permettant de créer la vue afin de pour pouvoir résoudre les instances.

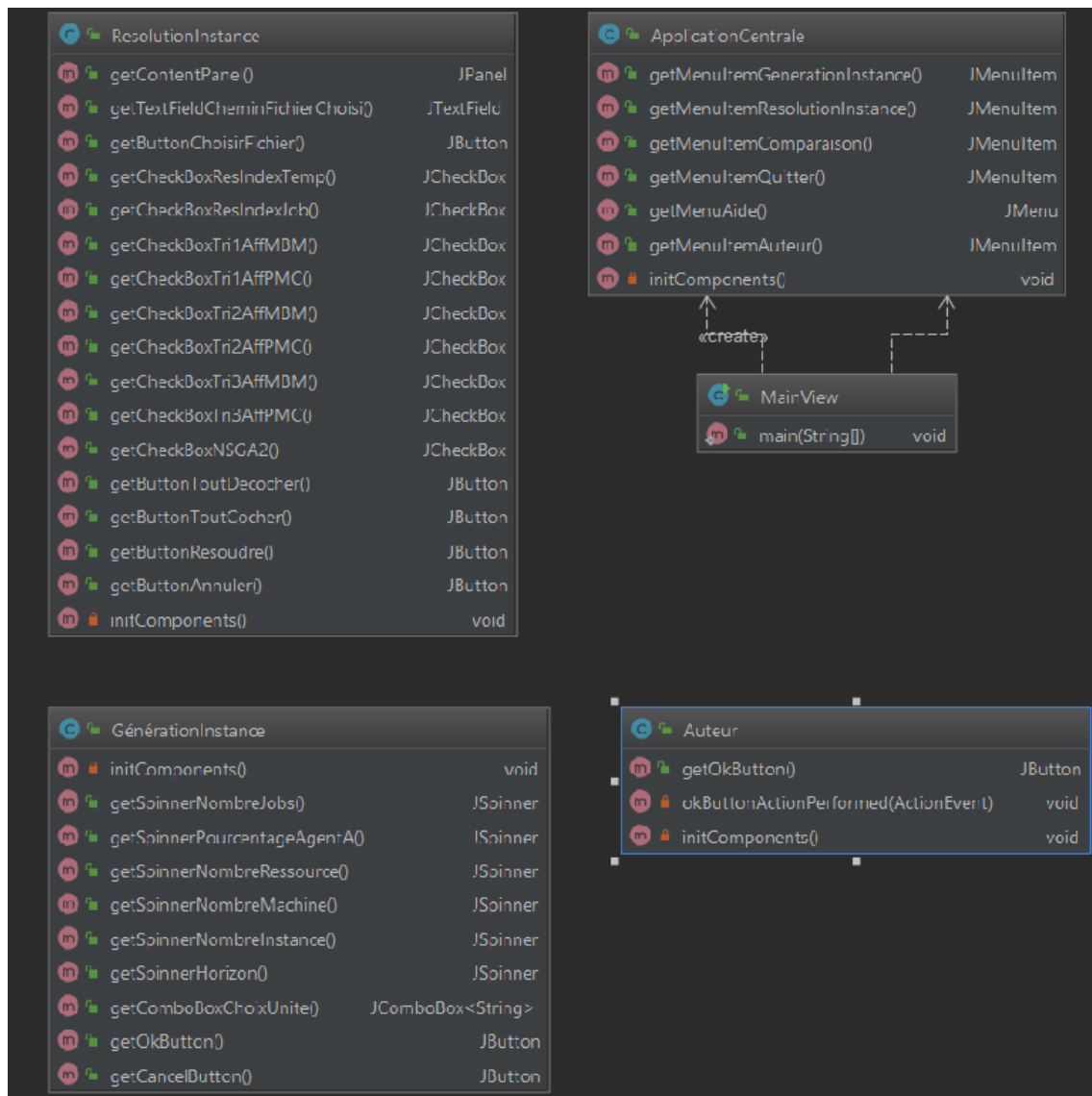


Figure 10 – les classes du Package Vue

- GenerationInstance : c'est à partir de cette classe que nous avons la vue pour la création d'instance.
- ApplicationCentrale : c'est la classe qui contient la vue pour l'accueil et contient les différents menus.
- Auteurs : elle permet de d'afficher les auteurs de l'applications.
- MainView : cette classe permet de lancer l'application.

#### 4.1.2 Package Contoleur

Son rôle est de gérer les différents actions initiés par un utilisateur sur l'une des vues du Package Vue. Il permet aussi de lier les vues aux modèles adéquats comme on peut le voir dans l'architecture de l'application [Figure 9](#). Donc chaque classe vue a une classe contrôleur qui va avec afin de gérer les événements de la vue qu'il gère.

Nous aurons comme classe du Package contrôleur [Figure 11](#) :

- ResolutionInstanceControler : est la classe contrôleur permettant de lancer la vue ResolutionInstance, de gérer les événements et lier en fonction des actions aux modèles afin de

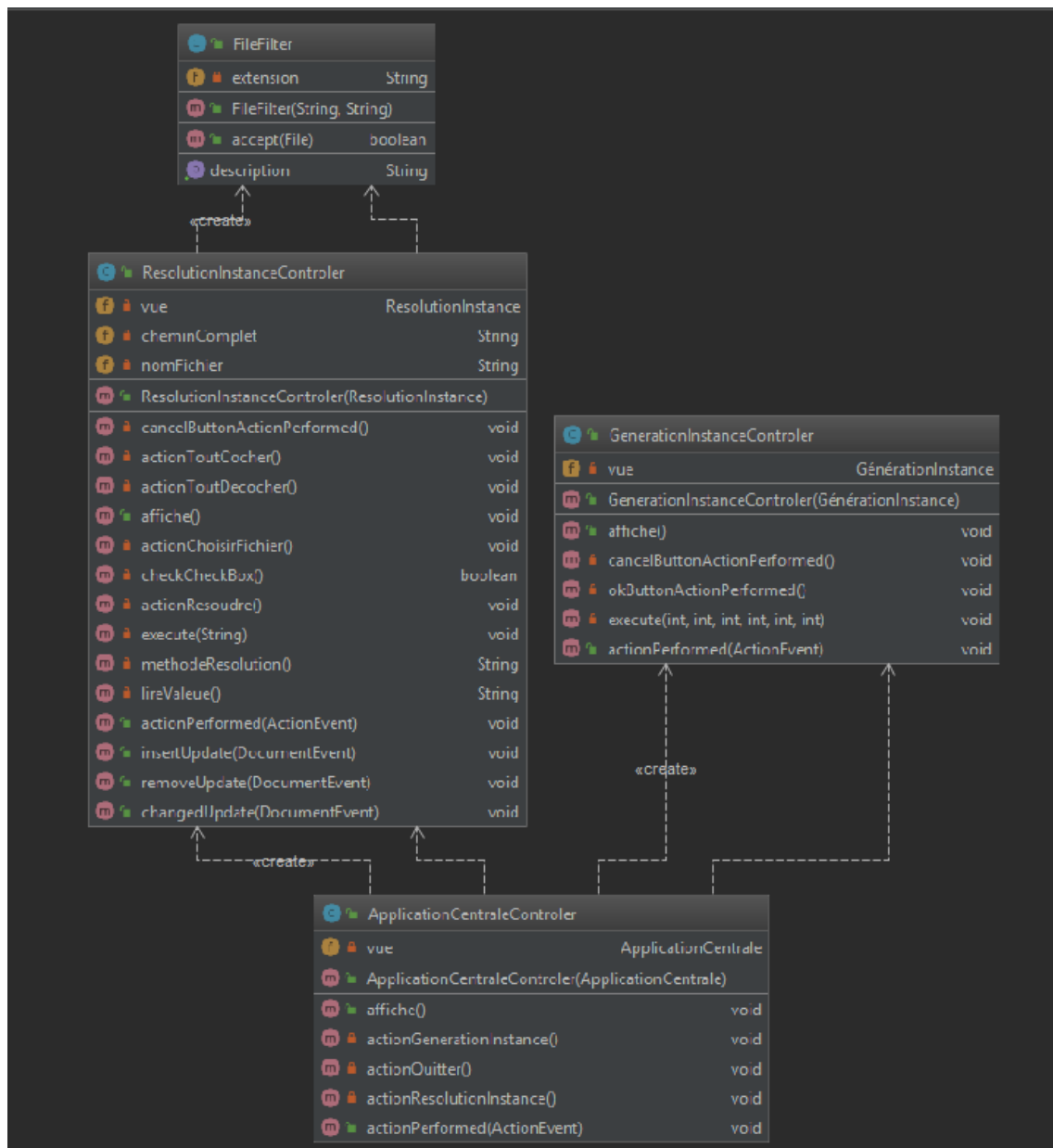


Figure 11 – les classes du Package Contrôleur

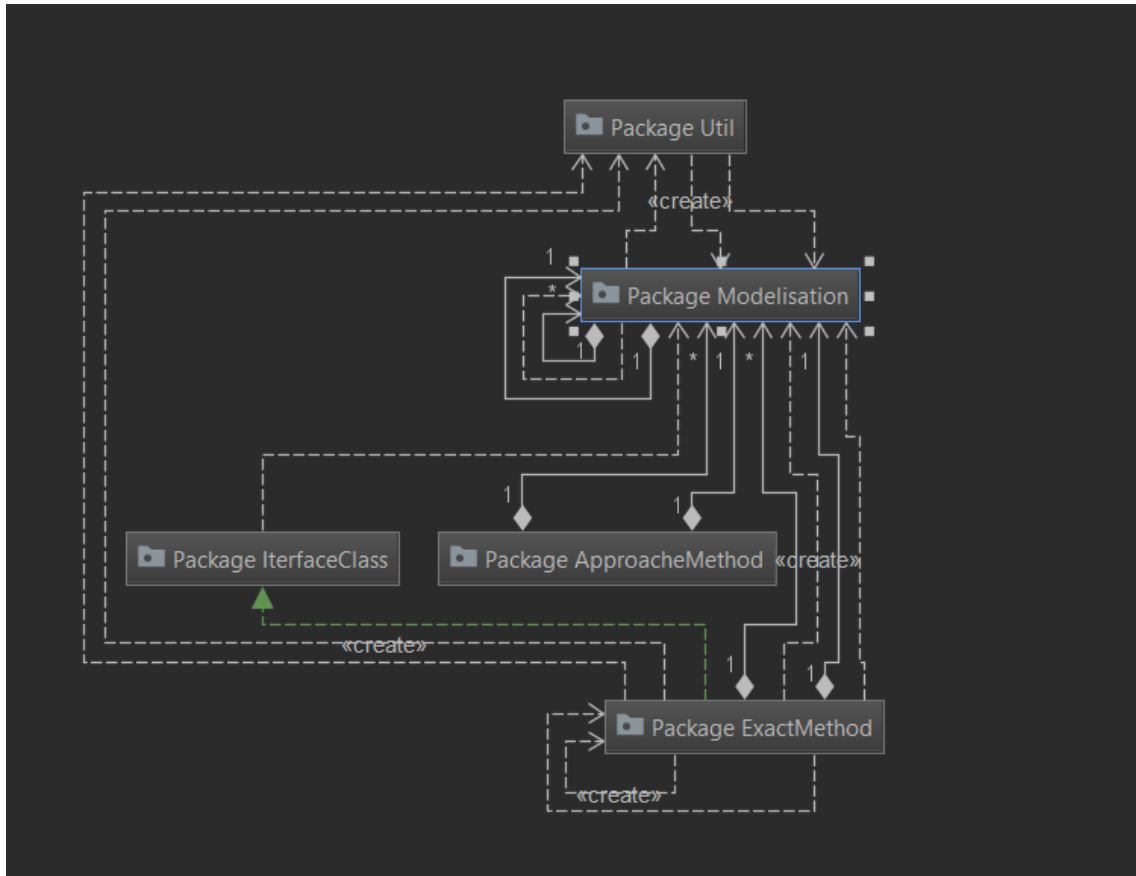
- pour pouvoir résoudre les instances.
- **GenerationInstanceController** : est la classe contrôleur permettant de lancer la vue **GenerationInstance**, de gérer les événements et lier en fonction des actions aux modèles afin de pouvoir créer des instances.
- **ApplicationCentraleController** : est la classe contrôleur principale permettant de lancer la vue **ApplicationCentrale**, de gérer les événements et les autres contrôleurs.

### 4.1.3 Package Modèle

C'est dans ce package se trouve l'implémentation de notre recherche et analyse sur le cas de l'ordonnancement et l'affectation multi critère et multi machine. Il contient toutes les classes et méthodes répondant ou solutionnant notre problématique. Dans notre architecture MVC, c'est le contrôleur qui lui fera appel pour répondre à une action déclenchée par un utilisateur lors de la manipulation de la vue.

Ce package utilise le Design Pattern Stratégie avec des Interfaces en JAVA pour que chacun des algorithmes (heuristique, méta heuristique, méthode exacte) puissent avoir le même comportement et une facilité de maintenance dans de futur évolution. Et pour une question de lisibilité, le package modèle est subdivisé en plusieurs petits packages afin de réunir les classes qui ont le même sens.

Tous les sous packages du package Modèle **Figure 12** :



**Figure 12** – Les sous Packages du Package Model

- Package Util : Cet dossier contient toutes les classes communes à toutes les autres, il est composé des classes suivante :
  - Commun : cette classe contient les méthodes de création d’instance, d’écriture des résultats dans un fichiers et de lire le contenu d’une instance.
  - Timer : cette classe permet de calculer le temps d’exécution d’un programme.
- Package Modelisation : Cet dossier contient les classes , qui modélise un job, une machine, une instance et une solution de Pareto.
- Package InterfaceClass : Cet dossier contient toutes les classes interfaces de l’application qui définissent les comportements des autres classes. Il est le dossier phare de la mise en place du Design Pattern Stratégie.
- Package ApproachMethod : cet dossier contient toutes classes qui implémentent les heuristiques et l’algorithme génétique NSGA2.
- Package ExactMethod : Cet dossier contient les classes qui implémentent la méthode exacte avec l’épsilon contrainte.

# 5

## Mise en Oeuvre

### 1 Implémentation de l'application

Pour mettre en pratique notre analyse de notre problème et la conception du programme, j'ai dû utiliser des outils afin d'atteindre les objectifs qui ont été fixés par la MOA.

D'un premier temps, je devais coder l'application et le programme en C++ et pour la partie interface graphique QT. Ce choix était fait à cause d'un ancien PRD qui avait fait une application dans ce langage. C'est sur la base de cette application que j'avais commencé à coder le programme comme le développement de la méthode exacte et l'algorithme génétique NSGA2, en parlant de ce dernier mon programme était lent et prenait du temps avant d'obtenir un résultat. Ce problème venait de la gestion de la mémoire, mais vu ma faible connaissance de la gestion de la mémoire nous décidâmes avec l'encadrant de continuer sur une nouvelle application avec un autre langage.

Le langage choisi à la suite de ça est le JAVA par sa facilité d'utilisation, sa portabilité et ses performances en terme de gestion automatique de la mémoire. Ce langage a à sa disposition plus d'API qui me permettront de réaliser mon projet aisément.

La modélisation de la méthode exacte, il fallait utiliser une bibliothèque pour le calcul linéaire PLNE avec un solveur pour obtenir une solution optimale : les fronts de Pareto non dominés. La bibliothèque adéquate que mon encadrant m'a proposée est CPLEX, cette bibliothèque est vraiment documentée par IBM, nous pouvons vite la prendre en main et on peut trouver des solutions à des erreurs rapidement, à partir de là j'ai créé une modélisation du problème d'ordonnancement et d'affectation multi machine et multi critère. Une classe lui a été dédiée afin de l'isoler et penser à l'évolution du programme. Puis j'ai intégré le programme CPLEX au projet.

L'implémenter de la méthode exacte passe par l'épsilon contrainte dont le principe est de varier la valeur de  $\epsilon$  afin de trouver pour chaque agent des solutions ce qui forme un front de Pareto à la fin. Et pour le faire il faut dans un premier temps calculer la valeur de fonction objectif de l'agent A  $f^A : \text{valeurObjA}$ , puis de calculer la valeur de la fonction de l'agent B  $f^B : \text{valeurObjB}$  avec ces deux fonctions objectifs nous cherchons la limite de  $\epsilon$  en posant ce cas :  $f^B \leq \text{valeurObjB} : \text{valeurObjBound}$ . Avec cette valeur, l'épsilon  $\epsilon$  varie entre  $\{\text{valeurObjA}, \text{valeurObjBound}\}$  et de là nous allons résoudre tous les cas de ce type  $f^A \leq \epsilon$ . Cette politique me permet d'obtenir un front de Pareto avec des points non dominés.

Avec 100 jobs à ordonnancer dont 50 jobs pour l'agent A et les 50 jobs restant pour l'agent B sur

7 machines disposant chacune 3 ressources de taille 1000. Nous avons dans l'image **Figure 1** le front de Pareto des points non dominés c'est la valeur de la fonction objectif de chaque agent qui forme un point. Et comme nous l'avons dans la section Analyse et conception, une valeur de la fonction objectif d'un agent est le nombre de jobs rejetés pour l'agent.



**Figure 1** – Exemple de front de Pareto obtenue par epsilon contrainte d'une instance de 100 jobs et 7 machines

La deuxième étape après avoir implémenter la méthode exacte qui peut prendre du temps en fonction de la taille d'un fichier d'instance (le nombre de job qui est référencé) , j'ai développé l'algorithme génétique NSGA2 et 12 heuristiques qui suivent l'architecture indiquer dans le chapitre Analyse et conception. Ces heuristiques et méta heuristiques peuvent nous donner parfois une solution optimale pour des petites instances ou des solutions réalisables et de bonne qualité en moins de temps.

A partir de l'analyse et la conception de ces algorithmes nous obtenons aussi des solutions qui représentent des points et de là nous pouvons en déduire un front de Pareto avec des points non dominés car chaque solution passe par une méthode qui construit un front de Pareto. Avec une instance de 50 jobs dont 25 jobs de A et 25 jobs de l'agent B et 4 machines on peut obtenir un front de Pareto de NSGA2 **Figure 2** et pour une des heuristiques ( Trie par MakeSpan et Affectation par machine) **Figure 3**.

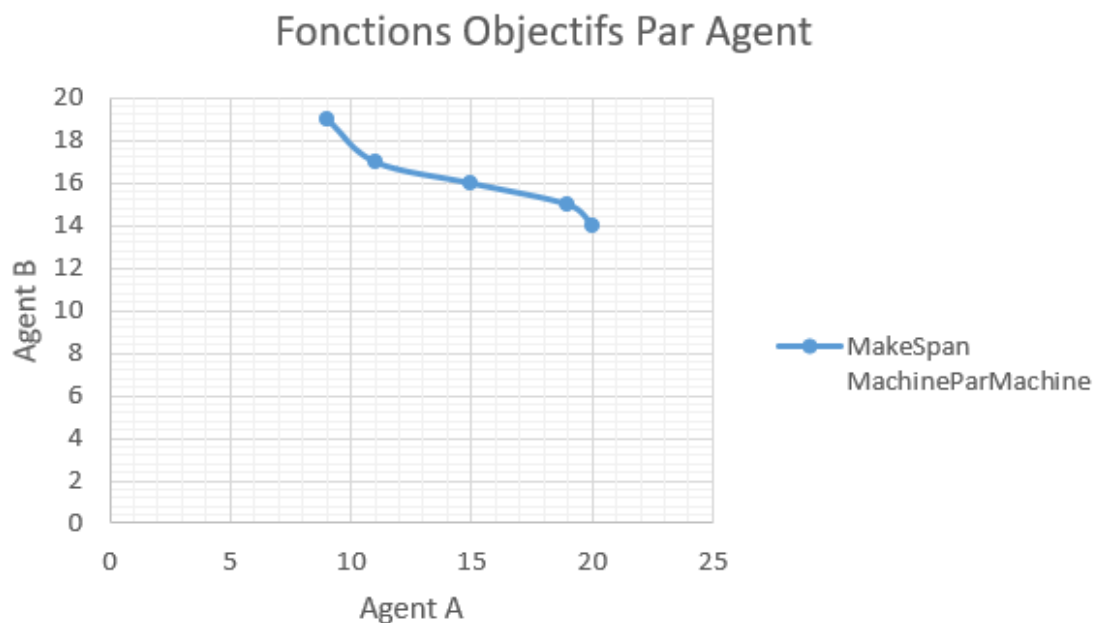
Avant de résoudre une instance, elle doit être créer, ce qui m'a conduit à mettre en place un programme de génération d'instance.

Avec l'obtention des résultats, j'ai mis en place un programme qui permet d'écrire ces résultats dans un fichier tests.

Une application a toujours une partie interface graphique, du coup le module Swing en JAVA est la meilleure option car il est embarqué dans le packaging de JAVA. Avec ce module, j'ai développer une application pour gérer facilement les interactions entre les fonctionnalités spécifiés dans le cahier de spécifications et l'utilisateur.



**Figure 2** – Exemple de front de Pareto obtenue par NSGA2 d'une instance de 50 jobs et 4 machines



**Figure 3** – Exemple de front de Pareto obtenue par Make Span MachineParMachine d'une instance de 50 jobs et 4 machines

## 2 Expérimentation et des résultats

L'analyse des résultats est un des objectifs du PRD après la conception et le développement des différentes approches pour résoudre le problème d'ordonnancement et d'affectations des jobs pour le cas multi critère et multi agent, dans cette étape nous devons comparer les différents résultats obtenus à partir de la méthode exacte Epsilon contrainte et les heuristiques et méta heuristiques, cela nous permettra de voir les performances de chaque approche.

l'analyse se base sur la durée de l'exécution d'une approche, le nombre de solution qu'elle trouve, le pourcentage de solution optimale, le pourcentage de solution faible, la distance euclidienne entre les solutions et l'hypervolume.

Afin de tirer une analyse pertinente des différentes approches, nous avons utilisé 30 instances dont le nombre de jobs sont comprises entre {10,100} avec un écart de 10 jobs, le nombre de machines varie entre 4, 7 et 10 où chaque machine possède 3 ressources (CPU, RAM, DISQUE) de capacité 1000 et nous avons 2 agents A et B qui ont chacun 50 % de jobs.

Les instances sont enregistrer dans un fichier texte comme le montre **Figure 1** (Chapitre 7). Ces 30 différentes instances sont passer à un programme qui lit le contenu de chaque fichier puis les résout avec la méthode exacte - Epsilon contrainte après avec l'algorithme génétique NSGA2 et avec tous les 12 heuristiques de là nous pouvons comparer les résultats obtenue par les approches.

Un autre programme récupère les résultats et écrit dans un fichier Excel la durée de l'exécution, le nombre de solution trouvé, le pourcentage de solution optimale, le pourcentage de solution faible, la distance euclidienne entre les solutions et l'hypervolume de chacune des approches. Cette comparaison est faite avec les résultats de la méthode exacte - Epsilon contrainte. A la fin nous avons un tableau Excel avec des données sur lesquelles nous pourrions faire des statistiques et des courbes pour mieux visionner les performances selon différents critères.

## 2.1 Les Performances de NSGA2 :

Comme expliqué plus haut, nous avons résolu toutes les 30 instances avec l'algorithme génétique NSGA2, à travers cette résolution, nous obtenons un tableau excel contenant le temps d'exécution par instance, le nombre de solution Pareto trouvé, le pourcentage de solution optimale de Pareto trouvé comparer à la méthode exacte, le pourcentage de solution de Pareto faible trouvé comparer à la méthode exacte puis l'hypervolume et la distance euclidienne entre les solutions obtenus par NSGA2 et Epsilon Contrainte.

l'exécution c'est passé avec les paramètres suivantes : nombre d'itération 500, taille de la populations 100, probabilité de croisement et de mutation 0.8, le nombre d'individu à sélectionner lors de la sélection 10.

Nous obtenons cet tableau **Figure 4** :

Nous remarquons dans ce tableau que plus la taille du fichier est petit plus nous trouvons un meilleur pourcentage de solution optimale voir 100 %, c'est à dire que NSGA2 trouve les mêmes solutions que celle de la méthode exacte qui nous donne les solutions optimales. Pour les instances de grandes tailles, nous trouvons presque 100 % de la solution de Pareto faible, une solution de Pareto est faible est que on trouve de très bonnes solutions mais elles sont dominées par celle obtenue par la méthode exacte pour un écart très faible soit une distance euclidienne moyenne de 1.21 et un hypervolume plus petit (on trouve des valeurs négatives) ce qui décrit que si NSGA2 ne trouve pas directement toutes les solutions optimales, il trouve des solutions très proches des solutions optimales c'est pourquoi on dit solution de Pareto faible.

Ces données montrent que nous trouvons moins de nombre de solution  $|S|$  que le nombre de solution de la méthode exacte  $|S^*|$ ,  $|S^*| > |S|$  mais NSGA2 trouve des solutions de qualité en moins de temps. le temps d'exécution d'une instance varie en fonction de la taille du fichier d'entrée.

En somme NSGA2 donne le front de Pareto est très proche du front de Pareto de la méthode exacte.

Instances	NSGA2					
	Time CPU	S	%S	%S Weak	HyperV	GD
Instance-1-10-3-4	12,252	1	100%	0,0%	0	0
Instance-1-10-3-7	16,877	1	100%	0,0%	0	0
Instance-1-10-3-10	11,611	1	100%	0,0%	0	0
Instance-1-20-3-4	11,98	4	100%	0,0%	0	0
Instance-1-20-3-7	17,104	4	75%	25,0%	0	0,25
Instance-1-20-3-10	19,225	1	100%	0,0%	0	0
Instance-1-30-3-4	14,097	3	100%	0,0%	-8	0
Instance-1-30-3-7	20,453	5	83%	0,0%	0	0
Instance-1-30-3-10	14,097	1	100%	0,0%	0	0
Instance-1-40-3-4	18,2	5	0%	100,0%	3	1,08284271
Instance-1-40-3-7	20	5	50%	0,0%	-15	0,28284271
Instance-1-40-3-10	18,2	2	0%	100,0%	1	1
Instance-1-50-3-4	15,582	5	0%	100,0%	12	1,4
Instance-1-50-3-7	34,618	7	0%	100,0%	21	1,73271289
Instance-1-50-3-10	28,866	5	0%	100,0%	6	1,2
Instance-1-60-3-4	27,663	5	0%	100,0%	-70	1,53137085
Instance-1-60-3-7	40,136	4	0%	100,0%	9	2,38415776
Instance-1-60-3-10	64,815	3	0%	100,0%	-5	1,41421356
Instance-1-70-3-4	140,653	7	0%	100,0%	-141	1,11834673
Instance-1-70-3-7	35,068	6	0%	83,3%	-40	2,23606798
Instance-1-70-3-10	46,765	2	0%	100,0%	-39	1,41421356
Instance-1-80-3-4	27,766	5	0%	80,0%	-124	1,9772699
Instance-1-80-3-7	48,109	3	0%	100,0%	-101	2,82842712
Instance-1-80-3-10	42,294	3	0%	100,0%	-24	2,15737865
Instance-1-90-3-4	32,573	5	0%	80,0%	-208	1,69574173
Instance-1-90-3-7	30,552	3	0%	100,0%	-75	1,41421356
Instance-1-90-3-10						
Instance-1-100-3-4	28,424	4	0%	75,0%	-394	3,22817177
Instance-1-100-3-7	53,177	3	0%	100,0%	-105	3,60555128
	31,8270357					1,21262581

Figure 4 – Tableau de données obtenu par NSGA2

## 2.2 Les Performances globales :

Pour analyse globale des différentes approches, j'ai dû comparer les résultats avec la méthode exacte l'épsilon contrainte en terme de temps d'exécution, les gaps, l'hypervolume et la distance euclidienne.

Cette expérimentation est faite sur les 30 instances décrit précédemment, à partir des résultats nous avons fait un tableau de données pour des statistiques pertinents

- temps d'exécution :

Nous voyons ici dans l'image [Figure 5](#), le temps d'exécution de chacune des approches, on remarque que les instances de 10 à 30 jobs sur 4 machines nous un temps d'exécution de la méthode exacte meilleure quelle de NSGA2 mais quand le nombre de jobs augmente le temps CPU de la méthode exacte grimpe conséquent et grimpe encore plus quand on augmente le nombre de machine avec le nombre de job (cas de 7 machines et 10 machines) alors que NSGA2 n'évolue pas trop. Les meilleurs temps appartient aux heuristiques qui s'exécutent en moins d'une seconde et cela quelque soit le nombre de jobs et de machines.

- les gaps :

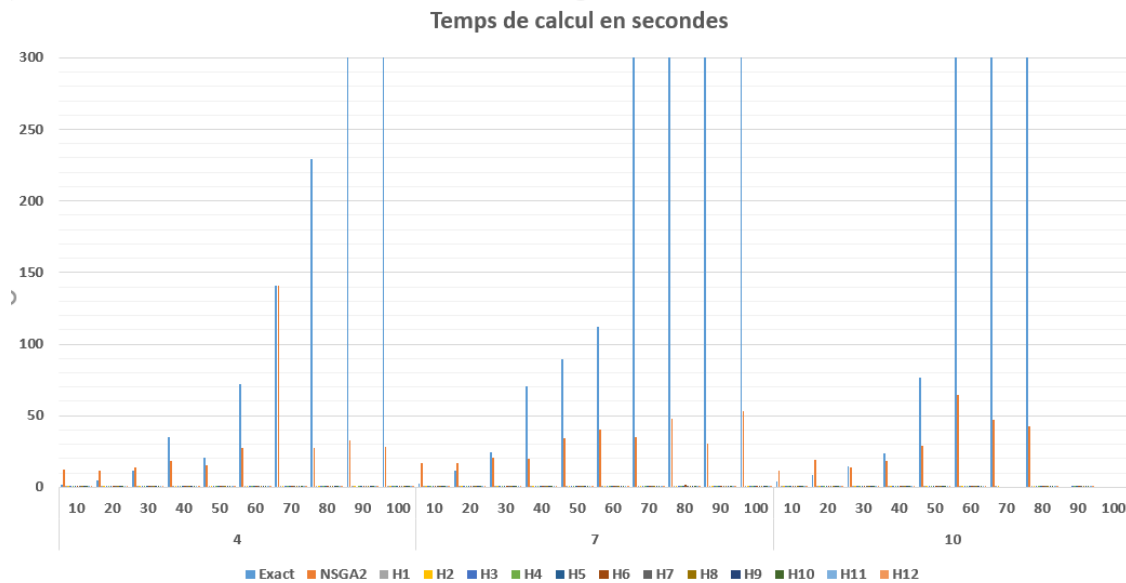


Figure 5 – Temps CPU par approches

Dans un premier temps nous avons l'image Figure 6 qui représente le gap entre le pourcentage de point optimale trouvé par NSGA2 et les heuristiques puis l'image Figure 7 qui représente le gap entre le pourcentage de point faible (dominé) trouvé par NSGA2 et les heuristiques.

Nous remarquons que pour les jobs de 10 à 30 sur 4 machines, le gap est égal à 0 pour l'heuristique H1 et NSGA2 car ils trouvent les mêmes solutions que la méthode exactes. Nous voyons aussi que jusqu'avant le job 100 NSGA2 a un gap entre 0 et 20 car trouve les mêmes solutions ou plus de points que la méthode exacte mais à partir de 100 jobs l'écart du gap devient important par contre sur l'image Figure 7 nous remarquons que NSGA2 trouve les points de Pareto faible à 100 % donc le gap est à 0. Pour les heuristiques H1 et H2 nous avons des gaps élevés pour le pourcentage de solutions optimale mais des petits gaps pour le pourcentage de solutions faible.

— l'hypervolume et la distance euclidienne :

Après avoir étudié le pourcentage de solutions trouvé par NSGA2 et les heuristique, on cherche à connaître la distance entre deux points, celui des approches et celui de la méthode puis la surface de séparation des solutions de la méthode exacte avec celles de NSGA2 et des heuristiques.

Le but est de savoir la proximité qui existe entre les deux solutions. Dans l'image Figure 8 nous voyons que l'hypervolume de NSGA2 est entre 1 et des valeurs négatives que nombre ces solutions sont très proches que celle de la méthode exacte par contre l'hypervolume des heuristiques pour les petits instance de 10 à 30 jobs sont petits mais quand on augmente le nombre de job l'hypervolume augmente et plus nous augmentons le nombre de machine l'hypervolume augmente aussi plus que la précédente par exemple la moyenne des hypervolumes sur 4 machines avec 10 à 100 jobs est inférieur à celui de hypervolumes sur 7 machines avec 10 à 100 jobs.

Nous remarquons le même phénomène que pour la distance euclidienne, la distance euclidienne des différents heuristiques augmentent aussi augmente en fonction du nombre de job et du nombre de machine par exemple la moyenne des distances euclidiennes sur 4 machines avec 10 à 100 jobs est inférieur à celle de la moyenne des distances euclidiennes sur 7 machines avec 10 à 100 jobs, comme le montre l'image Figure 9.

Avec NSGA2 sur l'image Figure 9, nous avons une courbe comprise entre 0 et 3 ce qui nous conforte dans l'idée de la proximités des resultats de NSGA2 et la méthode exacte.

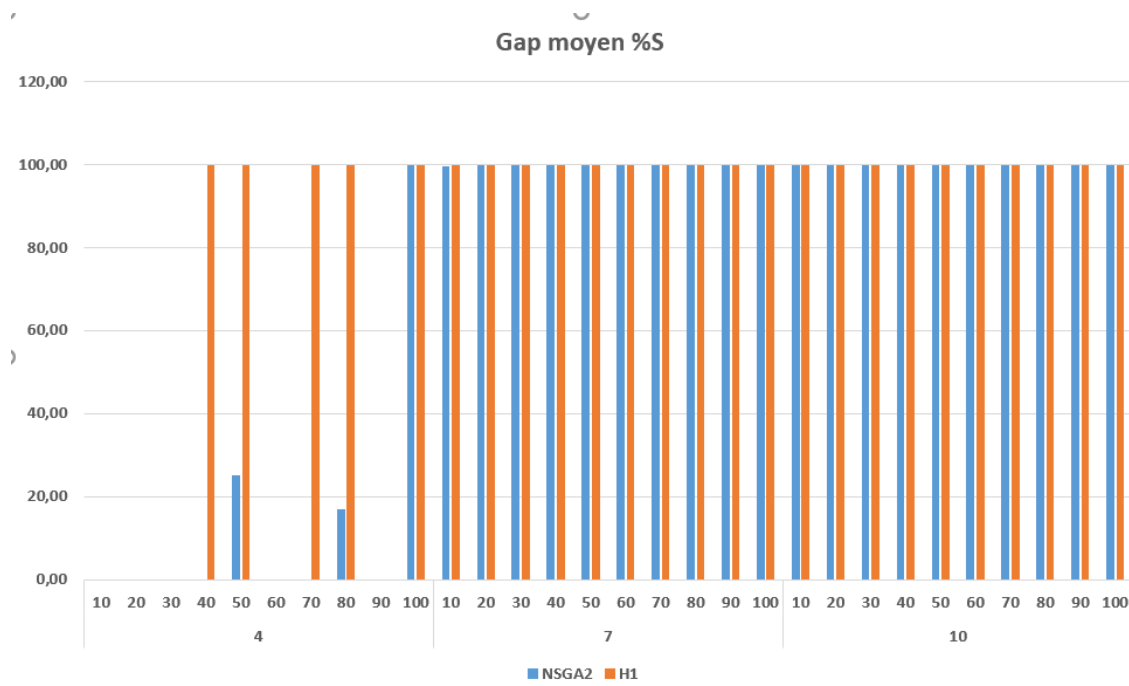


Figure 6 – Gap Pourcentage Moyen de %S

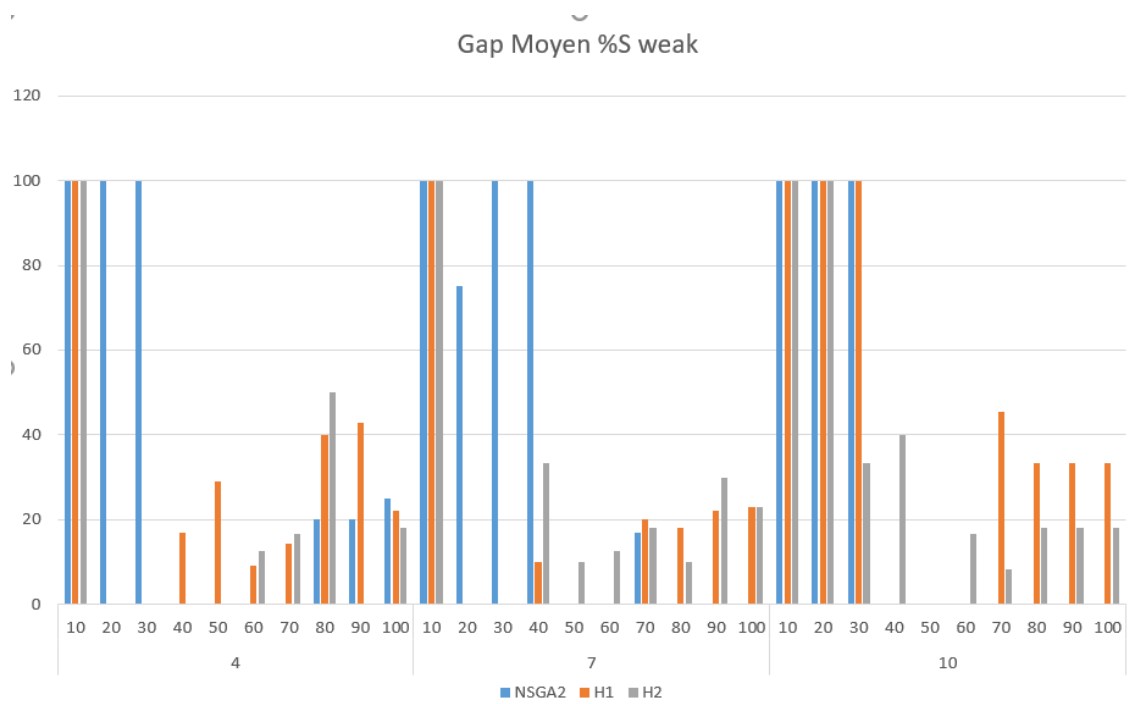


Figure 7 – Gap Pourcentage Moyen de %S weak

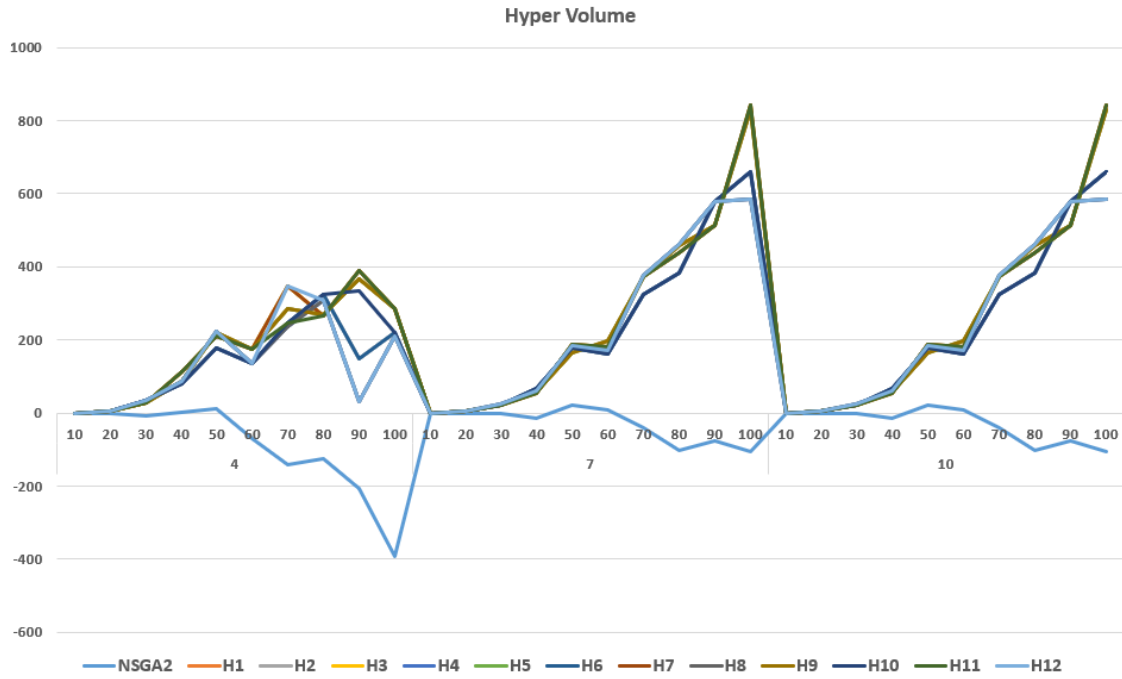


Figure 8 – Hypervolume entre Méthode exacte et NSGA2 avec les heuristiques

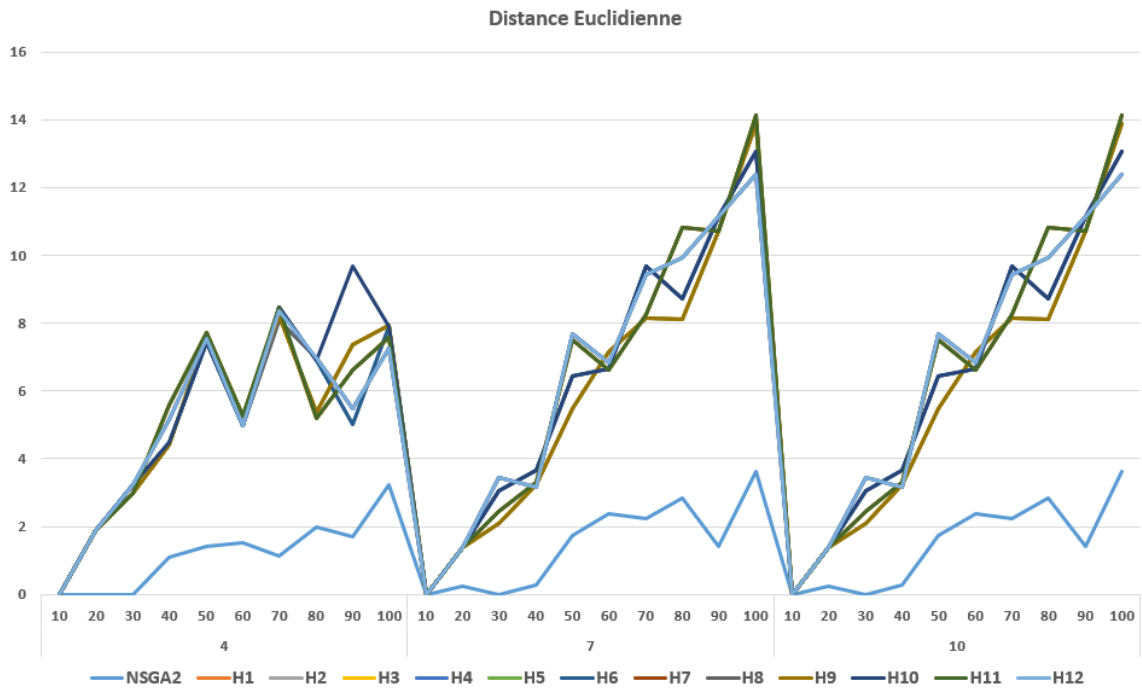


Figure 9 – la distance euclidienne entre Méthode exacte et NSGA2 avec les heuristiques

# 6

## Bilan et conclusion

### 1 Réalisé :

Les travaux que j'ai eu à faire lors de mon PRD :

- La compréhension du contexte et les objectifs à atteindre.
- L'étude de l'existant afin de comprendre comment l'application fonctionne et ce qui a été fait par le précédent PRD.
- La modélisation mathématique du problème d'ordonnancement et d'affectation multicritère et multi machine.
- L'étude de définition de solution de Pareto et de front de Pareto : J'ai étudié la définition de solution de Pareto et de front de Pareto. Ces deux théories sont utilisées en cas multicritères.
- L'étude de Combinaison Linéaire : La Combinaison Linéaire est une méthode exacte pour résoudre un problème de type multi critères et utilise la formulation mathématique. Mais on ne peut obtenir que un front de Pareto supporté.
- L'étude de Epsilon Contrainte Approche : Epsilon Contrainte Approche est une méthode exacte pour résoudre le problème de type multi critères. Cette méthode peut retourner une solution de Pareto faible. Il faut résoudre un problème symétrique pour obtenir une solution de Pareto optimale.
- Étude des heuristiques pour trouver des solutions proches de la solution optimale.
- L'étude de l'algorithme génétique NSGA2 : Cet algorithme génétique peut être utilisé à résoudre le problème multicritères. J'ai appris la démarche et les points importants de cet algorithme.
- Maîtrise de la bibliothèque CPLEX à utiliser.
- Mise en place de l'architecture et de la structure de l'application.
- l'écriture des résultats d'une instance obtenu à partir d'un algorithme dans un fichier texte.
- le développement de la méthode exacte l'épsilon contrainte, en testant différentes bornes pour les valeurs d'épsilon.
- Le développement de l'algorithme génétique et l'utilisation des stratégies pour diversifier les individus d'une population.
- l'amélioration des performances de NSGA2 avec un jeu sur les paramètres afin de trouver le bon paramètre.
- le développement des différents heuristiques.

- l'implémentation de l'interface graphique de l'application et la liaison avec les algorithmes développer.
- L'analyse complètes des performances des différents algorithmes sur un ensemble d'instance dont le nombre de job varie et le nombre de machine varie. Puis y tirer des conclusions.
- Finalisation du PRD par la finissions du rapport, des support de soutenances

## 2 Reste à faire :

Nous avons atteints l'objectif du PRD , mais il reste des choses à améliorer :

- liaison des fonctionnalités de comparaison des résultats obtenus des différents heuristique et méta heuristique avec l'interface graphique.
- Mettre tous les différents cas d'ordonnancement et d'affectation sous une seule application.

## 3 Bilan sur la qualité

L'un des objectifs du PRD est non seulement d'implémenter les recherches, mais aussi faire preuve de réflexion sur la manière de coder, c'est à dire penser à l'architecture de l'application, à l'évolution du programme et à la qualité du rendu au client.

Du coup avant de me lancer , j'ai écrit des documents pour recueillir les besoins du client et dès le début du developpement il m'est été nécessaire de faire une bonne conception de l'application, en respectant la qualité du logiciel( la compréhension rapide des classes et méthodes par les commentaires) à travers mes formations suivis et l'évolution du projet d'où l'utilisation du design pattern MVC (Modèle Vue Contrôleur) et stratégie.

Pour vérifier l'exactitude des résultats fournis par les algorithmes développer, j'ai mis en place des tests unitaires et fonctionnels sur ces algorithmes pour prouver la fiabilité de mes algorithmes implémenter.

Tout cela sous le contrôle de Mr Jean Yves Ramel, qui vérifie l'aspect qualité de mon développement.

## 4 Bilan auto-critique sur la gestion de projet

Pour un bilan générale de ma gestion de projet, je dirais mettre bien organiser sur la gestion des tâches à effectuer et leur date limite tout en respectant les spécifications des tâches.

j'ai utilisé une méthode de gestion de projet : Cycle en V, avec des échanges fréquents avec le client (MOA).

la gestion de version passe par GitLab, le suivie des tâches avec Trello, la documentation de l'application avec Google Drive.

Toutes les tâches lier à un semestre étaient planifié et il n'y a eu d'énorme changement sur la planification initiale visible dans la section planning.

En terme d'amélioration, je pourrais mieux calculer la charge de travail lier à une tâche et faire plus attention aux risques globales lors du developpement de l'application.

## 5 Conclusion

C'est avec une grande fierté et un sentiment de mission accompli que je finis ce PRD qui m'a tant appris en terme d'analyse des problèmes, de conception d'architecture de projet et de solution. Ce projet touche tous les domaines de développement d'application de la phase de recherche, de développement et de gestion de projet

A travers ce projet , j'ai réussi à acquérir des compétences dans l'analyse des problèmes et en gestion de projet du début à la fin, et à consolider mes acquis dans le developpement d'application en JAVA et à être de plus en plus autonome.

J'aimerais finir par un immense remerciement à mon encadrant et MOA **Mr Boukhalifa Zahout** de sa disponibilité tout le long de mon PRD, toujours là pour répondre à mes questions, à me débloquer en me donnant des idées, son envie de me faire passer son savoir sur le sujet et sans oublier ces conseils que je garde pour ma vie professionnelle. j'ai tisser de très bonne relation avec mon MOA et il reste un acteur clé dans la réussite de mon projet de fin d'étude.

# 7

## Annexe

### 1 Description des interfaces externes du logiciel

#### 1.1 Interfaces matériel/logiciel

L'une des caractéristiques du matériel est sa capacité de mémoire et sa puissance de calcul. La puissance de calcul quant à elle est primordiale pour obtenir des résultats dans un temps acceptable surtout dans le cas de l'exécution de la méthode exacte qui est NP-Difficile, le temps de calcul croît de manière exponentielle en fonction des données d'entrées.

Le matériel doit supporter le logiciel et le programme sera lancé dans l'IDE choisie pour compiler le projet.

#### 1.2 Interfaces homme/machine

L'interface graphique du programme permet l'interaction avec la MOA qui doit être informée sur l'utilisation des différents menus et l'utilité de chaque action faite car chaque est liée à une fonctionnalité du programme.

L'ergonomie n'est pas primordiale dans notre cas, elle vise plutôt le niveau fonctionnel parce que notre programme est plus dans la recherche scientifique, permettre au MOA de faire des tests scientifiques. L'interface doit être intuitive.

### 2 Spécifications fonctionnelles

#### 2.1 Description des fichiers d'entrée

Une instance sera dans un fichier texte, à l'intérieur duquel nous trouvons les jobs de chaque agent où l'agent A possède  $\pi^A\%$  des jobs et l'agent B possède  $\pi^B\%$  des jobs, par exemple l'agent A possède 50 % et l'agent B possède 50 % ce qui veut dire que si nous avons 8 jobs alors les 4 premiers jobs appartiennent à A et les 4 autres jobs à B.

Nous respectons la modélisation déjà faite lors du précédent PRD, dont la modélisation est la suivante :

```

Nombre_Jobs  Nombre_Ressources  Nombre_Jobs_A  Nombre_Machine

QuantitéRessourceMachine1  QuantitéRessourceMachine1  ...  QuantitéRessourceMachine1
QuantitéRessourceMachine2  QuantitéRessourceMachine2  ...  QuantitéRessourceMachine2
.
.
.
QuantitéRessourceMachinem  QuantitéRessourceMachinem...  QuantitéRessourceMachinem

NumJob1 depart Job1  finJob1  Agent  QuantitéRessource  QuantitéRessource  .....
.....

NumJob2 depart Job2  finJob2  Agent  QuantitéRessource  QuantitéRessource  .....
.
.
.

NumJobn depart Jobn  finJobn  Agent  QuantitéRessource  QuantitéRessource  .....

```

**Figure 1** – Structure pour les fichiers d'instances

Exemple : 8 jobs , 2 machines ayant chacun 3 Ressources , 4 jobs de l'agent A et 4 jobs de l'agent B.

```

8 3 4 2
1000 1000 1000
1000 1000 1000
0 0 4 A 500 300 400
1 1 4 A 125 600 300
2 4 8 A 125 250 200
3 5 9 A 250 500 300
4 0 6 B 125 500 250
5 1 5 B 250 800 600
6 3 6 B 500 700 250
7 3 8 B 500 300 700

```

**Figure 2** – Exemple de fichier d'instances

Si nous nous basons sur l'exemple **Figure 2**, avec les 8 jobs et que l'agent A a 50 % et l'agent B a 50 % alors A possède  $Job_A = \{0, 1, 2, 3\}$  et B possède  $Job_B = \{4, 5, 6, 7\}$  et la lettre de l'agent est indiqué devant le job à la quelle il possède.

## 2.2 Description des fichiers de sortie ou fichier de résultats

Vu que mon problème est différent de celui de l'ancien PRD alors la génération de fichier de résultat est la suivante **Figure 3**, elle contient plusieurs Solution conçu avec les valeurs des fonctions objectives de A et B ce qui constituent le front de Pareto, puis la liste par agent, par machine des jobs ordonnancés avec leur date de début et de fin. Et enfin le nombre de job Ordonné par Agent :

```

Instance: nom du fichier de resultat
Front de Pareto: (Xa,Yb)
IdMachine  IdJob  IdProprietaire  Si  Ci
0           1      A           771  1118
0
1
1
1
IdProprietaire  NombreJobOrdonnance
A                2
B                3
Front de Pareto: (Xa,Yb)
IdMachine  IdJob  IdProprietaire  Si  Ci
0           1 |    A           771  1118
0
1
1
1
IdProprietaire  NombreJobOrdonnance
A                2
B                3

```

Figure 3 – Structure pour les fichiers de résultat d'une instance

```

Instance : /home/kake/Documents/Projet_recherche_et_developpement/PRD_JAVA/prdordonnementmulticriterejava/PRD/Ressource/Instances/Dossi
Type de résolution : Optimale Solution NSGA2
Optimal Value (A - B) = 0.0 - 2.0
temps écoulé (en secondes): 2.99
IdAgent IdMachine  N°Job  Si  Fi
1 1 2 1 4
1 1 4 5 9
1 2 1 0 4
1 2 3 4 8
2 1 8 3 8
2 2 7 3 6
IdAgent Nombre Job Ordonné
1 4
2 2
Instance : /home/kake/Documents/Projet_recherche_et_developpement/PRD_JAVA/prdordonnementmulticriterejava/PRD/Ressource/Instances/Dossi
Type de résolution : Optimale Solution NSGA2
Optimal Value (A - B) = 2.0 - 1.0
temps écoulé (en secondes): 2.991
IdAgent IdMachine  N°Job  Si  Fi
1 2 1 0 4
1 2 3 4 8
2 1 5 0 6
2 1 8 3 8
2 2 7 3 6
IdAgent Nombre Job Ordonné
1 2
2 3

```

Figure 4 – Exemple de fichier de résultat d'une d'instances

Ces valeurs nous permettront de faire la comparaison des résultats provenant des heuristiques

ou des méta heuristique et de la méthode exacte.

## 2.3 Fonctionnalité du système

Notre but est d'utiliser des fonctionnalités qui existe dans l'existant fait lors d'un ancien PRD comme la lecture d'un fichier d'instance et la création d'une instance à partir de ce fichier. Puis allons ajouter mes nouvelles fonctionnalités. Et toutes ces fonctionnalités sont liées au cas multicritère et multi machine.

### 2.3.1 Définition de la fonction 1 : Exécution méthode Exacte

1. Identification de la fonction 1 :  
Cette fonction permet d'exécuter un algorithme basé sur la méthode exacte (Epsilon-Contrainte, Combinaison linéaire) de notre problème.
2. Présentation de la fonction 1 :
  - Nom : ExecuteMethodeExacte;
  - Priorité : primordiale.
3. Description de la fonction 1 :  
Cette fonctionnalité permet d'implémenter une méthode exacte et exécuter une des algorithmes choisie par l'utilisateur et de retourner un ensemble de résultat : un front de Pareto , soit une solution de Pareto optimale comme le résultat de traitement et les données liés aux agents et les machines.
4. Décrire précisément :
  - Entrées : une instance contenant la description du programme ( jobs, machines,...).
  - Sorties : un ensemble des solutions optimales (front de pareto).

### 2.3.2 Définition de la fonction 2 : Écriture du résultat dans Fichier

1. Identification de la fonction 2 :  
Cette fonction permet d'écrire les résultats d'un algorithme choisit par l'utilisateur qui retourne un front de Pareto dans un fichier texte.
2. Présentation de la fonction 2 :
  - Nom : EcritureResultatFichier;
  - Priorité : primordiale.
3. Description de la fonction 2 :  
Cette fonctionnalité permet d'écrire dans un fichier l'ensemble des agents, des machines et des jobs ordonnancés et le front de Pareto obtenu à partir d'un algorithme exécuté dans un fichier texte. la structure de ce fichier sera comme : **Figure 3**
4. Décrire précisément :
  - Entrées : l'ensemble des résultats contenant le front de Pareto.
  - Sorties : une variable booléenne qui gère la bonne exécution de l'écriture avec succès.

### 2.3.3 Définition de la fonction 3 : Exécution Heuristique

1. Identification de la fonction 3 :  
Cette fonction permet d'exécuter un algorithme basé sur la ou les méthodes approchées(des heuristiques basées sur le tri et l'affectation des jobs à aux machines) de notre problème.
2. Présentation de la fonction 3 :

- Nom : ExecuteHeuristique ;
  - Priorité : primordiale.
3. Description de la fonction 3 :
- Cette fonctionnalité permet d'implémenter des heuristiques et exécuter des algorithmes qui donnent des résultats se rapprochant de ceux obtenu par les méthodes exactes puis de retourner un ensemble de résultat : un front de Pareto , soit une solution de Pareto optimale comme le résultat de traitement et les données liés aux agents et les machines.
4. Décrire précisément :
- Entrées : une instance contenant la description du programme ( jobs, machines,...).
  - Sorties : un ensemble des solutions optimales (front de pareto).

#### 2.3.4 Définition de la fonction 4 : Obtention du Front Pareto optimale

1. Identification de la fonction 4 :
- Cette fonction permet de modéliser le front de Pareto qui lit le résultat obtenu par les algorithmes pour résoudre le problème, elle retourne un objet qui permettra d'avoir les fronts de Pareto optimale ou non-dominés.
2. Présentation de la fonction 4 :
- Nom : FrontPareto
  - Priorité : primordiale.
3. Description de la fonction 4 :
- Une fois qu'on a obtenu les différents résultats de front de Pareto, nous structurons ces résultats et obtenir une solution optimale. Une solution optimale est d'avoir des fronts de Pareto non-dominés avec les coordonnées x (fonction objective de A) et y (fonction objective de B).
4. Décrire précisément :
- Entrées : Un liste de résultat contenant des front de Pareto.
  - Sorties : Les fronts de Pareto optimale.

#### 2.4 Programme de comparaison de front de Pareto :

l'exécution des différents approches nous donne de multiple résultat,dans cette partie le but est de trouver l'écart qui existe entre les solutions obtenues par les méthodes exactes avec celles des heuristiques en faisant une comparaison.

##### 2.4.1 Définition de la fonction 1 : Pourcentage de la Solution Exacte

1. Identification de la fonction 1 :
- Cette fonction permet de calculer le pourcentage de solutions de la méthode approchée,égales à celle de la solution exacte par rapport au nombre total de solutions de la méthode approchée.
2. Présentation de la fonction 1 :
- Nom : PourcentageExacteSolution ;
  - Priorité : primordiale.
3. Description de la fonction 1 :
- En utilisant le méthode approchée, on peut obtenir un ensemble de solutions de Pareto optimales. Donc pour bien savoir la qualité de ce résultat, on va calculer le pourcentage de solutions de Pareto optimales par rapport au nombre total exact de solutions de Pareto optimales. On compare chaque solution de Pareto de la méthode approchée avec les

solutions de pareto de la méthode exacte. Si on trouve la même solution dans l'ensemble des solutions de Pareto de la méthode exacte, le compteur incrémente par un. Après le nombre de solutions de Pareto communes est divisé par le nombre de solutions de Pareto totales.

Cette fonction prend en entrée la liste des solution de la méthode exacte et la liste des solutions non dominées de la méthode approchées. Elle donne en sortie le pourcentage souhaité.

4. Décrire précisément :

- Entrées : la liste des solution de la méthode exacte et la liste des solutions non dominées de la méthode approchées.
- Sorties : Un pourcentage de comparaison.

### 2.4.2 Définition de la fonction 2 : Distance Moyenne

1. Identification de la fonction 2 :

Cette fonction permet de calculer la moyenne des distances euclidiennes les plus courtes entre les solutions du premier et du second front de Pareto.

2. Présentation de la fonction 2 :

- Nom : MoyenCourtDistance;
- Priorité : primordiale.

3. Description de la fonction 2 :

Cette fonction prend en entrée la liste des solutions non dominées de chaque front et donne en sortie la moyenne souhaitée.

## 2.5 Programme IHM et l'affichage des résultats

Dans cette section, nous visons à intégrer notre développement dans l'interface graphique et afficher les résultats sous formes graphique. Les fonctionnalités sont secondaire.

### 2.5.1 Définition de la fonction 1 : Mise en place d'une Application

1. Identification de la fonction 1 :

Cette partie permet de mettre en place une interface graphique, permettant de choisir des menus : de générations d'instance, de résolutions d'instances avec les différentes méthodes (Exacte, Heuristique, Méta-Heuristique) et la comparaison des résultats.

2. Présentation de la fonction 1 :

- Nom : Interface graphique
- Priorité : primordiale

3. Description de la fonction 1 :

Une interface graphique ou une application interactive , donnant l'opportunité à l'utilisateur de résoudre un fichier d'instance et voir les résultats obtenus

4. Décrire précisément :

- Entrées : Rien
- Sorties : Rien

### 2.5.2 Définition de la fonction 2 : Graphisme du Front de Pareto

1. Identification de la fonction 2 :

Cette fonction permet d'afficher sous forme de nuage de point un front de Pareto.

2. Présentation de la fonction 2 :
  - Nom : GraphismeFrontPareto;
  - Priorité : secondaire.
3. Description de la fonction 2 :  
Cette fonctionnalité nous affiche sous forme graphique un front de Pareto optimale.
4. Décrire précisément :
  - Entrées : un front de pareto
  - Sorties : représentation graphique du front de Pareto.

### 2.5.3 Définition de la fonction 3 : Diagramme de Gantt des jobs ordonnés

1. Identification de la fonction 3 :  
Cette fonction permet d'afficher sous forme de diagramme de Gantt l'organisation des jobs.
2. Présentation de la fonction 3 :
  - Nom : DiagrammeGantt;
  - Priorité : secondaire.
3. Description de la fonction 3 :  
Cette fonctionnalité nous affiche sous forme de diagramme de Gantt, pour chaque machine comment les jobs y sont ordonnancés et leurs appartenance à un agent.
4. Décrire précisément :
  - Entrées : L'organisation des jobs appartenant à un agent sur les machines.
  - Sorties : représentation graphique de l'organisation.

## 3 Spécifications non fonctionnelles

### 3.1 Contraintes de développement et conception

Il n'y a pas de contrainte particulière sur le matérielle mais il est important d'avoir un IDE comme IntelliJ pour le développement du module de calcul de solutions et de la comparaison. Il est cependant important de préciser que plus la machine de développement aura une puissance de calcul importante, plus les tests de calculs de solutions seront rapides.

Le langage de programmation adopté sera le JAVA. C'est en effet un langage orienté Objet et optimisé au niveau de la gestion automatique de la mémoire qui permettra d'obtenir de meilleures performances. Pour le développement des calculs de solutions exactes, le solveur cplex216 sera utilisé. Le développement se fera sous Linux, distribution Ubuntu car il est plus facile d'intégrer le solveur cplex216 sur ce système d'exploitation. Le développement de d'autre interface graphique ( l'ajout des fonctionnalités dans le projet initiale) se fera avec le Module Swing de JAVA.

### 3.2 Performances

Dans notre situation le calcule des solutions pour des cas d'ordonnancements multicritère et multi machines avec ressources multiples, les performances de notre application dépendra fortement la taille des données reçu en entré et affectées par la capacité de calcul de la machine sur laquelle le programme s'exécute.

Le problème d'ordonnancement étant NP-Difficile, des méthodes heuristiques et méta Heuristique seront implémentées pour résoudre efficacement et plus rapidement les instances ayant une taille importante contrairement à une méthode exacte.

### 3.3 Capacités

Le programme est prévue de s'exécuter sur la machine où se trouve l'exécutable sans limite fixées au niveau de la taille des instances et du nombre d'instance. Néanmoins le temps de traitement d'un algorithme dépend de la taille des instances.

### 3.4 Modes de fonctionnement

Pour utiliser le programme, il suffit de lancer l'exécutable en ligne de commande ou dans un IDE permettant de compiler le code JAVA comme IntelliJ ou Éclipse , puis interagir avec l'interface graphique.

### 3.5 Contrôlabilité

Afin de suivre l'exécution du programme, les messages d'exécution vont être affichées dans le console. Normalement la sortie standard du système va être utilisé.

### 3.6 Sécurité

L'application étant utilisée uniquement en interne par des doctorants et des enseignants chercheurs, il ne sera pas nécessaire d'implémenter un système de mot de passe ou de sécurité particulière.

### 3.7 Intégrité

Il n'est pas nécessaire de mettre en place des protections contre la déconnexion imprévue car c'est programme en local sur une machine. En effet, le programme n'effectue que du calcul pour obtenir des solutions qui seront ensuite directement sauvegardées dans un fichier texte. Si un problème survenait durant le calcul d'une solution ou un comportement imprévu alors il faudrait relancer le calcul. Il n'y a donc pas besoin de se soucier de la récupération des données.

### 3.8 Maintenance et évolution du système

Le but est de mettre un environnement de développement qui permettra à d'autre développeur de réutiliser le programme et ajouter des fonctionnalités comme le cas multicritère mais avec plus de deux agents.

## 4 Gestion de projet

La gestion de projet se fait avec la méthode de cycle en V. les rendez-vous avec l'encadrant sont fait chaque semaine où je dois indiquer mon avancement et mes problèmes rencontrés afin qu'il puisse m'expliquer et ensemble trouvons une solution adéquate et faire un compte rendu chaque Jeudi.

Ensuite, l'encadrant donne les objectifs suivants et les attendus ou livrables. Des livrables sont demandés par l'encadrant :

- un rapport de l'état de l'art
- Le premier résultat de la méthode exacte avec Epsilon-Contrainte.
- un cahier de spécifications.
- l'analyse des résultats des différents algorithmes (Heuristiques, algorithme génétique, méthode exacte).
- l'application complète avec l'interface graphique et l'implémentation des différents algorithmes.

Les outils de gestion de projet sont :

- Le diagramme de Gantt : pour le planning.
- L'application Trello : pour suivre les tâches en respectant le délais.
- GitLab : pour le contrôle de version.
- GitKraken : logiciel qui interface Git.

### 4.1 Découpage en tâche

#### 4.1.1 Tâche 1 : Gestion de projet et version

- Description de la tâche : Cette tâche est pour déterminer la méthode et les outils de la gestion de projet.
- Estimation de charge : Cette tâche est estimée à 1 jour/homme.

#### 4.1.2 Tâche 2 : Compréhension des objectifs et le contexte

- Description de la tâche : Cette tâche est pour comprendre le sujet du projet. Pour comprendre les objectifs et le contexte, un rendez-vous avec l'encadrant est obligatoire.
- Estimation de charge : Cette tâche est estimée à 3 jour/homme.

#### 4.1.3 Tâche 3 : Étude de l'existant

- Description de la tâche : Cette tâche est pour comprendre la structure et le fonctionnement du programme ou l'application existant.
- Estimation de charge : Cette tâche est estimée à 4 jour/homme.

#### 4.1.4 Tâche 4 : Étude de la documentation

- Description de la tâche : Cette tâche est divisée en plusieurs sous tâches : étude du cas Mono machine, étude du cas Multi machine et étude du cas multi critère afin d'avoir une source et un point de départ pour trouver des solutions de modélisation à notre problème.
- Estimation de charge : Cette tâche est estimée à 6jour/homme.

#### 4.1.5 Tâche 5 : Rédaction de l'état de l'art

- Description de la tâche : Cette tâche est pour décrire les documents que j'ai lu et les points clés que ces documents m'apporte dans l'avancé de mon PRD.
- Estimation de charge : Cette tâche est estimée à 6 jour/homme.
- Livrable : l'état de l'art.

#### 4.1.6 Tâche 6 : Recherche, Analyse et conception du cas Multi critère

- Description de la tâche : Cette tâche est pour rechercher la modélisation de notre problème, d'analyser des heuristiques et des méta heuristiques qui pourront donner des solution à notre problème.  
Cette Tâche est divisée en sous tâche comme : analyse de la combinaison linéaire, analyse de l'épsilon-contrainte, analyse de l'algorithme de Greedy.
- Estimation de charge : Cette tâche est estimée à 3 jour/homme.
- Livrable : explication de mes approches à mon encadrant.

#### 4.1.7 Tâche 7 : Rédaction de Cahier de Spécification

- Description de la tâche : Cette tâche est pour rédiger le cahier de spécification. Le cahier de spécification sera modifié lors de chaque consultation avec l'encadrant.
- Estimation de charge : Cette tâche est estimée à 3 jour/homme.
- Livrable : Un cahier de spécification sera à rendre.

#### 4.1.8 Tâche 8 : Développement de l'épsilon-contrainte

- Description de la tâche : Cette tâche est pour la validation de la modélisation mathématique du problème, voir si nous obtenons une solution optimale qui doit être retourné. Ce résultat est des différents fronts de Pareto avec différentes instances d'entrées.
- Estimation de charge : Cette tâche est estimée à 2 jour/homme.
- Livrable : Un code et présentation de la solution.

#### 4.1.9 Tâche 9 : Étudier l'algorithme génétique NSGA2

- Description de la tâche : Cette tâche est pour étudier l'algorithme génétique NSGA2. Il faut prendre du temps pour chercher les littératures. Il faut aussi du temps pour analyser cet algorithme et comprendre la démarche de cet algorithme.
- Estimation de charge : Cette tâche est estimée à 1 jour/homme.

**4.1.10 Tâche 10 : Rédaction du rapport final mi-parcours**

- Description de la tâche : Cette tâche est pour rédiger un rapport final pour le semestre 9
- Estimation de charge : Cette tâche est estimée à 20 jour/homme.
- Livrable : Le rapport final de la semestre 9 avec l'outil Latex.

**4.1.11 Tâche 11 : Préparation de la soutenance mi-parcours**

- Description de la tâche : Cette tâche est pour préparer la soutenance mi-parcours. la soutenance aura lieu le 12/12/2018.
- Estimation de charge : Cette tâche est estimée à 2 jour/homme.
- Livrable : Un exposé en Power Point.

**4.1.12 Tâche 12 : Développement de l'épsilon contrainte**

- Description de la tâche : Cette tâche est d'implémenter la méthode exacte par la politique d'épsilon contrainte.
- Estimation de charge : Cette tâche est estimée à 4 jour/homme.
- Livrable : Programme et résultat obtenu.

**4.1.13 Tâche 13 : Écriture des résultats dans un fichier**

- Description de la tâche : Cette tâche est d'écrire la ou les solutions obtenus dans un fichier texte le plus clair que possible.
- Estimation de charge : Cette tâche est estimée à 1 jour/homme.

**4.1.14 Tâche 14 : Développement des Heuristiques**

- Description de la tâche : Cette tâche est d'implémenter les différents heuristiques.
- Estimation de charge : Cette tâche est estimée à 4 jour/homme.
- Livrable : Programme et résultat obtenu.

**4.1.15 Tâche 15 : Obtention du front de Pareto**

- Description de la tâche : Cette tâche est d'implémenter ou de représenter une solution de Pareto puis de choisir parmi une liste de solution de Pareto, le front de Pareto optimale c'est à dire des solutions non dominée.
- Estimation de charge : Cette tâche est estimée à 1 jour/homme.

**4.1.16 Tâche 16 : Développement de l'algorithme génétique NSGA2**

- Description de la tâche : Cette tâche est d'implémenter l'algorithme génétique NSGA2.
- Estimation de charge : Cette tâche est estimée à 4 jour/homme.
- Livrable : Programme et résultat obtenu.

#### 4.1.17 Tâche 17 :Interface graphique JAVA

- Description de la tâche : Cette tâche est d'implémenter une interface graphique permet de facilité la génération d'instance, la résolution d'instance et la comparaison des résultats. Puis lier l'interface à l'ensemble du programme.
- Estimation de charge :Cette tâche est estimée à 5 jour/homme.
- Livrable : Programme et test du fonctionnement.

#### 4.1.18 Tâche 18 :Validation des résultats

- Description de la tâche : Cette tâche est de voir avec l'encadrant l'exactitude des résultats.
- Estimation de charge :Cette tâche est estimée à 3 jour/homme.

#### 4.1.19 Tâche 19 :Analyse des différents algorithmes

- Description de la tâche : Cette tâche est de voir le fonctionnement ou le comportement d'un algorithme.
- Estimation de charge :Cette tâche est estimée à 10 jour/homme.
- Livrable : résultat obtenu.

#### 4.1.20 Tâche 20 Rédaction du Rapport final

- Description de la tâche : Cette tâche est de finir la rédaction du rapport final.
- Estimation de charge :Cette tâche est estimée à 10 jour/homme.
- Livrable : rapport final.

#### 4.1.21 Tâche 21 :Préparation de la soutenance finale

- Description de la tâche : Cette tâche est de préparer les diapos et supports pour la présentation.
- Estimation de charge :Cette tâche est estimée à 2 jour/homme.
- Livrable : diapositive

### 4.2 Analyse de faisabilité

La gestion de projet doit permettre de garantir une meilleure fiabilité de réalisation. Nous sommes partie d'un cahier des charges puis restreindre les idées et besoins avec un cahier de spécification dans le but de définir les livrables et les attentes en termes de qualité logiciel.

Dans ce projet, nous n'avons pas de coûts ni de budget associé mais une date limite. Si à la fin de la date limite, le projet n'est pas terminé, les recherches et analyse apportées aux problèmes seront utiles à un autre PRD afin de finir ce projet très complexe.

Les impacts sont amortis lors d'un changement d'avis ou de besoins car le weekend est mise en place pour rattraper les retards.

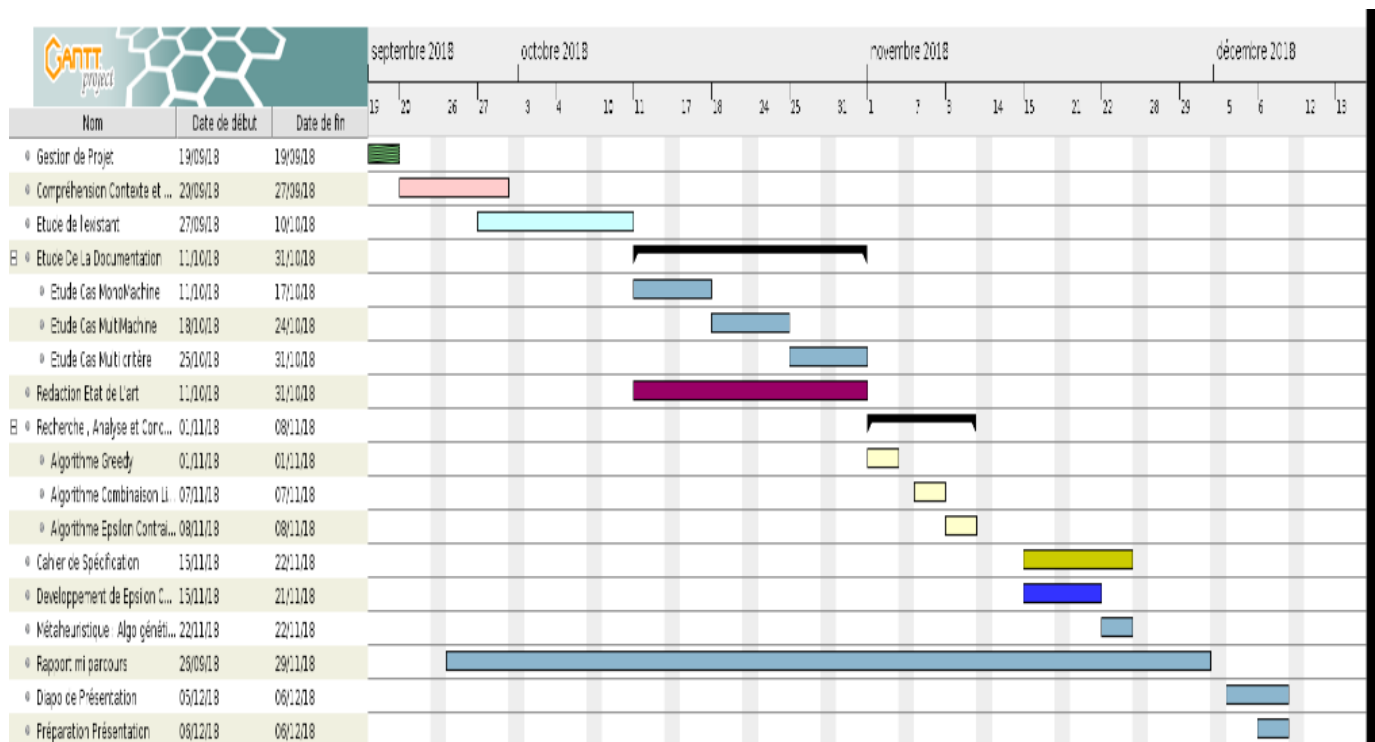
### 4.3 Analyse de risque

Le risque vient de la dépendance de certaines tâches donc le blocage d'une des tâches peut engendrer un retard considérable à l'ensemble du projet. Afin de trouver solution à ce risque, chaque tâche à un charge de travail réduit et l'augmentation des deadLines pour me laisser plus de temps à finir les tâches, si je finis avant la date limite alors je passe à la tâche suivante sans tarder ce qui permet de gagner du temps ou s'il y a blocage chercher de l'aide avec l'encadrant pour avancer rapidement.

### 4.4 Planning

#### 4.4.1 Le planning de la partie recherche

Voici la diagramme de Gantt comme **Figure 5** pour la partie Recherche.



**Figure 5** – diagramme de Gantt du planning de la recherche

#### 4.4.2 Le planning de la partie Développement

Voici la diagramme de Gantt comme **Figure 6** pour la partie Developpement.

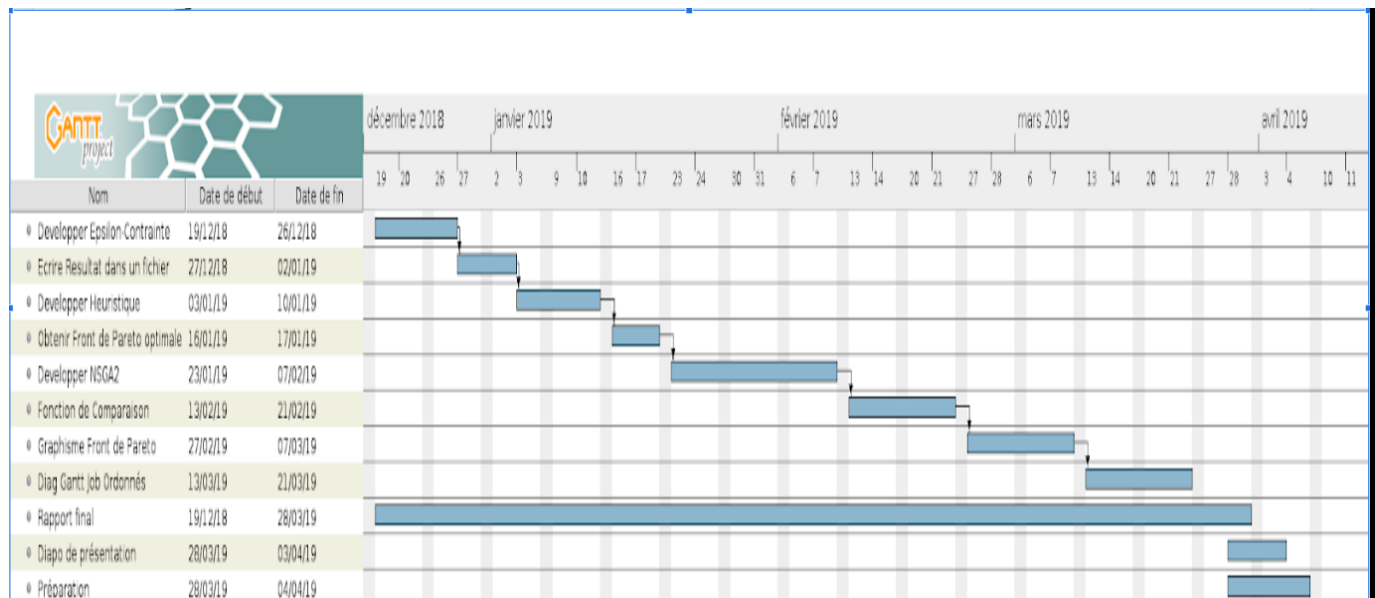


Figure 6 – diagramme de Gantt du planning de la partie developpement

# 8

## Document d'installation

Dans ce document, j'énumère les différentes manipulations à faire afin d'avoir un environnement de travail adéquat pour lancer l'application.

Les explications se baseront sur l'éditeur de code IntelliJ avec la version 8 de JAVA.



Figure 1 – Version de l'éditeur IntelliJ

Pour cette application, nous utilisons une librairie externe **Cplex** utile pour son fonctionnement pour les calculs exacts de notre modélisation mathématique en somme la méthode exacte.

les différentes étapes à suivre :

1. Récupérer le code source.
2. Avoir la librairie Cplex216, téléchargeable sur IBM ou fournit avec le projet.
3. Ouvrir le code Source avec l'éditeur **IntelliJ** qui est éditeur de code en JAVA. Dû à sa facilité d'utilisation, et d'insertion de librairie externe.

## 4. Avec L'IntelliJ les étapes à suivre :

- Cliquez droit sur la racine du projet , puis aller sur Module Setting ou taper F4.
- Une fenêtre est affichée , aller sur le Menu Dependencies de la fenêtre, nous verrons une bibliothèque Cplex, le but est de changer le chemin vers notre dossier Cplex216 à nous enregistrer dans un répertoire de la machine hôte ou à la racine du projet.
- Double cliquez sur Cplex : une nouvelle fenêtre s'ouvre et là on supprime les chemins existants pour les classes et la localisation de la librairie en cliquant sur le bouton (-) et les remplacer par ceux qui suivent en cliquant sur le bouton (+) :

**Classes :** -CheminDossier-/cplex-Version/cplex/lib/cplex.jar.

**Natives Library Locations :** — Pour Unix : -CheminDossier-/cplex-Version/cplex/bin/x86\_64-linux qui contient des fichiers .so.

- Pour Windows : -CheminDossier-/cplex-Version-/cplex/bin/x64-win64 qui contient des fichiers .dll

- Appliquer et fermer puis compiler votre programme pour voir qu'il n'y a pas d'erreur.

# 9

## Document d'utilisation

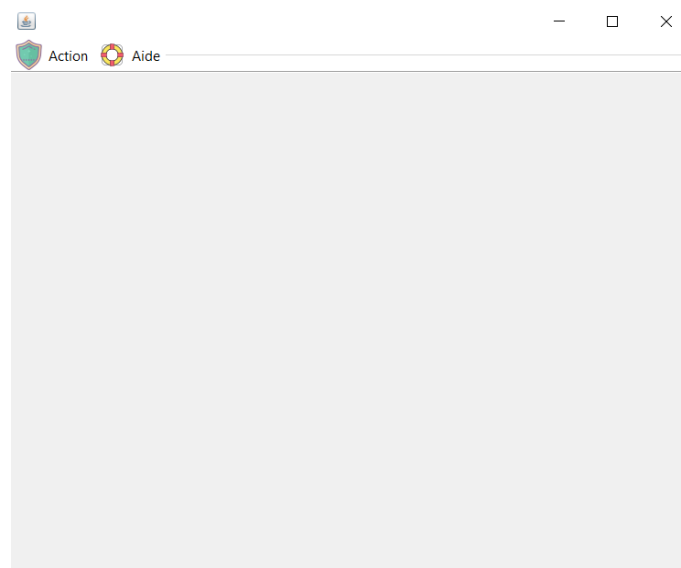
A la fin de mon projet de recherche et de développement, nous avons une application en java avec une interface graphique qui contient toutes les fonctionnalités défini dans la spécification.

L'utilisation de l'application passe par une interface graphique, qui contient les fonctionnalités de l'application à partir de la quelle, on peut manipuler pour faire des actions. Le lancement de l'application est à travers la classe View.MainView.java.

Avec l'interface graphique coder en java, l'utilisation deviens simple mais nous avons mis des classes Mains qui permettent de lancer un heuristique, un méta heuristique et une méthode exacte.

Pour un bon fonctionnement, l'utilisateur ne doit pas modifier la structure de l'application sauf personne ayant droit.

- Application centrale : c'est le premier onglet , elle représente l'accueil.



**Figure 1** – Fenêtre d'accueil de l'application.

- Les menus de l'application : cette partie, nous pouvons choisir les actions de générations d'instances , de résolution des instance , de comparaison des résultats , voir les auteurs de l'application ou encore de quitter l'application.

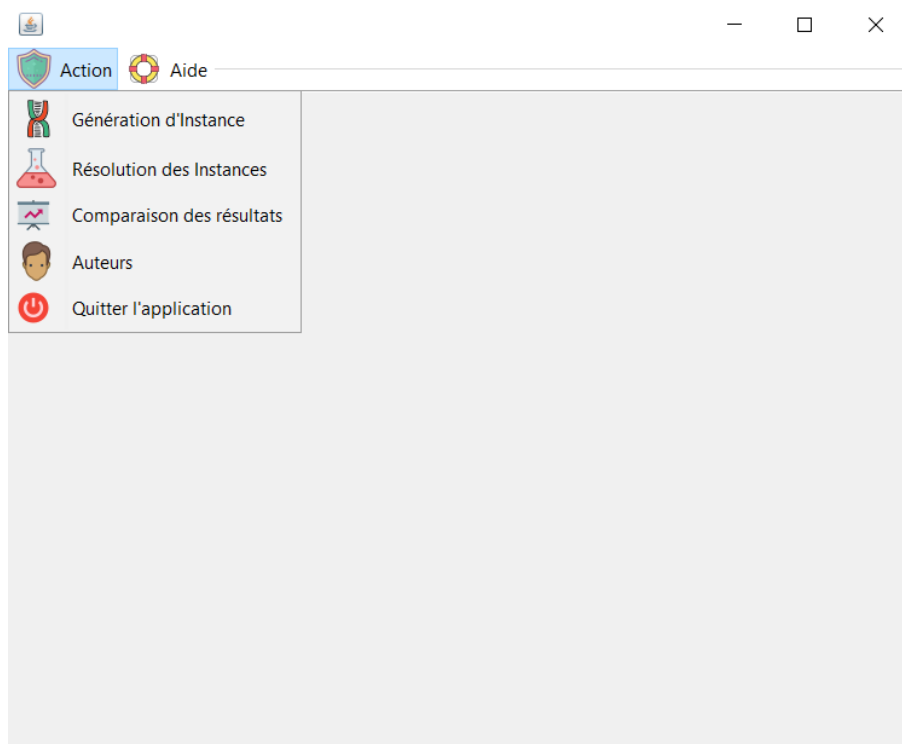


Figure 2 – Les différents menu de l'application

- La fonctionnalité de génération d'instance : Il faut rentrer les paramètres (le nombre de jobs, le pourcentage de jobs de l'agent A, le nombre de ressources, le nombre de machines, le nombre d'instance à générer et l'horizon de planification pour l'instance ) ou modifier ceux qui existent pour créer un fichier d'instance comme ceci **Figure 2** (Chapitre 10) avec la même structure expliqué dans la section pour les fichiers d'entrées dans le dossier **Instances/numéroInstance/NombreDeJobs/NombreRessources/NombreMachines/**.

 A screenshot of a dialog box titled 'Génération d'instance de Jobs'. The main heading inside is 'Génération d'Instance'. Below it, there are several labeled input fields:
 

- Nombre de Jobs :** A numeric input field with the value '1'.
- Pourcentage de Jobs de l'agent A :** A numeric input field with the value '100' followed by a '%' symbol. There is a circular icon with a double-headed arrow next to it.
- Nombre de Ressources :** A numeric input field with the value '3'.
- Nombre de Machines :** A numeric input field with the value '1'.
- Nombre d'Instance à Générer :** A numeric input field with the value '1'.
- Horizon de Planification :** A numeric input field with the value '1 440' and a dropdown menu set to 'secondes'.

 At the bottom right, there are two buttons: 'OK' and 'Annuler'.

Figure 3 – Fonctionnalité de résolution d'instance.

- La fonctionnalité de résolution d'instance : Avec cette fenêtre, on choisit un fichier d'instance et cocher une ou plusieurs méthodes de résolutions (Heuristique , Méthode Exacte , Méta heuristique) puis un ou plusieurs fichiers de résultat sont fournis contenant les

résultats des différentes méthodes choisit (exemple : **Figure 3** (Chapitre 7) ). Ainsi nous avons le front de Pareto de chaque instances à travers les heuristiques.

**Fonctionnalité**

Méthodes Exactes

☐ Résolution exact indexée Temps

☐ Résolution exact indexée Job

Méthode Approchées

Heuristiques

Trie Cmax basé sur la somme des ressource de chaque Job

☐ Affectation machine par machine

☐ Affectation priorisant la machine la moins chargée

Trie Cmax basé sur la ressource maximale de chaque Job

☐ Affectation machine par machine

☐ Affectation priorisant la machine la moins chargée

Trie sur La durée de chaque Job

☐ Affectation machine par machine

☐ Affectation priorisant la machine la moins chargée

Méta-Heuristiques

☐ Algorithme Génétique NSGA2

Tout Décocher Tout Cocher

**Résolution D'instances**

Fichier Choisi :

Choisir

Resoudre Annuler

**Figure 4** – Fonctionnalité de résolution d'instance de l'application

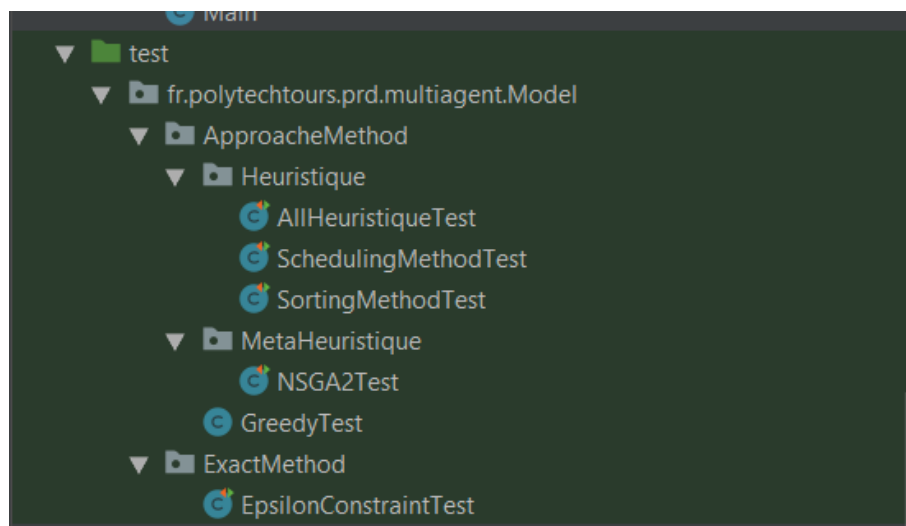
# 10

## Document de Test

L'un des objectifs de notre PRD, est aussi de tester le code fournit au client ici dans mon cas Mr **Boukhalfa ZAhout - Doctorant** et qui a été mon encadrant. Cela me permet aussi de vérifier l'exactitude des différents algorithmes développés.

Les tests permettent de mieux vérifier l'aspect fonctionnel de notre programme et le respect du cahier de spécification. Comme le developpement de l'application a été fait en java, j'ai utilisé la librairie **JUnit** consacrée aux tests (unitaire et fonctionnel) d'une application développer en JAVA.

l'architecture des tests **Figure 1** :



**Figure 1** – Architecture de tests avec JUnit du programme Java

Les tests ont été fait sur un fichier comme on peut le voir à **Figure 2** , contenant une instance de 8 jobs( avec 4 pour chacun des agents) , 3 ressources et 2 machines. le choix de ce fichier a été fait vu que nous connaissons les différents résultats attendus en fonctions des méthodes de tri, d'affectation, d'heuristique, de méta heuristique et la méthode exacte sans oublié la pertinence de cet exemple d'instance choisit avec l'encadrant et la taille du fichier qui est relativement petit pour que les tests ne prennent pas trop de temps.

Pour être efficace j'ai fait des tests unitaires et fonctionnels.

```

8 3 4 2
1000 1000 1000
1000 1000 1000
0 0 4 A 500 300 400
1 1 4 A 125 600 300
2 4 8 A 125 250 200
3 5 9 A 250 500 300
4 0 6 B 125 500 250
5 1 5 B 250 800 600
6 3 6 B 500 700 250
7 3 8 B 500 300 700

```

Figure 2 – Exemple de fichier de test.

- Test Unitaire : à ce niveau je teste les différentes méthodes de trie de jobs et d'affectation des jobs sur les machines.
  - SortingMethodTest** : cette classe contient les tests concernant les méthodes de trie (durée , MakeSpan , CCmax) et vérifier l'ordre des différents jobs.
  - SchedulingMethodTest** : cette classe teste les méthodes d'affectation des jobs sur les machines(affectation machine par machine , affectation sur machine moins chargées) et on vérifie les jobs qui ont été affecté et sur quelle machine.
- Test Fonctionnel : à ce niveau, j'ai utilisé la politique suivante : lire le contenu du fichier , passer l'instance lu à mon heuristique ou méta heuristique , ou méthode exacte, résoudre l'instance puis je vérifie que les résultats correspondent à ceux attendus(comme je connais déjà les différents résultats attendus).
  - AllHeuristiqueTest** : cette classe permet de tester tous mes heuristiques
  - NSGA2Test** : cette classe permet de tester l'algorithme génétique
  - EpsilonConstraintTest** : : cette classe permet de tester la méthode exacte avec l'épsilon contrainte.

Avec cette politique , je couvre la totalité de l'application surtout avec les tests fonctionnels car par exemple en testant NSGA2- l'algorithme génétique ( qui fait appel à plusieurs méthodes : initialisation, sélection, mutation, croisement, calcul du front de pareto), j'arrive à vérifier toutes les autres méthodes incluses.

État des tests **Figure 3** :

▼ ✓ <default package>	6 s 352 ms
▼ ✓ AllHeuristiqueTest	3 s 370 ms
✓ testHeuristique_Tri_Cmax_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachine	710 ms
✓ testHeuristique_Tri_MakeSpan_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachineMoinsCharger	372 ms
✓ testHeuristique_Tri_Cmax_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachineMoinsCharger	325 ms
✓ testHeuristique_Tri_Cmax_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachineMoinsCharger	290 ms
✓ testHeuristique_Tri_MakeSpan_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachine	242 ms
✓ testHeuristique_Tri_Cmax_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachine	179 ms
✓ testHeuristique_Tri_MakeSpan_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachineMoinsCharger	194 ms
✓ testHeuristique_Tri_ShotestTime_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachineMoinsCharger	193 ms
✓ testHeuristique_Tri_ShotestTime_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachineMoinsCharger	217 ms
✓ testHeuristique_Tri_MakeSpan_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachine	172 ms
✓ testHeuristique_Tri_ShotestTime_Affectataion1_resolutionParMachine_Affectation2_resolutionParMachine	269 ms
✓ testHeuristique_Tri_ShotestTime_Affectataion1_resolutionParMachineMoinsCharger_Affectation2_resolutionParMachine	207 ms
▼ ✓ SchedulingMethodTest	1 ms
✓ resolveByMachine	1 ms
✓ resolveMachineLessUsedMachine	0 ms
▼ ✓ SortingMethodTest	1 ms
✓ trierCCmaxMaxRessources	1 ms
✓ trierDuree	0 ms
✓ trierCCmaxSommeRessources	0 ms
▼ ✓ NSGA2Test	2 s 497 ms
✓ testNSGA2	2 s 497 ms
▼ ✓ EpsilonConstraintTest	483 ms
✓ testMethodeExacte	483 ms

Figure 3 – le résultat des test JUnit

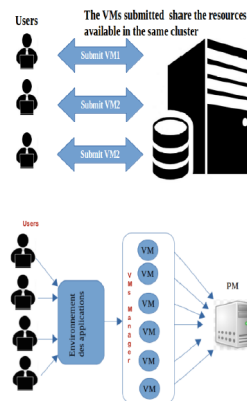
# Ordonnancement et affectation des ressources dans le cloud Computing : Cas multi critères et multi machines

Abdoulaye Kake

Encadrement : Boukhalifa Zahout, Ameer Soukhal et Patrick Martineau

## Objectifs

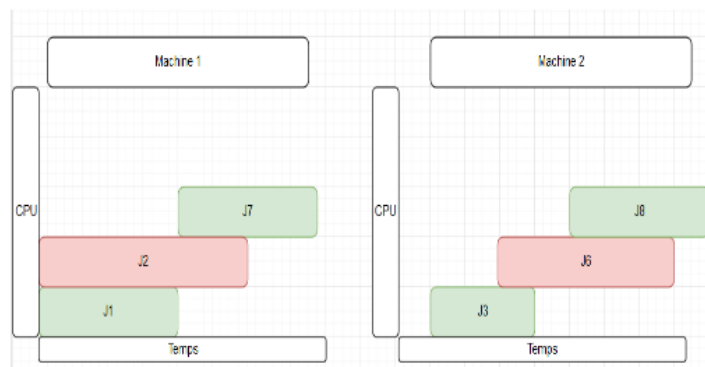
Le projet a pour objectif d'ordonnancement et d'affecter les machines virtuelles. La demande est tellement grande maintenant qu'il est nécessaire de trouver des méthodes d'affectation et d'ordonnancement des machines virtuelles afin de satisfaire le maximum de demandes en respectant les capacités de la machine physique qui doit réaliser.



Affectation des jobs sur le cloud ou sur le cluster

## Mise en œuvre

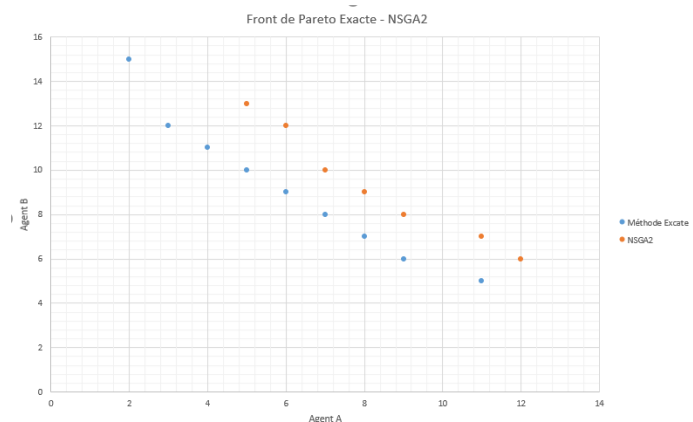
Après l'analyse, nous avons modéliser mathématiquement le problème avec PLNE puis trouver des méthodes approchés : NSGA et les heuristiques de Greedy.



Exemple d'ordonnancement et affectation de 8 jobs dont A(1,3,7,8) et B(2,4,5,6).

## Résultats attendus

le principe est de trouver des solutions optimale sous forme de front de Pareto avec la méthode exacte mais le problème est NP-Difficile dont il faut avoir des solutions proches de celle de la méthode exacte avec NSGA2 et les Greedy Heuristiques.



Exemple de front de Pareto obtenu une instance 50 jobs et 7 machines avec NSGA2 et la méthode Exacte

# Ordonnancement et affectation des ressources dans le cloud Computing : Cas multi critères et multi machines

Abdoulaye Kake

Encadrement : Boukhalfa Zahout, Ameer Soukhal et Patrick Martineau

## Objectifs

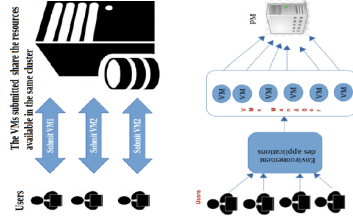
Le projet a pour objectif d'ordonner et d'affecter les machines virtuelles. La demande est tellement grande maintenant qu'il est nécessaire de trouver des méthodes d'affectation et d'ordonnancement des machines virtuelles afin de satisfaire le maximum de demandes en respectant les capacités de la machine physique qui doit réaliser.

## Mise en œuvre

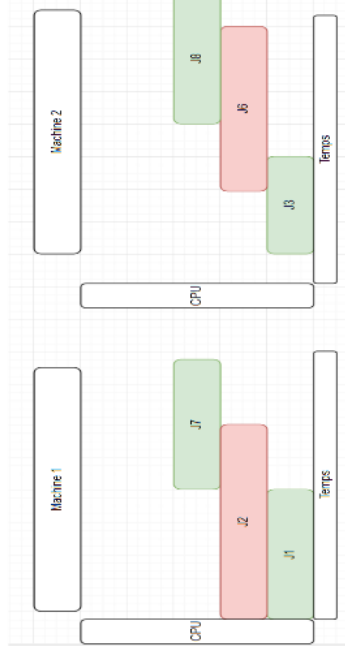
Après l'analyse, nous avons modéliser mathématiquement le problème avec PLNE puis trouver des méthodes approchées : NSGA et les heuristiques de Greedy.

## Résultats attendus

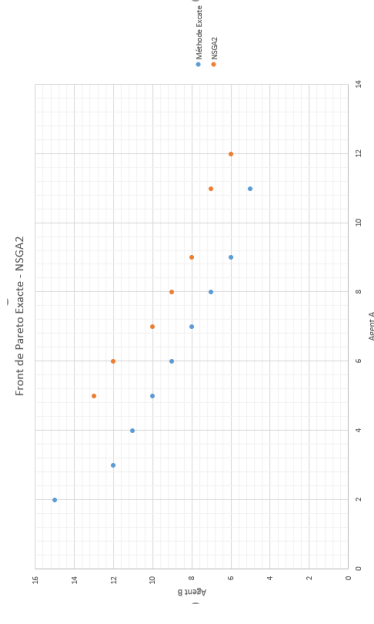
le principe est de trouver des solutions optimale sous forme de front de Pareto avec la méthode exacte mais le problème est NP-Difficile dont il faut avoir des solutions proches de celle de la méthode exacte avec NSGA2 et les Greedy Heuristiques.



Affectation des jobs sur le cloud ou sur le cluster



Exemple d'ordonnancement et affectation de 8 jobs dont  $A(1,3,7,8)$  et  $B(2,4,5,6)$ .



Exemple de front de Pareto obtenu une instance 50 jobs et 7 machines avec NSGA2 et la méthode Exacte

# Ordonnancement et affectation des ressources dans le cloud Computing

## Cas multi critères et multi machines

### Résumé

Ce document vise à présenter les recherches sur l'ordonnancement et l'affectation multicritère et multi machine. Il contient la formulation mathématique puis l'explication des différents heuristiques et méta heuristiques utiliser pour trouver des résultats approchés.

Ce rapport met aussi en avant une application développée qui incorpore toutes les recherches et permet de faciliter l'utilisation des instances de données. Il est composé de la partie état de l'art , analyse et conception et la spécification fonctionnelle de l'application.

### Mots-clés

Ordonnancement, Multicritère, Multi machines

### Abstract

The purpose of this document is to present research on sequencing and multi-criteria and multi-machine assignment. It contains the mathematical formulation then the explanation of the different heuristic and metaheuristic uses to find close results.

This report also highlights a developed application that incorporates all searches and facilitates the use of data instances. It consists of the state of the art part, analysis and design and functional specification of the application.

### Keywords

scheduling, multi-criteria, Multi machines

### Tuteurs académiques

Boukhalfa ZAHOUT

Ameur SOUKHAL

Patrick MARTINEAU

### Étudiant

Abdoulaye KAKE (DI5)