École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport Final de PFE

Méthodes approchées pour la résolution d'un problème d'ordonnancement avec travaux interférants

Encadrants
Faiza Sadi
faiza.sadi@univ-tours.fr
Ameur Soukhal
ameur.soukhal@univ-tours.fr

Étudiant
Baptiste Mille
baptiste.mille@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 5 mai 2014

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je souhaite remercier mon encadrante Sadi Faiza pour ses conseils et la disponibilité qu'elle aura eu tout au long de ce projet. De plus je souhaite aussi remercier Ameur Soukhal pour ses conseils notamment en début de projet.

Introduction

Ce document présente le travail effectué lors du PFE[1] réalisé durant la dernière année du cursus d'école d'ingénieur universitaire polytechnique Tours. L'intitulé de ce PFE est : *"Méthodes approchées pour la résolution d'un problème d'ordonnancement avec travaux interférants"*.
Celui-ci, proposé par l'équipe Ordonnancement et Conduite (OC) au Laboratoire Informatique de l'école, a été encadré par Faiza Sadi et Ameur Soukhal qui représentaient la MOA. Le projet a été réalisé par Baptiste Mille, élève ingénieur DI5 Polytech Tours qui représentait la MOE.

1. Projet de Fin d'Études

Présentation du projet

2.1 Contexte

2.1.1 Ordonnancement monocritère

Un problème d'ordonnancement monocritère consiste à organiser des tâches, en respectant des contraintes temporelles et de ressources dans le but d'optimiser une fonction objectif.

Exemple : $P_m||C_{max}$ (n travaux à ordonnancer sur m machines, le critère à minimiser est le maximum des dates de fin d'exécution des travaux)

2.1.2 Ordonnancement multicritère

Un problème d'ordonnancement multicritère diffère d'un monocritère en optimisant plusieurs fonctions objectifs au lieu d'une seule. La qualité de la solution est donc mesurée par plusieurs critères.

Exemple : $P2||C_{max}, \sum U_j$ (n travaux à ordonnancer sur m machines. Le premier critère à minimiser est le maximum des dates de fin d'exécution des travaux. Le second est de minimiser la somme des travaux en retard[1])

2.1.3 Ordonnancement avec travaux interférants

Nous nous intéressons dans ce projet à une classe particulière des problèmes d'ordonnancement multicritères. Ces problèmes sont appelés dans la littérature, ordonnancement avec travaux interférants. Dans leur formulation, plusieurs agents sont considérés, tel que chaque agent possède un sous-ensemble de travaux et une fonction objectif. Un critère global est aussi considéré (agent global) afin de mesurer la qualité de l'ordonnancement appliqué sur la totalité des travaux. Celui-ci est fixé suivant la notation à trois champs des problèmes d'ordonnancement présentée dans « Multicriteria scheduling [1] ». On note ces problèmes par $\alpha|\beta|\gamma$ tel que :

- α est représentatif de l'environnement machine
- β les contraintes
- γ critères d'optimalité du problème ($\gamma = f^0, ..., f^k$. Avec f^0 le critère de l'agent 0 et f^k le critère de l'agent k)

Si uniquement deux agents sont considérés, alors on note f^A l'agent global et f^B l'autre agent. Quand il s'agit d'environnement à machines parallèles, ces problèmes s'avèrent être NP-difficiles [4].

1. Un travail est dit en retard si, sa date de fin de réalisation dépasse sa date de fin souhaitée.

2.2 Objectifs

Dans cette étude nous considérerons plusieurs machines identiques et deux agents A,B. L'agent global A veut minimiser son C_{max}, quant à l'agent B, il cherche à minimiser le nombre de travaux en retard $\sum U_j$ (ainsi que le problème inverse : A veut minimiser $\sum U_j$ et B veut minimiser C_{max}). Avec U_j, une fonction booléenne qui prend 1 si le travail est en retard, 0 sinon.

Selon la formulation à trois champs $\alpha|\beta|\gamma$ des problèmes d'ordonnancement, les problèmes sujets de ce projet se notent :

$$Pm|d_i|C^A_{max}, \sum U^B_i$$
$$Pm|d_i|\sum U^A_i, C^B_{max}$$

Étant donnée la complexité du problème $Pm||C_{max}$ et de $Pm||\sum U_j$ (problèmes NP-difficiles), la minimisation des deux critères à la fois est donc aussi NP-difficile. Une méthode exacte serait coûteuse en temps machine. Dans ce travail nous allons favoriser les méthodes heuristiques. Ces méthodes ont un meilleur rapport qualité/temps de calcul que les méthodes exactes, même si elles ne garantissent pas l'optimalité. Néanmoins elles fournissent rapidement une solution dite "approchée".

On est dans un cas multicritère. Nous allons chercher l'ensemble des solutions non dominées ou un ensemble représentatif de celui-ci : front de Pareto. Ce front sera comparé à celui retourné par une méthode exacte. Puisque nous nous intéresserons à l'énumération de front de Pareto nos deux problèmes peuvent s'écrire de cette manière :

$$Pm|d_i|P(C^A_{max}, \sum U^B_i)$$
$$Pm|d_i|P(\sum U^A_i, C^B_{max})$$

2.3 Méthodes de résolution

2.3.1 Méthode exacte

Une méthode exacte permet de trouver une solution optimale à un problème donné. Toutefois, ces méthodes peuvent devenir rapidement couteuses en temps d'exécution, notamment pour les problèmes NP-difficiles. En effet, le temps de traitement et la complexité du problème sont généralement liés (plus c'est complexe, plus le temps d'exécution sera important). Ci-dessous, quelques méthodes exactes parmi les plus connues :

- Procédure par Séparation et Évaluation
- Programmation dynamique
- Programmation par contraintes
- Programmation linéaire

2.3.2 Méthode approchée

Les méthodes approchées (i.e. Heuristiques) permettent de trouver de manière rapide une solution réalisable à un problème donné. Cependant cette solution n'est pas forcément la solution optimale.

L'objectif d'une heuristique est donc de trouver une solution la plus proche possible de celle d'une méthode exacte tout en étant plus rapide. La qualité d'une méthode approchée va donc se calculer par rapport à l'écart obtenu entre sa solution et l'optimale. Plus ce résultat est proche de la solution optimale, meilleure est l'heuristique.

Il existe plusieurs classes d'heuristiques. Celles-ci sont définies par leur fonctionnement.

Déterministes VS Stochastiques

On dit d'une heuristique qu'elle est déterministe si elle ne fait pas appel au hasard. Pour chaque exécution sur un même jeu de données, on obtiendra le même résultat.

On dit d'une heuristique qu'elle est stochastique si elle fait appel au hasard. Pour chaque exécution sur un même jeu de données, on n'obtiendra pas forcément le même résultat.

Construction VS Amélioration

On dit d'une heuristique qu'elle est de construction si son exécution part de l'instance du problème pour construire une solution réalisable.

On dit d'une heuristique qu'elle est d'amélioration si son exécution part d'une solution réalisable et l'améliore par exploration du voisinage.

Solution VS Population

On dit d'une heuristique qu'elle est à la base de population si elle part/construit plusieurs solutions.

Quelques heuristiques

Heuristiques	Déterministes	Stochastiques	Construction	Amélioration	Solution	Population
Algorithme de liste	X		X		X	
Colonies de fourmis		X	X			X
Recherche locale	X			X	X	
Algorithme Génétique		X		X		X
Tabou	X			X	X	
Recuit Simulé		X		X	X	

2.4 L'approche ε-contrainte

Dans un premier temps, nous considérerons l'approche ε-contrainte. Par cette approche, nous chercherons une solution non-dominée dite solution optimale de Pareto. Il s'agit de chercher une solution minimisant le critère de l'agent A tout en bornant supérieurement la valeur de la fonction objectif de l'agent B. Les problèmes se notent donc :

$$Pm|d_i, \sum U_i^B \leq Q|C_{max}^A$$
$$Pm|d_i, C_{max}^B \leq Q|\sum U_i^A$$

Ces problèmes d'ordonnancement retournent une solution optimale de Pareto. Afin d'obtenir un front de Pareto il faudra faire varier la valeur Q en suivant l'algorithme ci-dessous :

Algorithm 1 Approche ε-contrainte avec $f^A = C_{max}$ et $f^B = \sum U_j$

```
Q = UB
while Q ≥ LB do
    Résoudre le problème
    if Pas de solution then
        break ;
    else
        noter(α, β) = (∑ U_j^B, C_max^A) la solution retournée.
    end if
    Poser Q = Q − 1
    S = S ∪ (α, β)
end while
Retourner "S"
```

2.5 Les algorithmes génétiques

Dans un second temps, nous considérerons la résolution de notre problème par l'application d'un algorithme génétique.

2.5.1 Description de la méta-heuristique [3]

Les algorithmes génétiques (AG) sont des algorithmes stochastiques qui se basent sur une simulation du processus de sélection naturelle : les individus les plus forts d'une espèce ont plus de chance de survivre que les plus faibles. Ainsi, afin de reproduire ce phénomène, ces algorithmes utilisent un ensemble de solutions candidates appelées « population d'individus ». Un individu est une solution du problème à résoudre. Pour pouvoir évaluer un individu, on lui attribue une fonction d'adaptation (fitness) qui va permettre de calculer sa qualité.

Dans notre population d'individus, on va sélectionner ceux qui nous paraissent les plus intéressants. Ensuite, ils vont créer des enfants qui pourront subir des mutations et croisements afin de donner une nouvelle population d'individus. On continue, ainsi de suite, jusqu'à atteindre un certain nombre de générations, un individu avec un fitness en dessous d'une valeur mise en paramètre ou un autre critère d'arrêt. Le but espéré est de trouver à chaque nouvelle génération, un individu meilleur pour obtenir finalement un individu proche de la perfection souhaitée.

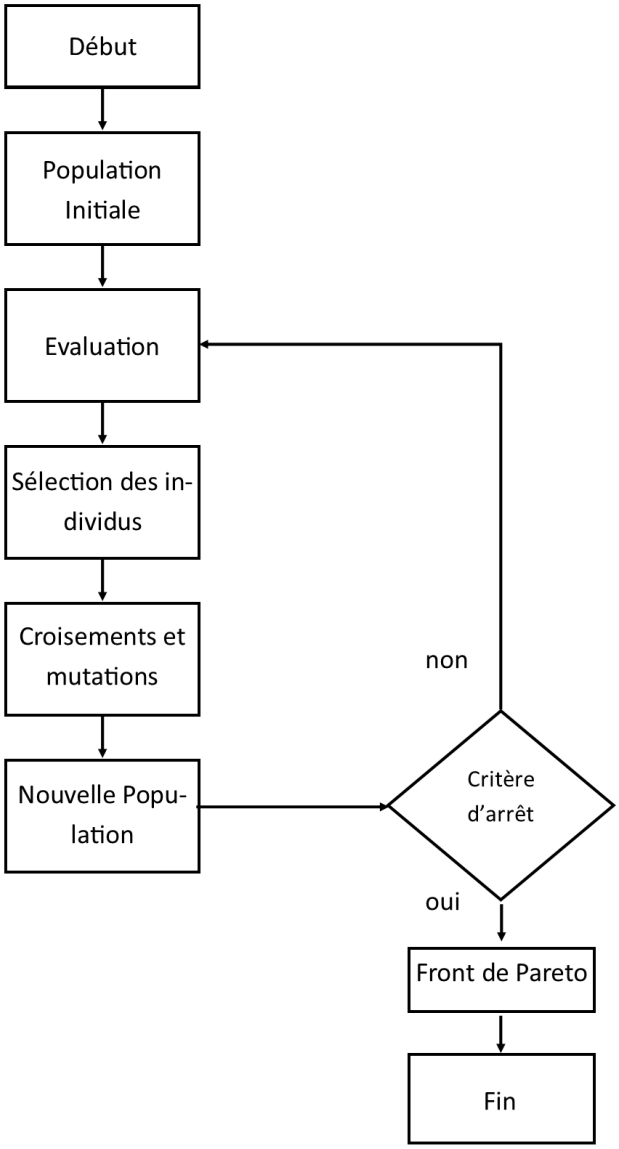


FIGURE 2.1 – Organigramme de l'algorithme génétique

2.5.2 Sélection

Tout d'abord, on part d'une population avec un certain nombre d'individus. Cette population est la première génération de notre algorithme. Dans celle-ci, on va choisir un certain nombre d'individus afin qu'ils deviennent les parents de notre future génération. Il existe plusieurs types de sélection.

L'élitisme

Cette sélection est "la plus simple". Elle va sélectionner les n premiers individus possédant le meilleur résultat de la fonction d'adaptation (fitness) dans la population.

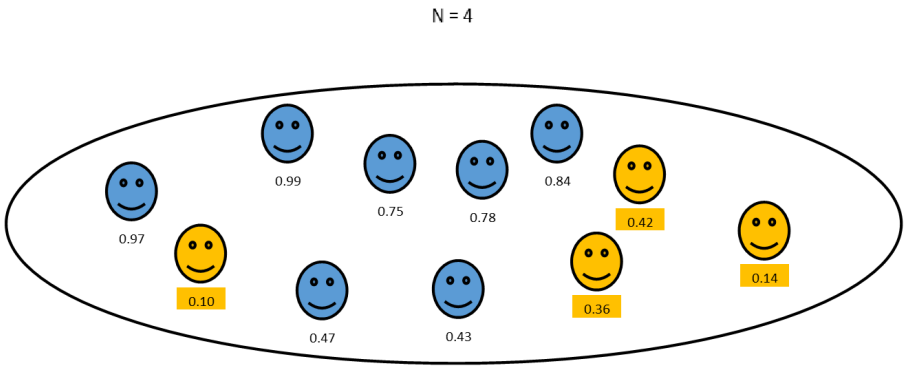


FIGURE 2.2 – Elitisme

Roue

Ici chaque individu à une chance d'être sélectionné pour devenir l'un des futurs parents. Toutefois, un individu possédant une meilleure fonction d'adaptation (fitness) aura plus de chance d'être sélectionné. La probabilité qu'un individu puisse être sélectionné est la suivante :

$$P(individu) = \frac{Fitness(invidivu)}{\sum_{individu=1}^{individuMax} Fitness(individu)}$$

Tournoi

La méthode ici consiste à sélectionner des sous-groupes de k individus aléatoirement dans la population. Puis de choisir le meilleur de chaque sous-groupe pour former les parents.

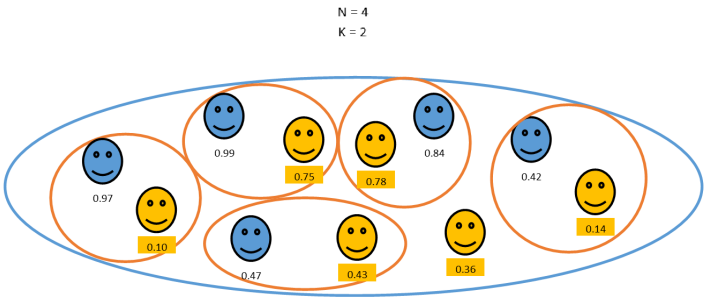


FIGURE 2.3 – Tournoi Binaire (k=2)

2.5.3 Croisement

À partir d'un père et d'une mère, on va créer un enfant qui va subir un croisement de ses deux parents. Le croisement consiste à sélectionner pour chacun des caractères de l'enfant, celui du père ou celui de la mère. Prenons un petit exemple de croisement humain :

Caractère	Père	Mère	Enfant
Couleur des yeux	Marron	Bleu	Marron
Forme des yeux	Amande	Rond	Amande
Couleur des cheveux	Brun	Blond	Blond
Type de cheveux	Lisse	Frisé	Frisé
Sexe	Masculin	Féminin	Masculin

On peut aussi tout à fait imaginer créer un nouvel enfant aléatoirement, et que chacun de ses gènes possède une probabilité de prendre soit celui du père, celui de la mère ou de garder le nouveau gène.

2.5.4 Mutation

La mutation concerne l'évolution des caractères de l'enfant. Elle permet de ne pas bloquer sur un optimum local. C'est une modification faible et avec une petite probabilité d'apparition. Si nous reprenons le tableau des croisements, on peut muter les caractères de l'enfant :

Caractère	Père	Mère	Enfant avec mutation
Couleur des yeux	Marron	Bleu	Marron-Vert
Forme des yeux	Amande	Rond	Amande
Couleur des cheveux	Brun	Blond	Blond
Type de cheveux	Lisse	Frisé	Frisé
Sexe	Masculin	Féminin	Masculin

On a eu ici une mutation sur la couleur des yeux de l'enfant. Si on travaillait sur un codage binaire, cette mutation pourrait être le passage d'un bit à 1 au lieu de 0 par exemple.

2.6 Réalisation des développements

Durant ce projet, l'ensemble des développements est réalisé en java 7. Au final, trois programmes auront été créés :

- un générateur d'instances
- un comparateur de fronts de Pareto
- un solver pour notre problème d'ordonnancement

L'environnement utilisé est Eclipse. Afin de pouvoir stocker les résultats du comparateur de fronts de Pareto dans des fichiers Excel, la librairie externe JExcelApi est utilisée.

FIGURE 2.4 – Eclipse

Travail réalisé

3.1 Compréhension du projet

La première étape de ce projet a consisté à comprendre celui-ci et ses objectifs. Pour cela des réunions ont eu lieu avec la MOA afin que celle-ci exprime son besoin et ses attentes. De plus diverses lectures (conseillées par la MOA) ont aussi permis de mieux comprendre le problème de ce projet et de voir ce qui se faisait dans la littérature.

Un rapport a été rendu sur la compréhension du projet et sur la compréhension du travail a effectué tout au long de l'année par l'étudiant. Ceci dans le but d'être en accord sur ces points.

3.2 Rédaction du cahier de spécifications

Une fois le problème et les attentes des encadrants compris pour ce PFE, le cahier de spécifications a été créé. Celui-ci avait pour but de présenter et expliciter, dans un premier temps, les différentes caractéristiques du projet : contexte, objectif, environnement et fonctionnalités. Puis dans un second temps, le plan de développement de ce projet : découpage du projet en tâches et planning associé.

3.3 Comparateur de fronts de Pareto

Afin de pouvoir comparer les résultats des méthodes approchées aux méthodes exactes, on a développé un programme permettant de comparer les fronts de Pareto obtenus. Ce programme a pour but d'évaluer l'efficacité des méthodes approchées.

Rappelons qu'une solution (x,y) est dite Pareto optimale ou non-dominée s'il n'existe pas de solution (x',y') tel que : $x \geq x' \ \& \ y > y' \ || \ x > x' \ \& \ y \geq y'$.

3.3.1 Critères de comparaison

Temps de calcul

Il est important de comparer les temps d'exécution entre les deux méthodes. En effet, une méthode heuristique doit trouver une solution réalisable plus rapidement qu'une méthode exacte.

Nombre de solutions

Le nombre de solutions de Pareto trouvées nous donne un avis sur l'étendue du front de Pareto de chaque méthode. Il est intéressant d'avoir un front diversifié afin d'avoir un plus grand nombre de choix (hormis si on trouve une solution idéale).

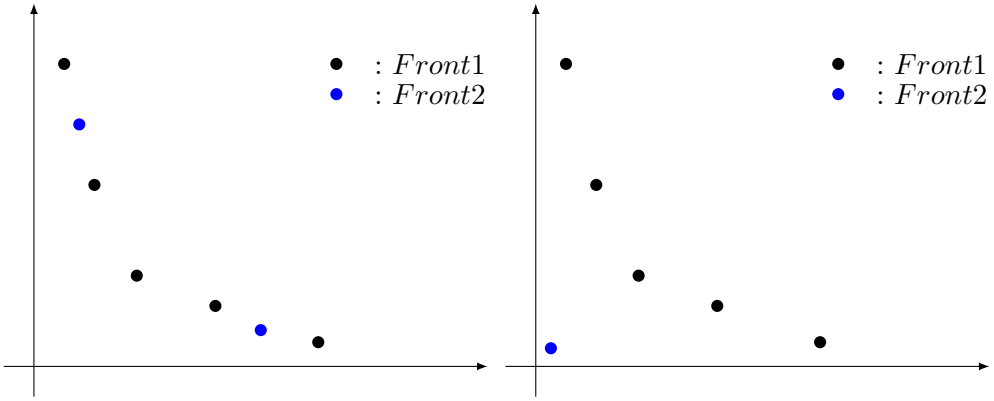


FIGURE 3.1 – Comparaison Front1 avec Front2

Sur la figure de gauche, Front1 est plus intéressant que Front2, car il possède plus de solutions de Pareto et il n'est pas dominé par Front2. Cependant, sur la figure de droite, bien que Front1 possède 5 fois plus de solutions de Pareto, il est dominé par Front2 qui possède une unique solution de Pareto. Front2 est donc plus intéressant que Front1.

Pourcentage de solutions exactes trouvées

Il est important de savoir combien de solutions de Pareto exactes la méthode approchée a réussi à trouver, car les mesures de distance sont corrélées avec le nombre de solutions.

On a donc $PESF$ [1] $= nbESF$ [2] $/nbES$ [3].

Moyenne des plus courtes distances entre les fronts

Cette comparaison calcul pour chaque solution de Pareto de Front2, la plus courte distance par rapport à une solution de Front1 puis, en fait la moyenne. Cette métrique permet d'avoir une idée de la distance qui sépare les deux fronts. Plus cette distance est courte, plus notre méthode approchée est bonne.

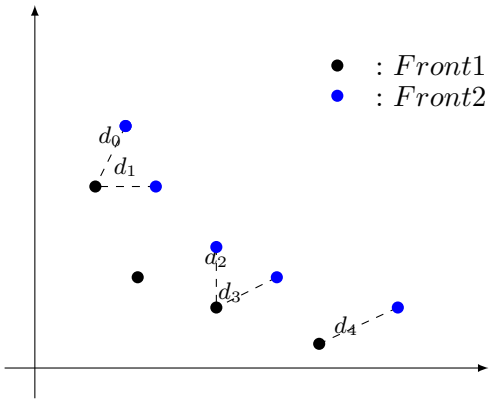


FIGURE 3.2 – Moyenne des distances les plus courtes entre les points du Front1 et ceux du Front2

On a donc ASD [4] $= (\sum_{i=0}^{n} d_i)/n$, telle que d_i est la plus petite distance qui sépare la solution i du Front

1. PESF : Percent of Exact Solution Found
2. nbESF : number of Exact Solution Found
3. nbES : number of Exact Solution
4. ASD : Average Shortest Distance

2 par rapport à une solution du Front 1.

Moyenne des projections orthogonales

Cette deuxième métrique donne plus ou moins la même information que la première. En effet elle permet d'avoir une idée sur la distance séparant les deux fronts. La présence de deux métriques indiquant la même chose est faite dans l'optique que ce programme sera utilisé par d'autres personnes par la suite.

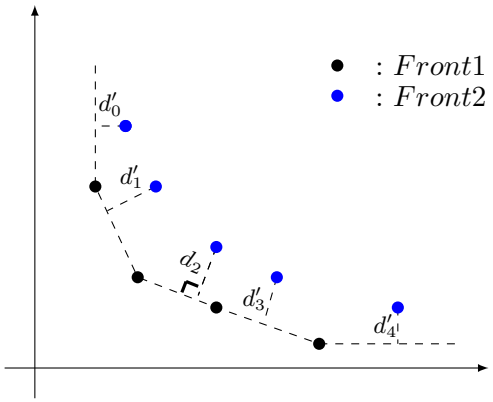


FIGURE 3.3 – Moyenne des distances les plus courtes entre les points du Front1 et le Front2

On a donc OP [5] $= (\sum_{i=0}^{n} d'_i)/n$ telle que d'_i est la distance la plus courte entre la solution i et le front 1.

Moyenne des distances entre les points d'un front

Cette métrique est là pour nous montrer si les solutions du front de Pareto sont concentrées ou étalées. On préfèrera un front qui possède des solutions étalées à un front possédant des solutions concentrées.

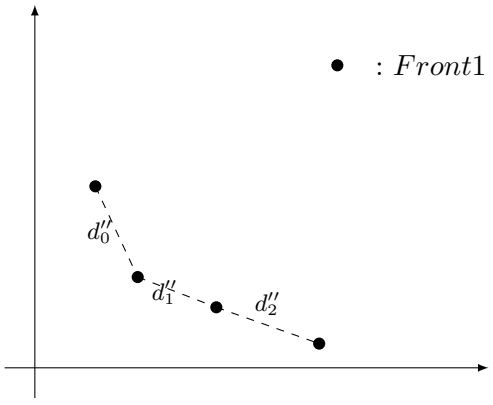


FIGURE 3.4 – Moyenne des distances entre les points d'un front

On a donc $ADPF1$ [6] $= (\sum_{i=0}^{n-1} d''_i)/(n-1)$ telle que d''_i est la distance séparant la solution i de la solution $i+1$

5. OP : Average Orthogonal Projection
6. ADPF1 : Average Distance Points Front1

Dominance

Une fonction de dominance est aussi présente. Voici l'algorithme que suit cette fonction :

Algorithm 2 Dominance

```
1: boolean F1dominant = true
2: boolean F2dominant = true
3: Dominance = "None"
4: if Un des points de F1 est dominé ou égal par un des points de F2 then
5:     F1dominant = false ;
6: end if
7: if Un des points de F2 est dominé ou égal par un des points de F1 then
8:     F2dominant = false ;
9: end if
10: if F1dominant && F2dominant then
11:     Dominance = None
12: else if F1dominant then
13:     Dominance = F1
14: else if F2dominant then
15:     Dominance = F2
16: end if
```

Cette dominance ne permet donc pas de savoir si un front domine totalement un autre, mais elle permet de donner un ordre d'idée.

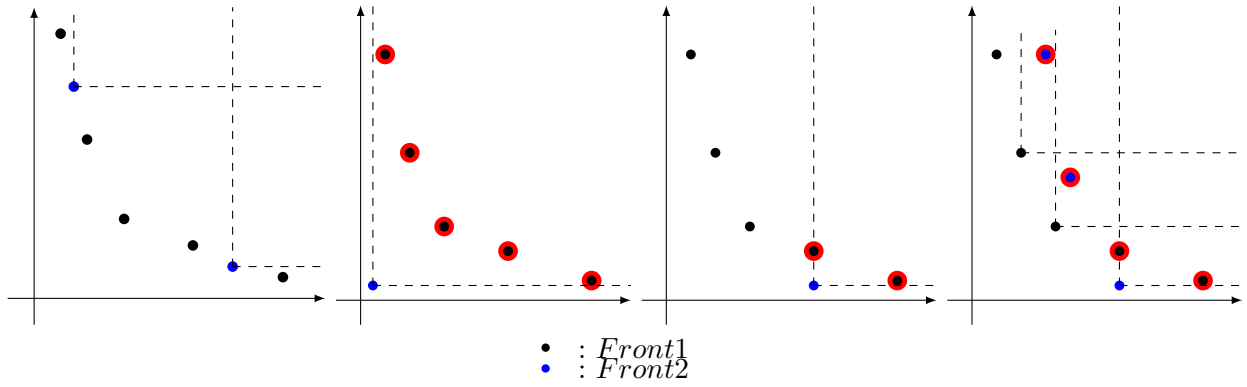


FIGURE 3.5 – Dominance

Sur la première figure, aucun point de Front2 ne domine un point de Front1. Et aucun point de Front1 ne domine un point de Front 2. La dominance donnera donc : None.

Sur la deuxième figure, on voit que le point de Front2 domine l'ensemble des points du Front1. La dominance donnera donc : Front2. Ici on est dans un cas ou le Front2 domine complètement le Front1.

Sur la troisième figure, on voit que des points de Front1 sont dominés par un point de Front2, mais aucun point de Front1 ne domine un point de Front2. La dominance donnera donc : Front2. Ici on n'est pas dans un cas ou Front2 domine totalement Front1.

Sur la dernière figure, on voit qu'une solution de Pareto du Front 2 est dominée par une du Front 1. De plus une solution de Pareto du Front 1 est dominée par une du Front 2. On est donc dans un cas où aucun front ne domine l'autre. La dominance donnera donc : None.

3.3.2 Fonctionnement du programme

Entrée du programme

Le programme prend en entrée 3 paramètres :
- le chemin du dossier contenant l'ensemble des résultats de la méthode approchée
- le chemin du dossier contenant l'ensemble des résultats de la méthode exacte
- le chemin du dossier où sera rangé le fichier de sortie.

Les dossiers contenant les résultats de chaque méthode doivent être constitués de la même manière. Pour cela une convention de nommage est à respecter.

Tout d'abord, le dossier fourni en paramètre contient des sous-dossiers spécifiant le nombre de machines de la manière suivante : "XMachines", X étant le nombre de machines.

Prenons un exemple : en paramètre de notre appel de programme, nous avons passé le chemin des résultats de la méthode exacte pour le problème 1 : D :/TimeIndexed/Probleme1.
On remarque que ce dossier contient bien 3 dossiers spécifiant chacun le nombre de machines sur lesquels les futures instances sont basées.

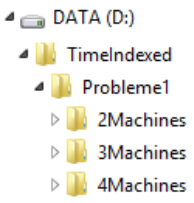


FIGURE 3.6 – Convention de nommage dossiers machines

Par la suite, chacun de ces dossiers machines est composé de dossiers exprimant le nombre de jobs qui seront appliqués sur les futures instances. Ces dossiers sont nommés de la manière suivante : "YJobs", Y étant le nombre de jobs.

Reprenons l'exemple ci-dessus et développons l'arborescence, on remarque que chaque dossier "XMachines" est composé de plusieurs dossiers "YJobs".

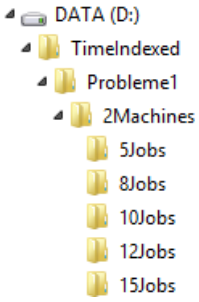


FIGURE 3.7 – Convention de nommage dossiers jobs

Enfin chacun de ces dossiers jobs contient les résultats correspondant aux n instances à X machines et Y jobs. Ils doivent être nommés de cette manière : "instance_Z" , Z étant le numéro de l'instance.

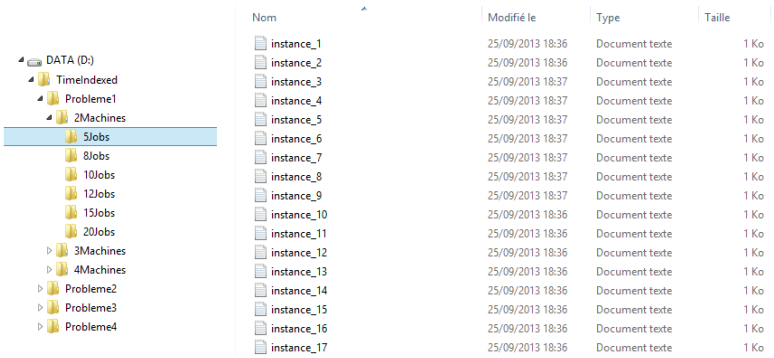


FIGURE 3.8 – Convention de nommage fichiers instances

On voit bien que les dossiers contiennent l'ensemble des résultats pour chaque instance. De plus ils contiennent aussi un autre fichier texte "temps" (cette convention de nommage est aussi à respecter) répertoriant les temps de calcul ainsi que le nombre de solutions trouvées pour chaque instance.

La présence de l'ensemble des fichiers est obligatoire pour le bon fonctionnement du programme. Si pour certaines instances on n'obtient pas de résultat, le fichier "instance_Z.txt" doit quand même être présent et doit être vide. Si ce n'est pas le cas, il y aura un décalage dans le fichier excel final et un message apparaitre afin d'informer l'utilisateur.

D'une manière générale, l'ensemble des dossiers "XMachines", "YJobs" et les fichiers "instance_Z" doivent être présents (sinon le programme ne fonctionnera pas correctement). On doit donc avoir pour nos deux méthodes, exactement le même nombre de dossiers et de fichiers.

Les fichiers "instance_Z" contiennent l'ensemble des solutions non dominées composant le front de Pareto. On a donc un fichier contenant les coordonnées (x,y) de chaque solution. Ces coordonnées doivent être séparées d'au moins un espace entre le x et le y. Par exemple "92 15".

Les fichiers "temps.txt" contiennent l'ensemble des temps de calcul et nombre de solutions obtenus pour chaque instance. Chaque ligne correspond à une instance, on a dans un premier temps le temps de calcul pour cette instance, puis le nombre de solutions trouvé pour cette instance. Si cette instance n'a pu être exécutée par la méthode pour diverses raisons (trop long, erreur ...) on laisse un espace et c'est tout.

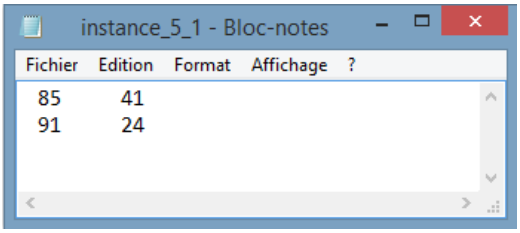


FIGURE 3.9 – Exemple de contenu d'un fichier instance

Pour plus d'informations sur l'utilisation du programme, on peut se référer à la documentation utilisateur.

Sortie du programme

En sortie nous obtenons un fichier Excel composé de plusieurs pages. Une page principale contenant un résumé des résultats (la moyenne des autres pages), et les pages détaillant les résultats sur chaque machine.

Tout d'abord la feuille principale :

	A	B	C	D	E	F	G	H	I	J	K
1	2 Machines										
2	Jobs	Méthode 1		Méthode 2		Comparaison					
3		TpsCal	NbSol	TpsCal	NbSol	PESF (%)	ASD	ADPM1	ADPM2	OP	
4	5	0,021	1	0,002	1	96,667	0,033	0	0	0	
5	10	0,043	1,033	0,002	1,133	43,333	1,155	0,024	0,247	0,117	
6	15	0,061	1	0,001	1,267	60	0,48	0	1,071	0,033	
7	20	0,067	1	0,001	1,367	50	1,014	0	2,553	0,033	
8	25	0,099	1	0,001	1,567	43,333	0,689	0	1,21	0,312	
9	30	0,1	1	0,002	1,667	53,333	0,667	0	4,261	0,214	

FIGURE 3.10 – Contenu feuille "Résultat"

Cette feuille présente pour chaque machine les moyennes des résultats obtenues par rapport au nombre de jobs. Nous avons :

- TpsCal = Temps de calcul pour la méthode correspondante
- NbSol = Nombre de solutions obtenues pour la méthode correspondante
- PESF = Le pourcentage de solution que la Méthode 1 a en commun avec la Méthode 2
- ASD = La distance moyenne entre les solutions de Pareto de la Méthode 1 et celle de la Méthode 2
- ADPM1 & ADMP2 = La moyenne des distances séparant les solutions de Pareto d'une méthode 2 à 2
- OP = La moyenne des distances en projection orthogonale de la Méthode 1 sur la Méthode 2.

Ensuite les autres feuilles : Ces feuilles permettent de voir en détail les résultats obtenus pour chaque instance. On a une colonne supplémentaire "Dominant" qui permet de dire si une méthode domine l'autre ou non.

Fonctionnement

Le programme récupère l'ensemble des résultats d'instances présents dans les arborescences des dossiers fournis en entrée et va les comparer un à un pour ensuite mettre les résultats dans le fichier Excel

	A	B	C	D	E	F	G	H	I	J	K	L
1	5 Jobs											
2	Instance	Méthode 1		Méthode 2		Comparaison						
3		TpsCal	NbSol	TpsCal	NbSol	PESF (%)	ASD	ADPM1	ADPM2	OP	Dominant	
4	1	0,015	1	0,006	1	100	0	0	0	0	None	
5	2	0,015	1	0,002	1	100	0	0	0	0	None	
6	3	0,018	1	0,002	1	100	0	0	0	0	None	
7	4	0,016	1	0,003	1	100	0	0	0	0	None	
8	5	0,022	1	0,002	1	100	0	0	0	0	None	
9	6	0,014	1	0,003	1	100	0	0	0	0	None	
10	7	0,034	1	0,002	1	100	0	0	0	0	None	
11	8	0,008	1	0,001	1	100	0	0	0	0	None	
12	9	0,017	1	0,001	1	100	0	0	0	0	None	
13	10	0,043	1	0,001	1	0	1	0	0	0	M1	

Résultat | 2Machines | 3Machines | 5Machines

FIGURE 3.11 – Contenu feuille "XMachines"

3.4 Générateur d'instances

Afin de pouvoir avoir des jeux de test, un générateur d'instances a été créé. Celui-ci permet de créer un lot d'instances.

L'important dans ce programme est de trouver une fonction générant des d_j en fonction des p_j. Pour cela on créé deux variables α et β qui vont nous permettre de fixer un intervalle de valeur que pourront prendre les d_j. Tout d'abord on calcule la somme des d_j divisée par le nombre de machines : $P = (\sum p_j)/m$. Ensuite les d_j prendront une valeur appartenant à l'intervalle $[(\alpha - \beta)P; (\alpha + \beta)P]$.

α et β sont fixés entre 0 et 1.

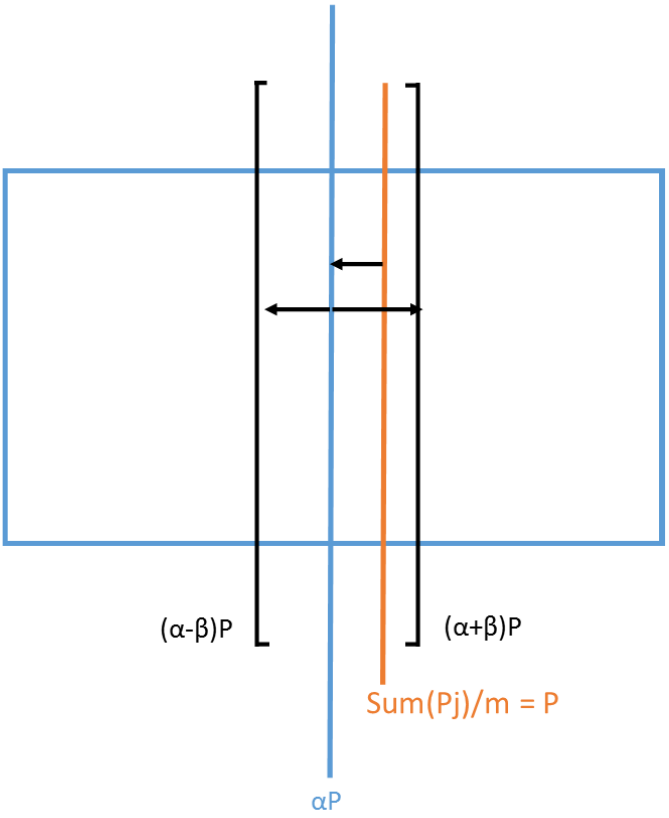


FIGURE 3.12 – Détermination de l'intervalle des d_j

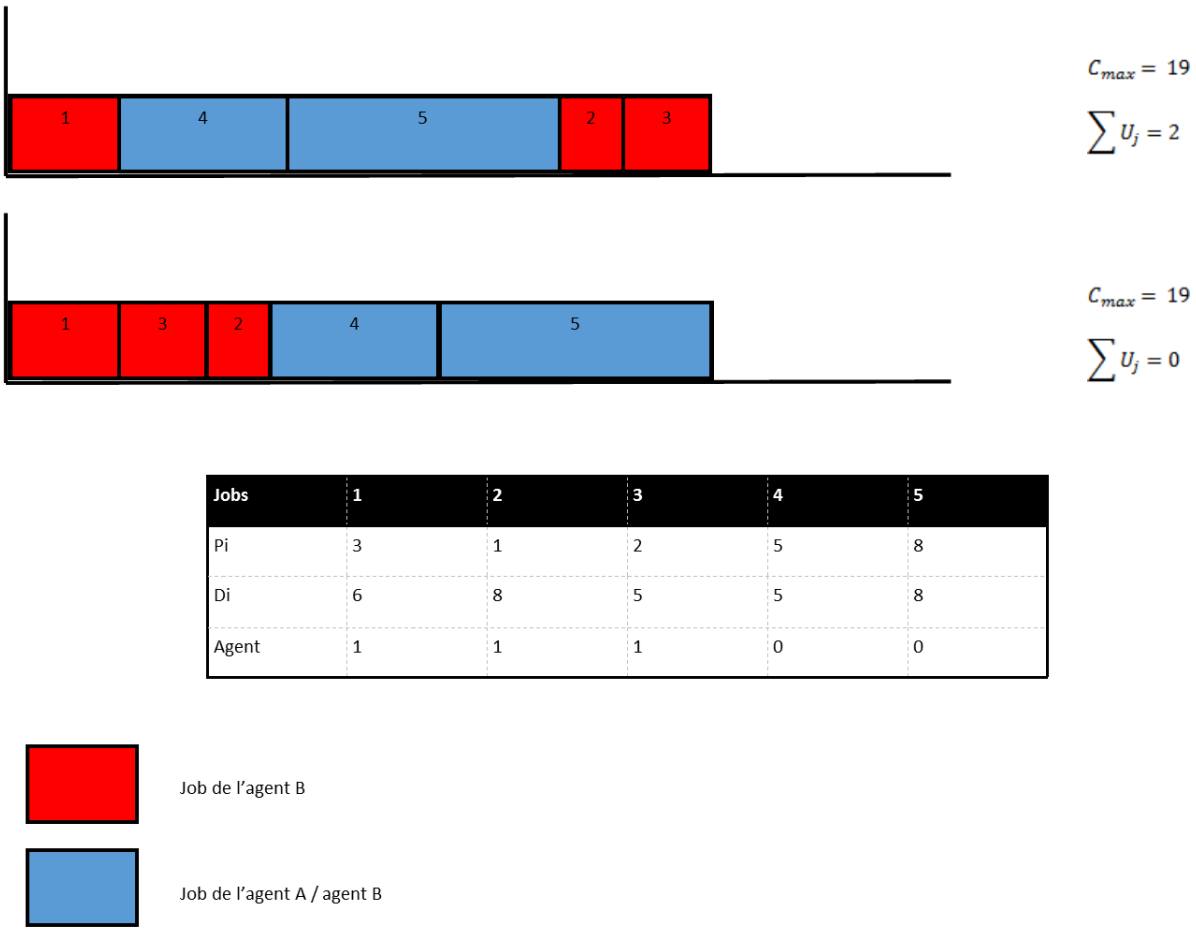
3.5 Problème 1 : $Pm|d_i|P(C^A_{max}, \sum U^B_i)$

L'objectif de ce problème est de minimiser le coût total de l'ensemble des jobs tout en minimisant le nombre de jobs en retard pour l'agent.

3.5.1 Quelques propriétés

Propriété 1 :

Une fois l'affectation des jobs sur les machines établie, il faut caler les jobs de l'agent sur la gauche en appliquant Moore afin de minimiser le nombre de jobs en retard pour l'agent. En effet cette modification ne changera pas le C_{max} on n'a donc pas une régression de la solution, mais bien une optimisation.



Jobs	1	2	3	4	5
Pi	3	1	2	5	8
Di	6	8	5	5	8
Agent	1	1	1	0	0

FIGURE 3.13 – Illustration propriété 1

3.5.2 Heuristique

Approche ε-contrainte

Pour UB, nous pouvons lui donner la valeur correspondant au nombre de jobs attribués à notre agent B. Ainsi, si notre agent possède n_B tâches, $UB = n_B$. Cette borne supérieure est bonne, car elle permettra de n'oublier aucune possibilité. C'est n_B qu'on utilise, car c'est uniquement les jobs de l'agent B qui sont concernés par cette borne.

Pour LB, nous pourrions tout simplement la fixer à 0. Cependant, il est facile de trouver une meilleure LB.

Une première solution serait de résoudre le problème $Pm||\sum U_j$. Ainsi nous obtiendrions la LB optimale. Malheureusement, résoudre ce problème peut rapidement être coûteux en temps de calcul.

La solution que je propose est de fixer notre LB au nombre de jobs ayant leur p_i supérieur à leur d_i : $LB = \#(p_i - d_i > 0)$ avec $i \in \{J_B\}$. On trouve ainsi une borne inférieure retirant les cas irréalisables (on retire une itération pour chaque tâche de l'agent B qui ne pourra jamais finir à temps).

Algorithm 3 Approche ϵ-contrainte avec $f^A = C_{max}$ et $f^B = \sum U_j$

```
Q = UB
while Q ≥ LB do
    Résoudre Pm|d_i, ∑U_i^B ≤ Q|C_max^A
    if Pas de solution then
        break ;
    else
        noter(α, β) = (∑U_j^B, C_max^A) la solution retournée.
    end if
    Poser Q = Q − 1
    S = S ∪ (α, β)
end while
Retourner "S"
```

Raisonnement pour l'heuristique

Pour cette heuristique nous allons travailler avec deux listes :
- une liste de jobs prioritaires composée d'un certain nombre de jobs de l'agent B
- une liste de jobs non prioritaires composée du reste des jobs

La première étape consiste à séparer les jobs de l'agent des autres. Ensuite, on trie les jobs de l'agent par EDD [7]. Cet ordre permet de minimiser de manière approchée le nombre de jobs en retard par la suite.

La seconde étape consiste à choisir les jobs qui vont faire partie de la liste prioritaire parmi les jobs de l'agent. Ce nombre de jobs va dépendre de Q. Cette variable nous force à avoir $n_B - Q$ jobs en avance. C'est donc de cette taille que va être notre liste prioritaire. On placera donc les $n_B - Q$ premiers jobs de l'agent B dans la liste prioritaire. Tous les autres jobs seront placés dans la liste non prioritaire qui sera triée par LPT [8]. Ce trie par LPT est utilisé, car c'est la meilleure heuristique pour résoudre le problème $P_m||C_{max}$

La troisième partie consiste à placer les jobs de la liste prioritaire sur les machines en utilisant FAM [9] puis à placer ceux de la liste non prioritaire en utilisant FAM aussi.

L'affectation des jobs sur les machines est faite, le C_{max} est maintenant fixé. Toutefois nous pouvons maintenant améliorer $\sum U_j$ (cf. Propriété 1). Nous prenons donc chaque machine une à une et nous appliquons Moore afin de minimiser le nombre de jobs en retard pour l'agent.

Enfin, il ne reste plus qu'a vérifié si notre solution respecte l' ε-contrainte Q.

Algorithme de l'heuristique

7. EDD : Earliest Due Date
8. LPT : Long Processing Time
9. FAM : First Available Machine

Algorithm 4 $Pm|d_j, \sum U_j^B \leq Q|C_{max}^A$

```
1:                                                          ▷ Préparation des listes
2: for (j = 0 ; j < nbJobs ; j++) do
3:     if (j ∈ J_B) then
4:         Ajouter le job j à ListJobB
5:     else
6:         Ajouter le job j à ListNonPrio
7:     end if
8: end for
9:                                                          ▷ Tri de ListJobB
10: Trier ListJobB par EDD
11:                                                         ▷ Choix des jobs prioritaires
12: for (j = 0; j < listJobB.size(); j++) do
13:     if ((p_j > d_j) && (listPrio.size() < (n_B − Q))) then
14:         Ajouter le job j à ListPrio
15:     else
16:         Ajouter le job j à ListNonPrio
17:     end if
18: end for
19:                                                         ▷ Tri de ListNonPrio
20: Trier ListNonPrio par LPT
21:                                                         ▷ Attribution des jobs sur les machines
22: Placer ListPrio en appliquant FAM
23: Placer ListNonPrio en appliquant FAM
24:                                                         ▷ On minimise le nombre de jobs de B en retard
25: for all Machines do
26:     Caler les jobs de l'agent B à gauche en appliquant Moore.
27: end for
28:                                                         ▷ On vérifie que la condition est respectée
29: if ∑U_j^B > Q then
30:     Retourner (Pas de solution trouvée)
31: end if
32: Retourner (∑U_j^B, C_max^A)
```

Tests et résultats

Pour effectuer nos tests, un jeu d'instances a été créé. Ces jobs ont une durée opératoire comprise entre 0 et 100. Les variables α et β sont fixées à 0,3 et 0,5. Ces choix ont été faits après de nombreux essais pour essayer d'obtenir des fronts de Pareto avec plusieurs points. Dans le cas de ce premier sous problème, il est rare de trouver plus d'une solution de Pareto. L'un des cas ou on peut avoir $n_B/2$ solutions est celui de la figure ci-dessous.

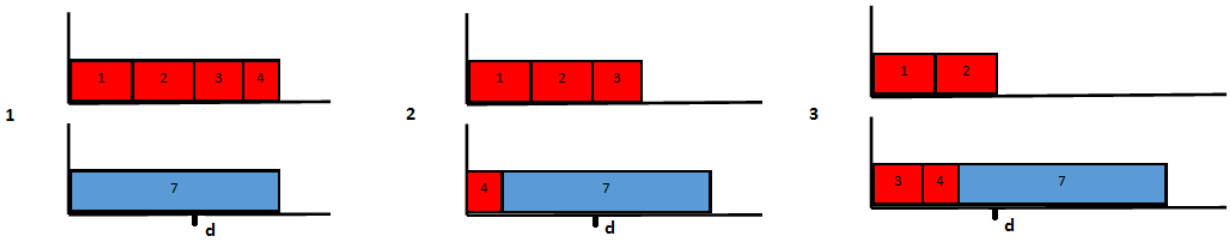


FIGURE 3.14 – Exemple donnant un front avec plusieurs solutions non dominées

Ce choix d'α et β a donc été fait par rapport au second sous problème pour que celui-ci nous donne un front avec plus de solutions.

Pour nos tests, nous les avons faits sur 2,3 et 5 machines pour un nombre de jobs allant de 5 à 100 jobs. C'est ces instances qui seront utilisées pour tous les tests.

Le front exact est trouvé grâce à l'application de la programmation linéaire.

Rappel :
- ADPF1 & ADPF2 : Average Distance Point Function 1 & 2
- nbSol : number of solution
- OP : orthogonal projection

		Méthode Exacte		Méthode Approchée		Comparaison			
m	n	CPU	nbSol	CPU	nbSol	% Exact	ADPF1	ADPF2	OP
2	5	0,021	1	0	1	80	0	0	0
	10	0,043	1,033	0,001	1,3	23,333	0,024	0,509	0,65
	15	0,061	1	0,002	1,167	30	0	0,22	0,133
	25	0,099	1	0,002	1,133	36,667	0	0,123	0,533
	50	0,114	1	0	1,133	36,667	0	0,092	1,6
	75	0,13	1	0,001	1,167	33,333	0	0,147	1,5
	100	0,128	1	0,002	1	53,333	0	0	2
3	5	0,037	1	0	1	96,667	0	0	0
	10	0,102	1,1	0	1,2	35	0,165	0,732	0,817
	15	0,1	1,033	0	1,3	20	0,024	0,817	0,55
	25	0,192	1	0	1,367	20	0	0,491	0,4
	50	0,389	1	0	1,2	23,333	0	0,198	1,267
	75	0,637	1	0,001	1,267	30	0	0,23	2,333
	100	0,604	1	0,002	1,1	36,667	0	0,084	2,833
5	5	0,012	1	0	1	100	0	0	0
	10	0,077	1	0	1	86,667	0	0	0
	15	0,189	1	0	1,1	13,333	0	0,343	0,3
	25	0,424	1	0	1,567	0	0	0,889	0,7
	50	1,411	1	0	1,667	3,333	0	0,722	0,871
	75	3,394	1	0,001	1,533	16,667	0	0,439	2,4
	100	4,432	1	0,002	1,467	10	0	0,543	4,133

TABLE 3.1 – Tableau résultats de l'heuristique pour $Pm||C^A_{max}, \sum U^B_j$

Comparaison des temps CPU : on peut voir que la méthode approchée est bien plus rapide que la méthode exacte. Cette différence est nette sur les instances avec 5 machines et 100 jobs. La méthode approchée remplit donc bien son rôle ici.

Comparaison du nombre de solutions de Pareto : globalement les fronts de la méthode approchée sont composés de plus de solutions de Pareto que ceux de la méthode exacte et avec des distances entre les solutions plus importantes. Cette distance permet d'avoir un choix plus simple. Toutefois si cet écart est trop important, le choix est compliqué, car soit on privilégie clairement un critère soit l'autre, mais il n'y aura pas de juste milieu.

Solutions exactes trouvées et écart entre les deux fronts : on peut voir que plus le nombre de machines augmente, plus il est difficile de trouver les solutions exactes. De plus, la moyenne des projections orthogonales reste faible en comparaison à l'intervalle des durées opératoires. Au vu du nombre de solutions exactes trouvées et de l'écart entre les fronts, on peut dire que cette heuristique donne d'assez bons résultats.

3.5.3 Algorithme Génétique

Évaluation & Rang

Nous commençons par l'évaluation, car il y a une notion importante à prendre en compte dans l'application de la méta-heuristique à notre problème : les rangs. En effet, puisque nous sommes dans un problème avec critères interférants, une solution peut être aussi importante qu'une autre même si les deux ne possèdent pas la même valeur de C_{max} et de $\sum U_j$. Les solutions appartenant à un même front possèdent la même importance. Il faut donc énumérer les différents fronts et leur attribuer un rang. Plus un front est bas, plus le rang sera faible. Chaque individu se verra donc attribuer le rang de son front. Le calcul de C_{max} et de $\sum U_j$ se font en suivant la propriété 1 ci-dessus.

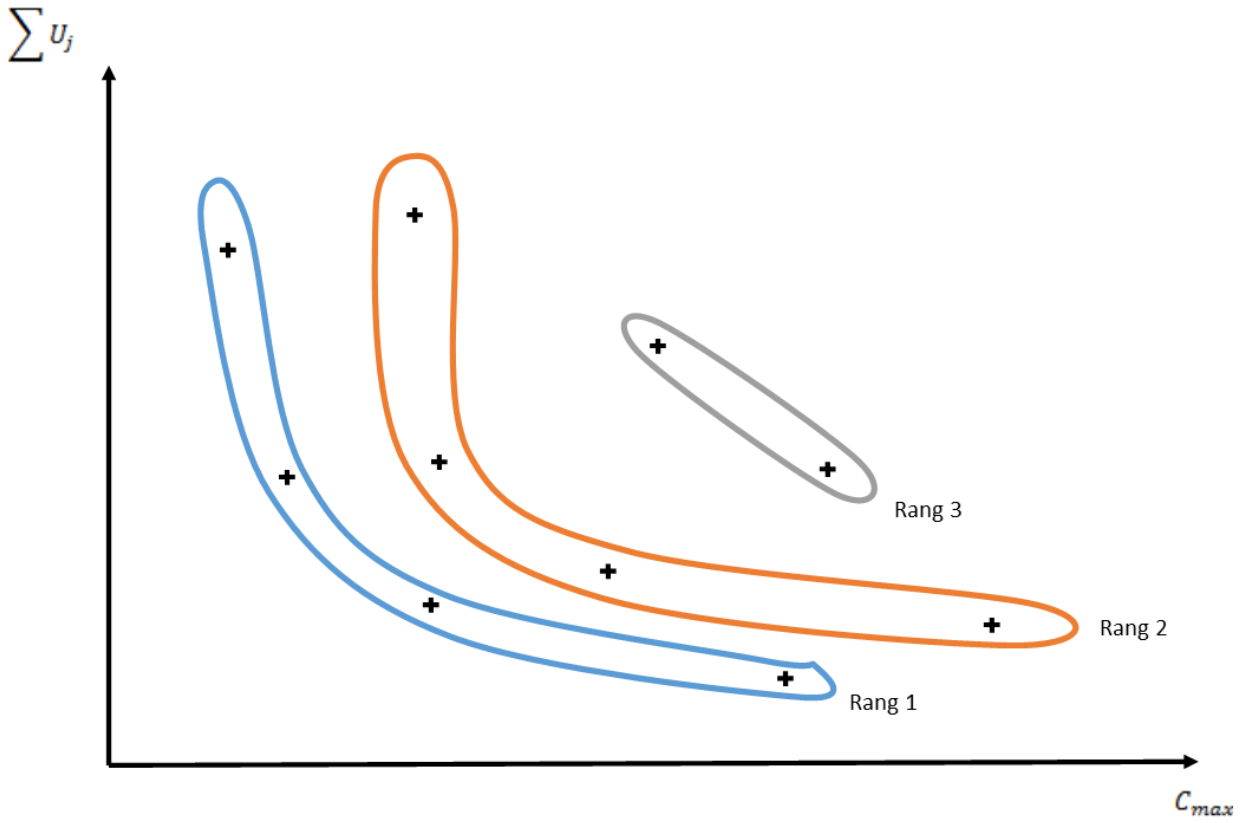


FIGURE 3.15 – Affectation des rangs

Individu

Afin d'adapter la méta-heuristique à notre problème, une classe d'individus est créée. Chaque individu est composé d'une liste de machines. Chaque machine est composée d'une liste de jobs. Cette structure est faite afin que, par la suite, le projet puisse être repris dans le cadre d'ajout d'agent de manière simple.

A cela, on lui rajoutera une variable stockant C_{max}, une variable stockant $\sum U_j$, et une variable stockant le rang de l'individu.

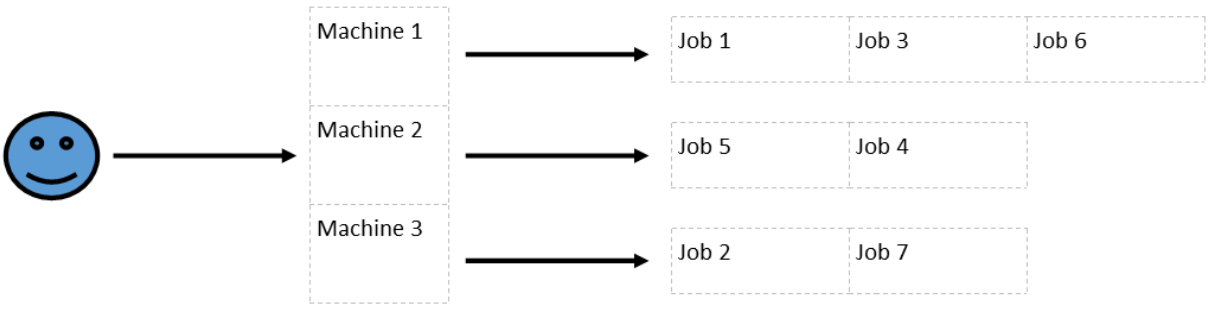


FIGURE 3.16 – Structure d'un individu

Population initiale

Pour générer cette population initiale, on créé tout d'abord deux individus particuliers :
- un individu créé en utilisant LPT + FAM
- un individu créé en utilisant SPT + FAM

Le reste des individus est créé en utilisant simplement FAM et en prenant les jobs aléatoirement.

Sélection

Pour la sélection des parents, on garde tous les individus du rang 1, puis on fait un tournoi binaire avec le reste des individus. Pour ce tournoi binaire, 2 cas sont possibles :
- si les deux individus possèdent le même rang alors on choisit au hasard l'un des deux
- si les deux individus ne possèdent pas le même rang, on gardera celui avec le rang le plus faible

Croisement

Pour le croisement, on commence par transformer nos affectations des jobs en une séquence pour le père et la mère. Dans cette séquence, on met les jobs de chaque machine dans l'ordre.

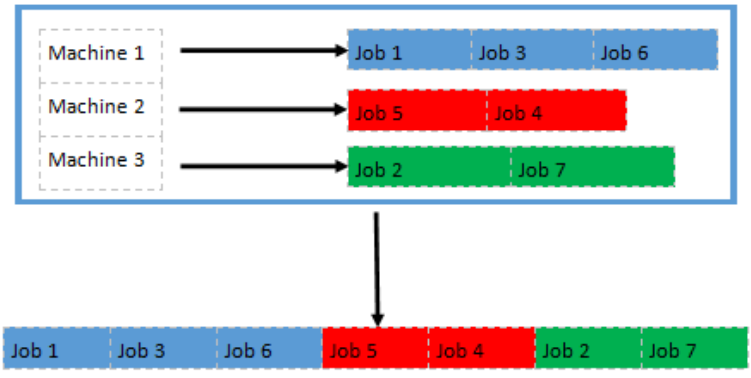


FIGURE 3.17 – Transformation en une liste

Une fois cette transformation faite pour les deux parents, on va créer les enfants avec un croisement à 2 points. Ces deux points sont calculés de la manière suivante :

- le premier point prend une valeur entre 1 et $nbJobs/3$
- le second point prend une valeur entre le premier point et $nbJobs$

Pour créer le fils/fille, on prend les jobs du père/mère jusqu'au premier point. Ensuite on prend les jobs entre le premier et second point du père/mère et on les place dans l'ordre d'apparition de la mère/père. Pour finir, on prend les derniers jobs du père/mère.

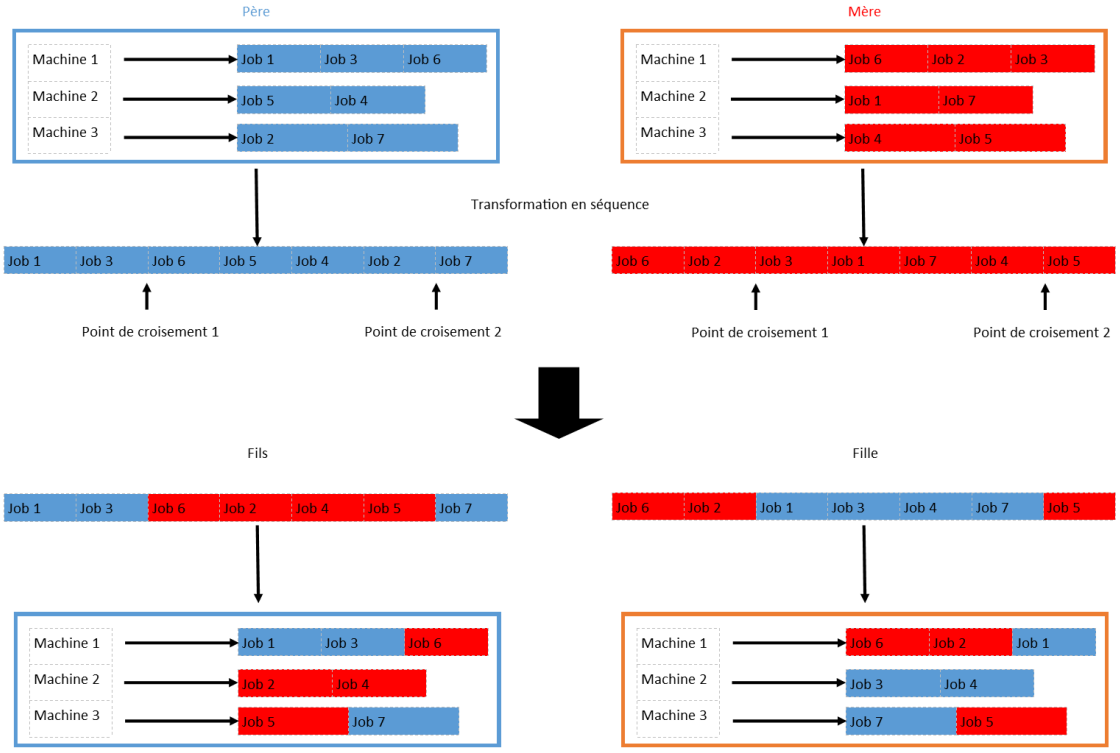


FIGURE 3.18 – Croisement

Mutation

Chaque individu à 2 % de chance d'être muté. Dans ce problème, la mutation consiste à choisir une machine aléatoirement, puis un job sur cette machine et de le déplacer sur une autre machine à un emplacement aléatoire.

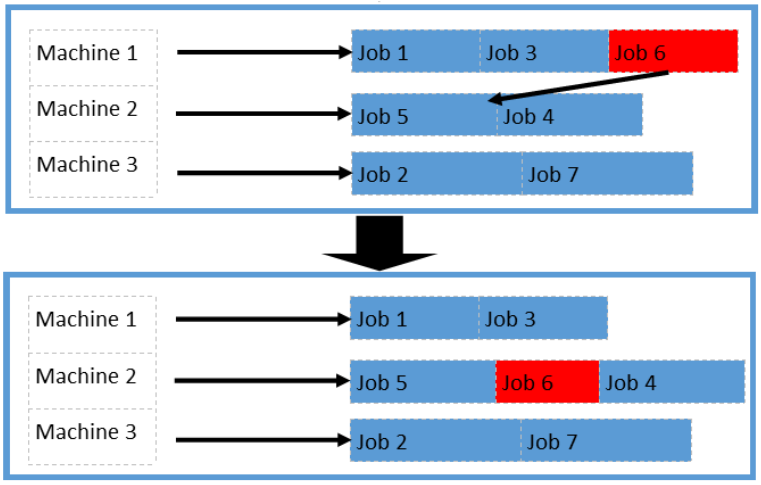


FIGURE 3.19 – Mutation

Tests et résultats

Pour ces tests, on a utilisé exactement les mêmes instances que pour l'approche ε-contrainte.

Après de nombreux tests, on a choisi de prendre une taille de population égale à deux fois le nombre de jobs et un nombre de générations égal à 10 fois le nombre de machines.

Rappel :
- ADPF1 & ADPF2 : Average Distance Point Function 1 & 2
- nbSol : number of solution
- OP : orthogonal projection

		Méthode Exacte		Méthode Approchée		Comparaison			
m	n	CPU	nbSol	CPU	nbSol	% Exact	ADPF1	ADPF2	OP
2	5	0,021	1	0,001	1	86,667	0	0	0
	10	0,043	1,033	0,002	1,167	25	0,024	0,402	0,117
	15	0,061	1	0	1,233	36,667	0	0,732	0,1
	25	0,099	1	0	1,7	26,667	0	3,049	0,406
	50	0,114	1	0,004	1,9	36,667	0	2,992	0,761
	75	0,13	1	0,011	1,767	46,667	0	3,908	0,679
	100	0,128	1	0,022	1,633	56,667	0	2,373	0,606
3	5	0,037	1	0	1	96,667	0	0	0
	10	0,102	1,1	0	1,233	31,667	0,165	1,627	0,383
	15	0,1	1,033	0	1,5	16,667	0,024	2,074	0,883
	25	0,192	1	0	1,733	10	0	3,558	1,067
	50	0,389	1	0,003	2,033	13,333	0	6,677	1,061
	75	0,637	1	0,011	2,267	10	0	7,144	1,705
	100	0,604	1	0,026	2,3	10	0	6,167	1,846
5	5	0,012	1	0	1	100	0	0	0
	10	0,077	1	0	1	90	0	0	0
	15	0,189	1	0	1,1	13,333	0	0,294	0,3
	25	0,424	1	0	1,667	0	0	3,526	1,867
	50	1,411	1	0,002	2,567	0	0	9,41	1,841
	75	3,394	1	0,01	2,733	3,333	0	8,91	2,133
	100	4,432	1	0,03	3	0	0	15,004	2,307

TABLE 3.2 – Tableau résultats de l'AG pour $Pm||C^A_{max}, \sum U^B_j$

Comparaison des temps CPU : de même que pour l'heuristique, on peut voir que l'algorithme génétique est bien plus rapide que la méthode exacte. Cette différence est nette sur les instances avec 5 machines et 100 jobs. La méthode approchée remplit donc bien son rôle ici. Toutefois, elle est un peu plus lente que l'approche ε-contrainte. En effet le temps d'exécution monte jusqu'à 0.03 seconde pour l'algorithme génétique alors qu'il ne monte qu'à 0.02 maximum pour l'approche ε-contrainte.

Comparaison du nombre de solutions de Pareto : globalement les fronts de la méthode approchée sont composés de plus de solutions de Pareto que ceux de la méthode exacte et avec des distances entre les solutions plus importantes. Ceci est normal du fait qu'ici, la méthode exacte trouve une seule solution pour la quasi-totalité des instances, puisque notre méthode approchée ne trouve pas la solution optimale tout le temps, elle va trouver une ou des solutions approchées.

Solutions exactes trouvées et écart entre les deux fronts : on peut voir que plus le nombre de machines augmente, plus il est difficile de trouver les solutions exactes. De plus, la moyenne des projections orthogonales reste faible en comparaison à l'intervalle des durées opératoires. Au vu du nombre de solutions exactes trouvées et de l'écart entre les fronts, on peut dire que cette heuristique donne d'assez bons résultats.

3.5.4 Comparaison entre l'heuristique et l'algorithme génétique

Solutions exactes trouvées

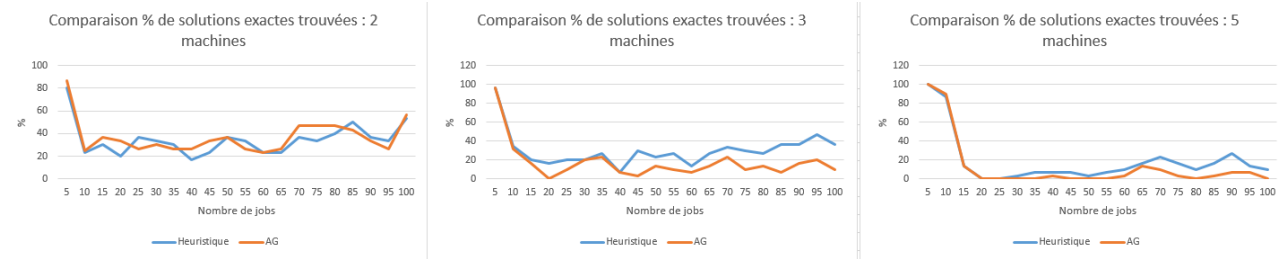


FIGURE 3.20 – Comparaison du % de solutions exactes trouvées

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne le nombre de solutions exactes trouvées. Sur chacun de ces graphes, on a le résultat du pourcentage de solutions obtenu pour chaque méthode en fonction du nombre de jobs.

On peut voir facilement que, hormis pour 2 machines, l'application de l'approche ε-contrainte sur notre heuristique domine l'algorithme génétique.

Espacement des fronts de Pareto

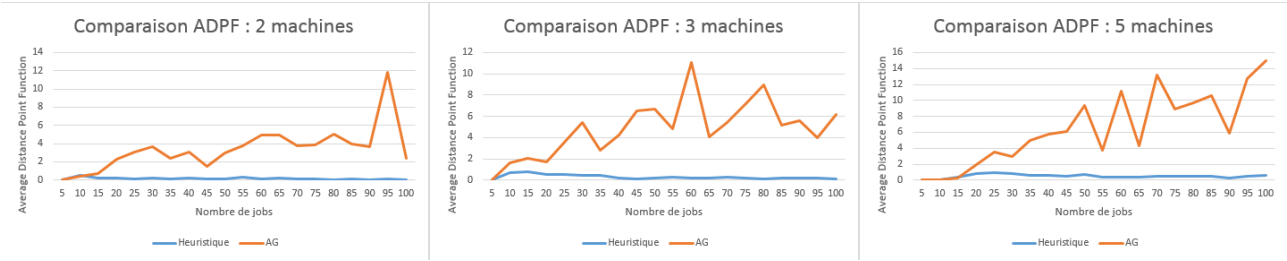


FIGURE 3.21 – Comparaison des "Average Distance Point Function"

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne la moyenne des distances entre les solutions pour chacun des fronts de Pareto. Sur chacun de ces graphes, on a le résultat de cette moyenne obtenue pour chaque méthode en fonction du nombre de jobs.

On voit nettement que l'algorithme génétique possède des fronts ayant des solutions de Pareto plus diverses que celles de l'heuristique. Ceci s'explique par le fait que l'heuristique trouve moins de solutions de Pareto (généralement seulement une ou deux) et qui sont très proches, tandis que l'algorithme génétique va trouver un peu plus de solutions qui seront un peu plus éloignées les unes des autres. L'algorithme génétique propose donc un plus large choix (par forcément meilleurs, mais plus diverses) de décisions que l'heuristique.

Écarts avec le front exact

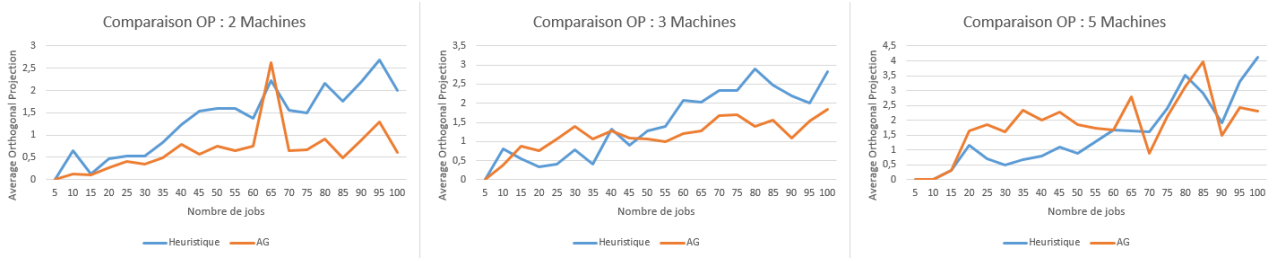


FIGURE 3.22 – Comparaison de l'écart entre les fronts approchés et le front exact

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne l'écart entre le front approché et le front exact. Cet écart est mesuré par la moyenne des distances les plus courtes entre les solutions du front approché et le front exact. Sur chacun de ces graphes, on a le résultat de cette moyenne obtenue pour chaque méthode en fonction du nombre de jobs.

Pour deux machines, il est clair que l'algorithme génétique trouve un front plus proche du front exact que l'approche ε-contrainte appliquée l'heuristique. Mais pour 3 et 5 machines, le verdict n'est pas aussi facile. En effet, il n'y a pas de véritable dominance. On peut dire que pour un nombre de jobs inférieur à 50, l'heuristique se comporte mieux que l'AG et qu'ensuite c'est l'inverse (pour 5 machines, après 50 jobs, il y a une alternance, il est donc difficile d'émettre un avis).

Conclusion

D'une manière générale, on peut dire avec les 3 études précédentes que, pour 2 machines, l'AG est meilleure que l'heuristique, car elle propose plus ou moins le même nombre de solutions exactes. De plus, les solutions proposées sont plus diverses, et elle propose un front approché plus proche de l'exact.

En ce qui concerne 3 et 5 machines, on ne peut pas vraiment se prononcer aussi facilement. On va avoir un plus grand nombre de solutions exactes avec l'heuristique, un choix plus divers avec l'AG, et aucune dominance d'une des deux méthodes en ce qui concerne l'écart avec le front exact. Ce sera donc suivant la volonté de l'utilisateur, ces préférences, qu'il faudra choisir l'une ou l'autre méthode.

3.6 Problème 2 : $Pm|d_i|P(\sum U_i^A, C_{max}^B)$

3.6.1 Heuristique

Approche ε-contrainte

Algorithm 5 Approche ϵ-contrainte avec $f^A = \sum U_i$ et $f^B = C_{max}$

```
Q = UB
while Q ≥ LB do
    Résoudre Pm|d_i, C^B_max ≤ Q| Σ U^A_i
    if Pas de solution then
        break ;
    else
        noter(α, β) = (C^B_max, Σ U^A_i) la solution retournée.
    end if
    Poser Q = Q − 1
    S = S ∪ (α, β)
end while
Retourner "S"
```

Pour UB, nous pouvons faire une répartition des jobs sur les machines en utilisant LPT+FAM tout en plaçant les jobs de B en dernier. On a donc : $UB = C_{max}((LPT + FAM)_{global}; (LPT + FAM)_B)$. Cette UB est une heuristique, car LPT n'est pas optimale pour le C_{max}, mais c'est une bonne approche.

Pour LB, nous appliquons le meilleur des cas : $LB = (\sum p_j)/m$ avec $j \in J_B$

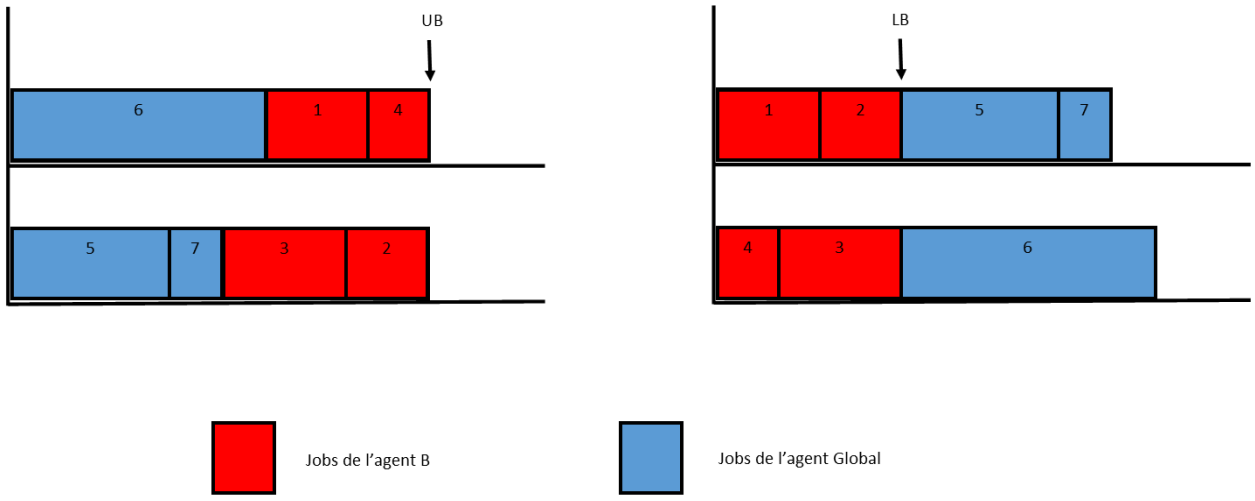


FIGURE 3.23 – $Pm|d_i|P(\sum U_i^A, C_{max}^B)$ UB & LB

Raisonnement pour l'heuristique

Le raisonnement pour cette heuristique est le suivant :
- première étape : on sépare les jobs de l'agent des autres. Ensuite, on trie les jobs de l'agent par LPT et ceux du global par EDD.
- deuxième étape : on place les jobs de l'agent puis ceux de l'agent global
- troisième étape : on applique Moore (avec la condition que le job en cours puisse finir à l'heure avec le futur changement) pour tous les jobs, dont la date de fin d'exécution est avant Q en plaçant ceux en retard le plus tard possible de manière à respecter la condition : $C_{max}^B \leq Q$.
- quatrième étape : on vérifie que notre solution respecte bien l'ε-contrainte Q

Algorithme de l'heuristique

Algorithm 6 $Pm|d_j, C_{max}^B \leq Q|\sum U_j^A$

▷ On récupère les jobs
ListJobB = récupérer la liste des jobs de l'agent B
ListJobGlobal = récupérer les autres jobs
▷ Tri des listes
Trier ListJobB par LPT
Trier ListJobGlobal par EDD
▷ Attribution des jobs sur les machines
Placer ListJobB en appliquant FAM
Placer ListJobGlobal en appliquant Moore et FAM
▷ Optimisation du résultat
for all Machine **do**
    Appliquer Moore (avec la condition que le job en cours puisse finir à l'heure avec le futur changement) pour tous les jobs dont la date de fin d'exécution est avant Q en plaçant ceux en retard le plus tard possible de manière à respecter la condition : $C_{max}^B \leq Q$.
end for

Tests et résultats

Pour ce problème, on utilise exactement les mêmes instances qu'avant. Toutefois, la méthode exacte n'a pas réussi à résoudre des instances de plus de 15 jobs. C'est donc sur des instances avec un maximum de 15 jobs que nous allons comparer les résultats. Pour 5 machines, la mémoire n'a pas tenu pour 15 jobs avec la méthode exacte.

Rappel :
- ADPF1 & ADPF2 : Average Distance Point Function 1 & 2
- nbSol : number of solution
- OP : orthogonal projection

		Méthode Exacte		Méthode Approchée		Comparaison			
m	n	CPU	nbSol	CPU	nbSol	% Exact	ADPF1	ADPF2	OP
2	5	0,107	1,167	0,004	1,3	40	1,22	3,962	0,717
	10	3,137	2,533	0,001	2,233	16,778	9,929	12,244	4,198
	15	34,336	3,3	0,001	2,6	12,333	13,25	17,332	5,516
3	5	0,086	1	0	1	66,667	0	0	0,333
	10	0,571	1,433	0	1,7	41,111	3,26	15,234	2,521
	15	16,073	2,9	0	2,467	6,5	9,586	15,462	2,428
5	5	0,047	1	0	1	100	0	0	0
	10	0,241	1	0	1	73,333	0	0	0,267
	15	0,032	-	0	1,633	-	-	11,608	0

TABLE 3.3 – Tableau résultats de l'heuristique pour $Pm||\sum U_j^A, C_{max}^B$

Comparaison des temps CPU : une fois de plus, les temps d'exécution sont plus courts pour la méthode approchée. Mais dans ce problème, la différence est vraiment frappante. Sur ce point l'heuristique est vraiment bonne.

Comparaison du nombre de solutions de Pareto : cette fois-ci sur ce problème, on a plus d'une solution par front d'une manière générale. Cependant, on peut voir que globalement, la méthode exacte possède plus de solutions de Pareto par front que l'heuristique. On a donc moins de choix. Par contre, l'écart entre les solutions de Pareto des fronts est plus important sur la méthode approchée que sur la méthode exacte. On a donc finalement moins de solutions, mais avec de plus grands écarts de choix.

Solutions exactes trouvées et écart entre les deux fronts : on voit clairement ici que plus le nombre de machines augmente, meilleure se comporte la méthode approchée en ce qui concerne le nombre de solutions exactes trouvées. De plus les distances des projections orthogonales restent relativement faibles par rapport aux durées opératoires. On a donc une bonne méthode approchée ici.

3.6.2 Algorithme Génétique

Pour cet algorithme génétique, le calcul des rangs, les individus, la sélection, et le croisement sont les mêmes que pour l'autre algorithme génétique.

Évaluation de C_{max} et de $\sum U_j$

Afin de calculer le C_{max}, il suffit de prendre le plus grand Cj des jobs de l'agent. Et pour le $\sum U_j$ il suffit de faire la somme des jobs en retard (i.e. le nombre de jobs dont $C_j > d_j$).

Population initiale

Pour générer cette population initiale, on créé tout d'abord deux individus particuliers :
- un individu créé en utilisant Moore sur l'ensemble des jobs + FAM
- un individu créé en utilisant Moore sur les jobs de l'agent + FAM

Le reste des individus est créé en utilisant simplement FAM et en prenant les jobs aléatoirement.

Mutation

Chaque individu à 1 % de chance de faire la même mutation que pour l'autre algorithme génétique. Mais chaque individu possède aussi 1% de chance de subir une autre mutation. Cette seconde mutation consiste à prendre un job et le déplacer à un autre endroit sur la même machine.

Tests et résultats

Dans les tests effectués sur la génétique, on se base sur les mêmes instances que pour l'heuristique.

Rappel :
- ADPF1 & ADPF2 : Average Distance Point Function 1 & 2
- nbSol : number of solution
- OP : orthogonal projection

		Méthode Exacte		Méthode Approchée		Comparaison			
m	n	CPU	nbSol	CPU	nbSol	% Exact	ADPF1	ADPF2	OP
2	5	0,107	1,167	0,001	1,267	40	1,22	2,795	0,683
	10	3,137	2,533	0,001	2,167	12,778	9,929	11,789	4,115
	15	34,336	3,3	0,001	1,933	4,167	13,25	21,341	13,216
3	5	0,086	1	0	1	66,667	0	0	0,333
	10	0,571	1,433	0	1,733	32,222	3,26	4,292	4,211
	15	16,073	2,9	0	2,3	2,778	9,586	17,171	6,202
5	5	0,047	1	0	1	100	0	0	0
	10	0,241	1	0	1,033	70	0	0,134	0,267
	15	0,032	-	0	1,9	-	-	3,733	0

TABLE 3.4 – Tableau résultats de l'AG pour $Pm||\sum U_j^A, C_{max}^B$

Nous pouvons faire les mêmes remarques que pour l'heuristique avec ces résultats. C'est donc une bonne méthode approchée qui fournit de bons résultats d'une manière bien plus rapide que la méthode exacte.

3.6.3 Comparaison entre l'heuristique et l'algorithme génétique

Solutions exactes trouvées

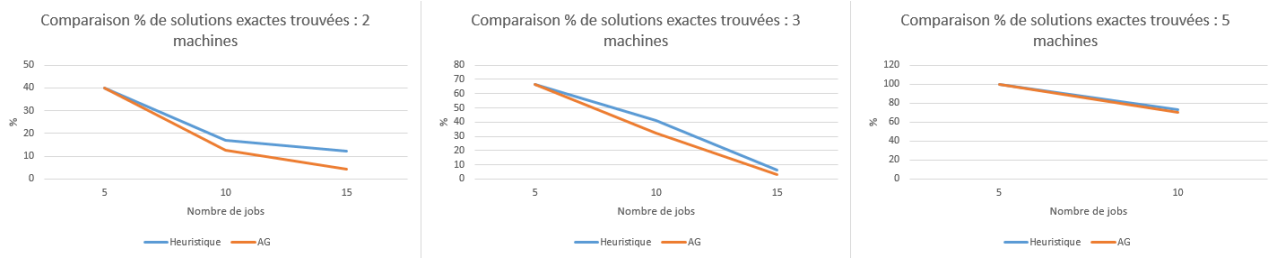


FIGURE 3.24 – Comparaison du % de solutions exactes trouvées

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne le nombre de solutions exactes trouvées. Sur chacun de ces graphes, on a le résultat du pourcentage de solutions obtenu pour chaque méthode en fonction du nombre de jobs.

Ici, l'heuristique domine l'AG. Toutefois, on peut voir que l'écart entre les courbes de l'AG et de l'heuristique se réduit au vu du nombre de machines pour presque se chevaucher sur 5 machines. On peut donc supposé que pour un grand nombre de machines, l'AG tant à devenir meilleure que l'heuristique, mais ceci n'est qu'une supposition.

Espacement des fronts de Pareto

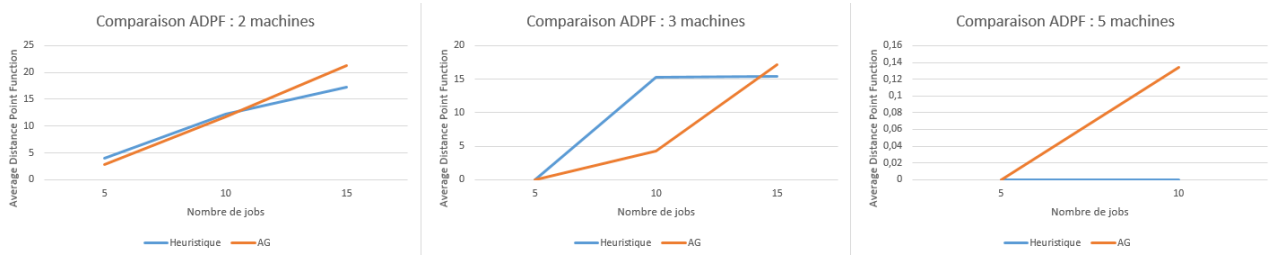


FIGURE 3.25 – Comparaison des "Average Distance Point Function"

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne la moyenne des distances entre les solutions pour chacun des fronts de Pareto. Sur chacun de ces graphes, on a le résultat de cette moyenne obtenue pour chaque méthode en fonction du nombre de jobs.

Pour 2 et 3 machines, on voit que l'heuristique propose des choix plus divers en dessous de 10 jobs. A 15 jobs c'est l'inverse. Pour 5 machines, on voit certes un écart, mais si on regarde la valeur maximale de cet écart, elle n'est que de 0.14, ce qui est très faible. Les deux méthodes proposent donc des solutions peu diverses les unes des autres.

Écarts avec le front exact

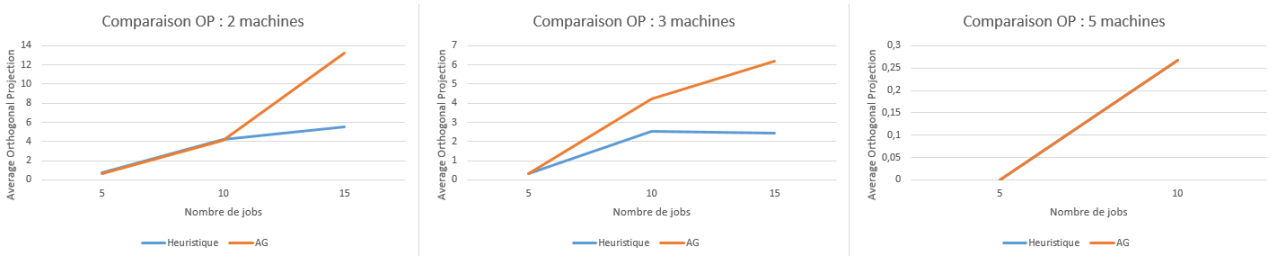


FIGURE 3.26 – Comparaison de l'écart entre les fronts approchés et le front exact

Ci-dessus, les trois graphes représentent la comparaison entre les deux méthodes approchées pour un certain nombre de machines en ce qui concerne l'écart entre leur front et le front exact. Cet écart est mesuré par la moyenne des distances les plus courtes entre les solutions du front approché et le front exact. Sur chacun de ces graphes, on a le résultat de cette moyenne obtenue pour chaque méthode en fonction du nombre de jobs.

Ici, on remarque facilement que l'heuristique se comporte mieux que l'AG pour 2 et 3 machines. En ce qui concerne 5 machines, les deux courbes se chevauchent, elles sont donc équivalentes.

Conclusion

Pour ce problème, on peut dire, suite aux trois études ci-dessus, que l'application de l'approche ε-contrainte se comporte mieux que l'AG. Elle donne plus de solutions exactes, un choix aussi divers que l'AG et un front proche du front exact.

3.7 Le programme

3.7.1 Entrées

Le programme prend plusieurs données en entrée :
- le chemin du dossier racine contenant les instances
- le chemin du dossier de sortie
- le type de problème à résoudre
  - 1 : $Pm|d_i|P(C^A_{max}, \sum U^B_i)$ avec l'approche epsilon-contrainte
  - 2 : $Pm|d_i|P(\sum U^A_i, C^B_{max})$ avec l'approche epsilon-contrainte
  - 3 : $Pm|d_i|P(C^A_{max}, \sum U^B_i)$ avec l'algorithme génétique
  - 4 : $Pm|d_i|P(\sum U^A_i, C^B_{max})$ avec l'algorithme génétique

3.7.2 Sorties

Le programme retourne plusieurs fichiers en sortie :
- pour chaque instance, un fichier de résultats est placé dans le dossier de sortie
- pour chaque nombre de jobs, un fichier "temps.txt"
- un fichier répertoriant les séquences trouvées pour chaque instance

3.7.3 Fonctionnement du programme

Le programme se lance en ligne de commande en fournissant l'ensemble des informations nécessaires (Cf : Entrées). Une fois lancé, il va récupérer l'ensemble des fichiers dans l'arborescence du dossier racine (il faut respecter certaines conventions, pour cela se référer à la doc utilisateur du programme).

Une fois toutes les instances récupérées, il exécute le problème suivant le type fourni en entrée. Les résultats sont mis dans un fichier. Ces fichiers contiennent donc les solutions de Pareto du problème. De plus un fichier "temps.txt" regroupant les durées et le nombre de solutions trouvées pour chaque instance est créé pour chaque nombre de jobs.

Déroulement du projet

4.1 Gestion du projet

4.1.1 Les séances

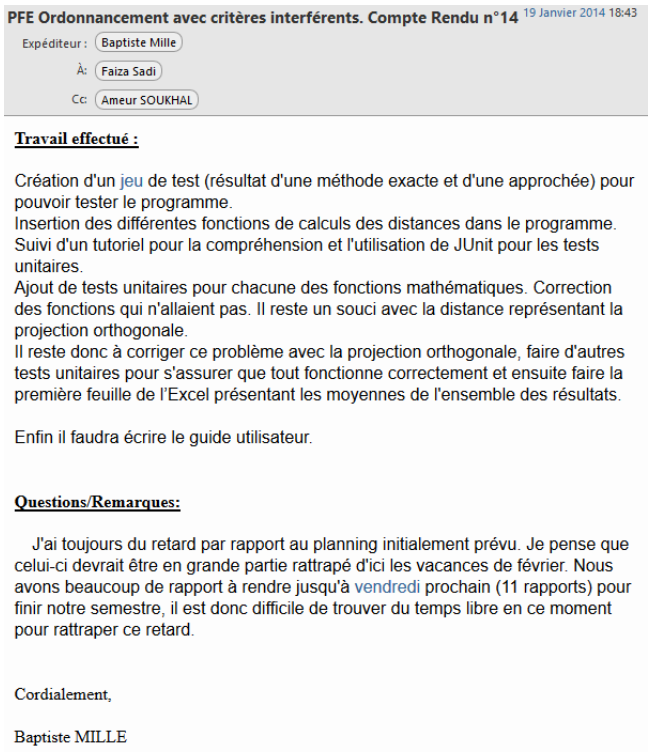
La plus grande partie du travail de ce PFE (90% environ) se sera effectué durant les séances prévues à cet effet. Pendant le premier semestre, elles ont eu lieu tous les jeudis de 8.15 à 18.15. Lors du second semestre, il y avait trois jours réservé pour le PFE : mardi, mercredi et jeudi de 8.15 à 18.15.

En plus de ces horaires, du travail a été effectué pendant des temps libres afin de rattraper du retard ou prendre de l'avance par exemple (10% environ).

4.1.2 Communication MOA MOE

Afin de tenir au courant la MOA de l'avancement du PFE, un mail hebdomadaire leur était envoyé. Celui était composé de la manière suivante :

- Travail réalisé
- Questions/Remarques



PFE Ordonnancement avec critères interférents. Compte Rendu n°14 19 Janvier 2014 18:43

Expéditeur : Baptiste Mille
À: Faiza Sadi
Cc: Ameur SOUKHAL

Travail effectué :

Création d'un jeu de test (résultat d'une méthode exacte et d'une approchée) pour pouvoir tester le programme.
Insertion des différentes fonctions de calculs des distances dans le programme.
Suivi d'un tutoriel pour la compréhension et l'utilisation de JUnit pour les tests unitaires.
Ajout de tests unitaires pour chacune des fonctions mathématiques. Correction des fonctions qui n'allaient pas. Il reste un souci avec la distance représentant la projection orthogonale.
Il reste donc à corriger ce problème avec la projection orthogonale, faire d'autres tests unitaires pour s'assurer que tout fonctionne correctement et ensuite faire la première feuille de l'Excel présentant les moyennes de l'ensemble des résultats.

Enfin il faudra écrire le guide utilisateur.

Questions/Remarques:

J'ai toujours du retard par rapport au planning initialement prévu. Je pense que celui-ci devrait être en grande partie rattrapé d'ici les vacances de février. Nous avons beaucoup de rapport à rendre jusqu'à vendredi prochain (11 rapports) pour finir notre semestre, il est donc difficile de trouver du temps libre en ce moment pour rattraper ce retard.

Cordialement,

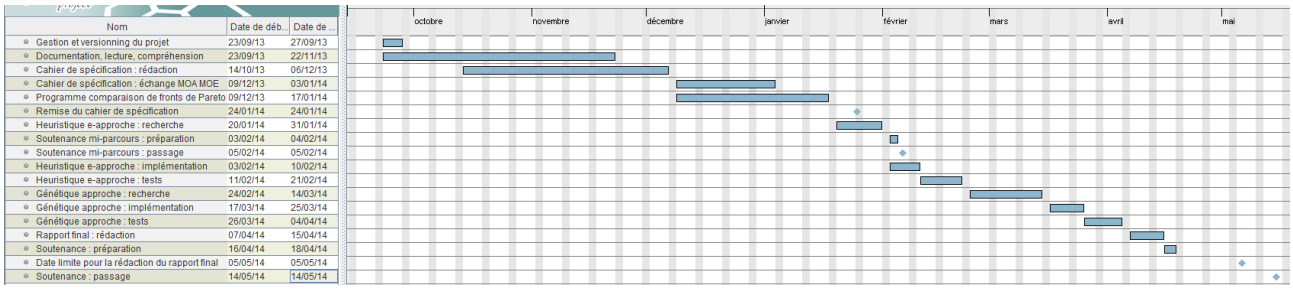
Baptiste MILLE

FIGURE 4.1 – Exemple de compte rendu hebdomadaire

En plus de ces échanges, de nombreuses réunions ont eu lieu afin que la MOA et la MOE soient en accord sur le travail en cours, sur les choix effectués en les critiquant afin de les améliorer, etc.

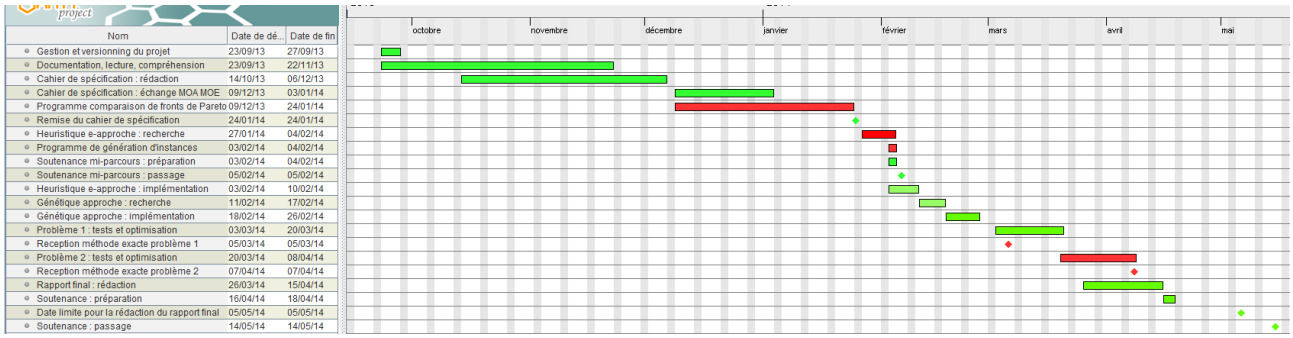
Tout ceci a permis d'avoir une bonne communication entre la MOA et la MOE tout au long de ce projet et de rester dans le périmètre défini dans le cahier de spécification.

4.2 Diagrammes de Gantt



Nom	Date de déb...	Date de ...
Gestion et versionning du projet	23/09/13	27/09/13
Documentation, lecture, compréhension	23/09/13	22/11/13
Cahier de spécification : rédaction	14/10/13	06/12/13
Cahier de spécification : échange MOA MOE	09/12/13	03/01/14
Programme comparaison de fronts de Pareto	09/12/13	17/01/14
Remise du cahier de spécification	24/01/14	24/01/14
Heuristique e-approche : recherche	20/01/14	31/01/14
Soutenance mi-parcours : préparation	03/02/14	04/02/14
Soutenance mi-parcours : passage	05/02/14	05/02/14
Heuristique e-approche : implémentation	03/02/14	10/02/14
Heuristique e-approche : tests	11/02/14	21/02/14
Génétique approche : recherche	24/02/14	14/03/14
Génétique approche : implémentation	17/03/14	25/03/14
Génétique approche : tests	26/03/14	04/04/14
Rapport final : rédaction	07/04/14	15/04/14
Soutenance : préparation	16/04/14	18/04/14
Date limite pour la rédaction du rapport final	05/05/14	05/05/14
Soutenance : passage	14/05/14	14/05/14

FIGURE 4.2 – Diagramme de Gantt initial



Nom	Date de dé...	Date de fin
Gestion et versionning du projet	23/09/13	27/09/13
Documentation, lecture, compréhension	23/09/13	22/11/13
Cahier de spécification : rédaction	14/10/13	06/12/13
Cahier de spécification : échange MOA MOE	09/12/13	03/01/14
Programme comparaison de fronts de Pareto	09/12/13	24/01/14
Remise du cahier de spécification	24/01/14	24/01/14
Heuristique e-approche : recherche	27/01/14	04/02/14
Programme de génération d'instances	03/02/14	04/02/14
Soutenance mi-parcours : préparation	03/02/14	04/02/14
Soutenance mi-parcours : passage	05/02/14	05/02/14
Heuristique e-approche : implémentation	03/02/14	10/02/14
Génétique approche : recherche	11/02/14	17/02/14
Génétique approche : implémentation	18/02/14	26/02/14
Problème 1 : tests et optimisation	03/03/14	20/03/14
Reception méthode exacte problème 1	05/03/14	05/03/14
Problème 2 : tests et optimisation	20/03/14	08/04/14
Reception méthode exacte problème 2	07/04/14	07/04/14
Rapport final : rédaction	26/03/14	15/04/14
Soutenance : préparation	16/04/14	18/04/14
Date limite pour la rédaction du rapport final	05/05/14	05/05/14
Soutenance : passage	14/05/14	14/05/14

FIGURE 4.3 – Diagramme de Gantt final

Dans la figure ci-dessus, les tâches en vert représentent celles qui étaient prévues et qui ont fini dans les temps et en respectant le nombre de jours/homme. En rouge se sont des tâches qui ont soit finies en retard, soit qui n'étaient pas prévues ou qui on du consommer plus de jours/homme.

4.2.1 Tâche 1 : gestion et versionning du projet

Description de la tâche

Cette tâche a servi à déployer une solution pour la gestion et le versionning du projet.Pour cela un redmine ainsi qu'un svn ont été mis à disposition par l'école. Une petite formation a été donnée à la MOA sur l'utilisation de ces outils.

Charge

Cette tâche a couté 0,5 jour/homme. Elle s'est déroulée du 23/09/2013 au 27/09/2013.

4.2.2 Tâche 2 : documentation, lecture et compréhension

Description de la tâche

Cette tâche a permis de comprendre le projet et d'en définir son périmètre. Tout au long du projet, cette tâche a été reprise pour obtenir des informations supplémentaires et des idées pour les algorithmes notamment. Elle s'est composée de lecture d'articles et de documents concernant notre sujet, mais aussi d'informations fournies par la MOA et de réunions entre la MOA et la MOE.

Livrables

Un rapport résumant l'article [2] a été fourni, les algorithmes présents dans celui-ci ont servi de base de réflexion pour nos heuristiques.

Charge

Cette tâche a couté 5 jours/homme. Elle s'est déroulée du 23/09/2013 au 22/11/2013.

4.2.3 Tâche 3 : rédaction du cahier des spécifications

Description de la tâche

Cette tâche a consisté en la rédaction du cahier de spécifications. Celui-ci permet de définir le contexte, le périmètre, les systèmes et leurs fonctionnalités ainsi que le déroulement du projet à l'aide d'un planning.

Livrables

Un cahier de spécification a été rendu.

Charge

Cette tâche a couté 4.5 jours/homme. Elle s'est déroulée du 14/10/2013 au 10/12/2013.

4.2.4 Tâche 4 : échange du cahier des spécifications entre la MOA et la MOE

Description de la tâche

Une fois le premier jet du cahier de spécification fait, celui-ci a subi des navettes entre la MOA et la MOE afin que ces deux partis se mettent en accord.

Livrables

Un cahier de spécification final a été rendu le 24/01/2014. Celui-ci a été validé par la MOA et la MOE. Il a été rendu plus tard, car la date de remise initialement prévue par l'école a été décalée. Du coup quelques échanges supplémentaires entre la MOA et la MOE ont eu lieu.

Charge

Cette tâche a couté 1 jour/homme. Elle s'est déroulée du 09/12/2013 au 03/01/2014.

4.2.5 Tâche 5 : Programme de comparaison de fronts de Pareto

Description de la tâche

Cette tâche a consisté à effectuer un programme. Le but de ce programme était de pouvoir comparer deux fronts de Pareto et de retourner les résultats dans un fichier .xls.

Livrables

Un guide d'utilisateur et le programme ont été fournis.

Charge

Cette tâche a couté 5 jours/homme. Cette estimation prenait en compte le développement du programme ainsi que la rédaction du guide d'utilisateur. Elle s'est déroulée du 19/12/2013 au 24/01/2014. Pour cause de maladie, cette tâche a pris une semaine de plus que prévu.

4.2.6 Tâche 6 : recherche d'une heuristique basée sur l'ϵ-approche

Description de la tâche

Nos premières heuristiques sont basées sur l'ϵ-contrainte. Cette tâche était composée de recherche dans l'état de l'art et de création d'heuristiques appropriées à nos deux sous problèmes.

Livrables

Le livrable de cette tâche a été les deux algorithmes.

Charge

Cette tâche a couté 6 jours/homme. Cette estimation tenait compte des échanges qui ont été effectués entre la MOA et la MOE sur ces algorithmes. Elle s'est déroulée du 20/01/2013 au 04/02/2013. Elle a donc été décalée d'une semaine due au retard de la tâche 5.

4.2.7 Tâche 7 : Préparation soutenance mi-parcours

Description de la tâche

Préparation de la soutenance de mi-parcours.

Livrables

Soutenance avec PowerPoint. La soutenance s'est déroulée le 05/02/2014.

Charge

Cette tâche a couté 0.5 jour/homme. Elle s'est déroulée du 03/02/2014 au 04/02/2014.

4.2.8 Tâche 8 : Générateur d'instance

Description de la tâche

Création d'un programme pour générer des instances

Livrables

Un guide utilisateur a été fourni ainsi que le programme.

Charge

Cette tâche n'avait pas été prévue, elle a couté 1 jour/homme. Cette journée prend en compte les renseignements pour la génération d'instances, l'implémentation et le guide utilisateur. Elle s'est déroulée du 03/02/2014 au 04/02/2014.

4.2.9 Tâche 9 : implémentation des heuristiques

Description de la tâche

Une fois les deux heuristiques validées, celles-ci ont été implémentées.

Livrables

Un guide d'utilisateur a été fourni ainsi que le programme.

Charge

Cette tâche a couté 3.5 jours/homme. Elle s'est déroulée du 03/02/2014 au 10/02/2014.

4.2.10 Tâche 10 : application d'une approche génétique

Description de la tâche

Une fois l'approche epsilon contrainte implémentée, il fallait mettre en place des algorithmes génétiques pour résoudre nos problèmes. Des recherches ont donc été effectuées pour appliquer la génétique à nos problèmes.

Livrables

Le livrable de cette tâche a été des algorithmes.

Estimation de charge

Cette tâche était estimée à 6 jours/homme. Cette estimation tenait compte des échanges effectués entre la MOA et la MOE sur ces algorithmes. Elle s'est déroulée du 24/02/2014 au 14/03/2014.

4.2.11 Tâche 11 : implémentation des algorithmes génétiques

Description de la tâche

Une fois les algorithmes génétiques validés par la MOA ils ont été implémentés.

Livrables

Un guide utilisateur a été fourni ainsi que le programme.

Estimation de charge

Cette tâche a couté 4 jours/homme. Elle s'est déroulée du 17/03/2014 au 25/03/2014.

4.2.12 Tâche 12 : tests, correction et optimisation sur le premier problème

Description de la tâche

Cette tâche a consisté à effectuer de nombreux tests sur notre implémentation afin de vérifier le bon fonctionnement des algorithmes. Puis d'optimiser pour améliorer les résultats obtenus.

Livrables

Un guide d'utilisateur a été fourni ainsi que le programme.

Estimation de charge

Cette tâche a couté 6 jours/homme. Elle devait se dérouler du 11/02/2014 au 21/02/2014. Toutefois sachant que la méthode exacte ne serait pas disponible a cette période, elle a été reportée pour être effectuée entre le 03/03/2014 au 20/03/2014. Les résultats de la méthode exacte ont été fournis le 05/03/2014.

4.2.13 Tâche 13 : tests, correction et optimisation sur le second problème

Description de la tâche

Cette tâche a consisté à effectuer de nombreux tests sur notre implémentation afin de vérifier le bon fonctionnement des algorithmes. Puis d'optimiser afin d'améliorer les résultats obtenus.

Livrables

Un guide d'utilisateur a été fourni ainsi que le programme.

Charge

Cette tâche a couté 6 jours/homme. Elle devait se dérouler du 26/03/2014 au 04/04/2014. Les résultats de la méthode exacte sont arrivés le 06/04/2014, il a donc fallu déborder un petit peu sur cette tâche. Elle se sera donc terminée le 09/04/2014.

4.2.14 Tâche 14 : rédaction du rapport final

Description de la tâche

Cette tâche consistait en la rédaction du rapport final de ce PFE, afin de rendre compte du projet. Celui-ci pouvait être, malgré la date de début, rédigé tout au long de l'année.

Livrables

Le livrable de cette tâche était le rapport en lui-même.

Charge

Cette tâche a couté à 4 jours/hommes. Elle s'est déroulée du 07/04/2014 au 15/04/2014.

4.2.15 Tâche 15 : préparation de la soutenance

Description de la tâche

Cette tâche consistait à préparer la soutenance de ce PFE avec la réalisation d'un diaporama sur le projet et les résultats obtenus.

Livrables

Le livrable de cette tâche était le code source, les exécutables du projet et le PowerPoint de la soutenance.

Estimation de charge

Cette tâche a couté 2 jours/hommes. Elle s'est déroulée du 16/04/2014 au 18/04/2014.

Conclusion

Ce projet m'a permis de mettre en pratique et d'approfondir l'ensemble des enseignements reçu lors de mon cursus scolaire concernant : l'ordonnancement, les algorithmes, le développement java et la gestion de projet.

Ce projet a été le plus complet de tout mon cursus scolaire. En effet il a regroupé toutes les étapes d'un cycle en V à effectuer seul, et ce, sur 52 jours/homme.

J'ai eu la chance de faire un projet qui m'a plu et qui m'a confirmé l'envie de travailler dans ce domaine un jour dans ma vie professionnelle.

En perspectives d'amélioration dans le futur, il pourrait être intéressant de tester une autre méthode de croisement pour la génétique. De plus, coupler pour chaque problème l'heuristique avec la génétique améliorerait les résultats obtenus (en ajoutant à la population initiale de l'AG, les solutions trouvées par l'heuristique). Pour allez plus loin avec la résolution du problème 2, une amélioration de la méthode exacte sera nécessaire pour qu'elle puisse traiter de plus grosses instances.

Pour terminer, nous avons travaillé tout au long de ce projet sur des problèmes multicritères avec 2 agents. Mais l'ensemble du projet a été pensé pour qu'il puisse par la suite, facilement être modifié pour rajouter d'autres agents. Toutefois, l'ajout d'agents augmente grandement la complexité du problème et il sera difficile de trouver une heuristique fournissant de bons résultats dans un temps raisonnable, et encore plus pour les méthodes exactes.

Bibliographie

[1] Jean-Charles Billaut and Vincent T'kindt. Multicriteria scheduling. *Springer*, 2006.

[2] Johnny C.Ho and Yih-Long Chang. Minimizing the number of tardy jobs for m parallel machines. *ELSEVIER*, 1995.

[3] Bassem Jarboui, Patrick Siany, and Jacques Tegha. Metaheuristiques pour l'ordonnancement multicritere et les problemes de transports. *Hermes Science*, 2014.

[4] Faiza Sadi, Ameur Soukhal, and Jean-Charles Billaut. Solving multi-agent scheduling problems on parallel machines with a global objective function (page 255 - 269). *RAIRO*, 2014.

Méthodes approchées pour la résolution d'un problème d'ordonnancement avec travaux interférants

Département Informatique
5e année
2013 - 2014

Rapport Final de PFE

Résumé : Dans ce PFE, nous nous sommes intéressés à une classe particulière des problèmes d'ordonnancement, appelée "Ordonnancement avec travaux interférants". Ces problèmes supposent que les travaux sont répartis sur un ensemble d'agents et que chaque agent vise à minimiser un critère appliqué sur ses travaux. Dans ce projet, nous avons considéré deux types de critère : minimisation de date maximum de fin et minimisation du nombre de travaux en retard. Deux problèmes sont analysés. Des méthodes heuristiques et algorithmes génétiques ainsi qu'une comparaison avec les résultats exacts sont proposés.

Mots clefs : Ordonnancement avec travaux interférants, Approche ε-contrainte, Front de Pareto, Algorithme Génétique, Heuristiques

Abstract: In this PFE, we are interested in a particular class of scheduling problems, called "Scheduling interfering jobs." These problems imply that jobs are distributed over a set of agent and each agent aims at minimizing a criterion applied to its jobs. In this project, we considered two types of criteria: minimization of maximum end date and minimizing the number of late works. Two problems are analyzed. Heuristics and genetic algorithms and a comparison with the exact results are available.

Keywords: Scheduling problem with Interfering jobs, ε-constraints approch, Pareto Front, Genetic Algorithm, Heuristic

---

Encadrants
Faiza Sadi
faiza.sadi@univ-tours.fr
Ameur Soukhal
ameur.soukhal@univ-tours.fr

Étudiant
Baptiste Mille
baptiste.mille@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours