

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

Projet Recherche & Développement
2017-2018

PRD Ordonnancement et affectation des ressources dans le Cloud Computing

Tuteurs académiques
Boukhalfa ZAHOUT
Ameur SOUKHAL
Patrick MARTINEAU

Étudiant
Pierre FERVault (DI5)



Liste des intervenants

Nom	Email	Qualité
Pierre FERVAULT	pierre.fervault@etu.univ-tours.fr	Étudiant DI5
Boukhalfa ZAHOUT	boukhalfa.zahout@etu.univ-tours.fr	Tuteur académique, Département Informatique
Ameur SOUKHAL	ameur.soukhal@univ-tours.fr	Tuteur académique, Département Informatique
Patrick MARTINEAU	patrick.martineau@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Pierre Fervault susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Boukhalfa 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

Pierre Fervault, *PRD Ordonnancement et affectation des ressources dans le Cloud Computing*,
Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais
de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Fervault, Pierre},
  title={PRD Ordonnancement et affectation des ressources dans le Cloud Computing},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```



Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	v
Liste des Algorithmes	vii
I Introduction	1
1 Acteur du projet :.....	1
2 Contexte et étude de l'existant	2
2.1 Contexte	2
2.2 Étude de l'existant	2
3 Objectifs	3
4 Hypothèses	3
5 Bases méthodologiques	3
II Description générale	4
6 Environnement du projet	4
7 Caractéristiques des utilisateurs	4
8 Fonctionnalité du système.....	5

III	État de l'art	7
9	Publication mono-machine	8
10	Publication multi-machine.....	10
10.1	Computers & Operations Research (Enrico Angelelli, Nicola Bianchessi, Carlo Filipp).....	10
10.2	Interval Scheduling : A Survey (Antoon and Al.)	12
IV	Analyse et conception	13
11	Résolution exacte du problème indexée temps	14
12	Résolution exacte du problème basé sur des sous-ensembles maximaux de jobs	15
13	Résolution du problème en utilisant des heuristiques	17
13.1	Méthodes de tri.....	18
13.2	Méthodes d'affectation	18
14	Conception.....	20
V	Mise en œuvre	21
15	Outils utilisés.....	21
16	Implémentation	22
16.1	Partie génération d'instances.....	22
16.2	Partie résolution d'instances	23
16.3	Partie comparaison des méthodes de résolution.....	25
17	Analyse des résultats	26
17.1	Qualité de l'implémentation	26
17.2	Résultats obtenus	28
VI	Bilan et conclusion du semestre 9	32
18	Bilan des tâches effectuées	33
19	Tâches restantes à réaliser	33
20	Conclusion	34
VII	Bilan et conclusion du semestre 10	35
21	Bilan des tâches effectuées	36
22	Bilan sur la qualité	36
23	Bilan sur la gestion du projet	37
24	Conclusion	38

VIII	Annexes	39
25	Description des interfaces externes du logiciel	39
25.1	Interfaces matérielles/logicielles.....	39
25.2	Interfaces homme/machine.....	39
26	Spécifications fonctionnelles.....	40
26.1	Description des fichiers d'entrées :	40
26.2	Description des fichiers de sorties :.....	41
26.3	Description des classes du système :	41
26.4	Description des fonctionnalités du module de résolution d'instances :.....	41
26.5	Description des fonctionnalités du module de comparaison de solution : ...	42
27	Spécifications non fonctionnelles.....	43
27.1	Contraintes de développement et conception.....	43
27.2	Performances.....	43
27.3	Capacités.....	43
27.4	Contrôlabilité.....	43
27.5	Sécurité	43
27.6	Intégrité.....	43
28	Plan de développement	44
28.1	Tache 1 : Lecture de document et compréhension.....	44
28.2	Tache 2 : Résumé première publication	44
28.3	Tache 3 : Recherche et compréhension du MIP1	44
28.4	Tache 4 : Recherche et compréhension du MIP2	44
28.5	Tache 5 : Recherche d'heuristiques	44
28.6	Tache 6 : Rédaction du cahier de spécification	44
28.7	Tache 7 : Rédaction état de l'art.....	45
28.8	Tache 8 : Rédaction rapport de mi-parcours.....	45
28.9	Tache 9 : Préparation de l'oral de mi-parcours	45
28.10	Tache 10 : Développement du programme de résolution.....	45
28.11	Tache 11 : Développement des méthodes de résolution exactes	45
28.12	Tache 12 : Développement des méthodes de résolution heuristique	45
28.13	Tache 13 : Test du programme de résolution	46
28.14	Tache 14 : Développement du module de comparaison de solution	46
28.15	Tache 15 : Test du programme de comparaison de solution.....	46
28.16	Planning réel pour la mise en place des tâches	46
29	Gestion de projet	47
29.1	Méthode de Gestion de Projet utilisée	47
29.2	Identification des livrables et tâches	47
29.3	Analyse de faisabilité.....	48
29.4	Analyse de risque.....	48

29.5	Suivi de projet.....	48
29.6	Outils utilisés pour améliorer la gestion de projet	49
30	Documentation d'installation pour le programme.....	49
31	Documentation d'utilisation du programme :.....	51
32	Plan des différents tests effectués :	54
32.1	Tests Unitaires :.....	54
32.2	Tests Fonctionnelles.....	55
33	Cahier du développeur.....	55
33.1	Langage de programmation utilisé :.....	56
33.2	Librairies utilisées :.....	56
33.3	Convention de nommage :.....	56
33.4	Documentation :	56
33.5	Structure du système :	57
Comptes rendus hebdomadaires		59
Bibliographie		67

Table des figures

Liste des Algorithmes

1	Diagramme de cas d'utilisation du programme de résolution d'instances.....	5
2	Diagramme de cas d'utilisation du programme de comparaison de solutions.....	5
3	Exemple montrant les pourcentages de solution optimale obtenus pour 3 heuristiques différentes en fonction du nombre de jobs par instances (cas mono-machine)	6
4	Exemple montrant le gap moyen obtenu pour 3 heuristiques différentes en fonction du nombre de jobs par instances (cas mono-machine)	6
5	Formulation du problème de manière temporelle	8
6	Algorithme pour obtenir les sous-ensembles maximaux	9
7	Formulation du problème basé sur des sous-ensembles maximaux de job	9
8	Formulation du problème à plusieurs machines avec ressource unique basé sur des sous-ensembles maximaux de job.....	11
9	Formulation du problème multi-machines avec ressources multiples de manière temporelle.....	14
10	Exemple d'instance de 8 jobs	15
11	Liste des événements classés afin d'obtenir les sous-ensembles maximaux	16
12	Application de l'algorithme permettant d'obtenir les sous-ensembles maximaux ..	16
13	Formulation du problème multi-machines avec ressources multiples indexées job	17
14	Diagramme de classe de l'application.....	20
15	Logos des outils utilisés.....	21
16	Fenêtre permettant la génération d'instances et arborescence pour la création d'instances	22
17	Fenêtre permettant la résolution d'instances suivant différentes méthodes	23
18	Schéma de l'arborescence des résultats pour la résolution d'un dossier d'instance à 8 jobs, 3 ressources et 3 machines avec tous les types de résolutions	24
19	Fichier de résultat pour une instance résolue, chaque instance étant ajoutée au fur et à mesure au fichier lors de la résolution	24

20	Fenêtre permettant la comparaison des heuristiques (gap, pourcentage de solution optimale et pourcentage du temps de résolution exacte)	25
21	Graphiques de comparaison de méthodes de résolution, pour un nombre de jobs allant de 10 à 100 avec 3 types de ressources et 3 machines	26
22	Structure du programme	27
23	Différence du temps d'exécution entre différentes instances	27
24	Classes sur lesquelles il est possible d'ajouter des méthodes de résolution	28
25	Classe où il est possible d'ajouter de nouvelles méthodes pour comparer les méthodes de résolution	28
26	Graphique représentant le gap des différentes méthodes de résolutions en fonction du nombre de jobs de chaque instance	29
27	Tableau récapitulatif des différents gaps selon le nombre de jobs	29
28	Graphique représentant le pourcentage de solutions exactes par méthode de résolution en fonction du nombre de jobs de chaque instance	30
29	Graphique représentant le pourcentage de temps de résolution par rapport à la résolution exacte pour chacune des heuristiques en fonction du nombre de jobs de chaque instance	30
30	Tableau récapitulatif des différents pourcentages de temps de résolution par rapport à la résolution exacte suivant le nombre de jobs	31
31	Diagramme de Gantt actuel	34
32	Diagramme de Gantt du S10 prévu au S9	37
33	Diagramme de Gantt des tâches réalisées au S10	38
34	Structure à respecter pour les fichiers d'instances	40
35	Exemple de fichier d'instance à 8 jobs, 3 ressources, 3 machines	40
36	Exemple de fichier de résultat à 8 jobs, 3 ressources, 3 machines	41



Liste des Algorithmes

- 1 Algorithme Heuristique 1 (On affecte le maximum de jobs sur une première machine et on renouvelle l'opération sur les autres machines) 19
- 2 Algorithme Heuristique 2 (On affecte chaque job sur la machine étant la moins chargée, en cas d'égalité, l'ordre lexicographique est utilisé) 19

Première partie

Introduction

Le Projet de Recherche et Développement (PRD), réalisé en 5ème année, s'inscrit dans la formation dispensée à l'École Polytechnique de Tours et constitue une réelle expérience en termes de conduite de projet. Le PRD se déroule du 20 septembre 2017 jusqu'à début avril 2018 (la date de fin n'a pas encore été décidée) à raison de 2 jours par semaine à temps plein. Il donne lieu à la rédaction d'un rapport avec le cahier de spécification et de deux présentations du travail effectué lors de deux soutenances.

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. Le projet est réalisé par Pierre FERVAULT, élève ingénieur en 5ème année de l'école Polytech Tours qui représente la MOE.

Le projet s'inscrit dans un cadre théorique qui fait l'objet d'une étude de document scientifique en amont, pour aboutir au développement d'un algorithme.

Ce document constitue les spécifications concernant le projet de recherche et développement "Ordonnancement et affectation des ressources dans le Cloud Computing ».

Il permet de développer les différentes caractéristiques du projet, de définir 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.

1 Acteur du projet :

Plusieurs acteurs ont participé à l'élaboration et la réalisation de ce projet :

- La maîtrise d'œuvre (MOE) : Pierre FERVAULT, étudiant en 5ème année de l'école d'ingénieur Polytech Tours au sein du département Informatique.
- La maîtrise d'ouvrage (MOA) : L'équipe de recherche ROOT du laboratoire d'informatique de Tours représentée par Boukhalfa ZAHOUT, Ameer SOUHKAL et Patrick MARTINEAU. Cette équipe de recherche se spécialise dans l'étude et la résolution des problèmes d'ordonnancement et de planification. Ce sont des problèmes d'optimisation pour lesquels des techniques de Recherche Opérationnelle et d'Aide à la Décision sont mises en œuvre. (www.li.univ-tours.fr)

2 Contexte et étude de l'existant

2.1 Contexte

Les systèmes d'informations jouent un rôle déterminant dans notre quotidien et dans le monde professionnel. L'utilisation des ressources de nos machines est de plus en plus prononcée au fil du temps. De plus, il est nécessaire de satisfaire le maximum des demandes des utilisateurs des systèmes d'informations.

Cependant, nos machines actuelles ne sont généralement pas assez puissantes pour satisfaire l'ensemble des demandes si aucun travail d'ordonnancement n'est réalisé. En effet, en ordonnant intelligemment les demandes des clients (que nous appellerons par la suite jobs), nous pouvons en réaliser un nombre plus important en exploitant au maximum les capacités de nos machines.

Dans ce PRD, nous supposons qu'il existe des méthodes d'ordonnancement efficaces pour un cas mono-machine. Toutefois, nous allons par la suite rechercher des solutions d'ordonnancement utilisant plusieurs machines. L'application du parallélisme pourrait dans ce cas augmenter significativement le nombre de jobs que nous pourrions effectuer et ainsi, nous aurions la possibilité de contenter un nombre plus important de clients. Cette amélioration des performances pourrait bien évidemment être un enjeu économique particulièrement important pour les sociétés proposant des services de Cloud Computing.

2.2 Étude de l'existant

Ce projet s'appuie sur un cas d'ordonnancement et d'affectation de ressources sur une machine unique [2].

Il s'agit d'un problème d'ordonnancement où nous devons affecter des jobs ne pouvant pas être déplacés et qui nécessitent certaines quantités de ressources pour fonctionner. Ce problème possède les contraintes suivantes :

Les jobs ont un instant de début (s_i), un instant de fin (f_i), une durée d'exécution ($p_i = f_i - s_i$) et ne peuvent pas être déplacé. Si le job ne peut pas être ordonnancé, il est rejeté. Il y a n jobs notés J_i .

Il y a 3 types de ressources R_j avec $j = 1, 2, 3$ (CPU, MEMOIRE, STOCKAGE)

- Q_j est l'unité de ressource renouvelable de type R_j
- q_{ij} est l'unité de ressources nécessaire pour le job i et la ressource j
- $s_i < f_i$ et $q_{ij} \leq Q_j$ pour tout job $i = 1, \dots, n$ et ressources $j = 1, 2, 3$

La publication montre qu'il est possible de trouver une solution optimale pour le problème du cas mono-machine. Cependant, la résolution exacte est NP-Difficile, ce qui signifie qu'elle demande un temps de calcul qui croît de manière exponentielle suivant la taille des données d'entrées. Afin de résoudre plus rapidement le calcul, plusieurs algorithmes de classification de jobs ont été proposés.

Ces algorithmes ont permis d'obtenir des résultats se rapprochant des solutions optimales avec un temps de calcul bien plus faible en comparaison de la résolution exacte.

L'étude de cette publication sera plus développée dans la section "État de l'art" du rapport.

La MOA a implémentait la résolution exacte du problème mono-machine via deux programmes en lignes de commandes Linux. Ces programmes sont codés en C++ et utilise le solveur cplex_216 afin de résoudre le problème.

3 Objectifs

Ce projet a pour objectif de proposer un algorithme permettant l'ordonnancement de jobs fixés dans le temps, avec k type de ressources sur plusieurs machines. Il s'agit donc de proposer une résolution similaire au cas mono-machine cité précédemment, mais avec comme spécificité l'utilisation de plusieurs machines à la fois.

Il est donc ici important de répartir les différents jobs sur différentes machines tout en minimisant le nombre de jobs rejetés (ou en maximisant le nombre de jobs exécutés). L'utilisation de machines en parallèle devrait permettre un gain de performances conséquent et pourrait augmenter de manière significative le nombre de jobs ordonnancés.

Il serait donc nécessaire de reprendre les modèles mathématiques exposés dans la publication traitant du cas mono-machine et de les adapter à notre problème multi-machines.

Tout comme le cas mono-machines, ce problème est NP-Difficile, ce qui signifie que le temps d'exécution croît de manière exponentielle suivant la taille des données d'entrées. Il sera donc nécessaire de rechercher des heuristiques permettant d'obtenir une solution proche du cas optimal avec un temps d'exécution bien plus faible.

La liste des objectifs à réaliser est la suivante :

- Obtenir un modèle mathématique indexé temps (dénommé MIP1) permettant de trouver des solutions optimales à notre problème d'ordonnancement multi-machine.
- Obtenir un modèle mathématique indexé jobs (dénommé MIP2) permettant de trouver des solutions optimales à notre problème d'ordonnancement multi-machine.
- Trouver différentes heuristiques permettant d'obtenir des résultats proches de la solution optimale 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.
- Implémenter un module de comparaison de solution afin de comparer les performances et les résultats obtenus en utilisant les heuristiques par rapport aux méthodes exactes.

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 d'heuristique par exemple est essentielle à notre projet et il ne serait pas concevable de développer une solution de résolution avant d'avoir trouvé des heuristiques permettant de résoudre efficacement notre problème. Néanmoins, il est important d'arrêter la phase de recherche suffisamment tôt afin d'avoir le temps de développer des outils de résolutions et de comparaisons de solutions.

5 Bases méthodologiques

Ce projet sera effectué avec un plan de gestion de projet agile. Il sera important de voir le client régulièrement pour faire valider les choix de conception et de développement. Les outils de gestion de projet utilisés seront la plateforme de gestion Trello, le diagramme de Gantt et l'outil de versionning Git pour le développement.

Deuxième partie

Description générale

6 Environnement du projet

Ce projet fait suite au cas d'ordonnancement mono-machine proposé par Boukhalifa ZAHOUT, Ameer SOUHKAL et Patrick MARTINEAU. De ce fait, le développement futur du projet fera suite à ce qui a déjà été produit et utilisera les mêmes technologies que pour le cas mono-machines.

Le développement se fera sur le système d'exploitation Linux, distribution Ubuntu. Pour ce qui est du langage de programmation utilisé, le C++ 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 cplex_216. Il est plus facile d'utiliser ce solveur sous Linux car il ne nécessite pas beaucoup de configuration et est facilement utilisable avec le langage c++.

Le programme s'exécutera directement en ligne de commande Linux, les utilisateurs devront alors connaître l'environnement Linux et savoir utiliser les commandes bash de base.

7 Caractéristiques des utilisateurs

Les utilisateurs seront la MOA. Ils auront une bonne connaissance de l'application et connaîtront son fonctionnement. Étant des experts en informatique, ils connaissent l'environnement Linux et l'utilisation de commande dans un terminal.

8 Fonctionnalité du système

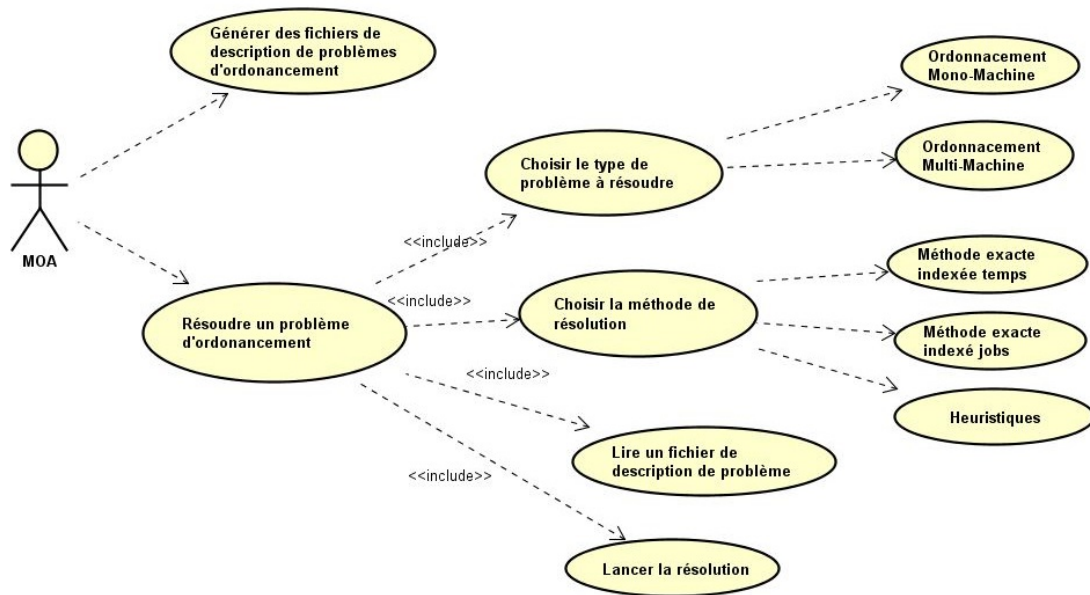


Figure 1 – Diagramme de cas d'utilisation du programme de résolution d'instances

Le système de résolution d'instance permettra de générer des fichiers d'instances ainsi que de trouver des solutions pour des instances spécifiées. Pour résoudre des problèmes d'ordonnancement, la MOA devra choisir le type de problème à résoudre (ordonnancement mono-machine ou multi-machine). Ensuite, elle choisira la méthode de résolution qu'elle souhaite utiliser (méthode exacte indexée temps, méthode exacte indexée jobs ou heuristiques). Puis, elle pourra spécifier un fichier de description de problème et lancer la résolution pour ce fichier.

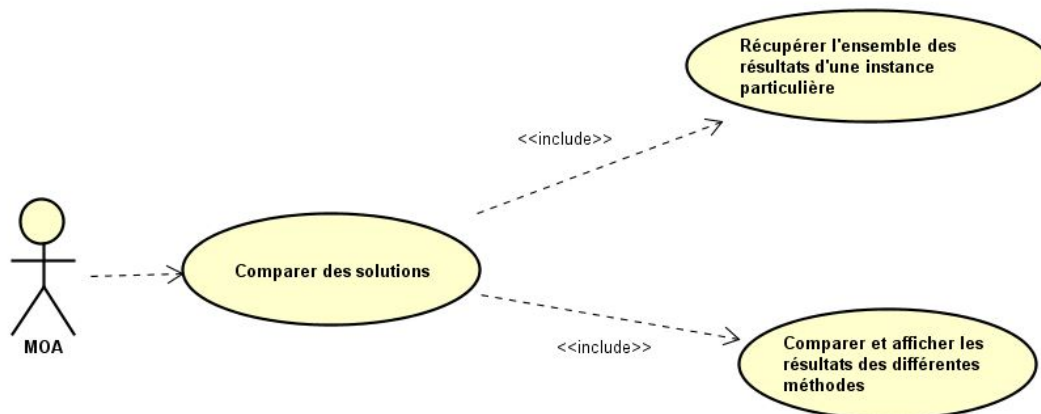


Figure 2 – Diagramme de cas d'utilisation du programme de comparaison de solutions

Un module de comparaison de solution sera mis en place. Cela permettra de prendre une instance particulière et de comparer tous les résultats des différentes méthodes de résolution. Ce module est particulièrement important, car il nous permettra de voir l'efficacité des heuristiques que nous avons trouvées jusqu'à maintenant.

Le module de comparaison de solution devra permettre d'obtenir des graphiques permettant de montrer le pourcentage de solution optimale obtenu à partir d'heuristiques particulières en fonction du nombre de jobs par instance.

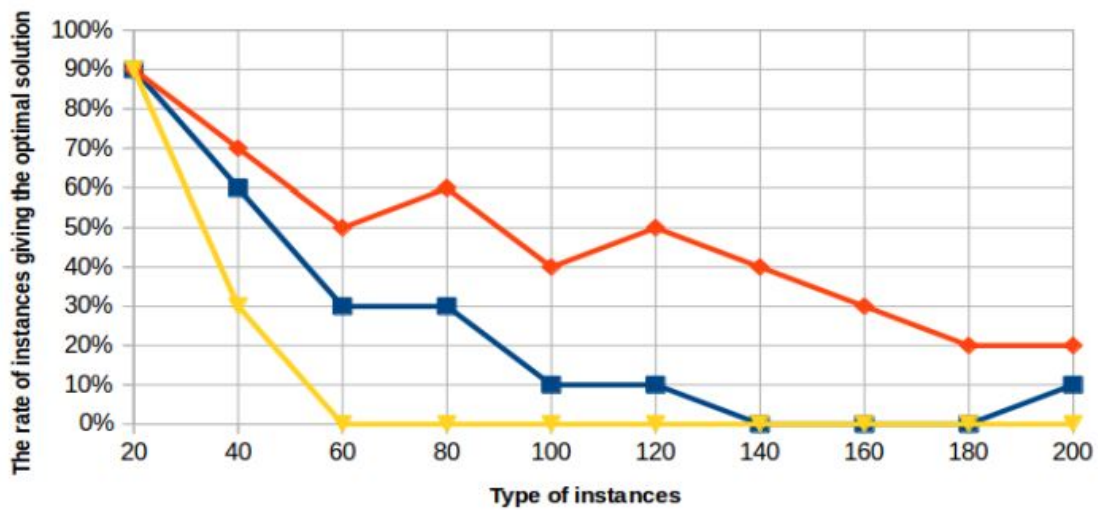


Figure 3 – Exemple montrant les pourcentages de solution optimale obtenus pour 3 heuristiques différentes en fonction du nombre de jobs par instances (cas mono-machine)

Le module devra permettre également d'obtenir le gap de chaque heuristique en fonction du nombre de jobs par instance. Le gap correspond au pourcentage de détérioration d'une solution par rapport à la solution optimale.

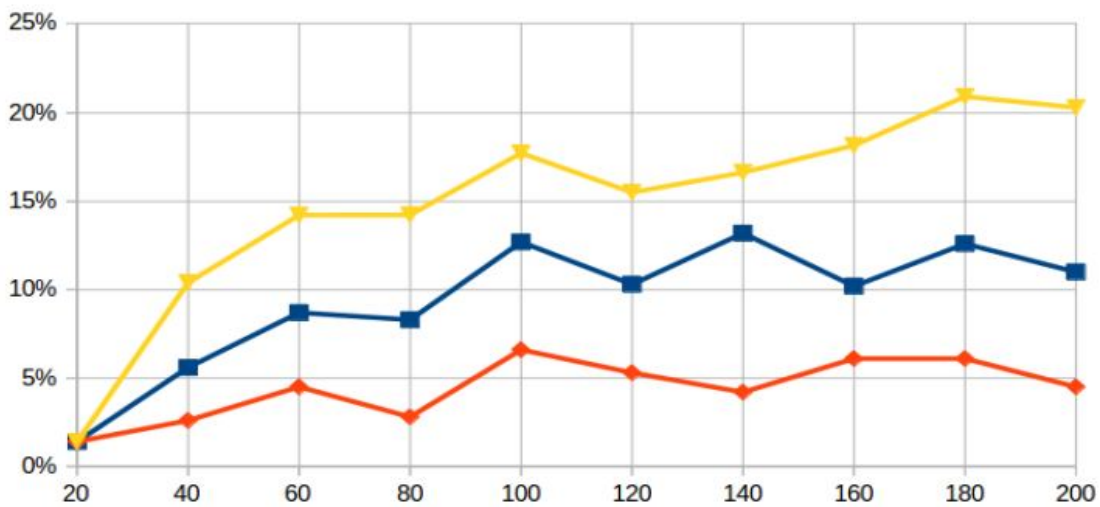


Figure 4 – Exemple montrant le gap moyen obtenu pour 3 heuristiques différentes en fonction du nombre de jobs par instances (cas mono-machine)

Une convention de nommage sera appliquée ultérieurement, afin que nous puissions classer efficacement les résultats des différentes instances. Le classement des instances sera effectué ainsi : les premiers sous-dossiers permettront d'identifier le nombre de machines pouvant ordonnancer des jobs, chacun de ces sous-dossiers contiendra des sous-dossiers permettant d'identifier le nombre de ressources spécifiées lors de l'ordonnancement. Puis dans chacun de ces sous -dossiers seront présent d'autres sous-dossiers avec le nombre de jobs à ordonnancer qui contiendront les résultats des instances correspondantes pour chaque type de résolution.

Troisième partie

État de l'art

Notre problème est un cas de fixed interval scheduling (ordonnancement à intervalle fixé), ce qui signifie d'un cas d'ordonnancement où les jobs ont des dates de début et de fin fixées et qu'elles ne peuvent par conséquent pas être modifiées. Ce problème a également comme caractéristique le fait que chaque job possède k types de ressources avec pour chacune d'entre elles une quantité fixée. Le cas d'ordonnancement de ce PRD a également comme spécificité le fait que nous devons ordonnancer les jobs sur plusieurs machines avec k ressources limités, un job ne pouvant être exécuté que sur une seule machine. L'objectif principal étant d'ordonnancer le maximum de jobs à partir d'une liste de jobs à exécuter.

Il existe diverses publications à propos de l'ordonnancement de jobs sur machines parallèles, mais généralement il s'agit de cas spécifique qui ne possède pas les mêmes contraintes que le problème proposé lors du PRD. Nous mentionnerons néanmoins les documents les plus pertinents qui ont permis de faire avancer la phase de recherche et nous essaierons d'en extraire le maximum d'informations.

Premièrement, j'analyserai la publication initiale du cas mono-machine ainsi que ces modèles de résolutions exactes et heuristiques qui ont servi de point de départ à ce PRD.

Je m'attarderai ensuite sur les divers documents annexes traitant de cas multi-machine qui m'ont permis de développer mes modèles exacts et heuristiques.

9 Publication mono-machine

J'ai pu en premier lieu observer le cas mono-machine, dont la publication a été rédigée cette année [2]. Les auteurs sont : Boukhalfa ZAHOUT, Ameer SOUKHAL et Patrick MARTINEAU. Grâce à celle-ci, j'ai pu comprendre que le problème comprenait la gestion de n ressources différentes, ce qui différencie le problème par rapport aux autres publications.

Il s'agit d'un problème d'ordonnancement où nous devons affecter des jobs ne pouvant pas être déplacés et qui nécessitent certaines quantités de ressources pour fonctionner. Ce problème possède les contraintes suivantes :

- Les jobs ont un instant de début (s_i), un instant de fin (f_i), une durée d'exécution ($p_i = f_i - s_i$) et ne peuvent pas être déplacé. Si le job ne peut pas être ordonnancé, il est rejeté. Il y a n jobs notés J_i .
- Il y a dans cet exemple 3 types de ressources R_j avec $j = 1, 2, 3$ (CPU, MEMOIRE, STOCKAGE), cela peut être généralisé avec k ressources
- Q_j est l'unité de ressource renouvelable de type R_j
- q_{ij} est l'unité de ressources nécessaires pour le job i et la ressource j
- $s_i < f_i$ et q_{ij} inférieur ou égal à Q_j pour tout job $i = 1, \dots, n$ et ressources $j = 1, 2, 3$

Deux types d'approches peuvent être utilisés pour obtenir une solution optimale au problème.

En premier lieu, nous pouvons décomposer le problème de manière temporelle.

Dans cette expression deux types de variables sont définies :

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

La formulation est la suivante :

$$\begin{aligned} \text{Minimize: } & \sum_{i=0}^{\mathcal{N}} x_i \\ \text{subject to: } & \sum_{t=s_i}^{f_i} y_{it} = (f_i - s_i) * (1 - x_i); \forall i \in \mathcal{N} \end{aligned} \quad (1)$$

$$\begin{aligned} & \sum_{i=0}^{\mathcal{N}} y_{it} * q_{ij} \leq Q_j; \forall j \in \mathcal{R}; \forall t \in \mathcal{T} \\ & x_i \in \{0, 1\}, \quad y_{it} \in \{0, 1\}, \forall i \in \mathcal{N}, \forall t \in \mathcal{T}. \end{aligned} \quad (2)$$

Figure 5 – Formulation du problème de manière temporelle

Il s'agit donc de minimiser le nombre de jobs rejetés. La première contrainte assure que si le job J_i n'est pas rejeté, alors il est ordonnancé durant sa période d'exécution et y_{it} vaut 1 pour $t = [s_i, f_i]$. La seconde contrainte vérifie qu'il n'y a pas plus de ressources consommées sur la machine à l'instant t que la valeur de ressource Q_j . Cette formulation est importante pour obtenir le modèle de résolution exacte pour notre cas d'affectation de jobs sur machines multiples.

La deuxième méthode de résolution est un modèle qui ne se base plus sur le temps, mais sur des sous-ensembles de jobs obtenus via un tableau d'événement (début ou fin de jobs). Ici, l'idée est de créer des sous-ensembles de jobs possédant un maximum de jobs qui se chevauchent. Chaque sous-ensemble ne pouvant pas être contenu dans un autre.

```

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

```

Figure 6 – Algorithme pour obtenir les sous-ensembles maximaux

Nous récupérons ainsi chaque sous-ensemble J et nous pouvons résoudre ce problème grâce à une résolution linéaire en nombre entier.

Dans cette expression :

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

La formulation est la suivante :

$$\begin{aligned}
 &\text{Maximize: } \sum_{i=0}^{\mathcal{N}} x_i \\
 &\text{subject to: } \sum_{l \in J_h} x_l * q_{lj} \leq Q_j ; \forall j \in \mathcal{R} ; h = 1 \dots k \quad (1) \\
 &x_i \in \{0, 1\}, \forall i \in \mathcal{N}.
 \end{aligned}$$

Figure 7 – Formulation du problème basé sur des sous-ensembles maximaux de job

Contrairement au modèle temporel, il s'agit ici de maximiser le nombre de jobs non rejetés. La première contrainte vérifie que pour chaque sous-ensemble de jobs, il n'y a pas plus de ressources consommées sur la machine j à l'instant t que la valeur de ressource Q_j .

Les deux méthodes nous permettent de résoudre ce problème de manière optimale, cependant ce problème est NP-Difficile, ce qui signifie qu'il demande un temps de calcul qui croît de manière exponentielle suivant la taille des données d'entrées. Afin de résoudre plus rapidement le calcul, plusieurs algorithmes de classification de jobs ont été proposés.

Algorithmes utilisés :

- Shortest Processing Time (SPT) : les jobs sont triés par ordre croissant de temps d'exécution $p_i = f_i - s_i$. En cas d'égalité, le job se terminant en premier est exécuté en premier, sinon, on se réfère à l'ordre lexicographique (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 nécessaire par unités de temps entre $[s_i \text{ et } f_i]$. 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.

Résultats obtenus grâce aux différents algorithmes : De manière générale, l'algorithme CC_{max} est le plus performant pour les tests avec un nombre n de job allant de 20 à 200. Pour 40 jobs par exemple il réussit à ordonnancer 40 % des jobs contre 0 % pour SPT et ARC. ARC est clairement inférieur par rapport aux autres algorithmes à partir d'un nombre de jobs $n = 40$.

10 Publication multi-machine

10.1 Computers & Operations Research (Enrico Angelelli, Nicola Bianchessi, Carlo Filipp)

J'ai ensuite étudié une référence qui a servi à l'élaboration du cas mono-machine. Elle a été écrite par Enrico Angelelli, Nicola Bianchessi, Carlo Filipp en 2014 [3].

Nous avons affaire ici à un cas d'ordonnancement multi-machine similaire à notre problème, mais avec la gestion d'une ressource unique pour chaque machine.

Ici, on considère 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 unité de ressource renouvelable, nécessaire pour l'exécution des jobs. Un job a un instant de départ s_j et un instant de fin f_j .

Le fait d'exécuter un job j requière r_j unité de ressources sur la machine qui l'exécutera et produit la valeur v_j (priorité du job).

Chaque machine peut exécuter plusieurs jobs tant qu'il ne consomme pas plus que la valeur de ressource R .

Une méthode de résolution exacte a été trouvée. Il s'agit ici de créer des sous-ensembles maximaux de jobs (sous-ensemble ou un maximum de jobs se chevauchent) et maximiser le nombre de jobs à exécuter sur m machines. L'algorithme de tri utilisé pour obtenir les sous-ensembles maximaux est le même que pour le cas mono-machine.

Une solution optimale a été trouvée :

$$x_{ij} = \begin{cases} 1 & \text{if machine } i \text{ processes job } j; \\ 0 & \text{otherwise.} \end{cases}$$

We may formulate problem OISRC as follows:

$$\max \sum_{i=1}^m \sum_{j=1}^n v_j x_{ij}$$

$$\text{subject to } \sum_{i=1}^m x_{ij} \leq 1 \quad (j = 1, \dots, n)$$

$$\sum_{j \in J_h} r_j x_{ij} \leq R \quad (i = 1, \dots, m; h = 1, \dots, k)$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \dots, m; j = 1, \dots, n)$$

Figure 8 – Formulation du problème à plusieurs machines avec ressource unique basé sur des sous-ensembles maximaux de job

Tout comme le cas mono-machine, nous cherchons ici à maximiser le nombre de jobs exécuté. Cependant, ici le cas est différent, nous n'avons qu'un seul type de ressources et plusieurs machines. Il sera donc nécessaire de modifier les contraintes pour quelle conviennent à notre cas d'ordonnancement multi-machine avec ressources multiples.

Ainsi la première contrainte vérifie qu'un job ne peut être exécuté que sur une seule machine à la fois. La seconde contrainte vérifie que pour chaque sous-ensemble de jobs, il n'y a pas plus de ressources consommées sur la machine j à l'instant t que la valeur de ressource Q_j .

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'ordonner un maximum de jobs.

Quatre algorithmes ont été définis :

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

BestGreedy-JS est défini comme étant le meilleur résultat pour chaque instance pour lesquelles nous aurions appliqué les 4 algorithmes précédents.

Tous ces algorithmes de trie possède 3 attributs et on un coût en temps d'exécution $O(n \log n)$. De plus, l'affectation ou le rejet de chaque job sur une machine disponible possède un temps d'exécution $O(nmR)$. De ce fait la complexité d'exécution des algorithmes de trie est dominé par l'affectation de jobs sur les différentes machines qui représente un coût total $O(n^2mR)$.

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 : BestGreedy MS. Au final, le meilleur algorithme est défini comme étant $\max(\text{BestGreedy JS}, \text{BestGreedy MS})$.

Créer un sous-ensemble maximal coûte $O(n^2)$. Le tri est toujours de coût $O(n \log n)$. La complexité de l'algorithme maximal-set based est toujours dominé par l'affectation de jobs de complexité $O(n^2 m R)$.

10.2 Interval Scheduling : A Survey (Antoon and Al.)

Ce document a été écrit par Antoon and Al. et publié en 2007 [1]. C'est un survey sur l'ordonnancement par intervalle qui correspond à un ordonnancement où les jobs à affecter ont une date de début et de fin définie au préalable et que ces dates ne peuvent pas être modifiées. En tant que survey, cette publication fait un résumé de la littérature existante pour les problèmes d'ordonnancement par intervalle. De ce fait, nous ne nous intéresserons qu'aux parties en lien avec notre projet.

Nous apprenons tous d'abord qu'il existe des cas d'ordonnancement où les jobs doivent forcément être exécutés tout en minimisant le nombre de machines nécessaires à l'exécution des jobs (Partie 2.2.1 de la publication). Ce type de problèmes est proche du notre, mais nous cherchons dans notre cas à minimiser le nombre de jobs rejetés tout en ayant un nombre de machines fixé.

La partie décrivant notre problème (2.2.2) nous informe que le problème est nommé « Interval Scheduling With Given Machines » (Ordonnancement par intervalle sur machines données). Nous apprenons qu'il s'agit de maximiser le nombre de jobs (avec un poids) ordonnancés sur un cas multi-machine. La différence avec notre problème vient du fait qu'ici il y a une notion de poids, ce qui n'est pas le cas de notre problème où chaque job n'a pas une priorité d'affectation supérieure à un autre.

Nous apprenons que ce type de problèmes a été résolu via une formulation de flot minimum. Dans le cas où les jobs ont des poids, des algorithmes gloutons ont été trouvés. Si chaque job ne peut être exécuté que par un sous-ensemble de machines défini arbitrairement, le problème devient NP-Difficile et des heuristiques ont été proposées. De plus, un algorithme de coût $O(n^{m-1})$ qui maximise le nombre de jobs ordonnancés à également était défini.

Nous apprenons que ces types de problèmes peuvent être retrouvés lors de l'affectation de membres d'équipages d'un train ou d'un avion par exemple. Les intervalles de job sont ici les tâches que doivent effectuer les membres d'équipage et les membres sont représentés comme des machines devant effectuer ces tâches. Ces types de problèmes sont également présents dans le domaine des télécommunications où chaque utilisateur communique avec d'autres personnes sur un réseau. Une communication entre utilisateurs demande une certaine quantité de bande passante durant un temps donné. Il faut ici gérer les communications qui représentent les jobs sur les différents liens du réseau qui représentent les machines.

Quatrième partie

Analyse et conception

11 Résolution exacte du problème indexée temps

Durant ce PRD, j'ai pu analyser plusieurs documents que j'ai décrits dans mon état de l'art et j'ai ainsi pu obtenir des résultats.

Dans un premier temps, je me suis concentré sur la résolution exacte du problème d'ordonnement multi-machines avec ressources multiples. J'ai en premier lieu étudié le modèle temporel de la première publication (publication mono-machine). J'ai pu déduire qu'il fallait ajouter ou modifier des contraintes pour obtenir une expression de résolution adaptée à notre problème.

J'ai également pu remarquer que la formulation du problème d'ordonnement de la publication d'Angellelli et al. était assez proche de notre problème. La principale différence étant que dans cette publication les jobs ont une priorité de départ et une valeur de ressource unique. De plus la formulation n'est pas basée sur un modèle temporel mais basée sur des sous-ensembles maximaux de jobs.

Après avoir réfléchi à comment résoudre le problème, je suis arrivé à la formulation suivante :

Maximize :

$$\sum_{i=0}^N \sum_{m=0}^M x_{im}$$

Subject to :

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

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

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

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

Figure 9 – Formulation du problème multi-machines avec ressources multiples de manière temporelle

Ici, il s'agit de maximiser le nombre de jobs ordonnancés sur l'ensemble des machines. Si le job i est ordonnancé sur la machine m alors $x_{im} = 1$, sinon $x_{im} = 0$.

Ensuite, nous assurons que la somme des valeurs de 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) pour t allant de s_i à f_i est égale à $(f_i - s_i) * x_{im}$ pour tous les jobs i et toutes les machines m .

Puis, nous vérifions que pour chaque job i , la somme des $Y_{imt} * q_{ijm}$ est inférieure ou égale à Q_{jm} pour toutes les ressources j , pour toutes les machines m et pour tous t .

De, plus, il faut également vérifier que la somme sur toutes les machines de x_{im} soit inférieure ou égale à 1 pour tout job i . Cela permet de nous assurer que chaque job n'est exécuté que par une seule et unique machine lorsque celui-ci est ordonnancé.

Après avoir exposé ma formulation du problème de manière temporelle, et que la MOA l'ait validé, je me suis concentré sur la formulation du problème basé sur des sous-ensembles maximaux de jobs.

Tout comme le cas mono-machine, nous recherchons des sous-ensembles de jobs maximaux (il s'agit de sous-ensembles contenant le maximum de jobs se chevauchant en même temps). Le principe ici est de créer un tableau d'événement e_h avec h compris entre 1 et $2n$ (n étant le nombre de jobs). Les événements sont les instants de départ des jobs s_i et les instants de fin des jobs f_i . Il s'agit ici de classer ces événements par ordre d'arrivée et en cas de valeurs égales, l'événement concernant le job ayant l'instant de fin le plus faible est privilégié.

L'algorithme permettant d'obtenir les sous-ensembles de jobs maximaux est le même que celui présenté dans l'état de l'art [Figure 6].

Nous voyons donc ici que pour chaque événement contenu dans la liste que nous avons trouvée précédemment, différentes actions sont effectuées. Au départ, les sous-ensembles J sont vides, le compteur de sous ensembles maximaux est vide et le compteur d'événement de départ est vide (il s'incrémente dès qu'un événement de type départ est parcouru).

Si l'événement parcouru est un événement de départ, le compteur d'événement de départ est incrémenté et on ajoute le job correspondant à l'événement dans le sous-ensemble J_k .

Si l'événement parcouru est un événement de fin, si le compteur d'événement de départ est égal au nombre total de jobs, on crée un nouveau sous-ensemble J_k correspondant au sous-ensemble courant J_k auquel nous enlevons le job correspondant à l'événement parcouru et nous récupérons tous les sous-ensembles trouvés. Dans le cas où le compteur d'événement de départ est inférieur au nombre total de jobs, on crée un nouveau sous-ensemble J_{k+1} correspondant au sous-ensemble courant J_k auquel nous enlevons le job correspondant à l'événement parcouru et nous incrémentons k .

Voici un exemple montrant comment sont obtenus les sous-ensembles maximaux :

Prenons une instance de 8 jobs :

Jobs	s_j	f_j	CPU	MEMORY	STORAGE
1	0	6	250	500	250
2	1	4	125	600	300
3	3	8	500	300	700
4	3	6	500	700	250
5	4	8	125	250	200
6	5	9	250	500	300
7	0	4	500	300	400
8	1	5	250	800	600

Figure 10 – Exemple d'instance de 8 jobs

Nous obtenons une liste d'événement qui correspondent aux instants de début et de fin de chaque job. Les événements sont classés par ordre croissant suivant le moment où ils appa-

raissent et en cas d'égalité, l'événement correspondant au job se terminant en premier est prioritaire.

Nous obtenons la liste suivante :

h	e_h	h	e_h
1	Job 7 ; $s_j = 0$	9	Job 5 ; $s_j = 4$
2	Job 1 ; $s_j = 0$	10	Job 8 ; $f_j = 5$
3	Job 2 ; $s_j = 1$	11	Job 6 ; $s_j = 5$
4	Job 8 ; $s_j = 1$	12	Job 1 ; $f_j = 6$
5	Job 4 ; $s_j = 3$	13	Job 4 ; $f_j = 6$
6	Job 3 ; $s_j = 3$	14	Job 3 ; $f_j = 8$
7	Job 2 ; $f_j = 4$	15	Job 5 ; $f_j = 8$
8	Job 7 ; $f_j = 4$	16	Job 6 ; $f_j = 9$

Figure 11 – Liste des événements classés afin d'obtenir les sous-ensembles maximaux

En appliquant l'algorithme permettant d'obtenir les sous-ensembles maximaux à partir de la liste d'événements, nous obtenons cette liste d'étape (chaque numéro en début de ligne correspond au numéro de l'événement (entre 1 et 16)) :

0 : $J = \emptyset$; $k = 0$; $nb_{\text{start.event}} = 0$
1 : $nb_{\text{start.event}} = 1$; $J_0 = \{\text{job7}\}$
2 : $nb_{\text{start.event}} = 2$; $J_0 = \{\text{job7} ; \text{job 1}\}$
3 : $nb_{\text{start.event}} = 3$; $J_0 = \{\text{job7} ; \text{job 1} ; \text{job 2}\}$
4 : $nb_{\text{start.event}} = 4$; $J_0 = \{\text{job7} ; \text{job 1} ; \text{job 2} ; \text{job 8}\}$
5 : $nb_{\text{start.event}} = 5$; $J_0 = \{\text{job7} ; \text{job 1} ; \text{job 2} ; \text{job 8} ; \text{job 4}\}$
6 : $nb_{\text{start.event}} = 6$; $J_0 = \{\text{job7} ; \text{job 1} ; \text{job 2} ; \text{job 8} ; \text{job 4} ; \text{job 3}\}$
7 : $nb_{\text{start.event}} = 6$; $J_1 = \{\text{job7} ; \text{job 1} ; \text{job 8} ; \text{job 4} ; \text{job 3}\}$; $k = 1$
8 : $nb_{\text{start.event}} = 6$; $J_2 = \{\text{job 1} ; \text{job 8} ; \text{job 4} ; \text{job 3}\}$; $k = 2$
9 : $nb_{\text{start.event}} = 7$; $J_2 = \{\text{job 1} ; \text{job 8} ; \text{job 4} ; \text{job 3} ; \text{job 5}\}$
10 : $nb_{\text{start.event}} = 7$; $J_3 = \{\text{job 1} ; \text{job 4} ; \text{job 3} ; \text{job 5}\}$; $k = 3$
11 : $nb_{\text{start.event}} = 8$; $J_3 = \{\text{job 1} ; \text{job 4} ; \text{job 3} ; \text{job 5} ; \text{job 6}\}$
12 : $nb_{\text{start.event}} = n$; $J_4 = \{\text{job 4} ; \text{job 3} ; \text{job 5} ; \text{job 6}\}$; on retourne les J_h

Figure 12 – Application de l'algorithme permettant d'obtenir les sous-ensembles maximaux

Les sous-ensembles maximaux trouvés sont alors : $J_0 = \{7, 1, 2, 8, 4, 3\}$, $J_1 = \{7, 1, 8, 4, 3\}$, $J_2 = \{1, 8, 4, 3, 5\}$, $J_3 = \{1, 4, 3, 5, 6\}$ et $J_4 = \{4, 3, 5, 6\}$.

Après avoir obtenu les sous-ensembles maximaux de jobs, nous devons trouver un modèle de résolution exacte. Le modèle mathématique permettant d'obtenir une solution optimale indexée jobs est le suivant :

Maximize :

$$\sum_{i=0}^N \sum_{m=0}^M x_{im}$$

Subject to :

$$\sum_{l \in J_h} x_{lm} * q_{ljm} \leq Q_{jm}; \forall j \in R; \forall m \in M; h = 1 \dots k$$

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

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

Figure 13 – Formulation du problème multi-machines avec ressources multiples indexées job

Ici, tout comme le modèle temporel, nous cherchons à maximiser le nombre de jobs ordonnancés, si le job i est ordonnancé sur la machine m alors $x_{im} = 1$, sinon x_{im} vaut 0.

Pour chaque sous-ensemble maximal de jobs (J_h), nous vérifions que la somme de $x_{lm} * q_{ljm}$ est inférieure ou égale à 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 .

Nous devons également vérifier qu'un job n'est exécuté que sur une seule machine et ainsi que x_{im} soit inférieur ou égal à 1, pour tous les jobs i appartenant à N .

13 Résolution du problème en utilisant des heuristiques

Nous avons vu que notre problème d'ordonnancement multi-machine peut être résolu avec des méthodes de résolutions exactes. Néanmoins, notre problème est NP-Difficile, cela signifie que le temps de calcul croît de manière exponentielle en fonction de la taille des données d'entrées. Il sera donc nécessaire de trouver des méthodes de résolutions approchées permettant de trouver des solutions proches de la solution optimale mais avec un temps de calcul beaucoup plus faible.

13.1 Méthodes de tri

Dans un premier temps, nous cherchons à utiliser des méthodes de tri sur les jobs composant l'instance que nous devons résoudre. Actuellement, les méthodes de tri retenues sont les suivantes :

- CC_{max} : L'ensemble des jobs de l'instance sont triés par ordre croissant d'espace occupé ($\sum_{j \in R} q_{ij} * (f_i - s_i)$). 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 tri a été testée dans la publication mono-machine [2] et il s'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 ($\sum_{j \in R} q_{ij}$), 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 [Figure 6], puis nous les trions par ordre croissant suivant la quantité moyenne de ressources du sous-ensemble ($\frac{\sum_{i \in J_h} \sum_{j \in R} q_{ij}}{\text{nombre de jobs de } J_h}$). 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 rajouté. Cette méthode est inspirée des heuristiques de la publication d'ordonnancement multi-machine à ressource unique [3]. Il s'agit de la partie maximal-set based job sorting [Figure 10.1]. Ici, nous cherchons à privilégier les sous-ensembles maximaux ayant une quantité de ressources moyenne la plus faible possible.

Ces méthodes de tri sont la première étape à effectuer avant l'affectation des jobs sur les différentes machines.

13.2 Méthodes d'affectation

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

- Soit 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.
- Soit nous affectons chaque job sur la machine la moins chargée (la charge est ici définie par la formule suivante $\sum_{t \in \tau} y_{imt} * q_{ijm}; \forall j \in R; \forall i \in N$ où 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.

Voici l'algorithme permettant de définir le premier cas d'affectation :

```

- 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

```

Algorithme 1 : Algorithme Heuristique 1 (On affecte le maximum de jobs sur une première machine et on renouvelle l'opération sur les autres machines)

Et voici l'algorithme définissant le second cas d'affectation :

```

- 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  $(sk, fk)$ ;
            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
        end
    end
    if  $\{k\} \in L$  then
         $L = L / \{k\}$  ( $J_k$  est rejeté);
    end
end

```

Algorithme 2 : Algorithme Heuristique 2 (On affecte chaque job sur la machine étant la moins chargée, en cas d'égalité, l'ordre lexicographique est utilisé)

14 Conception

Avec les différents éléments que nous avons pu voir précédemment, il est possible d'effectuer la conception de notre futur programme avant de pouvoir démarrer la mise en œuvre.

Ainsi, il a été nécessaire de réaliser un diagramme de classe pour pouvoir développer notre application et ce de manière à obtenir un programme fiable, facilement maintenable et évolutif dans l'optique d'une reprise du projet dans le futur.

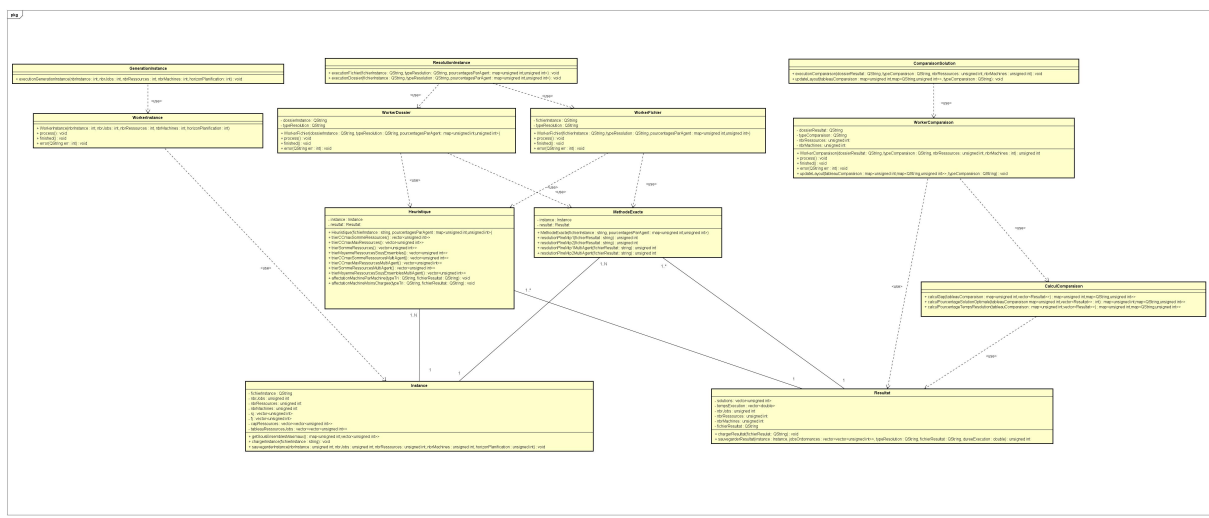


Figure 14 – Diagramme de classe de l'application

Comme nous avons pu le voir, il est nécessaire de réaliser trois types de tâches :

- La génération d'instances
- La résolution d'instances avec différentes méthodes de résolution
- La comparaison des différentes méthodes de résolution

Chacune des fenêtres permettant de réaliser une des tâches ci-dessus possède un ou plusieurs "Worker" qui lui sont liés. Un "Worker" est une classe qui permet d'effectuer plusieurs actions mais dans un thread séparé de l'application principale. Cela permet de ne pas bloquer l'application lorsqu'une génération ou une résolution est effectuée.

La génération créait plusieurs instances définies par la classe "Instance". Ces instances seront également utilisées par la partie de résolution qui les résoudra et créera des fichiers de résultat définis par la classe "Resultat". La comparaison des différentes méthodes de résolution utilisera les fichiers de résultats précédemment générés pour obtenir différents tableaux ou graphiques de comparaison.

Le fait d'utiliser des classes spécifiques aux fichiers d'instances ou de résultats est de rendre plus évidente la modification de celles-ci. Ce diagramme de classe a été réalisé dans l'optique de proposer une application respectant des standards de qualité de conception logicielle. Il sera par ailleurs bien plus évident d'ajouter de nouvelles fonctionnalités ou même de nouvelles classes avec ce type de conception qui permet d'assurer une certaine qualité au niveau de la maintenance et l'évolutivité.

Cinquième partie

Mise en œuvre

15 Outils utilisés

Pour ce qui est de la mise œuvre du projet, j'ai pu utiliser de nombreux outils afin de réaliser ce qui m'a été demandé.

Comme précisé précédemment, le langage de programmation utilisé sera le C++. Ce langage a été privilégié, car il est de bas niveau et offre de meilleures performances que les autres langages de manière générale. De plus, la résolution exacte du problème mono-machine avait déjà été codé en C++, de ce fait il m'a était plus facile de reprendre l'ancien code pour l'adapter au problème multi-machines.

Pour gérer le fait que mon application devait avoir un côté graphique, j'ai utilisé le framework QT afin de pouvoir gérer des IHM. Le fait d'utiliser QT permet de pouvoir rendre mon application plus évidente à utiliser et plus intuitive. J'ai également choisi ce framework, car il est encore très utilisé et reste une référence pour le développement C++ avec interface graphique. Le fait que son utilisation ait fait l'objet d'un module d'enseignement à Polytech Tours m'a également conforté dans mon choix.

La résolution exacte des problèmes d'affectations multi-machines est réalisée à partir du solveur cplex. De ce fait, j'ai dû intégrer cplex directement dans mon code et faire le lien entre QtCreator et la bibliothèque cplex. L'utilisation de celui-ci m'a permis de mettre en place tous les types de résolutions optimales que j'avais proposés dans la partie Analyse. Le solveur ayant été utilisé par Boukhalfa ZAHOUT pour la résolution mono-machine, il m'a était bien plus facile de l'utiliser et de comprendre son fonctionnement.

Pour la création de graphiques permettant la comparaison entre les différentes méthodes de résolution, j'ai pu utiliser la librairie Qwt. C'est une librairie spécifique à QT qui permet de tracer différents types de graphiques, et ce de manière assez simple et rapide. Son utilisation est primordiale pour l'implémentation de la partie qui permettra de comparer les vitesses d'exécutions et l'efficacité des différentes heuristiques que j'ai pu proposer précédemment.



Figure 15 – Logos des outils utilisés

16 Implémentation

16.1 Partie génération d'instances

Pour réaliser ce qui m'a été demandé lors du PRD, j'ai choisi de faire l'ensemble de mon programme de manière graphique (la résolution des instances ainsi que leurs générations se faisant à la base en ligne de commande). L'utilisation de l'application sera de ce fait beaucoup plus facile pour un public non initié et permettra simplement à l'utilisateur d'utiliser le programme beaucoup plus efficacement.

De ce fait la génération d'instance se fait désormais à l'aide d'une interface où l'on remplit via des champs le nombre de jobs, de ressources, de machines par instance ainsi que le nombre d'instances à générer et l'horizon de planification de celles-ci. Il est bien évidemment possible d'appeler directement la fonction de génération d'instances sans passer par l'interface afin d'effectuer plus facilement des tests par exemple.

La génération d'instance suit une arborescence très spécifique. À la racine du projet est créé le dossier Instance. Ensuite un sous-dossier portant un numéro est créé, il est ici pour éviter d'avoir des doublons lors de la génération de nouvelles instances. Par exemple, si l'on génère des instances de 20 jobs 3 ressources et 3 machines et qu'il existe des instances de 20 jobs 3 ressources et 3 machines dans un sous-dossier, alors un nouveau sous-dossier est créé afin de prendre en compte ces instances. Ensuite un sous-dossier est créé pour spécifier le nombre de jobs des instances, puis un autre sous-dossier pour le nombre de ressources et enfin un dernier sous-dossier pour le nombre de machines.

Chacune des instances est créée avec un nombre de jobs, de ressources et de machines fixes ainsi qu'avec une horizon de planification. Ensuite les instants de début et de fin de chaque job sont générés aléatoirement entre 0 et l'horizon de planification spécifiée auparavant (avec date de début < date de fin). Les machines générées ont chacune une valeur de 1000 pour chaque ressource. De ce fait les valeurs de ressources pour chaque job sont comprises entre 0 et 1000 de manière aléatoire. Toutes les valeurs aléatoires (date de début, de fin ou valeur) suivent une loi de distribution uniforme.

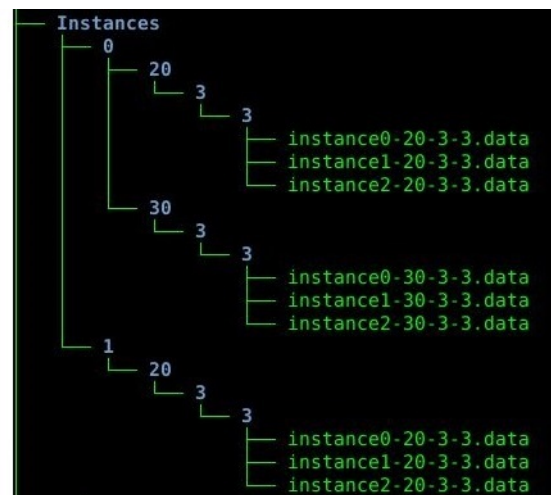
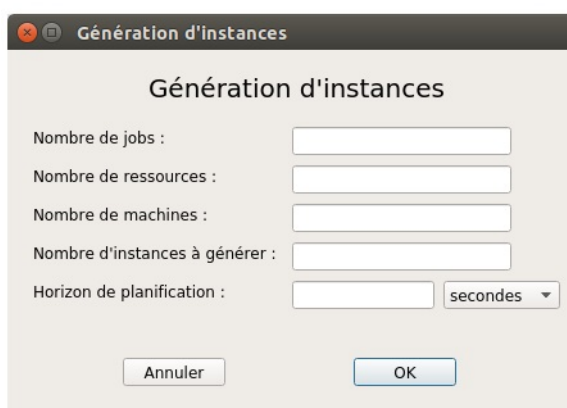


Figure 16 – Fenêtre permettant la génération d'instances et arborescence pour la création d'instances

Cette arborescence est très importante, car elle nous permet de classer les différentes instances de manière simple et compréhensible et cela nous servira également pour la suite pour générer l'arborescence pour les fichiers de résultat.

16.2 Partie résolution d'instances

La résolution d'instance doit être simple à utiliser et surtout rapide à lancer. De ce fait, une interface permettant la résolution d'instance suivant différentes méthodes a été développée :

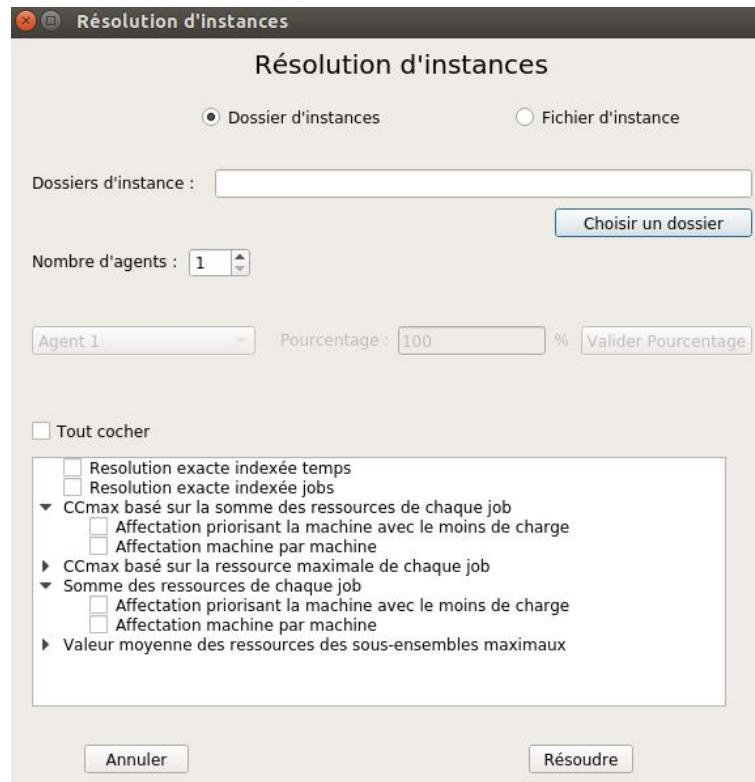


Figure 17 – Fenêtre permettant la résolution d'instances suivant différentes méthodes

Dans cette interface, nous pouvons choisir si nous voulons résoudre un dossier d'instances entier ou simplement un fichier d'instance seul à l'aide d'un bouton de choix. Ensuite un explorateur de fichier a été mis en place pour choisir le dossier ou le fichier en question lors de l'appui sur le bouton "Choisir un dossier" (ou "Choisir un fichier" si la case est cochée).

Une partie supplémentaire pour la résolution multi-agent a été mise en place sur l'interface, mais elle ne sera pas présentée ici, car cela concerne les méthodes de résolution d'un autre PRD (notre problème étant un cas mono-agent le nombre d'agents est de 1).

Ensuite, il suffit de cocher toutes les méthodes de résolution que l'on souhaite effectuer. Il y a au choix :

- La résolution exacte indexée temps appelée Mip1
- La résolution exacte avec contraintes de ressources appelée Mip2
- Les différentes heuristiques basées sur des méthodes de trie (par ordre croissant) :
 - CCmax basé sur la somme des ressources de chaque job
 - CCmax basé sur la valeur maximale des ressources de chaque job
 - Somme des ressources de chaque job
 - Valeur moyenne des ressources des sous-ensembles maximaux

Chaque méthode de tri possède deux types d'affectation qui peuvent être effectuée. La première consiste à affecter le maximum de jobs sur la première machine et de répéter l'opération sur les suivantes. La seconde affecte les différents jobs sur la machine la moins chargée en priorité, si cela n'est pas possible, la seconde machine la moins chargée est choisie et ainsi de suite jusqu'à ne plus avoir la possibilité d'ordonnancer des jobs.

Dès que la résolution est lancée, une nouvelle arborescence est créée. Elle est basée sur l'arborescence précédente (celle où sont stockées les instances), ce qui signifie que chaque fichier d'instance présent dans le dossier "Instances/numéroGénération/nombreDeJobs/nombreDeRessources/nombreDeMachines" sera résolu et écrit dans des fichiers de résultat présents dans le dossier "Resultat/numéroGénération/nombreDeJobs/nombreDeRessources/nombreDeMachines".

Pour la suite, il est important d'utiliser la résolution par dossier (la résolution par fichier ne crée actuellement que des fichiers uniques à chaque instance). Si l'on résout par dossier, chacune des résolutions sera faite dans un fichier unique qui portera le nom de la méthode de résolution (Affectation1 signifie que l'on affecte machine par machine et Affectation2 que l'on affecte sur la machine la moins chargée).

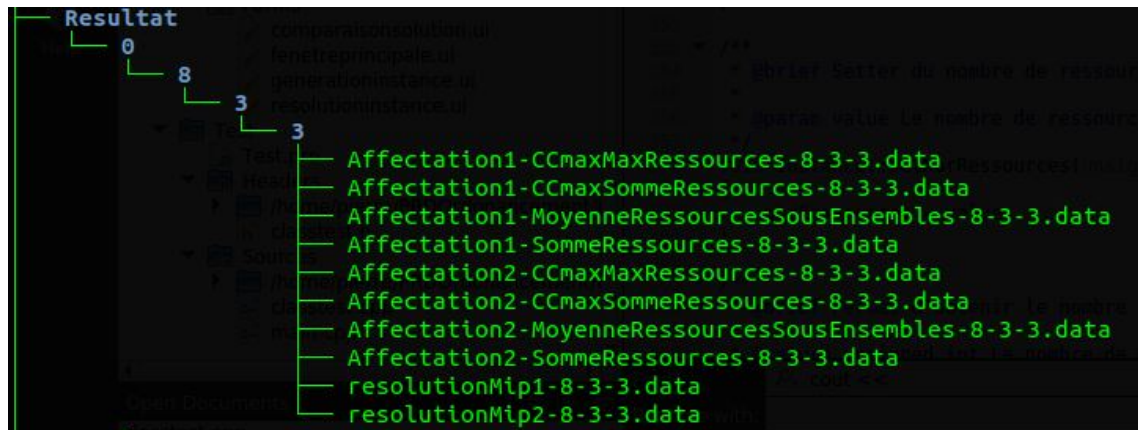


Figure 18 – Schéma de l'arborescence des résultats pour la résolution d'un dossier d'instance à 8 jobs, 3 ressources et 3 machines avec tous les types de résolutions

Les fichiers de résultats ainsi créés sont de la forme suivante :

- Le chemin du fichier d'instance
- Le type de résolution qui a été utilisé
- Le résultat (nombre de jobs ordonnancés) trouvé
- Le nombre de jobs pour chaque machine
- Le descriptif de chaque job (machine où le job est exécuté, numéro du job, date de début, date de fin)

```
Instance:Instances/0/8/3/3/instance1-8-3-3.data
Solution status:Mip1
Optimal Value=5
temps écoulé (en secondes):0.0889931
IdMachine Nombre de jobs ordonnancés
0 2
1 1
2 2
IdMachine n°Job Si Fi
0 2 771 1118
0 4 430 736
1 0 820 1088
2 1 966 1188
2 3 835 874
```

Figure 19 – Fichier de résultat pour une instance résolue, chaque instance étant ajoutée au fur et à mesure au fichier lors de la résolution

Comme précisé précédemment, les fichiers d'instances sont résolus et les résultats sont mis au fur et à mesure dans un fichier de résultat spécifique à la méthode de résolution. Il est important de résoudre les instances par dossier, car cela permet de bien écrire toutes les informations dans un seul et unique fichier ce qui est important pour la comparaison.

16.3 Partie comparaison des méthodes de résolution

Enfin, il a fallu développer un module de comparaison de méthode de résolution. Ce module doit permettre de vérifier la modification du gap ainsi que du pourcentage de solution optimale suivant le nombre de jobs des instances (avec un nombre de ressources et de machines fixées). Il y a également une comparaison prenant en compte le temps d'exécution de chaque méthode de résolution, ici on prend en compte le pourcentage de temps de résolution par rapport à la résolution exacte (pourcentage du temps de résolution d'une heuristique par rapport au Mip1).

Par exemple, si la résolution exacte met 1 seconde à s'exécuter et qu'une heuristique ne met que 0,1 seconde à être résolue, le pourcentage renvoyé sera de 10 %. Il est nécessaire pour les méthodes d'heuristique d'avoir un pourcentage de temps de résolution par rapport à la résolution exacte inférieure à 100 % car elle n'aurait pas d'intérêt à être exécutée (la solution exacte donnant à coût sur la solution optimale, ce qui n'est pas le cas des heuristiques).



Figure 20 – Fenêtre permettant la comparaison des heuristiques (gap, pourcentage de solution optimale et pourcentage du temps de résolution exacte)

La fenêtre permet de spécifier le dossier de résultat, le nombre de ressources et de machines ainsi que le type de comparaison. Il est également possible de pouvoir choisir le rendu de la comparaison : un graphique (avec en abscisse le nombre de jobs et en ordonnées le pourcentage correspondant à la comparaison que l'on effectue) ou un tableau contenant les différents résultats de la comparaison. Lorsque l'on clique sur le bouton de comparaison, un graphique ou un tableau permettant la comparaison apparaît. On peut effacer le graphique ou le tableau de comparaison grâce au bouton "Effacer Comparaison".

Si un graphique ou un tableau est déjà présent et que l'on effectue une nouvelle comparaison, le nouvel élément sera ajouté en dessous du premier et l'ensemble sera redimensionné.

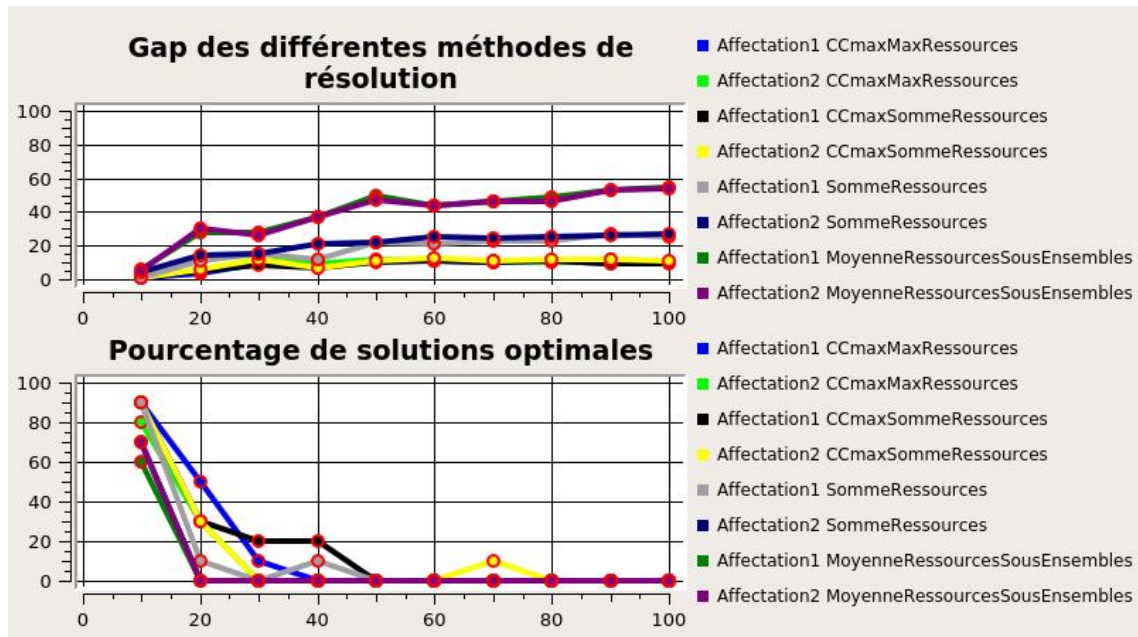


Figure 21 – Graphiques de comparaison de méthodes de résolution, pour un nombre de jobs allant de 10 à 100 avec 3 types de ressources et 3 machines

L'analyse de ces résultats est très important car il s'agit de valider les différentes heuristiques proposées lors de la partie Recherche du projet.

17 Analyse des résultats

17.1 Qualité de l'implémentation

Pour la partie "Développement" du PRD, il nous a été demandé de réaliser un travail respectant un certain standard au niveau de la qualité du code et de la documentation. Il est important de bien effectuer des tests sur le code, rédiger des documents d'installation, d'utilisation pour le programme ou encore un guide du développeur.

Le programme développé lors de ce PRD a été conçu de manière à être facilement maintenable et évolutif pour des projets futurs. Il est à noter que les différentes technologies utilisées lors de ce PRD sont encore aujourd'hui relativement utilisées et régulièrement mises à jour. Il est alors relativement important de tester les différents éléments de l'application pour obtenir un certain niveau de qualité. Il est ainsi beaucoup plus facile de reprendre une fonction sur laquelle des tests ont été effectués que sur une fonction non testée.

Des tests unitaires ont pu être intégrés au projet, en effet, le programme est découpé en deux parties distinctes : l'application principale et le sous-projet de test.



Figure 22 – Structure du programme

Ce sous-projet permet principalement d'effectuer des tests unitaires sur les différentes parties de l'application. Le principe est de réaliser un scénario d'utilisation standard de l'application. Ainsi nous testons :

- La génération d'instance (avec une génération aléatoire d'instances de 8 jobs pour que l'exécution ne soit pas trop longue).
- La résolution des instances précédemment générée en vérifiant qu'il n'y a pas d'erreurs liées à la résolution (Par exemple : les heuristiques donnent des résultats inférieurs ou égaux à ceux des méthodes de résolution exactes ou encore les méthodes de résolutions exactes doivent donner les mêmes résultats).
- La comparaison de solution en vérifiant que les différents pourcentages renvoyés soit cohérents (gap supérieur ou égal à 0 %, pourcentage de solutions exactes compris entre 0 et 100 %, pourcentage de temps de résolution par rapport à la résolution exacte supérieur à 0 %).

Des tests fonctionnels ont également été effectués pour vérifier par nous même que l'application fonctionne et que les méthodes de résolution sont cohérentes. Il est par exemple normal d'observer une augmentation du temps de résolution si le nombre de jobs augmente :

Instance:Instances/0/10/3/3/instance0-10-3-3.data Solution status:Mip1 Optimal Value=7 temps écoulé (en secondes):0.11207	Instance:Instances/0/100/3/3/instance0-100-3-3.data Solution status:Mip1 Optimal Value=38 temps écoulé (en secondes):4.73642
Instance:Instances/0/200/3/3/instance1-200-3-3.data Solution status:Optimal Optimal Value=56 temps écoulé (en secondes):345.944	

Figure 23 – Différence du temps d'exécution entre différentes instances

Nous pouvons ainsi remarquer que pour la résolution exacte indexée temps Mip1 par exemple, le temps d'exécution croit très largement suivant le nombre de jobs de départ (les temps indiqués varient fortement selon l'implémentation et l'environnement d'exécution).

Pour ce qui est de la maintenabilité et l'évolutivité de l'application, un certain nombre de choix ont été faits. En effet, il est important de prendre en compte le fait que l'on puisse dans le futur ajouter de nouvelles méthodes de résolution. Cela est possible assez facilement en créant des méthodes dans les classes "MethodeExacte" si l'on souhaite des méthodes exactes ou "Heuristique" si l'on souhaite des heuristiques.

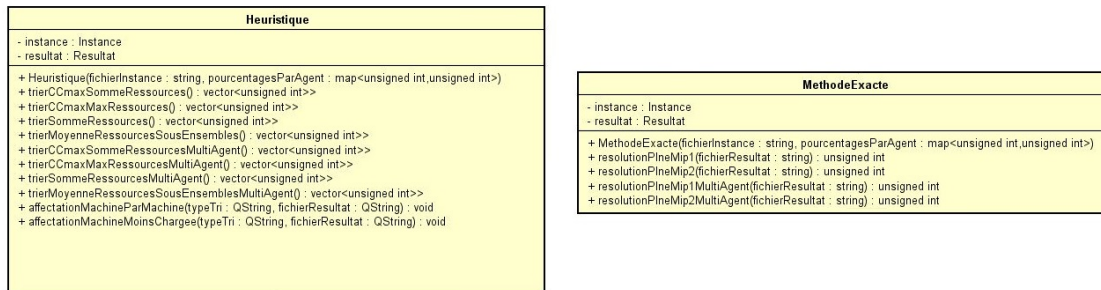


Figure 24 – Classes sur lesquelles il est possible d'ajouter des méthodes de résolution

Bien évidemment, il faudra également faire le lien entre les nouvelles méthodes de résolution et les vues de l'application (ajouter les nouvelles méthodes dans l'ensemble des checkboxes que l'ont peut cocher pour choisir la méthode de résolution). Mais l'ajout de nouveaux algorithmes dans les deux classes prévues a cet effet est très facile et cela rendra une possible reprise du projet bien plus facile.

Enfin, il est possible d'ajouter de nouvelles méthodes permettant de comparer les différentes méthodes de résolution dans la classe CalculComparaison :

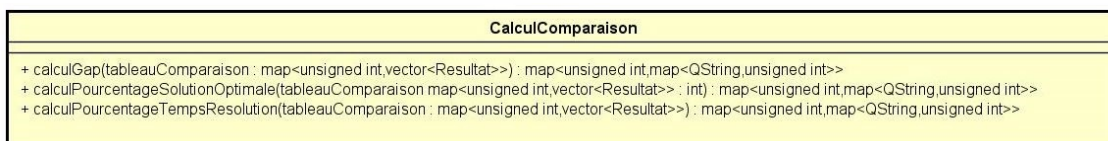


Figure 25 – Classe où il est possible d'ajouter de nouvelles méthodes pour comparer les méthodes de résolution

17.2 Résultats obtenus

Comme indiqué précédemment, l'objectif de ce PRD a était de développer une application permettant à la fois de générer et résoudre des instances à plusieurs machines, mais également de comparer les différentes méthodes de comparaison proposées dans la partie "Recherche" du projet.

Pour ce qui est de la partie génération d'instances, il s'agit de la tâche la plus simple du développement. En effet, il s'agit de créer des fichiers où l'on place différentes informations à propos des instances à résoudre. La génération s'effectue dans un thread séparé de l'application principale, mais l'exécution est particulièrement rapide et toutes les générations effectuées durant le projet ont paru instantanées pour l'utilisateur (il est évidemment possible de générer un très grand nombre d'instances, ce qui augmente le temps de génération).

La résolution d'instance quant à elle permet au final d'exécuter plusieurs méthodes de résolution les unes à la suite des autres directement en cochant des checkboxes sur la fenêtre prévue à cet effet. La résolution peut être faite sur un dossier complet d'instances ou simplement sur une instance unique. La résolution est très facile à effectuer et il est rapide de la lancer. Bien évidemment, le temps d'exécution pour la résolution dépend à la fois des méthodes de résolution cochées et du nombre de jobs, ressources et machines des instances.

La comparaison de méthode de résolution est la partie la plus importante du projet. En effet, elle sert à valider les différentes heuristiques proposées dans la partie "Recherche" du PRD. Grâce à cette partie, nous pouvons vérifier les 8 heuristiques que nous avons proposées (4 méthodes de tri différentes avec 2 affectations différentes pour chacune d'entre elles).

Il est dans un premier temps très important de calculer le gap de chaque heuristique pour se rendre compte de son efficacité. En effet le gap est en fait le pourcentage de dégradation de la solution trouvée par l'heuristique par rapport à la solution exacte. De ce fait, plus le gap est bas, plus la méthode de résolution est performante.

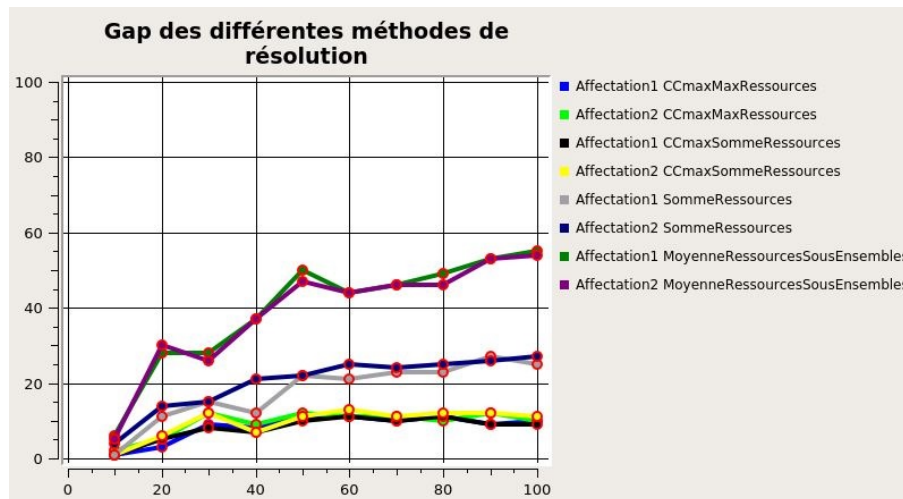


Figure 26 – Graphique représentant le gap des différentes méthodes de résolutions en fonction du nombre de jobs de chaque instance

Ici, nous avons lancé la résolution sur des instances ayant de 10 à 100 jobs (par pas de 10 jobs, avec à chaque fois 10 instances), 3 types de ressources et 3 machines. Nous remarquons que les heuristiques basées sur les méthodes de tri CCmax sont bien plus performantes que les autres méthodes de tri. De plus, nous remarquons également que les deux méthodes d'affectation donnent des résultats similaires pour une même méthode de tri.

Les différents pourcentages sont stockés dans le tableau si-dessous. Nous remarquons alors que les deux méthodes CCmax sont particulièrement similaires et que les deux méthodes d'affectations sont également très proches pour une même méthode de tri. De plus, nous remarquons que le tri basé sur la moyenne de ressources par sous-ensembles maximaux est particulièrement mauvais (gap d'environ 55 % pour des instances à 100 jobs). Les méthodes CCmax quant à elle restent particulièrement performantes avec un gap d'environ 10 % pour des instances à 100 jobs.

Nombre de jobs	10	20	30	40	50	60	70	80	90	100
CcmaxMaxRessource-MachineParMachine	1,00%	3,00%	9,00%	8,00%	11,00%	11,00%	10,00%	11,00%	9,00%	10,00%
CcmaxMaxRessource-MachineMoinsChargée	2,00%	5,00%	12,00%	9,00%	12,00%	11,00%	11,00%	10,00%	12,00%	10,00%
CcmaxSommeRessource-MachineParMachine	1,00%	5,00%	8,00%	7,00%	10,00%	11,00%	10,00%	11,00%	9,00%	9,00%
CcmaxSommeRessource-MachineMoinsChargée	1,00%	6,00%	12,00%	7,00%	11,00%	13,00%	11,00%	12,00%	12,00%	11,00%
SommeRessource-MachineParMachine	1,00%	11,00%	15,00%	12,00%	22,00%	21,00%	23,00%	23,00%	27,00%	25,00%
SommeRessource-MachineMoinsChargée	4,00%	14,00%	15,00%	21,00%	22,00%	25,00%	24,00%	25,00%	26,00%	27,00%
MoyenneRessourcesSousEnsemblesMaximaux-MachineParMachine	6,00%	28,00%	28,00%	37,00%	50,00%	44,00%	46,00%	49,00%	53,00%	55,00%
MoyenneRessourcesSousEnsemblesMaximaux-MachineMoinsChargée	5,00%	30,00%	26,00%	37,00%	47,00%	44,00%	46,00%	46,00%	53,00%	54,00%

Figure 27 – Tableau récapitulatif des différents gaps selon le nombre de jobs

De même, le pourcentage de solutions exactes est important, car c'est également un bon indicateur pour juger de la qualité d'une heuristique. Voici ce que donne la comparaison des méthodes de résolution suivant le pourcentage de solutions exactes (comparaison effectuée avec les mêmes instances que les graphiques précédents) :

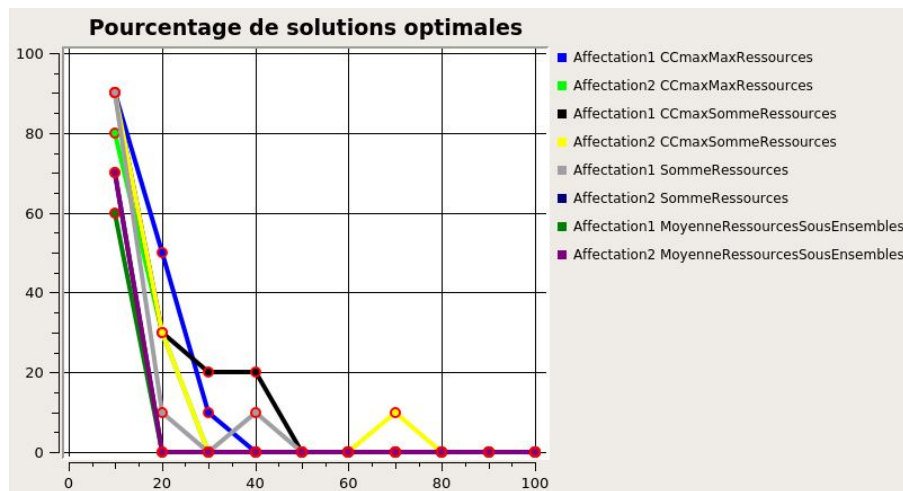


Figure 28 – Graphique représentant le pourcentage de solutions exactes par méthode de résolution en fonction du nombre de jobs de chaque instance

Nous remarquons encore une fois que les tris CCmax sont généralement meilleurs que les autres tris. Ici, la différence n'est pas particulièrement flagrante au vu du faible nombre d'instances (10 instances pour chaque pas de 10 jobs), il serait légitime d'effectuer cette comparaison avec un plus grand nombre d'instances.

Un autre type de comparaison est primordiale pour juger de la qualité de nos heuristiques. Il s'agit du pourcentage de temps de résolution par rapport à la résolution exacte. En effet, le principe d'une heuristique est de trouver une solution approchée tout en réalisant la résolution bien plus rapidement que les méthodes exactes. Il est donc primordial que nos heuristiques soient bien plus rapides que la résolution exacte. Voici le graphique montrant le pourcentage de temps de résolution par rapport à la résolution exacte pour chacune des heuristiques :

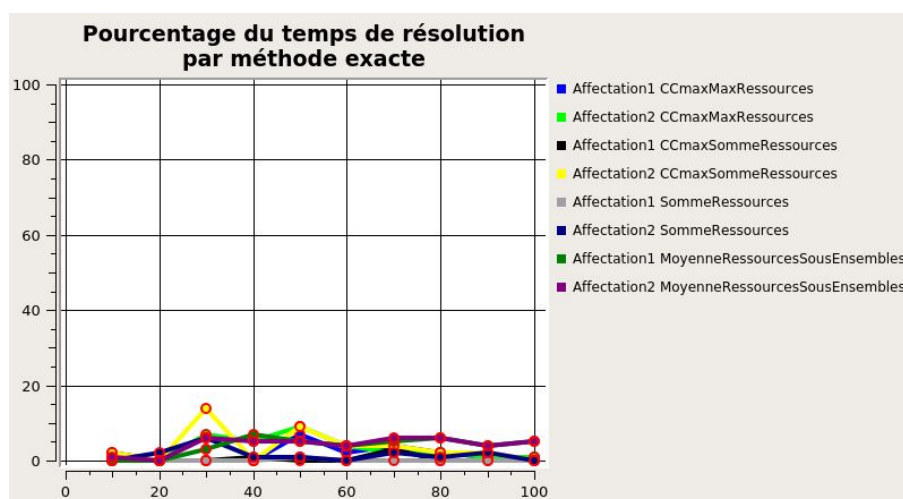


Figure 29 – Graphique représentant le pourcentage de temps de résolution par rapport à la résolution exacte pour chacune des heuristiques en fonction du nombre de jobs de chaque instance

Cette fois si, nous ne voyons pas vraiment de différences très importantes pour les méthodes de résolution. Il est d'ailleurs à noter que ces courbes ne sont jamais les mêmes lorsque l'on relance la résolution pour les mêmes instances (contrairement aux autres comparaisons qui sont toujours les mêmes, les solutions trouvées ne changeant pas). En effet, suivant l'état de la machine lors de la résolution, une heuristique peut voir son temps de résolution décuplé (c'est par exemple le cas avec la courbe jaune sur le graphique précédent où il y a des piques de valeurs). Afin de mieux, se rendre compte des différentes valeurs voici le tableau avec les différents pourcentages :

Nombre de jobs	10	20	30	40	50	60	70	80	90	100
CcmaxMaxRessource-MachineParMachine	0,00%	0,00%	0,00%	0,00%	7,00%	2,00%	4,00%	2,00%	1,00%	1,00%
CcmaxMaxRessource-MachineMoinsChargée	1,00%	0,00%	7,00%	5,00%	9,00%	4,00%	2,00%	2,00%	1,00%	1,00%
CcmaxSommeRessource-MachineParMachine	2,00%	0,00%	0,00%	1,00%	0,00%	0,00%	3,00%	0,00%	0,00%	0,00%
CcmaxSommeRessource-MachineMoinsChargée	2,00%	0,00%	14,00%	0,00%	9,00%	4,00%	4,00%	2,00%	2,00%	0,00%
SommeRessource-MachineParMachine	0,00%	0,00%	0,00%	0,00%	1,00%	0,00%	0,00%	0,00%	0,00%	0,00%
SommeRessource-MachineMoinsChargée	0,00%	2,00%	6,00%	1,00%	1,00%	0,00%	2,00%	1,00%	2,00%	0,00%
MoyenneRessourcesSousEnsemblesMaximaux-MachineParMachine	0,00%	0,00%	3,00%	7,00%	5,00%	4,00%	5,00%	6,00%	4,00%	5,00%
MoyenneRessourcesSousEnsemblesMaximaux-MachineMoinsChargée	1,00%	0,00%	6,00%	5,00%	5,00%	4,00%	6,00%	6,00%	4,00%	5,00%
Mip2	6,00%	85,00%	110,00%	143,00%	153,00%	145,00%	146,00%	166,00%	140,00%	187,00%

Figure 30 – Tableau récapitulatif des différents pourcentages de temps de résolution par rapport à la résolution exacte suivant le nombre de jobs

Nous remarquons que l'ensemble des heuristiques est particulièrement rapide face à la méthode de résolution exacte. Malheureusement, l'affichage étant en valeur entière, les pourcentages inférieurs à 1 % sont affichés à 0 % et il n'est donc pas possible d'avoir une analyse plus complète. Il sera possible d'améliorer cela en obtenant un affichage de nombre flottant.

Comme les heuristiques sont à chaque fois comparées à la méthode exacte, il est également légitime de comparer le temps de résolution de chaque méthode exacte. L'ensemble des pourcentages sont basés sur la méthode Mip1 (résolution exacte indexée temps) qui a le pourcentage de base : 100 %. La méthode par contraintes de ressources appelée Mip2 à également été comparée à Mip1. Nous pouvons remarquer que le Mip2 est meilleur que le Mip1 pour un nombre de jobs faible (inférieur ou égal à 20 jobs), mais reste moins performant au-dessus de 30 jobs.

Sixième partie

Bilan et conclusion du semestre 9

18 Bilan des tâches effectuées

Pour ce qui est du semestre 9, j'ai tout d'abord essayé de comprendre le problème qui m'était posé. Le problème posé étant un cas d'ordonnancement de job avec ressources multiples sur plusieurs machines, j'ai dans un premier temps étudié la publication du cas mono-machine réalisée au sein du laboratoire d'informatique de Tours [2].

Cette publication m'a en effet appris beaucoup de chose à propos du problème d'ordonnancement sur lequel je devais travailler. J'ai pu en faire un résumé et comprendre comment fonctionnait le modèle mathématique utilisé pour obtenir la solution optimale avec une résolution indexée temps. J'ai pu également comprendre les différentes heuristiques présentes dans l'article et les essayés à la main sur des instances avec peu de jobs. J'ai par ailleurs essayé de comprendre et d'interpréter les résultats obtenus avec ces heuristiques.

J'ai ensuite pu rechercher un modèle de résolution exacte indexée temps pour le cas multi-machines que je devais traiter. J'ai donc repris le modèle présent sur la première publication et je l'ai adapté au problème avec m machines. Ce modèle mathématique a pu être présenté et validé par la MOA.

Il a ensuite était décidé que je recherche également le modèle de résolution exacte indexé jobs où il est question de rechercher une solution optimale en se basant sur des sous-ensembles maximaux de jobs.

J'ai donc pu lire la publication écrite par Enrico Angelelli et al [3] où il est question de résoudre un problème d'ordonnancement similaire à celui que je dois traiter, mais avec des valeurs de priorité pour chaque job et avec une contrainte d'uniquement une ressource. Un modèle mathématique de résolution exacte indexé jobs permettait de résoudre le problème et cette formulation m'a aidée à trouver la formulation indexée jobs de mon problème. J'ai également pu m'inspirer des heuristiques que proposait la publication.

J'ai également essayé de comprendre comment fonctionner l'algorithme permettant de trouver les sous-ensembles maximaux de jobs présents dans une instance. J'ai alors pu obtenir un modèle de résolution exacte indexé jobs et le faire valider par la MOA.

J'ai alors pu commencer mes recherches sur les heuristiques permettant de résoudre ce problème NP-Difficile avec des méthodes de résolutions approchées, mais ces solutions devront être bien plus rapides que la résolution exacte.

La partie de recherche des heuristiques est encore en cours et prend un peu de retard par rapport à ce que j'espérer. La rédaction des différents documents (cahier de spécification, état de l'art) ainsi que certaines incompréhensions de ma part lors du PRD ont fait que la partie de recherche d'heuristique n'est pas encore très avancée.

Je remercie par ailleurs la MOA pour son aide lors de cette première partie de PRD et plus particulièrement Boukhalfa ZAHOUT avec qui j'ai pu échanger directement presque chaque semaine, lorsque je rencontrais des difficultés.

19 Tâches restantes à réaliser

Comme j'ai pu le préciser plus haut, la partie de recherche sur les heuristiques n'est pas encore terminée et se poursuivra jusqu'à mi-Janvier. Il sera également important que l'ensemble des heuristiques soient validées avant fin Janvier.

Il faudra ensuite que je développe les modules nécessaires à la génération et résolution d'instances ainsi que la comparaison des différentes méthodes de résolution et que je les teste ensuite.

Afin de décrire les différentes tâches et de leur donner une durée de réalisation, un diagramme de Gantt a été réalisé :

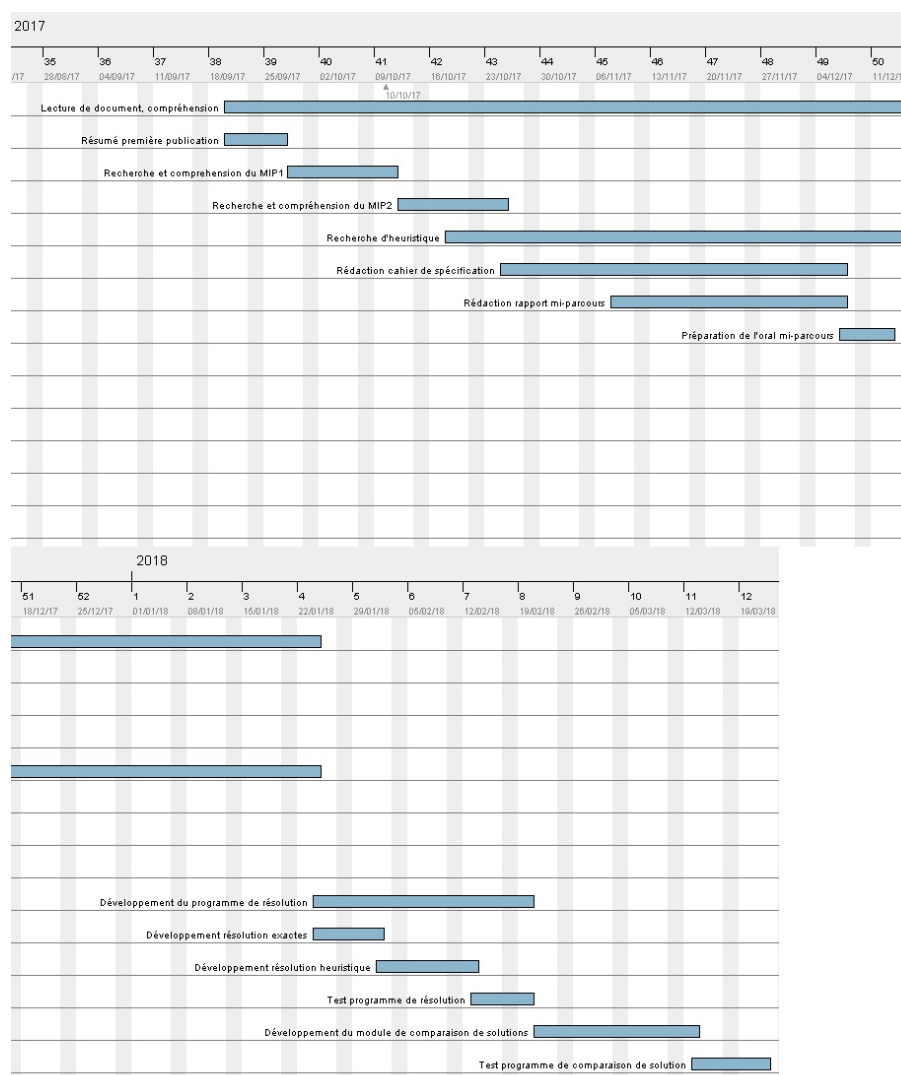


Figure 31 – Diagramme de Gantt actuel

Les tâches à réaliser au semestre 10 seront mises à jour au fur et à mesure de l'avancement du projet.

20 Conclusion

J'espère que la fin de la partie recherche me permettra de trouver des heuristiques fonctionnelles et servant à résoudre de manière plus rapide le problème d'ordonnancement traité lors de ce PRD. De même, il est évident que le développement de module de résolution d'instances et de comparaison de méthodes de résolutions permettra d'avoir des réponses sur les performances des solutions que je proposerai par la suite.

Je suis satisfait du travail que j'ai pu effectuer lors de cette première partie de PRD. J'ai pu obtenir des résultats à partir des recherches que j'ai pu effectuer et cela m'apporte une certaine satisfaction. Il est vrai que j'ai avancé que ce que j'imaginai, mais je suis convaincu que ce projet est très formateur et nous rapproche d'un contexte de production ou de recherche suivant le cas.

Septième partie

Bilan et conclusion du semestre 10

21 Bilan des tâches effectuées

Pour ce qui est du semestre 10, j'ai pu développer l'application qui m'avait été demandée pour ce PRD et que j'ai pu spécifier au semestre 9. J'ai rapidement débuté le développement et ceux plus tôt que ce que j'avais prévu initialement (développement débuté début Janvier tandis qu'il avait été prévu fin Janvier). J'ai pris cette décision, car je me suis rendu compte de la charge de travail à effectuer lors du semestre 10 et j'ai ainsi arrêté mes recherches d'heuristiques bien plus tôt.

J'ai dans un premier temps choisi un IDE et framework pour développer. Qt fut choisi et j'ai ensuite commencé l'implémentation. J'ai mis en place un générateur d'instance multi-machines qui permet comme son nom l'indique de générer des instances en spécifiant, le nombre de jobs, de ressources et de machines pour chaque instance. Il faut également spécifier le nombre d'instances à générer ainsi que leur horizon de planification (date de fin maximale des instances).

Ensuite, j'ai réalisé le module de résolution d'instances. Ce module permet de résoudre des instances soit en choisissant un dossier contenant plusieurs instances soit en choisissant une seule instance. Il est alors possible de choisir différentes méthodes de résolution en les cochant (cette méthode permet de choisir plusieurs types de résolution à la fois et ainsi de n'avoir à lancer la résolution qu'une seule fois). Une fois la résolution lancée, toutes les instances choisies sont résolues séquentiellement par chacune des méthodes de résolution.

Enfin, j'ai pu mettre en place un module de comparaison de méthodes de résolution qui permet au final de valider les heuristiques que j'ai proposées dans la partie Recherche. Ce module permet de choisir un dossier contenant des fichiers de résultats, de les filtrer selon le nombre de ressources et de machines, et de les comparer selon ce que l'on souhaite : Gap des heuristiques, pourcentage de solutions exactes, pourcentages de temps de résolution par rapport à la résolution exacte. Il est alors possible d'afficher la comparaison soit sous la forme de graphique, soit dans un tableau.

Lors de cette partie "Développement", j'ai également mis en place des éléments sur la qualité en mettant en place des tests et en rédigeant des documents pour faciliter la reprise du projet.

J'ai également dû effectuer un nombre assez important de recherches notamment pour installer les différentes librairies sur Qt. Le lien entre Qt et Cplex notamment a été une tâche à part entière et même prioritaire à un moment donné, car sans le solveur la résolution n'était plus possible.

Il en va de même pour Qwt qui permet de dessiner des graphiques directement sur Qt. J'ai passé beaucoup de temps pour réussir à l'installer correctement et cela m'a d'ailleurs fait perdre un peu de temps à ce moment.

Au final, l'ensemble des tâches prévues ont été effectuées dans les temps et le programme fonctionne bien. Il existe encore de nombreuses améliorations possibles et peut-être qu'il serait possible de poursuivre le projet pour un autre PRD.

22 Bilan sur la qualité

Dans le cadre du PRD, il a été nécessaire de faire attention sur la qualité de notre implémentation. Une évaluation a d'ailleurs été effectuée et c'est Monsieur Jean-Yves RAMEL qui a évalué les éléments sur la qualité de ce projet.

Dans un premier temps, il a été nécessaire de réaliser une bonne conception pour le projet. Un diagramme de classe complet a d'ailleurs été réalisé pour cela. La conception prend en compte

le fait qu'il doit être facile de modifier l'application sans pour autant avoir à modifier l'ensemble du code produit.

Ainsi il est surtout possible d'ajouter de nouvelles méthodes de résolution dans les classes "MethodeExacte" ou "Heuristique" selon ce que l'on cherche à faire. Ici, l'ajout de nouveaux algorithmes de résolution se fait simplement en ajoutant une méthode à la classe existante (tout en modifiant le lien avec la vue bien évidemment). Il est alors facile de mettre à jour l'application et de la maintenir.

De plus, des tests unitaires ont été effectués en ajoutant un sous-projet de test à l'application principale. Cela permet par exemple de vérifier qu'une fonctionnalité réalise correctement son travail même après avoir effectué des modifications sur celle-ci. Les éléments de la génération et la résolution d'instances ont été testés ainsi que la comparaison de méthodes de résolution.

Différents documents ont également pu être produits afin de faciliter une éventuelle reprise du projet. Ainsi, une documentation complète du code a pu être générée au format Doxygen et des documents écrits tels qu'un cahier du développeur [??], un guide d'installation [??], un guide d'utilisation [??] ou un plan de test [??] ont pu être produits.

Je suis particulièrement satisfait de ce que j'ai pu réaliser pour garantir la qualité de mon implémentation. En effet, j'ai le sentiment qu'il sera ainsi bien plus facile de reprendre mon code et de poursuivre le projet sans avoir à effectuer une longue phase de recherche et de compréhension du code.

23 Bilan sur la gestion du projet

La phase de développement a été réalisée en suivant un modèle de gestion de projet agile où je réalisais une ou plusieurs tâches chaque semaine et où j'effectuais une réunion hebdomadaire (dans la plupart des cas) avec mon encadrant et client (Boukhalifa ZAHOUT) qui vérifie le travail effectué, le valide et propose de nouvelles tâches pour la semaine suivante.

Comme j'ai pu le préciser précédemment, j'ai modifié le planning initialement prévu au S9 et j'ai commencé le développement de l'application 3 semaines en avance. Cela m'a laissé plus de temps pour le développement et j'avoue qu'avec le recul, je me rends compte que c'était une très bonne décision. Sans cette modification du planning initial, je n'aurais sans doute pas fini le projet à temps.

Voici le planning initialement prévu :

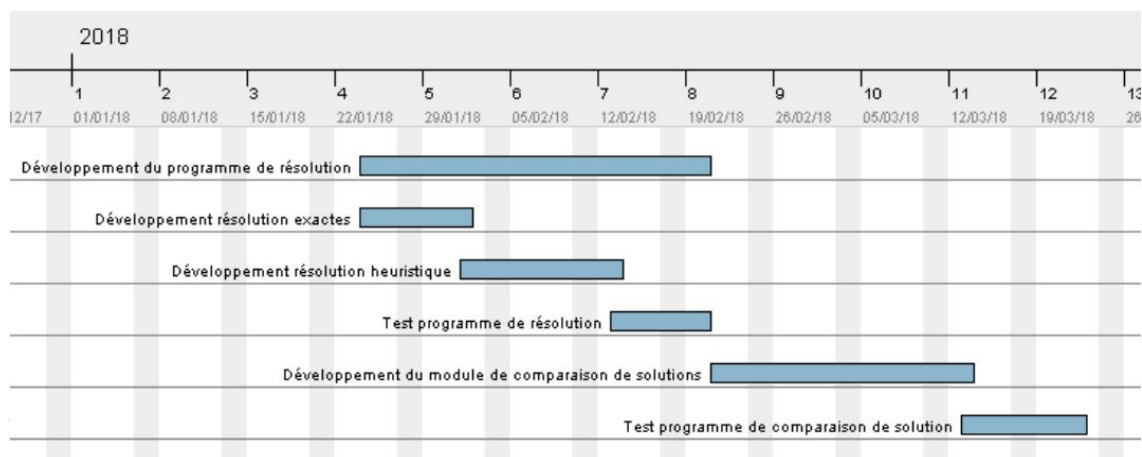


Figure 32 – Diagramme de Gantt du S10 prévu au S9

Et voici le planning réel des tâches effectuées :

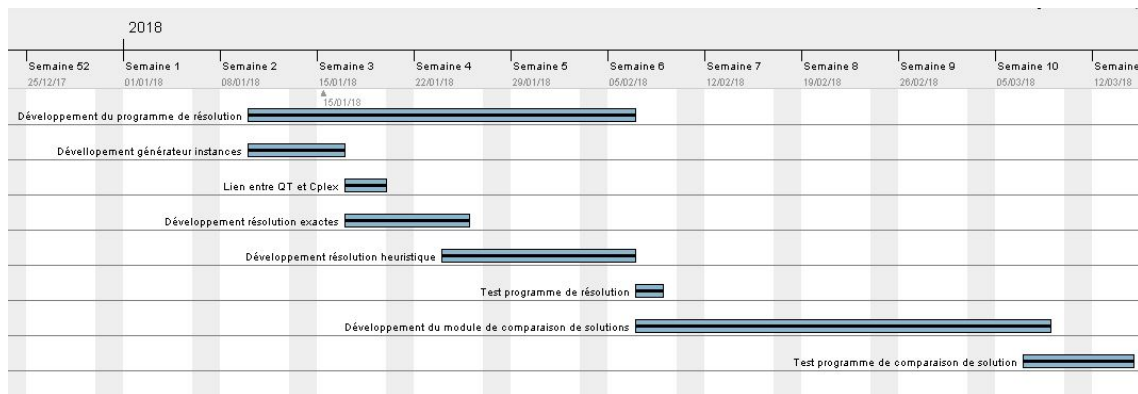


Figure 33 – Diagramme de Gantt des tâches réalisées au S10

Nous pouvons remarquer que les dates des différentes tâches ont pu être modifiées et que certaines tâches ont été ajoutées. Cependant, le planning réel proposé ici à était fait en début de semestre 10 et n'a absolument pas été modifié depuis. Les différentes tâches ont été effectuées dans les délais (un travail personnel externe aux séances de PRD a été nécessaire pour respecter les délais) et l'ensemble des tâches initiales ont été faites et validées par le client/encadrant.

Je suis particulièrement satisfait de la gestion de ce projet. En effet, le modèle agile m'a permis d'avancer rapidement en ayant des objectifs à court terme et en ayant des réponses à mes interrogations hebdomadairement avec les réunions effectuées avec mon encadrant/client.

Ainsi, le projet a pu être mené à terme et cela me procure une grande satisfaction non seulement pour avoir réussi à avancer tout au long du projet, mais également pour avoir réalisé une application avec un bon niveau de qualité et qui réalise les tâches qui ont été définies au cours du projet.

24 Conclusion

En conclusion, je suis particulièrement satisfait du travail que j'ai pu effectuer lors de PRD et surtout, j'ai pu réussir à développer une application de qualité tout en menant à bien les objectifs du projet. Ce PRD m'a permis d'acquérir des connaissances dans le domaine de la recherche opérationnelle, du développement et également de la gestion de projet.

En tant que projet de fin d'études, cette expérience a été tout de même assez proche d'un travail qui aurait pu être demandé dans un contexte professionnel. Il est vrai que ce projet a été compliqué et le fait de devoir réaliser un grand nombre de tâches en autonomie avec des éléments à rendre régulièrement est pour moi un exercice qui est loin d'être facile.

Dans tous les cas, ce projet est particulièrement formateur et permet vraiment de se rendre compte de la complexité du travail d'ingénieur à travers les phases de recherches, de développement et de gestion de projet. J'ai pu à travers ce projet être confronté à de nombreuses difficultés que j'ai pu surmonter grâce à mes connaissances, ma capacité à trouver l'information ou mon analyse. J'ai ainsi pu mettre en pratique ces capacités qui feront de moi un futur ingénieur et j'en suis particulièrement fier.

Pour finir, je tiens vraiment à remercier Boukhalfa ZAHOUT, mon encadrant et client sur ce projet qui a donné beaucoup de son temps et de sa personne pour me faire avancer à travers de nombreuses réunions pour gérer au mieux le projet. J'ai par ailleurs le sentiment d'avoir vraiment accompli quelque chose d'utile pour la suite, car j'ai entretenu de bonnes relations avec mon client tout au long du projet et je pense au final qu'avoir une relation de confiance avec son client et tout aussi important que d'avoir mené à bien le projet.

Huitième partie

Annexes

25 Description des interfaces externes du logiciel

25.1 Interfaces matérielles/logicielles

Le système de résolution d'instance sera en ligne de commande et devra donc se lancer à partir du terminal. La principale caractéristique du matériel ayant des influences sur le logiciel est la puissance de calcul de la machine. En effet, le logiciel ne nécessite pas une grande quantité de mémoire ou de stockage, les résultats étant fournis au format texte. La puissance de calcul quant à elle est primordiale pour obtenir des résultats dans un temps acceptable. Cela est d'autant plus vrai pour la résolution exacte du problème, car en tant que problème NP-Difficile, le temps de calcul croît de manière exponentielle si l'on augmente les données d'entrées.

Le système de comparaison de solution sera quant à lui une interface simple permettant de visualiser les différences entre les méthodes de résolutions heuristiques et exactes. Ici encore la puissance de calcul de la machine permettra d'obtenir plus rapidement les résultats de comparaison.

25.2 Interfaces homme/machine

Il n'y aura pas d'interface graphique pour le programme de résolution d'instance. Les instructions seront effectuées en ligne de commande et s'il y a une erreur, l'utilisateur en sera directement informé sur le terminal. Les utilisateurs de l'application seront la MOA, de ce fait ils connaîtront l'application et le problème que l'application résoudra.

Une interface graphique simple sera développée pour le module de comparaison de méthode. Il est ici nécessaire de développer une interface graphique afin que les résultats obtenus soient facilement interprétables.

26 Spécifications fonctionnelles

26.1 Description des fichiers d'entrées :

Les fichiers d'entrées seront au format txt. L'organisation à l'intérieur du fichier respectera le modèle suivant (avec des jobs allant d'un numéro entre 1 et j, des ressources ayant un numéro entre 1 et r et des machines ayant un numéro entre 1 et m) :

Nombre Jobs	Nombre Ressources	Nombres Machines
Quantité Ressource 1 Machine 1	Quantité Ressources 2 Machine 1	Quantité Ressources r Machine 1
...
Quantité Ressource 1 Machine m	Quantité Ressources 2 Machine m	Quantité Ressources r Machine m
Numero Job 1	Instant de départ Job 1	Instant de fin Job 1
...
Numero Job j	Instant de départ Job j	Instant de fin Job j
Quantité Ressource 1 Job 1	Quantité Ressource 2 Job 1	Quantité Ressource r Job 1
...
Quantité Ressource 1 Job j	Quantité Ressource 2 Job j	Quantité Ressource r Job j

Figure 34 – Structure à respecter pour les fichiers d'instances

Exemple (8 jobs compris entre 0 et 7, 3 ressources, 3 machines) :

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

Figure 35 – Exemple de fichier d'instance à 8 jobs, 3 ressources, 3 machines

26.2 Description des fichiers de sorties :

Les fichiers de sorties obtenus après avoir généré la solution sont de la forme suivante :

```
Instance:Instances/0/8/3/3/instance1-8-3-3.data
Solution status:Mip1
Optimal Value=5
temps écoulé (en secondes):0.0889931
IdMachine Nombre de jobs ordonnancés
0 2
1 1
2 2
IdMachine n°Job Si Fi
0 2 771 1118
0 4 430 736
1 0 820 1088
2 1 966 1188
2 3 835 874
```

Figure 36 – Exemple de fichier de résultat à 8 jobs, 3 ressources, 3 machines

Ces valeurs seront utilisées par le module de comparaison de solution afin de pouvoir comparer essentiellement les heuristiques par rapport aux méthodes exactes.

26.3 Description des classes du système :

Le module de calcul de solutions contiendra plusieurs classes :

- Une classe ChoixFichier permettant de choisir un fichier de description de problème. Ce fichier contient l'instance mise au bon format et pour laquelle nous chercherons une solution d'ordonnancement respectant notre problème.
- Une classe MethodeExacte qui contiendra toutes les méthodes permettant de calculer les solutions exactes d'instances données.
- Une classe Heuristique qui contiendra toutes les méthodes permettant de calculer des solutions exactes d'instances données à travers plusieurs heuristiques.
- Une classe GenerationSolution permettant de gérer l'ensemble du module.

Le module de comparaison de solution contiendra également différentes classes :

- Une classe ComparaisonSolution permettant de gérer l'ensemble du module.

26.4 Description des fonctionnalités du module de résolution d'instances :

Définition de la fonction 1 : generationInstance

Identification de la fonction 1 : Cette fonction permet de générer une instance suivant les données passées en paramètres. L'instance devra respecter le format des fichiers d'instance.

Description de la fonction 1 : Cette fonction prend en entrée, le nombre de jobs voulus, le nombre de ressources voulues, le nombre de machines voulues, la quantité maximale de ressources, la durée de l'instance. Elle créera ensuite en sortie un fichier d'instance.

Définition de la fonction 2 : calculExactTemps

Identification de la fonction 2 : Cette fonction permet de générer une solution exacte indexée temps pour une instance donnée.

Description de la fonction 2 : Cette fonction prend en entrée un fichier d'instance, va calculer une solution optimale au problème avec une méthode exacte indexée temps et écrira le résultat et le temps d'exécution dans un fichier en sortie.

Définition de la fonction 3 : calculExactJobs

Identification de la fonction 3 : Cette fonction permet de générer une solution exacte indexée jobs pour une instance donnée.

Description de la fonction 3 : Cette fonction prend en entrée un fichier d'instance, va calculer une solution optimale au problème avec une méthode exacte indexée jobs et écrira le résultat et le temps d'exécution dans un fichier en sortie.

Définition de la fonction 4 : calculHeuristique

Identification de la fonction 4 : Cette fonction permet de générer une solution approchée pour une instance donnée. Les heuristiques choisies sont encore en cours de recherche.

Description de la fonction 4 : Cette fonction prend en entrée un fichier d'instance, va calculer une solution approchée au problème avec une heuristique et écrira le résultat et le temps d'exécution dans un fichier en sortie.

26.5 Description des fonctionnalités du module de comparaison de solution :

Définition de la fonction 1 : pourcentageSolutionOptimale

Identification de la fonction 1 : Cette fonction permet de trouver le pourcentage de solution optimale obtenu avec la méthode utilisée pour un type d'instance donné.

Description de la fonction 1 : Cette fonction va rechercher les solutions obtenues lors de la résolution des instances à partir d'heuristiques particulières (présentes dans les fichiers de sorties après la résolution) et va les comparer avec les résultats obtenus avec la méthode exacte. La fonction retournera donc le pourcentage de solution optimale obtenu avec la méthode approchée proposée.

Définition de la fonction 2 : gapSolutionOptimale

Identification de la fonction 2 : Cette fonction permet de trouver le gap de la solution approchée par rapport à la solution optimale. Il s'agit du pourcentage de dégradation entre la solution optimale et la solution approchée.

Description de la fonction 2 : Cette fonction va rechercher les solutions obtenues lors de la résolution des instances à partir d'heuristiques particulières (présentes dans les fichiers de sorties après la résolution) et va les comparer avec les résultats obtenus avec la méthode exacte. La fonction retournera donc le pourcentage de dégradation global entre la solution optimale et la solution approchée.

Définition de la fonction 3 : differenceTempsSolution

Identification de la fonction 3 : Cette fonction permet de comparer le temps d'exécution nécessaire pour obtenir une solution approchée par rapport au temps d'exécution pour obtenir une solution exacte.

Description de la fonction 3 : Cette fonction va rechercher les temps d'exécution obtenus lors de la résolution des instances à partir d'heuristiques particulières (présentes dans les fichiers de sorties après la résolution) et va les comparer avec les temps d'exécution obtenus avec la méthode exacte. La fonction retournera donc la différence de temps d'exécution sous la forme d'un pourcentage (si une méthode exacte mets 10 secondes à s'exécuter et que l'heuristique mets 3 secondes pour trouver la solution, alors la fonction retournera 30%).

27 Spécifications non fonctionnelles

27.1 Contraintes de développement et conception

Il n'y a pas de contrainte matérielle pour le développement du module de calcul de solutions ni pour celui de 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 C++. C'est en effet un langage de bas niveau qui permettra d'obtenir de meilleures performances. Pour le développement des calculs de solutions exactes, le solveur cplex_216 sera utilisé. Le développement se fera sous Linux, distribution Ubuntu car il est plus facile d'intégrer le solveur cplex_216 sur ce système d'exploitation.

27.2 Performances

Étant donné que notre programme calcule des solutions pour des cas d'ordonnancements multi-machines avec ressources multiples, les performances de notre programme seront grandement affectées par la capacité de calcul de la machine sur laquelle le programme s'exécute. Le problème d'ordonnement étant NP-Difficile, des méthodes heuristiques seront implémentées pour résoudre efficacement les instances ayant une taille importante.

27.3 Capacités

L'application devra permettre de traiter des fichiers texte. Ce qui signifie que le volume de données à traiter est peu important. Néanmoins, selon la taille de ces fichiers, le temps de traitement peut très rapidement augmenter.

27.4 Contrôlabilité

L'application devra afficher les résultats à l'utilisateur directement sur le terminal.

27.5 Sécurité

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

27.6 Intégrité

Il n'est pas nécessaire de mettre en place des protections contre la déconnexion imprévue. 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 et qu'il y avait une déconnexion imprévue, alors il faudrait relancer le calcul. Il n'y a donc pas besoin de se soucier de la récupération des données.

28 Plan de développement

28.1 Tache 1 : Lecture de document et compréhension

Description de la tâche : Il s'agit de lire différents documents permettant de comprendre notre problème et d'en définir le périmètre. Cette tâche sera réalisée durant l'ensemble de la partie recherche du projet ainsi que pendant le début de la partie développement. Ces différents documents sont des articles scientifiques ou des informations communiquées par la MOA.

Estimation de charge : Cette tâche est estimée à 5 jours/homme.

28.2 Tache 2 : Résumé première publication

Description de la tâche : Il s'agit de résumer la première publication. Cette partie va permettre de comprendre le problème initial de notre projet et de découvrir comment a été étudié le problème d'ordonnancement mono-machine.

Estimation de charge : Cette tâche est estimée à 1 jours/homme.

28.3 Tache 3 : Recherche et compréhension du MIP1

Description de la tâche : Il s'agit de rechercher des informations sur le modèle de résolution exacte indexée temps du cas mono-machine afin qu'il soit valide pour le cas multi-machines

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.4 Tache 4 : Recherche et compréhension du MIP2

Description de la tâche : Il s'agit de rechercher des informations sur le modèle de résolution exacte indexée jobs du cas mono-machine afin qu'il soit valide pour le cas multi-machines

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.5 Tache 5 : Recherche d'heuristiques

Description de la tâche : Il s'agit de rechercher des informations sur les différentes heuristiques possibles pour résoudre notre problème de manière plus rapide. Cette tâche comprend également toute la phase de compréhension et des tests sur des petites instances qui seront effectués pour valider théoriquement les heuristiques.

Estimation de charge : Cette tâche est estimée à 6 jours/homme.

28.6 Tache 6 : Rédaction du cahier de spécification

Description de la tâche : Il s'agit de rédiger le cahier de spécification système. Ce document nous permet de définir les objectifs du projet, son contexte et les différentes fonctionnalités qui devront être implémentées par la suite.

Estimation de charge : Cette tâche est estimée à 4 jours/homme.

28.7 Tache 7 : Rédaction état de l'art

Description de la tâche : Il s'agit de rédiger l'état de l'art qui sert à évoquer les problèmes similaires à celui que nous étudions dans ce projet.

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.8 Tache 8 : Rédaction rapport de mi-parcours

Description de la tâche : Le rapport du PRD est la concaténation du cahier de spécification et de l'état de l'art. Il possède également une partie analyse et conception et une partie annexe. Il est nécessaire de le rendre.

Livrable : Rapport de la partie Recherche du PRD

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.9 Tache 9 : Préparation de l'oral de mi-parcours

Description de la tâche : Il s'agit ici de la préparation à l'oral de mi-parcours. Cette tâche est composée de l'élaboration d'un powerpoint et des différentes répétitions de cet oral.

Livrable : Powerpoint de la soutenance de mi-parcours

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.10 Tache 10 : Développement du programme de résolution

Description de la tâche : Il s'agit ici de réaliser toute la partie développement spécifique au programme de résolution d'instance. Il faudra dans un premier temps récupérer le code existant du cas mono-machine et l'adapter à notre problème.

Livrable : Code du module de résolution d'instances

Estimation de charge : Cette tâche est estimée à 8 jours/homme.

28.11 Tache 11 : Développement des méthodes de résolution exactes

Description de la tâche : Il s'agit ici de développer les méthodes de résolution exactes que nous avons trouvées dans la partie recherche. Il s'agira de coder les deux méthodes exactes de la partie analyse et conception.

Estimation de charge : Cette tâche est estimée à 3 jours/homme.

28.12 Tache 12 : Développement des méthodes de résolution heuristique

Description de la tâche : Il s'agit ici de développer les méthodes heuristiques que nous avons trouvées dans la partie recherche. Il s'agira de coder toutes les heuristiques que nous proposons.

Estimation de charge : Cette tâche est estimée à 3 jours/homme.

28.13 Tache 13 : Test du programme de résolution

Description de la tâche : Il s'agit ici de tester l'ensemble du programme de résolution d'instance. Il sera nécessaire de le tester avec différentes instances, de vérifier que les solutions correspondent bien à ce que l'on attend.

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.14 Tache 14 : Développement du module de comparaison de solution

Description de la tâche : Il s'agit ici de développer le module de comparaison de solution. Il faudra ici réaliser une interface graphique simple et implémenter différentes méthodes de comparaisons.

Estimation de charge : Cette tâche est estimée à 6 jours/homme.

28.15 Tache 15 : Test du programme de comparaison de solution

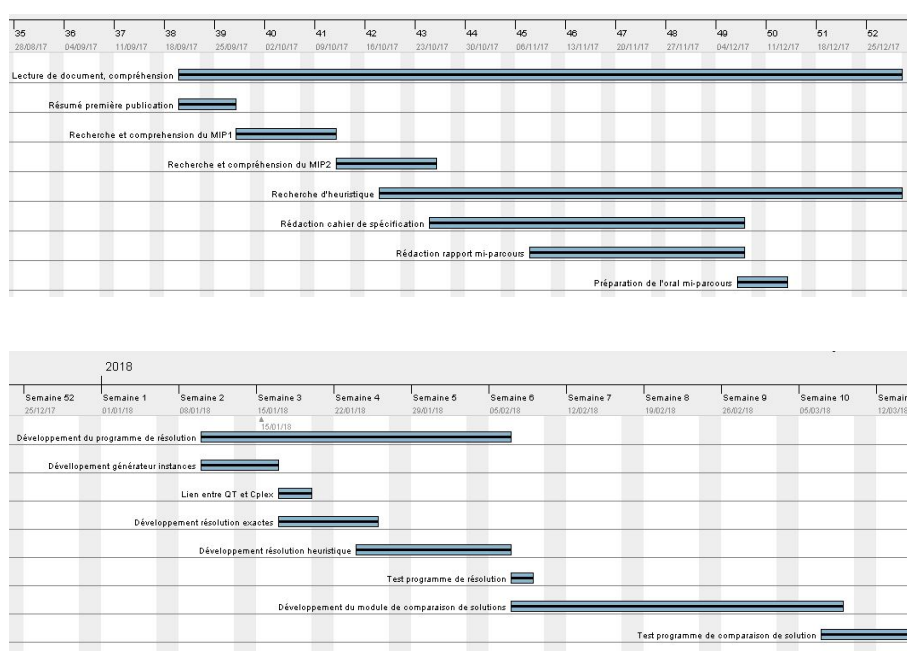
Description de la tâche : Il s'agit ici tester l'ensemble du programme de comparaison de solution. Il sera nécessaire de tester la comparaison de toutes les méthodes et cela nous permettra également d'obtenir des indices sur les performances de chaque méthode.

Estimation de charge : Cette tâche est estimée à 2 jours/homme.

28.16 Planning réel pour la mise en place des tâches

Les tâches décrites plus haut ont été effectuées.

Voici, le planning réel des tâches effectuées aux semestres 9 et 10 :



29 Gestion de projet

29.1 Méthode de Gestion de Projet utilisée

Ce projet consiste à trouver des méthodes de résolutions permettant de résoudre un problème d'affectation de jobs sur m machines où l'objectif est de maximiser le nombre de jobs ordonnancé sur chaque machine. Au cours de la partie recherche de ce projet, deux modèles de résolutions exactes du problème ont été trouvés. De plus, des heuristiques de type glouton ont été proposées pour résoudre le problème de manière plus rapide tout en se rapprochant au maximum de la solution exacte du problème.

L'objectif du projet pour le semestre 10 est de développer une application permettant de résoudre le problème d'affectation à partir de fichiers d'instances. Ces fichiers d'instances, au format txt, devront être créés à l'aide d'un générateur d'instances inclu dans l'application.

Il est également nécessaire de coder les différentes méthodes de résolutions (exactes et heuristiques) pour pouvoir avoir des éléments de comparaison entre ces différentes méthodes (temps d'exécution, solution optimale).

Enfin, il faudra ajouter à l'application des éléments permettant de comparer les différentes méthodes développées précédemment à travers des graphiques ou des métriques qui permettront d'analyser les différentes méthodes de résolution et d'effectuer un comparatif complet.

Pour ce qui est de la gestion de projet, nous avons choisi avec mon encadrant d'utiliser une méthode agile basée sur un modèle itératif. En effet, il nous semblait plus intéressant de développer des fonctionnalités chaque semaine (ou du moins sur une période de temps assez courte) et d'effectuer des réunions hebdomadaires pour faire le point sur ce qui avait été fait. Ces réunions servent avant tout à valider ou non ce qui a été développé et ensuite de proposer de nouvelles tâches à accomplir au cours de la semaine suivante.

Cette méthode de gestion est très adaptée à notre projet. En effet, il est important que le client (ici mon encadrant) participe à l'élaboration du produit final et donne son avis régulièrement sur ce qui a été fait et ce qu'il reste à faire. Il est de plus, beaucoup plus facile de détecter et corriger les éventuelles mauvaises interprétations du développeur par rapport à ce que voulait le client, car elles sont découvertes rapidement. Cette phase de rencontre avec le client est fondamentale, car cela permet de ne pas arriver à la fin du projet avec une application qui ne correspond pas à ce que voulait le client.

29.2 Identification des livrables et tâches

Comme cela a été précisé plus haut, le projet consiste à développer une application permettant de résoudre un problème d'affectation à partir d'instances comprises dans des fichiers txt. À partir de cela différents livrables correspondant à des fonctionnalités de l'application sont à prévoir :

- Un générateur d'instance qui permettra de générer automatiquement des instances avec un nombre de jobs, de ressources et de machines spécifiques. Les instances devront avoir également une horizon de planification (planification sur une journée, une heure ...).
- Des méthodes de résolutions qui permettront de résoudre selon différentes manières notre problème d'ordonnancement. D'une manière plus précise, les différentes méthodes de résolutions attendues sont les suivantes :

- Mip1 : Résolution linéaire en nombre entier indexé temps qui permet de trouver la solution exacte de notre problème (cette fonctionnalité nécessite l'utilisation du solveur Cplex qui devra être intégré à l'application)
- Mip2 : Résolution linéaire en nombre entier indexé jobs qui permet de trouver la solution exacte de notre problème (cette fonctionnalité nécessite l'utilisation du solveur Cplex qui devra être intégré à l'application)
- Une classe Heuristique qui contiendra toutes les méthodes permettant de calculer des solutions exactes d'instances données à travers plusieurs heuristiques.
- Une interface permettant de comparer les différentes méthodes de résolutions développées en amont. Cette partie est fondamentale, car elle correspond à l'objectif initial du PRD : Comparer les méthodes de résolutions d'instances.

29.3 Analyse de faisabilité

Le projet est normalement réalisable si l'on considère les contraintes de temps et de ressources liées à ce projet. Initialement, le projet étant plus axé sur la recherche, j'avais conclu que le développement pouvait commencer au plus tard fin Janvier. Cependant, je me suis vite rendu compte de la charge de travail que demande la partie Développement du projet et j'ai décidé de commencer l'implémentation des fonctionnalités dès début Janvier.

29.4 Analyse de risque

Il existe différentes tâches du projet qui sont dépendantes d'autres tâches et qui de ce fait ne peuvent pas être réalisées si les tâches qui leur sont liées ne sont elles-mêmes pas réalisées. Les livrables qui sont spécifiés plus haut dans ce rapport doivent être réalisés dans l'ordre défini, car chacun d'entre eux est lié au livrable qui le précède. Ainsi, sans générateur d'instances, il n'est pas possible d'implémenter les méthodes de résolutions (car sans données d'entrées, nous ne pouvons pas calculer de solution). De la même manière, il est impossible de mettre en place un outil de comparaison de méthodes de résolutions si les méthodes de résolutions n'ont pas été implémentées auparavant.

29.5 Suivi de projet

Le suivi de l'avancée du projet s'est fait à partir des réunions et des mails de compte-rendu hebdomadaires. Toutes les réalisations pour chaque semaine ont été envoyées par mail chaque jeudi soir et mises à jour sur le dépôt GitHub créé pour ce projet.

Les différentes étapes clés du projet ont été validées par mon encadrant (et également client). Chaque semaine, les différentes tâches effectuées ont été expliquées et les livrables remis. Lorsqu'une étape du projet est franchie, c'est mon encadrant qui confirme que la tâche a bien été réalisée et que je peux continuer les tâches ou livrables suivants.

Les aléas identifiés ont été généralement des problèmes de compréhension d'algorithme ou encore de mauvaises interprétations pour les tâches à effectuer. Généralement, ces problèmes sont identifiés en réunion et sont alors corrigés par la suite.

Pour gérer les aléas, il fut nécessaire de discuter des problèmes rencontrés avec le client et de les résoudre grâce à l'aide de mon encadrant ou encore de proposer des solutions alternatives.

29.6 Outils utilisés pour améliorer la gestion de projet

Dans un premier temps en début de projet, j'ai créé un dépôt Git afin de gérer les versions de mon programme. Il est important de noter que ce n'est pas vraiment un outil de gestion de projet, mais plus de versionning. Cependant, cet outil me permet d'avoir une trace de mon travail chaque semaine et à travers les différents commits que je peut effectuer, je laisse une trace de ce que j'ai pu effectuer entre deux commits.

J'ai également pu utiliser l'outil Trello pour pouvoir visualiser les tâches que j'avais à faire, ce qui était en cours, ce qui était à valider et ce qui avait été fait. J'ai utilisé Trello pour les tâches du semestre 9 et 10.

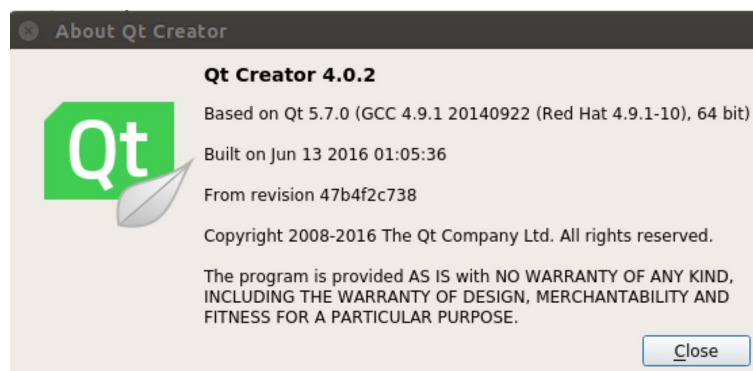
J'ai pu également mettre en place un diagramme de Gantt pour planifier ce qui doit être fait au fur et à mesure du projet. Bien évidemment, comme j'ai utilisé la méthode agile et que les demandes de mon encadrant pouvaient varier ou être différentes de ce que j'avais prévu initialement, le diagramme de Gantt n'a pas été suivi totalement.

30 Documentation d'installation pour le programme

Ce document énumère les différentes manipulations à effectuer afin de pouvoir installer et utiliser le programme développé lors du PRD Ordonnancement et affectation des ressources dans le Cloud Computing.

Pour ce qui est du système d'exploitation à utiliser, nous préconisons une distribution Linux (le développement ayant été fait sur Ubuntu 17.10). L'ensemble de la documentation d'installation se concentrera sur une installation pour une distribution Linux, mais il est normalement tout à fait possible d'effectuer cette installation sur une autre distribution.

Dans un premier temps, il est nécessaire d'installer Qt Creator (le développement a été effectué sur Qt Creator 4.0.2 avec QT 5.7.0). Il est tout à fait possible d'installer Qt Creator via la commande : `sudo apt-get install qtcreator` (cependant, il faudra faire attention que le dépôt Linux soit bien à jour au niveau des versions de Qt). Vous pouvez également télécharger Qt Creator à partir du site web : <https://www.qt.io/download> .



Afin d'obtenir les solutions exactes des instances créées par le programme, il est nécessaire d'obtenir le solveur cplex_216. C'est un dossier contenant Cplex, un solveur développé par IBM. Il s'agit d'un logiciel propriétaire qui doit être acheté pour être utilisé. Pour ce projet, le dossier cplex_216 a été fourni, l'installation sera donc concentrée sur cette version de cplex mais il doit être possible de l'appliquer avec des versions supérieures.

Après avoir copier le dossier cplex dans le répertoire de votre choix, ouvrir le fichier .pro spécifique au projet Qt Creator (ouvrir également le fichier .pro du sous-programme de test car il utilise également le solveur et vous devez faire les mêmes modifications) :



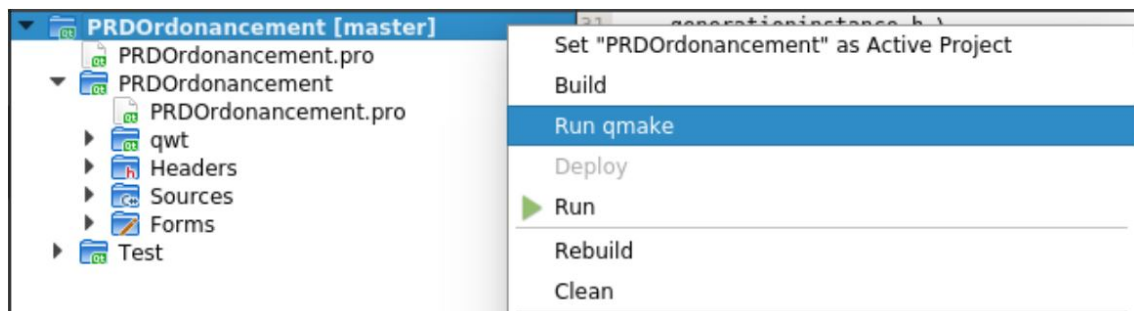
Ensuite, il est important d'ajouter les lignes suivantes dans vos fichiers .pro. Vous devez bien évidemment adapter les chemins spécifiés suivant le répertoire où vous avez choisi de copier cplex_216 :

```
INCLUDEPATH += /home/pierre/cplex_216/cplex/include
INCLUDEPATH += /home/pierre/cplex_216/concert/include

DEFINES += IL_STD

LIBS += -L/home/pierre/cplex_216/cplex/lib/x86-64_linux/static_pic -lliloplex -lconcert -lcplex -lm -lpthread
LIBS += -L/home/pierre/cplex_216/concert/lib/x86-64_linux/static_pic -lconcert
```

Il vous suffit ensuite de faire un qmake (clic droit à la racine du projet : « Run qmake ») et de recompiler pour que vous puissiez utiliser le solveur cplex_216 :



Enfin, le programme utilise une librairie permettant de tracer des courbes directement dans une fenêtre Qt, cette librairie s'appelle Qwt. Pour l'installer, vous devez aller sur le site de qwt : <http://qwt.sourceforge.net/qwtinstall.html> . Prenez bien soin de prendre la version spécifique à votre version de Qt et correspondant à votre distribution. Il vous suffit ensuite de suivre la notice d'installation pour votre distribution.

Ensuite, de la même manière que pour cplex_216, vous devez faire le lien entre Qt Creator et la librairie Qwt. Vous devez encore une fois ouvrir le fichier .pro du projet et ajouter les lignes suivantes (il faut que vous adaptiez les chemins vers Qwt suivant le répertoire où la bibliothèque a été installée) :

```
CONFIG += qwt
INCLUDEPATH += /usr/local/qwt-6.1.3/include
LIBS += -L /usr/local/qwt-6.1.3/lib -lqwt

include(/usr/local/qwt-6.1.3/features/qwt.prf)
```

Après avoir fait cela, faites encore une fois un qmake pour recompiler le fichier .pro et recompiler le programme.

N'oubliez pas d'inclure les bibliothèques dans les fichiers les utilisant :

```
#include <ilcplex/ilocplex.h>

using namespace std;

//Déclaration de l'ensemble des types utilisés par cplex
ILOSTLBEGIN
```

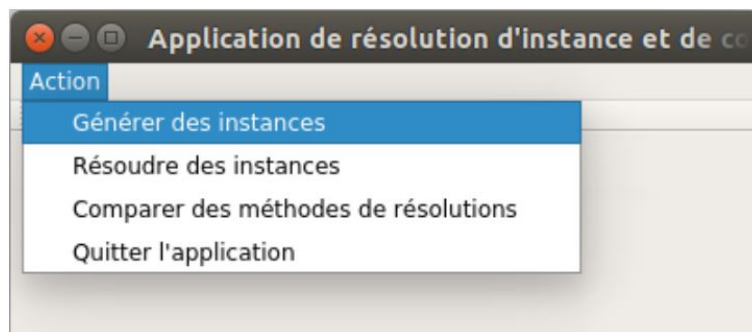
```
#include <qwt.h>
#include <qwt_plot.h>
#include <qwt_plot_curve.h>
#include <qwt_plot_grid.h>
#include <qwt_symbol.h>
#include <qwt_legend.h>
```

31 Documentation d'utilisation du programme :

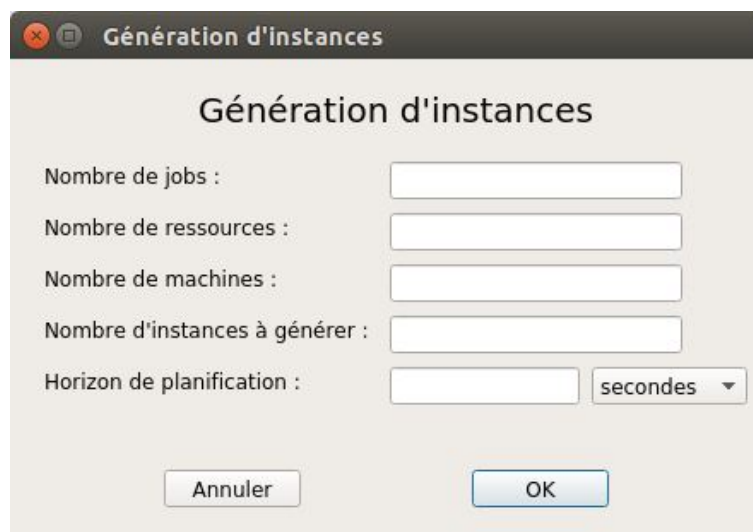
Ce document permet d'énumérer les différentes fonctionnalités de l'application développée lors du PRD Ordonnancement et affectation des ressources dans le Cloud Computing.

Cette application développée dans le cadre d'un projet de recherche et développement permet de générer et de résoudre des instances ayant un certain nombre de jobs avec un nombre r de ressources et qui peuvent être exécutés par m machines.

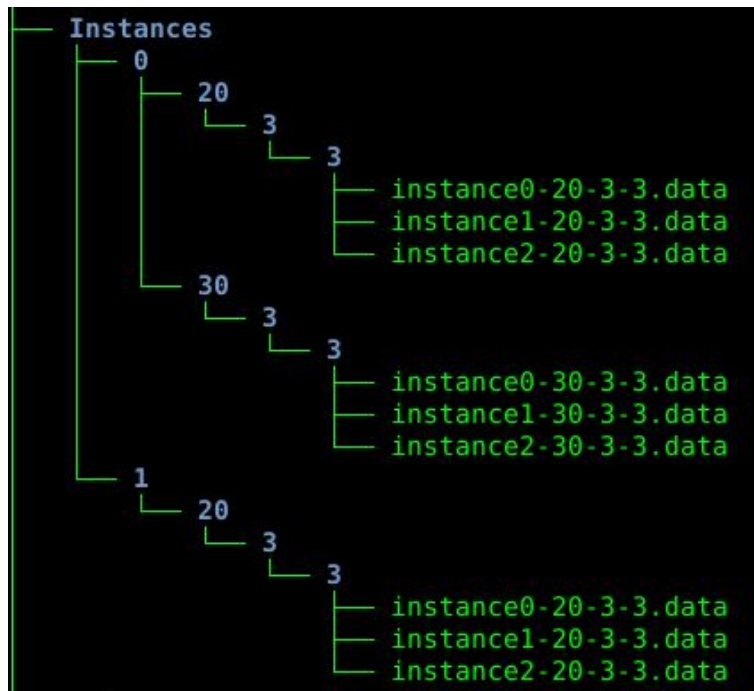
Lors du lancement de l'application, nous pouvons voir qu'il y a un onglet unique qui nous permet de générer des instances, de résoudre des instances, de comparer des méthodes de résolutions ou encore de quitter l'application.



Si nous cliquons sur le champ permettant de générer des instances (« Générer des instances »), nous pouvons spécifier le nombre de jobs, le nombre de ressources et le nombre de machines par instances à générer. Ensuite nous pouvons spécifier le nombre d'instances à générer ainsi que l'horizon de planification pour chaque instance.



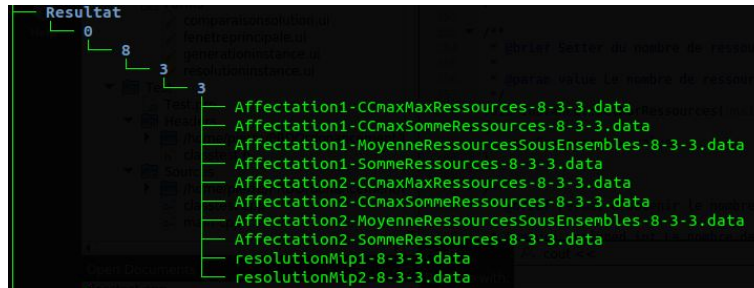
Cela crée une arborescence spécifique dans le dossier courant de l'application (Instances/numeroGeneration/numeroJobs/numeroRessources/numeroMachines) :



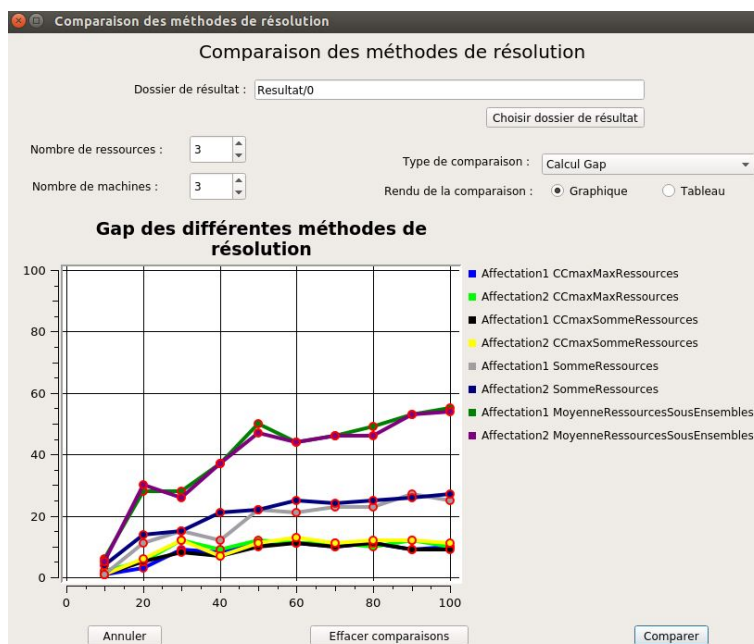
Pour ce qui est de la résolution d'instance, une fenêtre permettant de résoudre des instances est disponible (cliquer sur le champ « Résoudre des instances » de la fenêtre principale). Elle permet de choisir soit un dossier d'instances soit une instance unique à résoudre. Ensuite en cliquant sur le bouton de choix, une fenêtre apparaît, permettant de choisir un dossier ou un fichier. Il est possible de cocher plusieurs méthodes de résolution et de lancer la résolution ensuite.

La fenêtre intitulée 'Résolution d'instances' propose deux options de sélection : 'Dossier d'instances' (sélectionné) et 'Fichier d'instance'. Sous 'Dossier d'instances', il y a un champ de texte pour le dossier et un bouton 'Choisir un dossier'. Le 'Nombre d'agents' est réglé sur '1'. Une section pour 'Agent 1' permet de définir le 'Pourcentage' (actuellement à 100%) avec un bouton 'Valider Pourcentage'. Une case à cocher 'Tout cocher' est présente. Une liste de méthodes de résolution est affichée avec des cases à cocher : 'Resolution exacte indexée temps', 'Resolution exacte indexée jobs', 'CCmax basé sur la somme des ressources de chaque job' (avec sous-options 'Affectation priorisant la machine avec le moins de charge' et 'Affectation machine par machine'), 'CCmax basé sur la ressource maximale de chaque job' (avec sous-options 'Somme des ressources de chaque job' et 'Affectation priorisant la machine avec le moins de charge'), et 'Valeur moyenne des ressources des sous-ensembles maximaux'. Des boutons 'Annuler' et 'Résoudre' sont situés en bas.

Les fichiers d'instances sont alors résolus et les résultats sont stockés dans un fichier spécifique portant le nom de la méthode de résolution, suivi du nombre de jobs, de ressources et de machines. De la même façon que pour la génération d'instance, les fichiers de résultats sont stockés dans une arborescence spécifique (Resultat/numeroGeneration/numeroJobs/nombre-Ressources/ nombreMachines) :



Après avoir généré des fichiers de solutions, vous pouvez lancer la comparaison des différentes méthodes de résolution. Dans la fenêtre principale, cliquer sur le champ « Comparer des méthodes de résolutions ». Une nouvelle fenêtre s'ouvre, elle permet de choisir un dossier de résolution (choisissez alors un des sous-dossiers du dossier Resultat) et de comparer les différentes méthodes de résolution pour l'ensemble des fichiers de résolution contenu dans le sous-dossier. Ensuite, vous devez spécifier le nombre de ressources et de machines pour filtrer les fichiers de résolution. Vous pouvez ensuite choisir le type de comparaison à effectuer et le type de rendu de la comparaison.



Après avoir validé la comparaison, le résultat est affiché dans la fenêtre (soit un graphique, soit un tableau de pourcentage). Vous pouvez alors effacer la comparaison en cliquant sur le bouton « Effacer comparaisons » soit effectuer une nouvelle comparaison qui sera ajoutée en dessous de la première.

32 Plan des différents tests effectués :

Ce document énumère les différents tests effectués sur l'application de génération et résolution d'instance ainsi que de comparaison de méthode de résolution. Cette application a été développée afin de permettre d'étudier un cas d'affectation de jobs à intervalle fixe sur plusieurs machines. Les jobs tout comme les machines possèdent un nombre R de ressources, l'objectif étant que l'exécution des jobs ne consomme pas plus de ressources que ce qui est disponible sur chaque machine.

Cette application développée dans le cadre d'un projet de recherche et développement se doit d'être fiable et efficace. Un sous-projet de test a été créé pour valider les fonctions du projet principal du PRD. Les tests qui ont été effectués sont principalement des tests unitaires, mais ils permettent de se rendre compte de la fiabilité de l'application.

32.1 Tests Unitaires :

Les tests unitaires sont tous présents dans le sous-dossier de test dans la classe "classtest". Comme toute classe c++, elle possède un .h et un .cpp. De plus la librairie Qtest est utilisée pour effectuer les tests unitaires.



Tests de la génération d'instances :

Dans un premier temps, il est important de vérifier en tout premier lieu que la génération des instances se fait correctement. À chaque vérification, un certain nombre d'instances sont générées aléatoirement et nous vérifions ensuite qu'elles sont présentes à l'emplacement que l'on souhaite. Ensuite nous comptons le nombre d'instances présentes dans le dossier afin de vérifier qu'il y a bien le nombre d'instances souhaité. Cette phase de test nous permet de vérifier le nombre d'instances générées. Nous vérifions également le nombre de jobs, de ressources et de machines pour chaque instance en spécifiant notre chemin de recherche des instances. En effet, ce chemin est composé de chacune de ces valeurs.

Tests des méthodes de résolution d'instances :

Nous venons ensuite tester l'une des fonctions principales de notre application : la résolution d'instances. Pour résoudre les instances, différentes méthodes ont été développées et chacune d'entre elles doit être testée afin de vérifier la cohérence des résultats.

- Dans un premier temps, nous vérifions que nos heuristiques donnent des résultats inférieurs ou égaux au nombre de jobs totales de l'instance et ceux quel que soit la méthode d'affectation (deux tests distincts car il y a deux méthodes d'affectation).
- Dans un second temps, nous vérifions que nos méthodes de résolution exactes (permettant de trouver les solutions optimales) donnent également des résultats inférieurs ou égaux au nombre de jobs totales de l'instance.
- Ensuite, nous vérifions que nos deux méthodes exactes retournent le même résultat (la résolution indexée temps Mip1 doit être égale à la résolution indexée jobs Mip2).
- Les résultats des heuristiques sont ensuite comparés aux résultats d'une solution optimale. Nous vérifions que pour chaque instance, les heuristiques donnent des résultats inférieurs ou égaux aux solutions optimales.

Tests de la comparaison des méthodes de résolution :

Enfin, nous venons tester le module de comparaison des méthodes de résolution. Nous testons ici toutes les méthodes de comparaison définies dans le cahier de spécification.

- Nous vérifions que le calcul du gap des solutions trouvées par les heuristiques soit supérieur ou égal à 0 (si il était inférieur à 0 cela signifierait que l'heuristique est meilleur que la résolution exacte).
- Nous vérifions que le pourcentage de solutions optimales pour chaque heuristique est compris entre 0 et 100.
- Enfin nous vérifions que le pourcentage du temps de résolution par rapport à la méthode exacte pour chaque heuristique est positif.

32.2 Tests Fonctionnelles

L'application développée est susceptible de devoir créer et manipuler des instances de tailles variables et qui demandent des temps de traitements différents.

Nous devons donc tester directement avec l'application, la génération et la résolution d'instance de tailles différentes (nombre de jobs, ressources, machines). Cela permettra de confirmer que la taille des instances influe sur le temps de calcul (que cela soit pour les méthodes de résolutions exactes et heuristiques).

Le projet a pour but de comparer différentes méthodes de résolutions, de ce fait nous testons directement l'application en générant des instances et en les résolvant (les instances sont de toute manière générées aléatoirement à chaque fois). Nous prenons des instances de tailles différentes (instances de 10 à 100 jobs avec 3, 6 et 9 ressources et machines pour chaque génération). Nous nous limitons à des instances jusqu'à 100 jobs, car au-delà, les calculs sont longs.

Nous pouvons ainsi, grâce aux informations présentes dans les fichiers de résultats remarquer que les instances avec moins de jobs sont résolues en général bien plus rapidement que celles avec plus de jobs :

Instance:Instances/0/10/3/3/instance0-10-3-3.data Solution status:Mip1 Optimal Value=7 temps écoulé (en secondes):0.11207	Instance:Instances/0/100/3/3/instance0-100-3-3.data Solution status:Mip1 Optimal Value=38 temps écoulé (en secondes):4.73642
Instance:Instances/0/200/3/3/instance1-200-3-3.data Solution status:Optimal Optimal Value=56 temps écoulé (en secondes):345.944	

33 Cahier du développeur

Ce document permet de donner différentes informations liées au développement de l'application réalisé lors du PRD Ordonnancement et affectation des ressources dans le Cloud Computing. Ce programme a pour but de générer et résoudre des instances avec un certain nombre de jobs et plusieurs ressources à ordonnancer sur plusieurs machines. L'application devra également permettre de comparer les méthodes de résolutions d'instance entre elles, d'afficher des graphiques et tableaux de résultats pour la comparaison.

33.1 Langage de programmation utilisé :

Le langage de programmation utilisé lors de ce projet est le C++. Afin de réaliser un programme avec une interface graphique, le framework Qt a été privilégié. L'IDE qui a été utilisé durant le semestre 10 du PRD est Qt Creator 4.0.2 basé sur Qt 5.7.0. Il ne s'agit pas de la dernière version de Qt Creator disponible (actuellement Qt Creator 4.5.2 depuis le 13 mars 2018), il sera donc peut-être nécessaire d'effectuer une mise à jour de l'IDE si le projet est repris.

33.2 Bibliothèques utilisées :

Lors de ce projet, le solveur `cplex_216` a été utilisé afin de pouvoir obtenir des solutions exactes à notre problème pour les instances que nous avons pu générer. Pour l'installation de cette bibliothèque, veuillez vous référer au guide d'installation du projet.

Étant donné que notre programme se doit de pouvoir comparer des méthodes de résolution et donc de pouvoir afficher des graphiques de comparaison, il a été nécessaire d'utiliser une bibliothèque permettant de dessiner des graphiques. La bibliothèque `Qwt` a été choisie, il s'agit d'une bibliothèque permettant de dessiner différents types de graphiques. L'installation de cette bibliothèque est détaillée dans le guide d'installation du projet.

33.3 Convention de nommage :

Pour ce qui est de la convention de nommage, le Camel Case a été utilisé. Il s'agit d'écrire les mots sans espaces entre eux et sans signe de ponctuation. Il y a quelques différences selon ceux que nous avons à nommer.

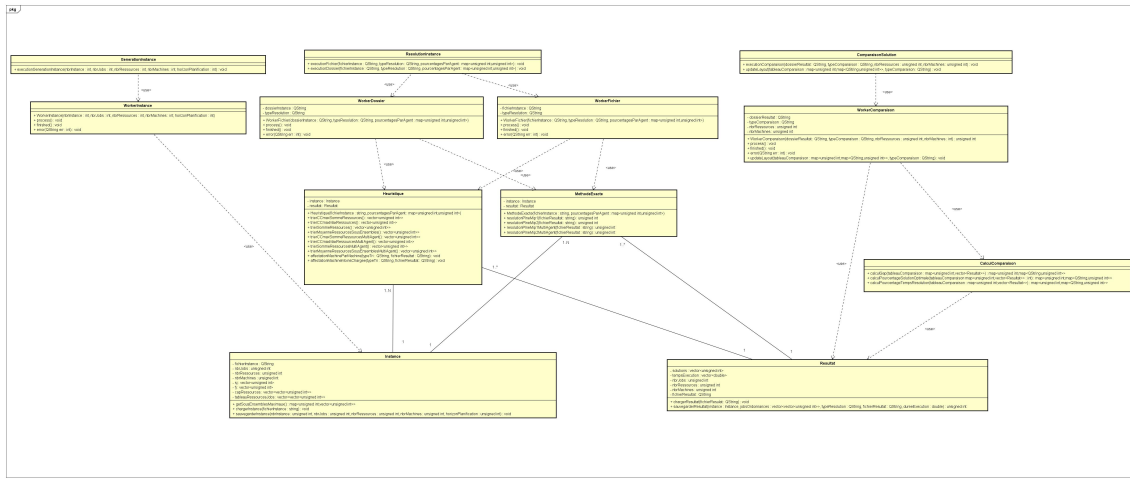
- Classes : Les classes sont nommées en CamelCase avec une majuscule au début du nom (exemple : `ComparaisonSolution`). De ce fait, les constructeurs de classe débutent également par une majuscule.
- Méthodes : Les méthodes sont nommées en lowerCamelCase, ce qui signifie qu'il y a une minuscule pour débiter le mot (exemple : `executionGenerationInstance`)
- Attribut et variables : Les attributs et variables situés dans les classes sont tous nommés en lowerCamelCase.
- Slots et signaux générés par Qt : Les slots et signaux générés par Qt ont une convention de nommage particulière, les mots sont séparés par des underscores (exemple : `on_choisirFichierPushButton_clicked()`, ici `choisirFichierPushButton` est un bouton nommé en lowerCamelCase).
- Élément d'interface graphique : Les éléments d'interface graphique sont nommés en lowerCamelCase. Cependant, ils sont généralement en français et anglais étant donné que les noms d'objet de QT sont en anglais (exemple : `choisirFichierPushButton` qui correspond à un objet de type `PushButton` qui permet de choisir un fichier).

33.4 Documentation :

Afin de générer la documentation pour le programme, la bibliothèque `Doxygen` a été utilisée. Cela nous permet de mettre en place facilement de la documentation en la générant et en la complétant facilement. La bibliothèque va générer la documentation au format html et cela nous permet facilement d'avoir des informations sur les classes et méthodes.

33.5 Structure du système :

Voici le diagramme de classe du système :



Le programme peut donc être séparé en 3 parties :

- Génération d'instances
- Résolution d'instances
- Comparaison des méthodes de résolutions

Chaque partie du programme possède une fenêtre qui lui est dédiée. Chaque fenêtre appelle une classe Worker qui exécute du code à l'intérieur d'un thread (cela empêche la fenêtre principale d'être bloquée lors de l'exécution).

Le générateur d'instance permet de générer des instances en précisant : le nombre de jobs, de ressources et de machines par instances, le nombre d'instances que l'on souhaite générer ainsi que l'horizon de planification maximale. Le Worker lié à la génération d'instances se chargera de mener à bien la génération.

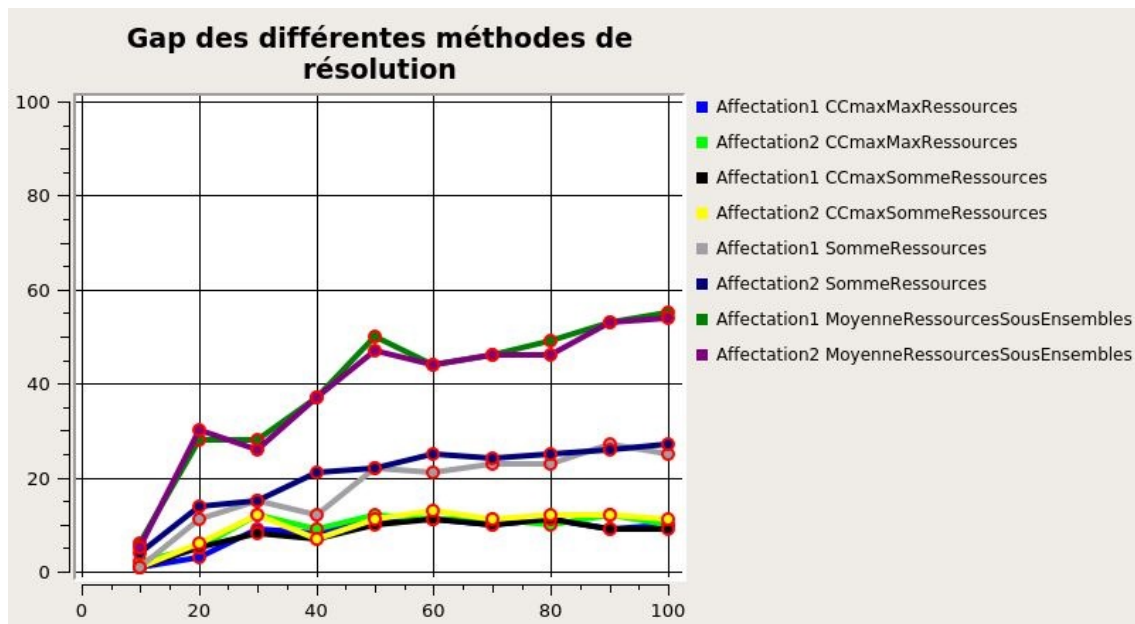
La résolution d'instances se fait à l'aide d'une fenêtre où l'on peut choisir le dossier d'instances (ou le fichier d'instance). Un ensemble de CheckBox est ensuite cochable pour chaque méthode de résolution. C'est cette fenêtre qu'il faudra modifier pour ajouter la possibilité d'avoir d'autres méthodes de résolutions. Il faudra également penser à modifier les classes WorkerDossier et WorkerFichier afin de prendre en compte l'ajout de nouvelles méthodes de résolution.

L'ajout des algorithmes pour les nouvelles méthodes de résolution se fait directement dans les classes prévues à cet effet : `Heuristique` et `MethodeExacte`. Il suffit alors uniquement d'ajouter soit une nouvelle méthode de tri, soit une nouvelle méthode d'affectation ou encore une méthode de résolution exacte supplémentaire :

Heuristique
<pre> - instance : Instance - resultat : Resultat + Heuristique(fichierInstance : string, pourcentagesParAgent : map<unsigned int,unsigned int>) + trierCmaxSommeRessources() : vector<unsigned int>> + trierCmaxMaxRessources() : vector<unsigned int>> + trierSommeRessources() : vector<unsigned int>> + trierMoyenneRessourcesSousEnsembles() : vector<unsigned int>> + trierCmaxSommeRessourcesMultiAgent() : vector<unsigned int>> + trierCmaxMaxRessourcesMultiAgent() : vector<unsigned int>> + trierSommeRessourcesMultiAgent() : vector<unsigned int>> + trierMoyenneRessourcesSousEnsemblesMultiAgent() : vector<unsigned int>> + affectationMachineParMachine(typeTri : QString, fichierResultat : QString) : void + affectationMachineMoinsChargees(typeTri : QString, fichierResultat : QString) : void </pre>

MethodeExacte
- instance : Instance
- resultat : Resultat
+ MethodeExacte(fichierInstance : string, pourcentagesParAgent : map<unsigned int, unsigned int>)
+ resolutionPineMip1(fichierResultat : string) : unsigned int
+ resolutionPineMip2(fichierResultat : string) : unsigned int
+ resolutionPineMip1MultiAgent(fichierResultat : string) : unsigned int
+ resolutionPineMip2MultiAgent(fichierResultat : string) : unsigned int

La comparaison des méthodes de résolution se fait à l'aide d'une fenêtre où l'on peut choisir le dossier contenant les résultats des différentes méthodes de résolution. Il suffit alors de filtrer les résultats selon le nombre de ressources et de machines pour avoir le graphique de comparaison :



De la même façon que pour l'ajout de nouvelles méthodes de résolution, il est possible d'ajouter facilement de nouvelles méthodes de comparaison, pour cela il faut créer de nouvelles méthodes directement dans la classe CalculComparaison :

CalculComparaison
+ calculGap(tableauComparaison : map<unsigned int,vector<Resultat>>) : map<unsigned int,map<QString,unsigned int>>
+ calculPourcentageSolutionOptimale(tableauComparaison map<unsigned int,vector<Resultat>> : int) : map<unsigned int,map<QString,unsigned int>>
+ calculPourcentageTempsResolution(tableauComparaison : map<unsigned int,vector<Resultat>>) : map<unsigned int,map<QString,unsigned int>>



Comptes rendus hebdomadaires

Compte rendu n°1 du 21/09/2017

Bonjour à vous,

J'ai commencé à faire des recherches pour mon état de l'art durant ces deux premiers jours (document en pièce jointe).

Je n'ai pas beaucoup de contenu pour le moment, mais je pense être sur la bonne voie (la littérature étant tous de même assez complexe).

Je n'ai dans mon état de l'art que 3 documents (finalement je n'ai développé que sur 2), mais j'ai lu quelques passages de thèse sur l'ordonnancement parallèle, mais je n'en est pas fait allusion dans le document, car cela n'était pas assez en lien avec le PRD (tout comme le deuxième document de l'état de l'art).

J'aurais besoin de vous voir assez rapidement tout de même pour avoir une meilleure compréhension du projet et des objectifs.

Je vous rappelle également que je ne serais pas disponible le jeudi 28 septembre (la semaine prochaine) de 13h30 à 16h30, car je passe le TOEIC sur le site Dassault.

Merci à vous et bonne soirée,

Pierre FERVAULT DI5

Compte rendu n°2 du 28/09/2017

Bonjour,

Je vous fais parvenir le résumé de la première publication que vous m'aviez envoyé (il reste peu être des fautes je corrigerai mercredi prochain).

J'ai pendant ces deux jours travaillé essentiellement sur le résumé que je vous envoie et également sur mon cahier des spécifications système (du moins le début).

Je n'ai pas eu spécialement de problème cette semaine.

Je continuerai à avancer sur mon cahier de spécification la semaine prochaine.

Si vous avez de la littérature qui pourriez m'être utile, je suis preneur.

Bonne fin de semaine à vous,

Pierre FERVAULT DI5

Compte rendu n°3 du 05/10/2017

Bonjour,

Cette semaine, j'ai travaillé sur mon cahier de spécification et j'ai fait quelques recherches sur le cas multi-machines (1ère référence de la publication mono-machine du PRD).

De plus, j'ai cherché à trouver une expression de programmation linéaire qui correspond au cas multi-machine (je me suis bien sûr inspiré de l'expression mono-machine). Vous pouvez trouver la formulation en pièce jointe.

Si possible, j'aimerais que l'on puisse se voir la semaine pour être bien au courant des documents attendus et ce qu'il faut que je fasse. Je vous avoue que je suis un peu perdu sur ce qu'il faut que je réalise dans cette première partie. Beaucoup de parties du cahier de spécification me posent d'ailleurs problème.

Bonne fin de semaine à vous,

Pierre FERVAULT DI5

Compte rendu n°4 du 12/10/2017

Bonjour à vous,

J'ai continué mon état de l'art sur la publication "Optimal interval scheduling with a resource constraint" qui est la première référence de la publication sur cas mono-machine.

J'ai reçu 2 autres publications de la part de Boukhalfa, dont celle avec le MIP2 (que j'étudierai la semaine prochaine).

Je continuerai mon état de l'art et je me pencherai sur le cas du MIP2 la semaine prochaine.

Bonne fin de semaine à vous,

Pierre FERVAULT DI5

Compte rendu n°5 du 19/10/2017

Bonjour à vous,

Je vous fais parvenir le modèle MIP2 pour le cas multi-machine, Boukhalfa l'a validé tout à l'heure avec moi.

Cette semaine, j'ai essentiellement travaillé sur le MIP2 et lu les documents qui m'ont été transmis et j'ai résumé ce que j'ai retenu afin de réaliser mon état de l'art.

Bonne semaine à vous,

Pierre FERVAULT DI5

Compte rendu n°6 du 26/10/2017

Bonjour à vous,

Cette semaine, j'ai essentiellement travaillé sur mes spécifications et résumé ce que j'ai retenu des publications afin de réaliser mon état de l'art.

Je n'ai pas encore trouvé d'idée précise pour les heuristiques (j'ai résumé celle qui se trouve sur la publication de Computers & Operations Research (Enrico Angelelli ...) mais j'ai un peu de mal pour certaines, je vais continuer à étudier ça).

Dans tous les cas, je vais continuer à étudier les papiers la prochaine fois.

Bonne semaine à vous,

Pierre FERVAULT DI5

Compte rendu n°7 du 09/11/2017

Bonjour à vous,

Suite à la discussion que j'ai eue avec Boukhalifa ce matin, j'ai commencé à tester des heuristiques sur papier (avec l'exemple des jobs proposé dans le cas mono-machine et un autre exemple de 8 jobs)

Je me suis basé sur la publication Optimal interval scheduling with a resource constraint (Enrico Angelelli et al).

Je n'ai pas encore formalisé les heuristiques, je vais en essayer d'autre et les présenterai à Boukhalifa la semaine prochaine.

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°8 du 16/11/2017

Bonjour à vous,

Cette semaine, j'ai cherché des heuristiques et des algorithmes.

- Algorithme Heuristique 1 : On affecte le maximum de jobs sur une première machine et on renouvelle l'opération sur les autres machines)
- Algorithme Heuristique 2 : On affecte chaque job sur la machine ayant le moins de ressources utilisées (entre s_k et f_k), en cas d'égalité, l'ordre lexicographique est utilisé.

Je les ai proposés à Boukhalifa cette après-midi, il y a encore pas mal de choses à améliorer.

Sinon, je vais continuer à chercher et proposer d'autres heuristiques et algorithmes la semaine prochaine.

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°9 du 23/11/2017

Bonjour à vous,

Cette semaine, je me suis concentré sur mon rapport et les spécifications.

J'ai d'ailleurs été voir Monsieur Ragot ce matin pour parler des spécifications.

Il me manque pas mal de choses au final du coup, je fais un bilan de ce qu'il manque.

Il faudra définir plus précisément ce qui devra être fait pour l'implémentation des algorithmes et heuristiques ainsi que les spécifications fonctionnelles.

Monsieur Ragot m'a également demandé quel type de protocole d'évaluation sera utilisé, quels outils sont disponibles pour la génération de solution (si cela existe déjà). Il faudra aussi voir le format des fichiers d'entrée et de sortie (pour le moment, je me base sur ce que j'ai vu sur l'implémentation du MIP1, mais peut-être qu'un autre format pourrait être plus adapté).

Voilà, je pense que je pourrai voir ça la semaine prochaine avec Boukhalifa, en espérant que je puisse finir le cahier de spécification la semaine prochaine étant donné que le rapport doit être rendu dans 2 semaines.

Bonne soirée à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°10 du 01/12/2017

Bonjour à vous,

Désolé de vous faire parvenir mon compte rendu de la semaine si tard, mais la messagerie de l'université était inaccessible hier soir.

Cette semaine, j'ai essentiellement travaillé sur mon rapport (il n'est pas encore terminé, je le continuerai ce week-end).

Je vous le fait parvenir en pièce jointe, il est encore incomplet, mais si vous avez le temps de le lire et que vous trouvez des problèmes ou des éléments qui manquent surtout niveau spécification, je vous serais reconnaissant de m'en informer.

La partie heuristique est actuellement manquante dans mon rapport, je vais tout de même ajouter ce que j'ai fait et proposé à Boukhalfa même si ce n'est pas encore validé (évidemment, je préciserai que ce n'est pas encore validé dans mon rapport).

L'état de l'art est incomplet sur la partie concernant le survey, je l'alimenterai avant de rendre le rapport.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVULT DI5

Compte rendu n°11 du 06/12/2017

Bonjour à tous,

Je vous fais parvenir une version plus à jour du rapport du PRD, la deadline a été reporté au dimanche 10 décembre,

Si vous avez le temps de le lire et de me faire parvenir vos remarques dessus d'ici le 10 décembre, cela m'aiderait beaucoup,

Bonne soirée à vous,

Pierre FERVULT DI5

Compte rendu n°12 du 14/12/2017

Bonjour à vous,

Cette semaine j'ai donc passé la soutenance mercredi et finalisé le diaporama.

Aujourd'hui, j'ai étudié un cas de bin packing à plusieurs dimensions dont voici le lien : <https://people.math.gatech.edu/tetali/PUBLIS/CKPT.pdf>

Je n'en est encore pas retenu grand-chose, mais je continuerai la semaine prochaine en espérant que cet article est pertinent, sinon je chercherai des heuristiques ailleurs.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVULT DI5

Compte rendu n°13 du 21/12/2017

Bonjour à vous,

J'ai continué à étudier des publications sur le bin packing et le fixed interval scheduling mais je n'ai pas de nouvelles solutions à proposer.

https://ac.els-cdn.com/S0377221706003559/1-s2.0-S0377221706003559-main.pdf?_tid=ab9067d0-e620-11e7-9d65-00000aab0f02&acdnat=1513841554_9fb6fe5bb533098332f34e975

<http://ttic.uchicago.edu/~cjulia/papers/bandwidth.pdf>

<https://people.math.gatech.edu/~tetali/PUBLIS/CKPT.pdf>

Je pense que je vais plutôt me concentrer sur les heuristiques que j'ai déjà et proposé d'autres méthodes de tri ou d'affectation à la rentrée, car je n'ai rien trouvé d'intéressant à ajouter cette semaine.

Dans tous les cas, je vous souhaite une bonne fin de semaine et de joyeuses fêtes,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°14 du 11/01/2018

Bonjour à vous,

Cette semaine, j'ai arrêté la lecture de document pour la recherche d'heuristique et j'ai du coup démarré le développement.

Comme vu avec Boukhalfa aujourd'hui, j'enverrai les sources de mon projet chaque semaine (au moins à Boukhalfa) afin que vous puissiez suivre mon avancement.

J'ai choisi d'utiliser QT afin de réaliser les tâches demandées de manière graphique sans pour autant perdre de temps.

Cette semaine, j'ai implémenté le générateur d'instance multi-machine et j'ai commencé à réaliser un squelette de l'application.

La semaine prochaine, j'essayerai de lier CPLEX avec QT, et j'essayerai d'implémenter au moins le MIP1 qui a déjà été effectué en ligne de commande, je verrais si je peux faire le MIP2 ensuite.

Dans tous les cas bonne fin de semaine, j'ai mis mes sources en pièce jointe, ainsi que l'exécutable (uniquement sous Linux),

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°15 du 18/01/2018

Bonjour à vous,

Cette semaine, j'ai continué à implémenter les modèles de MIP et j'ai réalisé le lien entre QT et le solveur Cplex.

Comme vu avec Boukhalfa aujourd'hui, je vais changer l'arborescence lors de la création de fichier d'instance et je vais également faire une arborescence spécifique pour les résolutions des fichiers d'instances.

Je dois donc finir cela la semaine prochaine et commencer l'implémentation des heuristiques définies dans le rapport.

Dans tous les cas, bonne fin de semaine,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°16 du 25/01/2018

Bonjour à vous,

Cette semaine, j'ai modifié mon code afin de réaliser une arborescence correspondant à la structure dont nous avons convenu avec Boukhalfa.

J'ai également bien commencé la mise en place des heuristiques.

J'ai cependant un problème, je pense qu'il y a des fuites mémoires sur le projet. En effet, lorsqu'on lance beaucoup d'instance et surtout de grandes instances il y a de grandes montées en mémoire et l'application crash.

Je vais essayer de modifier l'appli pour qu'elle ne plante plus et je continue l'implémentation des heuristiques.

En pièce jointe, les fichiers de mon projet.

Dans tous les cas, bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°17 du 01/02/2018

Bonsoir à vous,

Cette semaine, j'ai modifié mon code afin d'intégrer les différentes heuristiques que j'avais incluses dans mon rapport de S9.

Il y a donc les méthodes de tri et les méthodes d'affectation d'intégrer dans l'application et on peut facilement choisir ce que l'on veut exécuter.

Les modifications définies durant la réunion de mercredi ont été effectuées également et ont pris un peu plus de temps que prévu.

En pièce jointe, les fichiers de mon projet.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°18 du 08/02/2018

Bonsoir à vous,

Cette semaine, j'ai modifié mon code afin d'avoir un sous-projet permettant de gérer les tests unitaires.

J'ai également modifié mon interface pour prendre en compte des ajouts demandés par Boukhalfa.

J'ai changé un peu la structure du code et j'ai commencé à chercher des bibliothèques pour gérer les graphiques pour la comparaison des différentes méthodes de résolution, ce que je continuerai la semaine prochaine.

En pièce jointe, les fichiers de mon projet.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°19 du 15/02/2018

Bonsoir à vous,

Cette semaine, j'ai ajouté la librairie Qwt permettant de créer des graphiques en QT, cela permettra de faire des graphiques pour la comparaison des méthodes de résolution.

J'ai également vu Monsieur Ramel ce matin pour parler de la qualité de ma modélisation et de mon code. Suite à cela je vais sans doute modifier légèrement la structure de mon code.

J'ai également ajouté de la doc avec Doxygen.

J'ai remarqué un souci que je n'avais pas auparavant. En testant un début de comparaison entre deux méthodes de résolution, j'ai remarqué que mes méthodes heuristiques sont plus lentes que les méthodes exactes (ce qui n'était absolument pas le cas avant, les heuristiques étaient très largement plus rapides que les méthodes exactes). Je vais donc tout d'abord essayer de résoudre ce problème, car il est très embêtant.

La semaine prochaine, je vais continuer mon implémentation de la comparaison de solution.

En pièce jointe, les fichiers de mon projet.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°20 du 22/02/2018

Bonjour à vous,

Cette semaine j'ai réglé le souci de ralentissement des heuristiques et continué à utiliser la bibliothèque QWT.

J'ai également avancé sur mon rapport de S10. Il faudra que l'on voie avec Boukhalfa comment mettre en place la comparaison des méthodes de résolution.

En pièce jointe, les sources de mon projet.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°21 du 08/03/2018

Bonjour à vous,

J'ai pu obtenir un programme permettant de comparer les différentes méthodes de résolution que j'avais pu mettre en place.

J'ai également commencé à intégrer le fait de pouvoir résoudre des instances avec une approche multi-agent (les méthodes seront vues avec Boukhalfa la semaine prochaine).

J'ai également fait du ménage dans le code pour prendre en compte ce que Monsieur Ramel voulait pour la qualité.

J'avancerai sur mon rapport ainsi que mes interprétations au vu des résultats que j'ai pu obtenir suite à la résolution.

En pièce jointe, les sources de mon projet.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°22 du 16/03/2018

Bonjour à vous,

Cette semaine, je me suis avancé sur les différents documents nécessaires pour l'évaluation de la qualité du code ainsi que le rapport.

J'ai ajouté la possibilité de résoudre des résolutions multi-agents (les algorithmes ne sont pas implémentés mais les fonctions existent).

Je continuerai la rédaction des documents la semaine prochaine,

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°23 du 22/03/2018

Bonjour à vous,

J'ai pu avancer sur la rédaction des différents documents et commencer à me préparer pour l'évaluation de la qualité.

Je finaliserai le rapport et les documents la semaine prochaine.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5

Compte rendu n°24 du 29/03/2018

Bonjour à vous,

Cette semaine, j'ai travaillé principalement sur mon rapport et le powerpoint pour la soutenance de mercredi prochain.

J'ai également rendu mon code à Boukhalfa et nous avons installé les éléments qu'il fallait sur sa machine.

Je vous expliquerai mon travail lors de la soutenance de mercredi prochain et je rendrai mon rapport avant mardi.

Bonne fin de semaine à vous,

Cordialement,

Pierre FERVAULT DI5



Bibliographie

- [1] Antoon W.J. Kolen et AL. « Interval Scheduling : A Survey ». In : *Wiley InterScience* (2007).
- [2] Patrick MARTINEAU BOUKHALFA ZAHOUT Ameer SOUKHAL. « Fixed jobs scheduling on a single machine with renewable resources ». In : *Mista* (2017).
- [3] Carlo Filippi ENRICO ANGELELLI Nicola Bianchessi. « Optimal interval scheduling with a resource constraint ». In : *Computers & Operations Research* (2014).

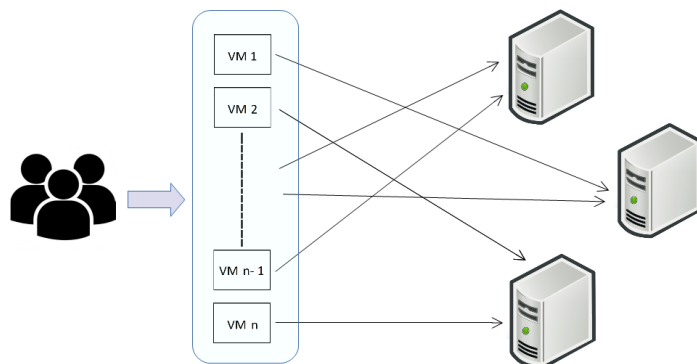
PRD Ordonnancement et affectation des ressources dans le Cloud Computing

Pierre Fervault

Encadrement : Boukhalfa Zahout, Ameer Soukhal et Patrick Martineau

Objectifs

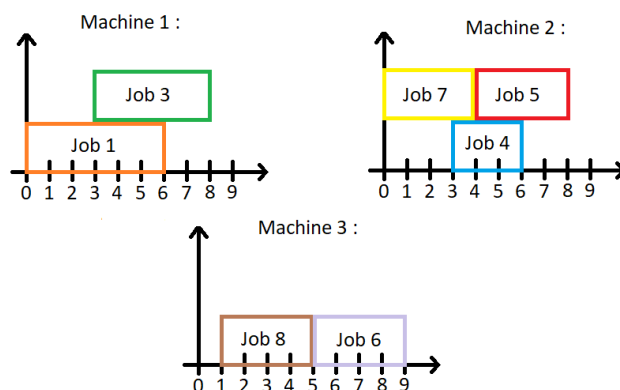
Ce projet a pour but de proposer des algorithmes permettant l'ordonnancement de jobs fixés dans le temps, possédant plusieurs ressources chiffrées sur des machines possédant le même nombre de ressources, mais en quantités limitées. L'objectif est de minimiser le nombre de jobs rejetés.



Objectif : Affecter le maximum de jobs sur des machines

Mise en œuvre

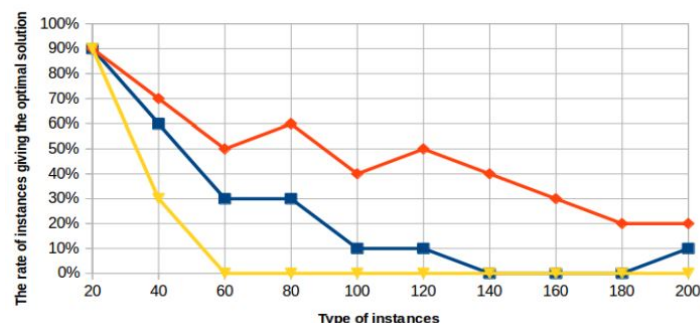
Après avoir trouvé des algorithmes permettant d'ordonnancer les jobs, il faudra les implémenter pour obtenir des solutions exactes et des heuristiques. Il sera important d'implémenter des solutions efficaces et de comparer les méthodes de résolution entre elles.



Exemple d'ordonnancement avec 8 jobs et 3 machines, seuls 7 jobs sont affectés

Résultats attendus

Il est important d'obtenir des solutions rapprochées les plus proches possible des solutions optimales. Le problème étudié étant NP-Difficile, il est évident que l'implémentation d'heuristiques devra permettre d'obtenir des résultats quasi optimaux avec un temps de calcul modeste.



Les solutions apportées par les heuristiques devront être les plus proches possible des solutions optimales

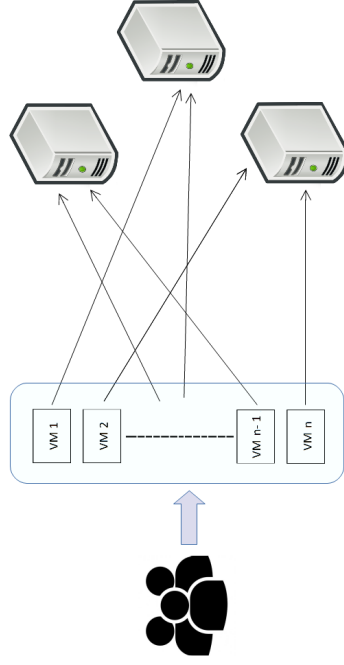
PRD Ordonnancement et affectation des ressources dans le Cloud Computing

Pierre Fervault

Encadrement : Boukhalfa Zahout, Ameer Soukhal et Patrick Martineau

Objectifs

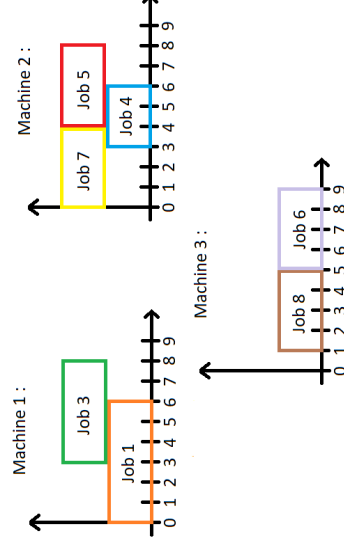
Ce projet a pour but de proposer des algorithmes permettant l'ordonnement de jobs fixés dans le temps, possédant plusieurs ressources chiffrées sur des machines possédant le même nombre de ressources, mais en quantités limitées. L'objectif est de minimiser le nombre de jobs rejetés.



Objectif : Affecter le maximum de jobs sur des machines

Mise en œuvre

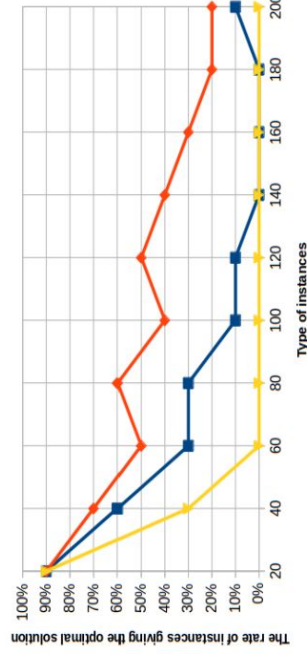
Après avoir trouvé des algorithmes permettant d'ordonner les jobs, il faudra les implémenter pour obtenir des solutions exactes et des heuristiques. Il sera important d'implémenter des solutions efficaces et de comparer les méthodes de résolution entre elles.



Exemple d'ordonnement avec 8 jobs et 3 machines, seuls 7 jobs sont affectés

Résultats attendus

Il est important d'obtenir des solutions rapprochées les plus proches possible des solutions optimales. Le problème étudié étant NP-Difficile, il est évident que l'implémentation d'heuristiques devra permettre d'obtenir des résultats quasi optimaux avec un temps de calcul modeste.



Les solutions apportées par les heuristiques devront être les plus proches possible des solutions optimales

PRD Ordonnancement et affectation des ressources dans le Cloud Computing

Résumé

L'objectif de ce document est de récapituler la phase de recherche effectuée au Semestre 9 pour le PRD Ordonnancement et affectation des ressources dans le Cloud Computing. Il est composé de la partie spécification système, de l'état de l'art et des éléments d'analyse et de conception obtenus pendant le projet.

Mots-clés

Ordonnancement, Parallélisation

Abstract

The aim of this document is to recapitulate the research state of the semester 9 for the PRD Scheduling and ressources affectation in Cloud Computing. This report is composed of the system specification part, the art state and the analyze and conception element obtain during the project.

Keywords

Scheduling, Parallelization

Tuteurs académiques

Boukhalfa ZAHOUT
Ameur SOUKHAL
Patrick MARTINEAU

Étudiant

Pierre FERVULT (DI5)