



Ecole Polytechnique de l'Université François Rabelais de Tours

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**

**2018-2019**

# **Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO**

**Entreprise**



**Tuteur entreprise**

**Sebastien LANNEZ** (Ingénieur)

**Étudiant**

**François GAUCHER** (DI5)

**Tuteur académique**

**Vincent T'KINDT**

# Liste des intervenants

## Entreprise

FICO  
Paris

[www.fico.com/fr/](http://www.fico.com/fr/)



Nom	Email	Qualité
François GAUCHER	<a href="mailto:francois.gaucher@etu.univ-tours.fr">francois.gaucher@etu.univ-tours.fr</a>	Étudiant DI5
Vincent T'KINDT	<a href="mailto:tkindt@univ-tours.fr">tkindt@univ-tours.fr</a>	Tuteur académique, Département Informatique
Sebastien LANNEZ	<a href="mailto:SebastienLannez@fico.com">SebastienLannez@fico.com</a>	Tuteur entreprise, Ingénieur



# Avertissement

Ce document a été rédigé par François GAUCHER susnommé l'auteur.

L'entreprise FICO est représentée par Sébastien LANNEZ susnommé le tuteur entreprise.

L'École Polytechnique de l'Université François Rabelais de Tours est représentée par Vincent T'KINDT susnommé le tuteur académique.

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

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

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

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

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

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

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



## Pour citer ce document

François GAUCHER, *Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={GAUCHER, François},
  title={Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	v
<b>1 Introduction</b>	<b>1</b>
1 Acteurs, enjeux et contexte	1
1.1 Acteurs	1
1.2 Enjeux et Contexte	1
2 Objectifs	1
3 Hypothèses	2
4 Bases Méthodologiques	2
<b>2 Description générale</b>	<b>3</b>
1 Environnement du projet	3
2 Caractéristiques des utilisateurs	3
3 Fonctionnalités du système	4
4 Structure générale du système	4
<b>3 État de l'art</b>	<b>6</b>
1 Optimisation multi-objectif	6
1.1 Définition	6
1.2 Détermination de la solution	7

1.2.1	Pour le décideur .....	7
1.2.2	Pour le concepteur.....	7
1.3	Résolution .....	7
2	Optimum de Pareto .....	8
2.1	Définition .....	8
2.2	Les types de solution de Pareto.....	8
2.3	Détermination de l'optimum de Pareto .....	9
2.3.1	Détermination par combinaison convexe de critères.....	9
2.3.2	Détermination par analyse paramétrique .....	9
2.3.3	Détermination au moyen de l'approche par contrainte $\varepsilon$ .....	9
2.3.4	Utilisation de la métrique Tchebycheff .....	10
2.3.5	Utilisation de la métrique Tchebycheff pondérée.....	10
2.3.6	Utilisation de la métrique Tchebycheff pondérée augmentée .....	10
2.3.7	Détermination par l'approche d'objectif.....	10
2.3.8	Détermination par utilisation d'un ordre lexicographique .....	10
2.3.9	Détermination par utilisation d'un ordre lexicographique avec objectifs .....	10
2.4	Détermination du front de Pareto .....	10
3	Algorithme de Kirlik et Sayin .....	11
3.1	Objectif.....	11
3.2	Initialisation .....	11
3.3	Définition de l'espace de recherche .....	12
3.4	L'approche par contrainte $\varepsilon$ en 2 étapes .....	13
3.5	Analyse des résultats .....	13
4	Outils du projet .....	13
4.1	Fico Xpress .....	13
4.2	Mosel .....	14
<b>4</b>	<b>Analyse et conception</b> .....	<b>15</b>
1	Analyse de la fonction de l'algorithme de Kirlik et Sayin.....	15
1.1	Etape 0 : initialisation .....	15
1.2	Etape 1 : définition des bornes de la solution .....	16
1.3	Etape 2 : Recherche de la solution non-dominée.....	17
1.4	Etape 3 : Redéfinition des rectangles de recherche.....	17
1.5	Etape 4 : Affichage de la solution.....	18
2	Analyse du fichier test de Sébastien LANNEZ.....	19
3	Analyse des performances de l'algorithme de Kirlik et Sayin .....	20

<b>5</b>	<b>Mise en œuvre</b>	<b>21</b>
1	Problème d'ordonnancement bi-objectif.....	21
1.1	Présentation du problème .....	21
1.2	Modélisation du problème : fichier "Flowshop-problem.mos" .....	22
1.3	Comparaison des résultats .....	24
1.3.1	Pré-traitement selon les critères de Johnson : fichier "Johnson.c"..	24
1.3.2	Programme de tests : "campagnepourpapier.c" .....	25
1.3.3	Fichier de sortie de comparaison "Comparaison.txt" .....	25
1.4	Analyse des résultats .....	26
1.4.1	Analyse du temps d'exécution.....	26
1.4.2	Analyse du nombre de solutions trouvées .....	27
2	Problème d'ordonnancement à machines parallèles uniformes .....	27
2.1	Présentation et modélisation du problème.....	27
2.2	Modélisation du problème : fichier "Scheduling problem.mos" .....	28
3	Amélioration de la fonction de l'algorithme de Kirlik et Sayin.....	28
3.1	Définition des limites de la solution.....	28
3.2	Modification du calcul du volume de la boîte .....	29
3.3	Modification du découpage de la boîte .....	29
3.4	Fichier de sortie Mosel : "XpressResults.txt" .....	30
<b>6</b>	<b>Bilan et conclusion</b>	<b>31</b>
1	État d'avancement .....	31
1.1	Semestre 1 .....	31
1.2	Semestre 2.....	31
1.2.1	Tâches effectuées .....	31
1.2.2	Tâches à réaliser .....	31
1.2.3	Retard.....	32
2	Planning.....	32
2.1	Semestre 1 .....	32
2.2	Semestre 2.....	32
3	Bilan qualité.....	32
4	Bilan gestion de projet.....	33
<b>7</b>	<b>Annexes</b>	<b>34</b>
1	Description des interfaces externes du système .....	34
1.1	Interfaces matériel/logiciel .....	34
1.2	Interface homme/machine.....	34
1.3	Interface logicielle/logicielle .....	34
2	Spécifications fonctionnelles .....	34
2.1	Initialiser le problème .....	34

2.1.1	Identification de la fonction « Initialiser le problème » .....	34
2.1.2	Description de la fonction « Initialiser le problème » .....	35
2.2	Exécuter l'algorithme de Kirlik et Sayin .....	35
2.2.1	Identification de la fonction « Exécuter l'algorithme de Kirlik et Sayin » .....	35
2.2.2	Description de la fonction « Exécuter l'algorithme de Kirlik et Sayin » .....	35
2.3	Comparer les résultats .....	35
2.3.1	Identification de la fonction « Comparer les résultats » .....	35
2.3.2	Description de la fonction « Comparer les résultats » .....	35
2.4	Améliorer la fonction de l'algorithme de Kirlik et Sayin .....	36
2.4.1	Identification de la fonction « Améliorer la fonction de l'algorithme de Kirlik et Sayin » .....	36
2.4.2	Description de la fonction « Améliorer la fonction de l'algorithme de Kirlik et Sayin » .....	36
2.5	Informations complémentaires .....	36
3	Spécifications non fonctionnelles .....	36
3.1	Contraintes de développement et conception .....	36
3.2	Contraintes de fonctionnement et d'exploitation .....	36
3.2.1	Performances .....	36
3.2.2	Capacité .....	36
3.2.3	Contrôlabilité .....	37
3.2.4	Maintenance et évolution du système .....	37
4	Gestion de projet .....	37
4.1	Outils .....	37
4.2	Diagramme de Gantt semestre 1 .....	37
4.3	Diagramme de Gantt final .....	38
4.4	Diagramme de ressources .....	38
4.5	Diagramme de ressources final .....	39
5	Documentation de tests .....	39
6	Documentation développeur .....	51
7	Documentation utilisateur .....	61
8	Campagne de tests .....	66
9	Résultats des tests de performance du problème d'ordonnancement bi-objectif .....	68
<b>8</b>	<b>Sources</b>	<b>70</b>
<b>9</b>	<b>Comptes rendus hebdomadaires</b>	<b>71</b>
<b>10</b>	<b>Bibliographie</b>	<b>75</b>
<b>11</b>	<b>Acronymes</b>	<b>78</b>



# Table des figures

## 1 Introduction

1	Schéma du cycle en V .....	2
---	----------------------------	---

## 2 Description générale

1	Schéma de l'environnement du projet .....	3
2	Diagramme de cas d'utilisation .....	4
3	Diagramme de composant .....	5

## 3 État de l'art

1	Espace décisionnel et espace objectif d'un problème d'optimisation multi-objectif (exemple avec deux variables de décision) .....	7
2	Représentation graphique des solutions de Pareto .....	8
3	Représentation graphique du front de Pareto sur un problème bi-objectif .....	9
4	Solutions dans $R^3$ et projection des points sur le plan $f_2 - f_3$ .....	11
5	Vue de l'environnement Xpress Workbench .....	14

## 4 Analyse et conception

1	Code Mosel de l'étape 0 .....	16
2	Code Mosel de l'étape 1 .....	16
3	Code Mosel de l'étape 2 (partie 1) .....	17
4	Code Mosel de l'étape 2 (partie 2) .....	17
5	Code Mosel de l'étape 3 .....	18
6	Code Mosel de l'étape 4 .....	18
7	Affichage console du résultat avec l'algorithme de Kirlik et Sayin .....	19

**5 Mise en œuvre**

1	Diagramme de classe de l'implémentation sous Xpress.....	23
2	"Conversion" Mosel du modèle mathématique.....	23
3	Diagramme de classe du projet "campagnepourpapier" .....	24
4	Exemple de fichier Statistiques générés .....	25
5	Exemple de fichier "Comparaison.txt" .....	26
6	Exemple de sortie d'exécution pour 1500 données.....	27
7	Définition des variables "HV" et "LV" .....	29
8	Fonction "updlogarea" modifiée .....	29
9	Fonction "mmoint kirliksayin split" modifiée.....	29
10	Exemple d'affichage de "XpressResults.txt" .....	30

**7 Annexes**

1	Diagramme de Gantt début du semestre 2 .....	37
2	Diagramme de Gantt fin du semestre 2 .....	38
3	Diagramme de Ressources .....	38
4	Diagramme de Ressources .....	39

# 1

## Introduction

### 1 Acteurs, enjeux et contexte

#### 1.1 Acteurs

La **maîtrise d'ouvrage (MOA)** est composée de 2 personnes :

- Sébastien LANNÉZ, le tuteur d'entreprise qui représente ici la société FICO ;
- Vincent T'KINDT, le tuteur académique.

La **maîtrise d'œuvre (MOE)** est François GAUCHER (moi)

#### 1.2 Enjeux et Contexte

Aujourd'hui, aucun solveur mathématique commercial n'inclus des outils pour résoudre des problèmes faisant intervenir plusieurs critères à optimiser. L'ambition de FICO est, en partenariat avec l'équipe Recherche Opérationnelle, Ordonnancement et Transport du LIFAT, d'inclure dans MOSEL des outils permettant de résoudre des problèmes d'optimisation multi-objectif.

### 2 Objectifs

L'objectif du PR&D, dans ce contexte, va être de :

- 1) valider et expérimenter au sein du langage Mosel un algorithme d'optimisation multi-objectif : l'algorithme de Kirlik et Sayin ;
- 2) comparer expérimentalement l'algorithme sur des problèmes multi-objectifs par rapport à des algorithmes existants et conçus pour ces problèmes ;
- 3) évaluer la pertinence de l'algorithme programmé (quitte à l'améliorer par la suite le cas échéant).

Suite à ça, L'entreprise devra avoir un compte-rendu des différentes expérimentations et des possibles changements effectués sur l'algorithme implémenté.

### 3 Hypothèses

Pour pouvoir évaluer les capacités de l'algorithme de Kirlik et Sayin, il faudra d'une part vérifier que l'algorithme a bien été implémentée dans MOSEL puis évaluer ses performances propres selon plusieurs problèmes multi-objectifs.

### 4 Bases Méthodologiques

Dans le cadre du développement de ce projet, nous utilisons le solveur Xpress avec le langage de programmation et de modélisation Mosel.

Le code programmé par le tuteur entreprise est récupéré à l'aide de l'outil "Github".

Au niveau de la gestion du projet, on utilisera une méthodologie en cycle en V classique.

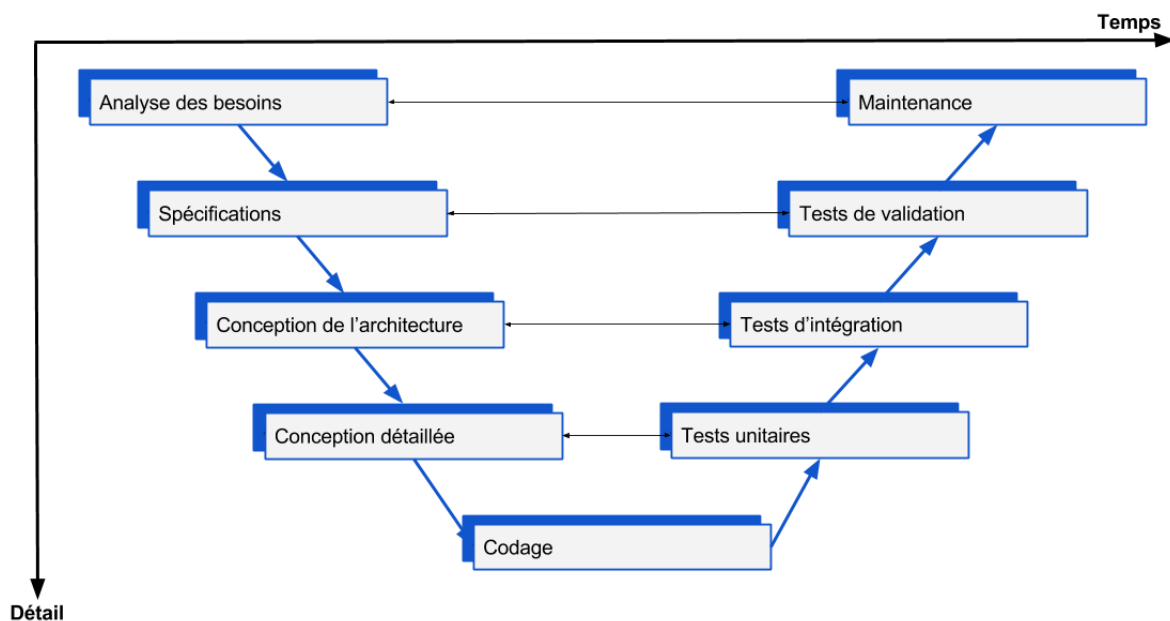


Figure 1 – Schéma du cycle en V

Un échange continu avec la MOA permettra de définir l'analyse des besoins et les spécifications. La partie conception et codage correspond à la compréhension et à l'analyse du code de l'algorithme de Kirlik et Sayin. Cette phase inclut de plus la compréhension des problèmes multi-objectifs à traiter et leur modélisation sur le solveur Xpress.

Enfin, les phases de tests et de maintenance correspondent à l'utilisation de l'algorithme sur des problèmes multi-objectifs et des possibles améliorations du code de l'algorithme. L'algorithme sera testé selon plusieurs critères (validité du résultat, temps d'exécution...) et une phase de maintenance aura lieu pour améliorer l'algorithme si les tests ne sont pas concluants.

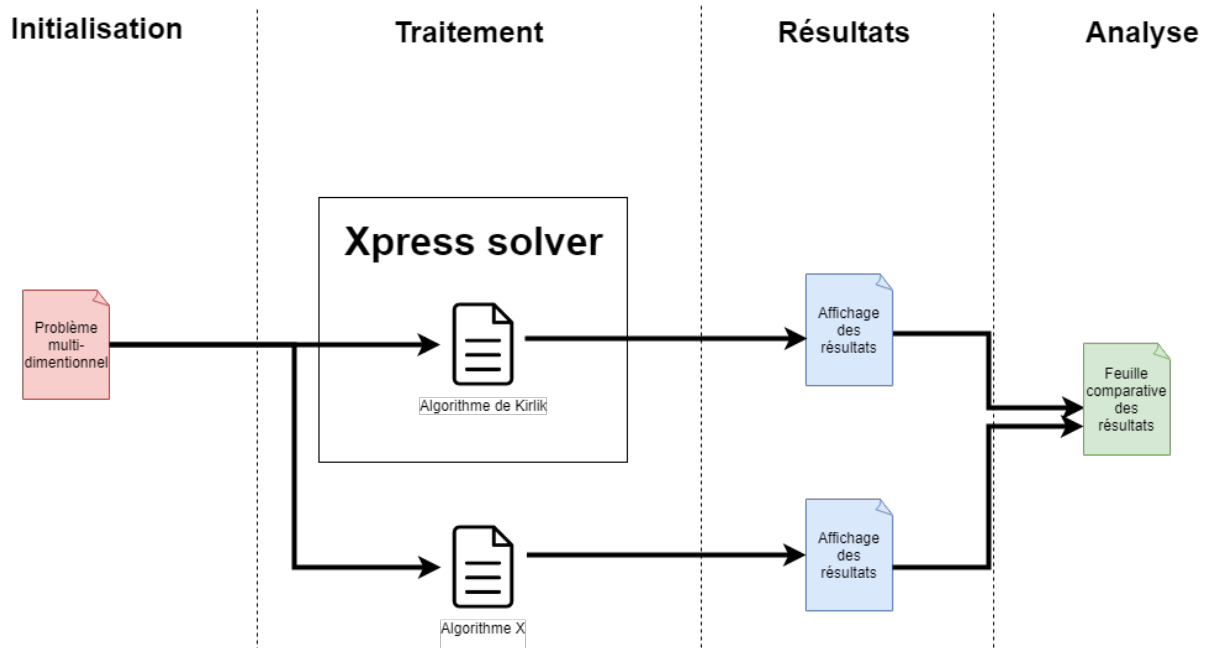
L'utilisation d'un diagramme de Gantt permettra de définir un planning à respecter pour la réalisation du projet (cf [Section 4.2](#) (Chapitre 7)).

# 2

## Description générale

### 1 Environnement du projet

L'ensemble du projet se déroulera essentiellement sur le solveur Xpress



**Figure 1** – Schéma de l'environnement du projet

Pour un problème multi-objectif donné, le solveur va exécuter l'algorithme de Kirlik et Sayin. Un algorithme X dédié à un problème type va aussi être exécuté.

A la fin du traitement, une feuille de résultat sera créée et va comparer les 2 algorithmes selon plusieurs critères (temps d'exécution, résultats donnés...) pour ainsi savoir si l'algorithme de Kirlik et Sayin est efficace.

### 2 Caractéristiques des utilisateurs

En se focalisant sur l'utilisation du solveur, nous pouvons identifier deux types d'utilisateur :

- **l'utilisateur "testeur"** : ce profil est destiné au chercheur qui teste les différents algorithmes proposées et analyse les résultats produits. Cet utilisateur modifie uniquement les paramètres du problème ;
- **l'utilisateur "développeur"** : ce profil est destiné aux développeurs qui testent les différentes fonctions basées sur les algorithmes. Cet utilisateur modifie le code afin d'améliorer les fonctions et/ou les résultats donnés.

Pour ce projet, nous serons d'une part "testeur" mais aussi "développeur" selon les résultats produits.

### 3 Fonctionnalités du système

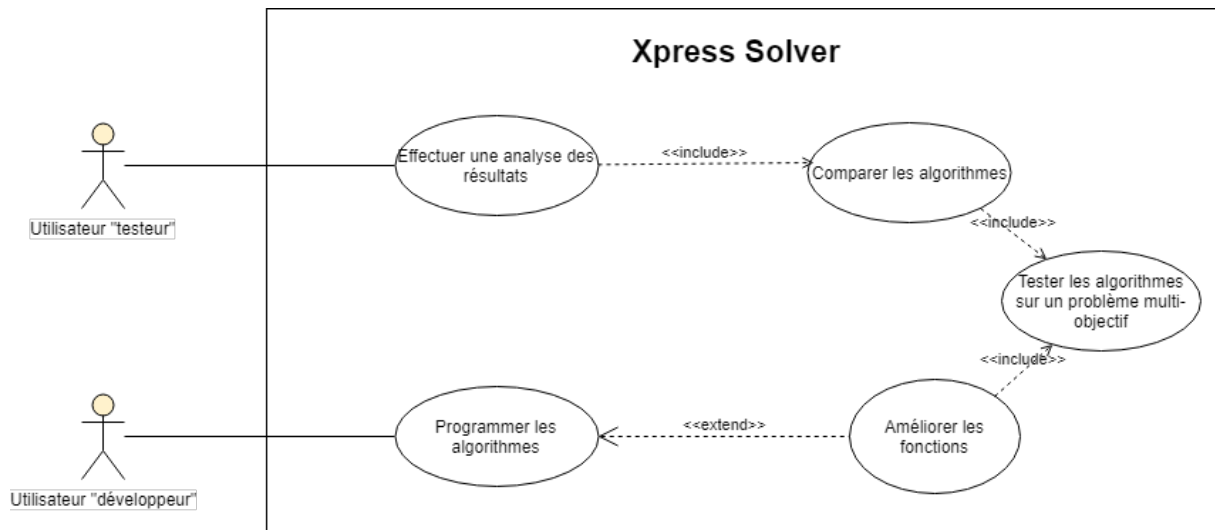


Figure 2 – Diagramme de cas d'utilisation

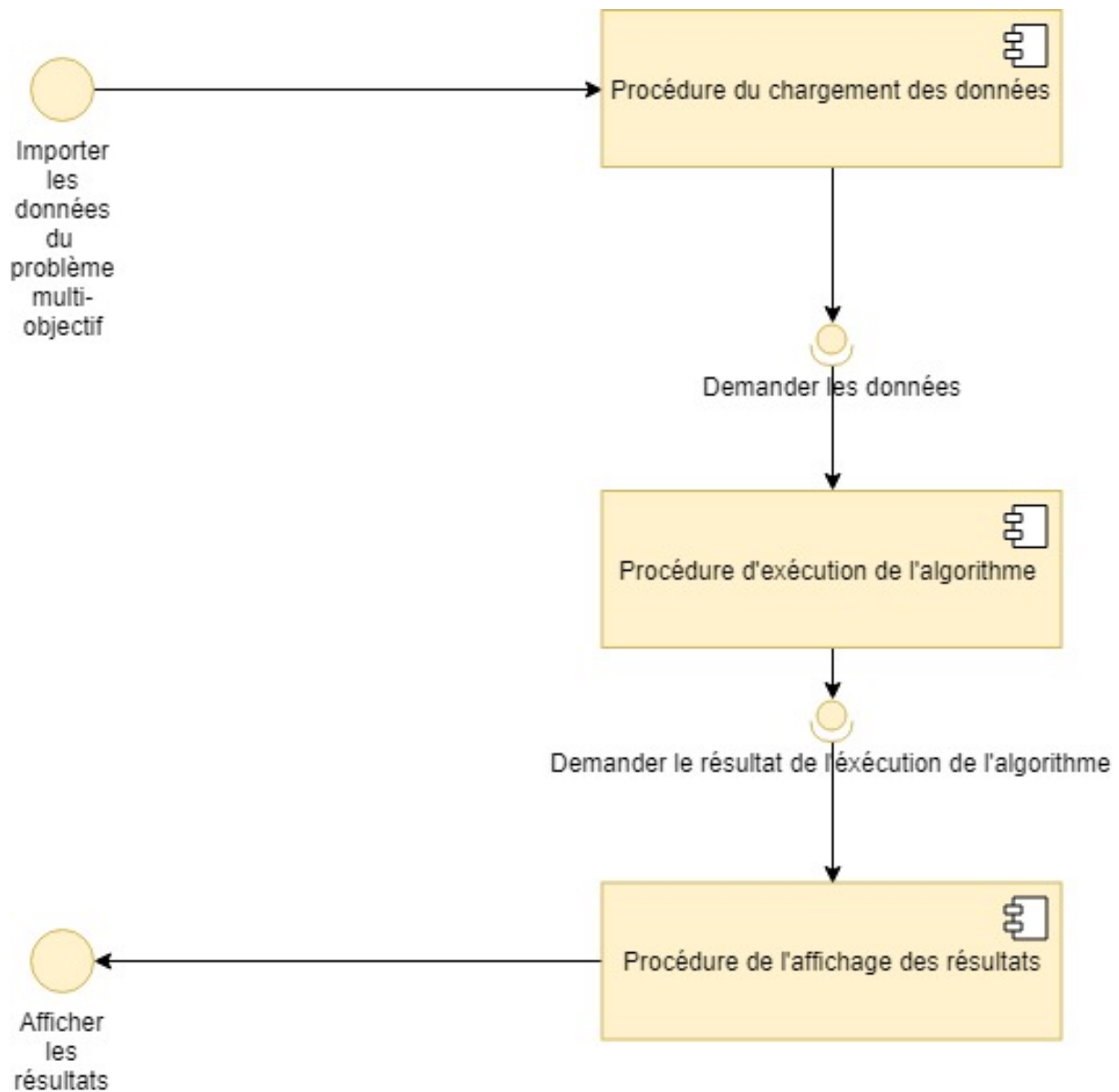
Ce diagramme présente les différentes actions exécutées par les deux types d'utilisateur énoncés précédemment.

Le système utilisé et créé par la société FICO permet aux utilisateurs de réaliser le projet.

Dans une optique d'amélioration, les deux types d'utilisateur doivent communiquer afin d'optimiser l'algorithme de Kirlik et Sayin.

### 4 Structure générale du système

Nous pouvons représenter la structure interne du solveur Xpress à l'aide d'un diagramme de composants.



**Figure 3** – *Diagramme de composant*

Dans ce diagramme, il y a 3 composants principaux, deux interfaces de service et deux interfaces de demande. L'utilisateur devra toujours importer les données du problème multi-objectif.

Lorsque le système exécute le composant « Procédure d'exécution de l'algorithme », il demande les données importées au composant « Procédure du chargement de données » par l'interface de demande «Demander les données », dont le composant « Procédure du chargement de données » fournit aussi une interface de service « Importer les données du problème multi-objectif » à l'utilisateur pour lui faire importer les données.

Après l'exécution de l'algorithme , le composant « Procédure de l'affichage de résultat » demande le résultat au composant « Procédure d'exécution de l'algorithme » et l'affiche dans la console du système.

# 3

## État de l'art

### 1 Optimisation multi-objectif

#### 1.1 Définition

La plupart des problèmes d'optimisation réels sont décrits à l'aide de plusieurs objectifs ou critères devant être optimisés simultanément.

Un **problème d'optimisation multi-objectif (MOP)** consiste à déterminer la solution correspondant au mieux aux préférences du décideur parmi les solutions de bonne compromis.

Il peut être défini ainsi :

$$\text{Min} f(x) = f_1(x), f_2(x), \dots, f_n(x) \quad (1)$$

$$\text{s.t. } x \in X \quad (2)$$

Où  $n$  est le nombre d'objectifs ( $n \geq 2$ ),  $X$  est l'ensemble des solutions réalisable dans l'espace décisionnel, et  $x = (x_1, x_2, \dots, x_k) \in X$  est un vecteur représentant les variables de décision. A chaque solution  $x$  est associé un vecteur objectif  $z \in Z$  sur la base d'un vecteur de fonction  $f : X \rightarrow Z$  tel que  $z = (z_1, z_2, \dots, z_n) = f(x) = (f_1(x), f_2(x), \dots, f_n(x))$ , où  $Z = f(X)$  représente l'ensemble des points réalisables.



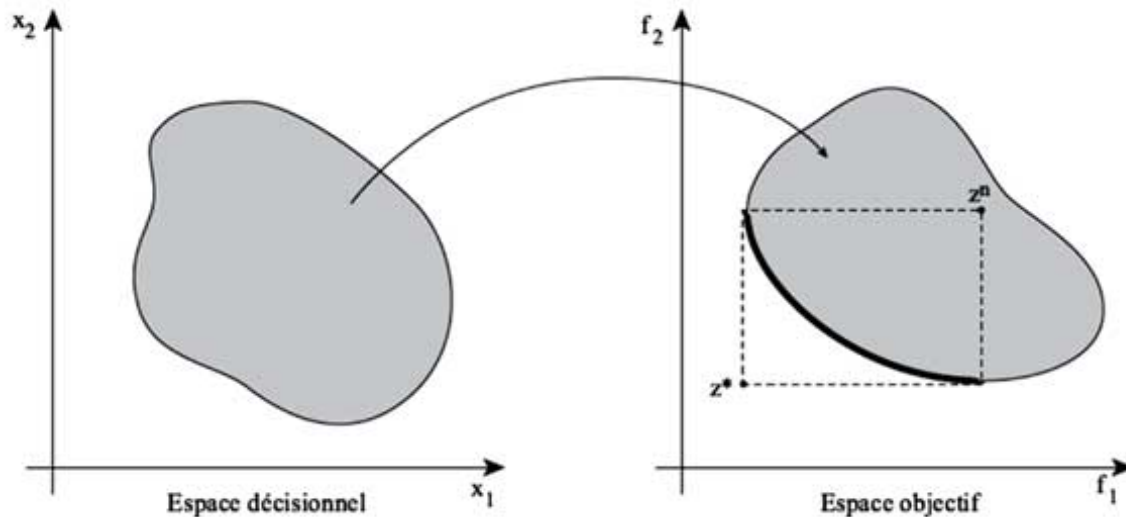


Figure 1 – Espace décisionnel et espace objectif d'un problème d'optimisation multi-objectif (exemple avec deux variables de décision).

## 1.2 Détermination de la solution

### 1.2.1 Pour le décideur

Il existe plusieurs méthodes permettant de déterminer la meilleure solution pour le “client” :

- 1) La méthode **a priori** permettant au décideur d'intervenir avant le processus de résolution. Cette méthode donne une vision précise des préférences du client.
- 2) La méthode **interactive** permettant au décideur d'intervenir pendant le cours du processus de résolution. Cette méthode donne une vision partielle mais réaliste de la demande. La méthode fonctionne selon un processus itératif où des discussions continues ont lieu avec le concepteur afin de trouver la solution adaptée.
- 3) La méthode **a posteriori** permettant au décideur d'intervenir après la résolution processus sont appelés. Cette méthode est la moins utilisée car elle demande à l'utilisateur de visualiser et choisir une solution parmi un grand nombre.

### 1.2.2 Pour le concepteur

Pour déterminer la solution, le concepteur peut décider d'adopter une approche de Pareto (cf [Section 2](#)) ou non. Il peut en effet transformer le problème multi-objectif en plusieurs problèmes mono-objectifs (approche non Pareto). Ce cas ne sera cependant pas étudié durant le projet.

## 1.3 Résolution

La résolution du problème multi-objectif se fait en 2 étapes :

- 1) La recherche des solutions de meilleur compromis : le concepteur cherche des solutions multi-objectifs optimales (cf [Section 2](#)) ;
- 2) le choix de la solution à retenir : le décideur choisit la solution qu'il utilisera.

## 2 Optimum de Pareto

### 2.1 Définition

Un optimum de Pareto est une allocation des ressources pour laquelle il n'existe pas une alternative dans laquelle tous les acteurs seraient dans une meilleure position. Une approche Pareto ne transforme pas les objectifs du problème, ceux-ci sont traités sans aucune distinction pendant la résolution. Cette approche permet de prendre en compte tous les critères du problème afin de trouver la solution la moins contraignante pour le décideur.

### 2.2 Les types de solution de Pareto

Il existe trois types de solution de Pareto :

#### La solution de Pareto stricte

Une solution  $a \in A$  est un optimum stricte de Pareto si il n'existe pas de solution  $b \in A$  telle que pour tout critère  $k$  :  $f^k(b) \leq f^k(a)$ . Ces solutions sont cherchées par l'algorithme de Kirlik et Sayin (cf [Section 3](#))

#### La solution de Pareto faible non stricte

Une solution  $a \in A$  est un optimum faible de Pareto si il existe une solution  $b \in A$  telle qu'il existe un critère  $k$  telle que, pour un critère  $k_1$  fixé, :  $f^{k_1}(b) = f^{k_1}(a)$  et  $f^k(b) < f^k(a)$

#### La solution dominée

Une solution  $a \in A$  est un optimum dominé de Pareto par une solution  $b \in A$  si il n'existe pas de critère  $k \in [1, \dots, n]$  telle que  $f^k(a) < f^k(b)$

Ainsi, si il n'existe pas de solution qui domine  $x$ , on dit que  $x$  est une solution de Pareto optimale ou stricte.

La figure ci-dessous permet de mettre en valeur la relation entre ces différentes solutions

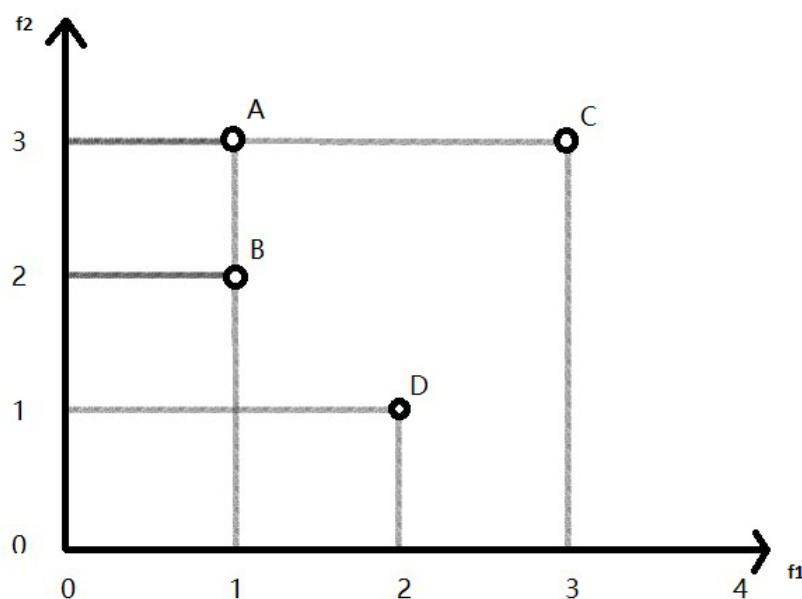
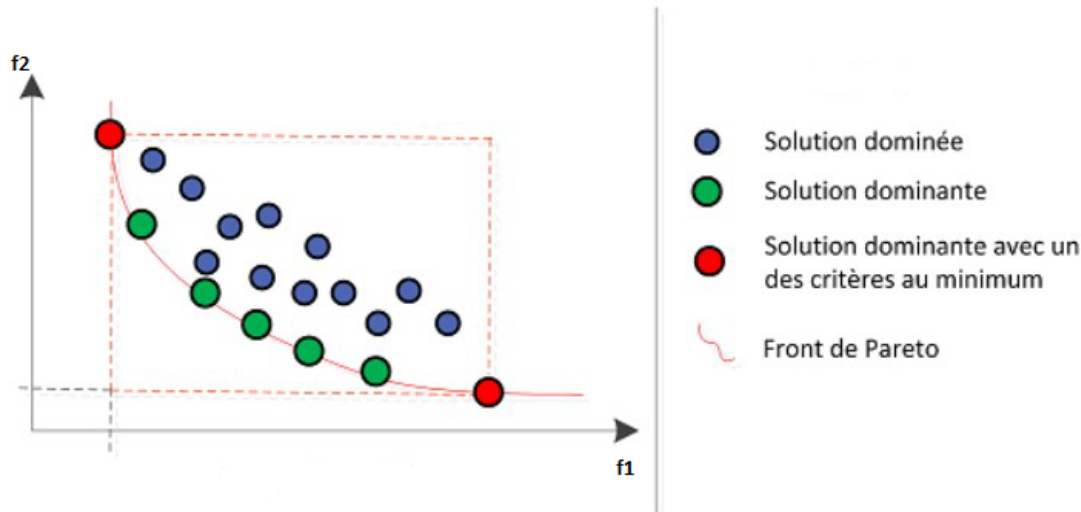


Figure 2 – Représentation graphique des solutions de Pareto

En reprenant les différentes définitions, on peut dire que :

- la solution A est une solution de Pareto faible non stricte ;
- la solution B est une solution de Pareto stricte ;
- la solution C est une solution de Pareto dominée ;
- la solution D est une solution de Pareto stricte.

En faisant la liaison de toutes les solutions de Pareto optimales, on peut obtenir une courbe spécifique appelée **front de Pareto**



**Figure 3** – Représentation graphique du front de Pareto sur un problème bi-objectif

Toutes les solutions sur le front de Pareto sont des solutions de Pareto faibles strictes ("Solution dominante" sur le schéma).

## 2.3 Détermination de l'optimum de Pareto

Il existe plusieurs méthodes (décrites dans l'ouvrage de Vincent T'KINDT et Jean-Charles BILLAUT [20]) permettant d'obtenir un optimum de Pareto.

### 2.3.1 Détermination par combinaison convexe de critères

Cette méthode est proposée par Geoffrion [9]. Elle repose sur la minimisation d'une combinaison convexe de critères pour laquelle le résultat de base est présenté dans le «théorème de Geoffrion».

### 2.3.2 Détermination par analyse paramétrique

Cette méthode est proposée par Soland [17] permet de calculer tous les optima de Pareto.

### 2.3.3 Détermination au moyen de l'approche par contrainte $\epsilon$

Dans cette méthode, l'un des objectifs est choisi comme fonction objective et les autres sont transformés en contraintes. Cette méthode est utilisée par le solveur Xpress avec l'algorithme de Kirlik et Sayin (cf [Section 3](#)).

### 2.3.4 Utilisation de la métrique Tchebycheff

il est possible d'utiliser une métrique et de rechercher la solution «la plus proche possible» d'un vecteur de critères de référence. Cette métrique s'appelle la métrique Tchebycheff [2].

### 2.3.5 Utilisation de la métrique Tchebycheff pondérée

Cette méthode correspond à une généralisation de la métrique Tchebycheff [2]. Ici, les critères sont associés à des poids que l'on peut varier selon l'importance qu'on veut lui donner.

### 2.3.6 Utilisation de la métrique Tchebycheff pondérée augmentée

Cette méthode correspond à une forme encore plus générale de la métrique de Tchebycheff [2]. Elle est cependant plus difficile à mettre en place.

### 2.3.7 Détermination par l'approche d'objectif

Une méthode similaire à celles utilisant la métrique de Tchebycheff est l'approche par objectifs de Gembicki [8] et Wierzbicki [22]. Elle consiste à définir un objectif pour les critères, et à rechercher la solution la plus proche de cela. La différence avec les approches basées sur les mesures de Tchebycheff [2] réside dans la manière dont cette solution est recherchée.

### 2.3.8 Détermination par utilisation d'un ordre lexicographique

Une méthode fréquemment utilisée pour minimiser plusieurs critères consiste à définir un ordre d'optimisation. Ce type de problème se produit quand aucun compromis entre les critères n'est autorisé.

### 2.3.9 Détermination par utilisation d'un ordre lexicographique avec objectifs

Dans cette méthode, nous recherchons une solution qui soit le plus proche de l'objectif que nous souhaitons atteindre.

## 2.4 Détermination du front de Pareto

De nombreux algorithmes (présentés dans l'article du journal Elsevier[13]) permettent de définir un front de Pareto :

- l'algorithme de Zadeh [23] ;
- l'algorithme de Eswaran, Ravindran et Moskowitz [7], basé sur les recherches de Bowman [2] ;
- l'algorithme de Haines, Lasdon et Wismer [11] ;
- l'algorithme de Sylva et Crema [19] ;
- l'algorithme de Laumanns, Thiele et Zitzler [14] ;
- l'algorithme de Tenfelde-Podehl [21] ;
- l'algorithme de Dhaenens, Lemesre et Talbi [4] ;
- l'algorithme de Ozlen et Azizog [15].

Certains de ces algorithmes ne peuvent être utilisés que selon des types de problèmes spécifiques, notamment selon le nombre de critères à traiter. La particularité de l'algorithme de Kirlik et Sayin (cf [Section 3](#)) est qu'il peut traiter tout type de problèmes multi-objectif, ce qui le rend intéressant.

### 3 Algorithme de Kirlik et Sayin

Cette section présente l'algorithme de Gokhan Kirlik présenté dans l'article du journal Elsevier[13]. Dans cet article, une nouvelle méthode est proposée pour résoudre les problèmes d'**optimisation discret multi-objectif (MODO)**. La méthode est conçue pour fonctionner avec un nombre quelconque de fonctions objectives. L'algorithme repose sur la méthode de la contrainte  $\epsilon$  ([Section 3.4](#)) et introduit un modèle simple effectué en deuxième étape pour éviter de trouver des solutions de Pareto faibles non strictes (cf [Section 2.2](#)). La méthode effectue une recherche sur toutes les valeurs pertinentes en définissant des rectangles de dimension  $p-1$  similaires à la définition de grille de Laumanns et al.[14] ([Section 3.3](#))

#### 3.1 Objectif

L'objectif est d'obtenir toutes les solutions non dominées (ou solutions de Pareto faibles strictes) ( $\Upsilon_N$ ) pour les problèmes **MODO** (cf [Section 2.2](#)). Dans la méthode de la contrainte  $\epsilon$ , l'un des objectifs est choisi comme fonction objective et les autres sont transformés en contraintes (Haimes et al., 1971 [10]).

La solution optimale donnée par la méthode de la contrainte  $\epsilon$  peut être de type faible non stricte (Ehrgott, 2005 [5]) Ce type de solution peut être évité en utilisant des formulations en deux étapes (Steuer, 1986 [18]).

#### 3.2 Initialisation

Pour simplifier la présentation de la méthode proposée, sans perte de généralité, nous nous plaçons dans un problème à 3 objectifs.

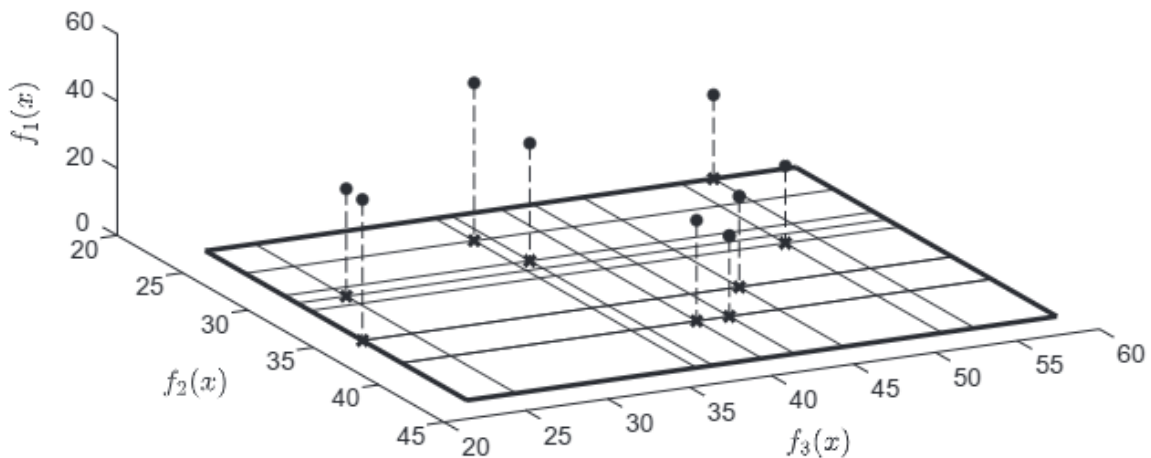


Figure 4 – Solutions dans  $R^3$  et projection des points sur le plan  $f_2 - f_3$

La **Figure 3.2** montre la projection des solutions sur le plan  $f_2 - f_3$  où l'espace de résultat est dans  $R^3$  et l'espace de recherche des solutions de Pareto faibles strictes est dans  $R^2$ . Les points projetés forment des rectangles de dimensions (p-1) dans l'espace de recherche. Les sommets inférieurs et supérieurs d'un rectangle sont désignés respectivement par  $l \in R^{P-1}$  et  $u \in R^{P-1}$ . Avec ces limites, un rectangle est défini comme :  $R(l, u) = \bar{y} \in R^{P-1}, l \leq \bar{y} \leq u$  avec  $\bar{y} = f(x) \in R^{P-1}$  où  $f(x) = (f_2(x), \dots, f_p(x))$  (Horst and Tuy, 1995 [16]). Pendant la recherche, un ensemble de rectangles doit être conservé. Le i-ème rectangle est désigné par  $R_i$  avec ses sommets inférieur et supérieur  $l_i, u_i \in R^{P-1}$ . Tous les rectangles qui doivent être recherchés sont conservés dans une liste L, i.e.  $L = \bigcup_{i=1}^{|L|} R_i$  où  $|L|$  représente le nombre de rectangles dans L.

### 3.3 Définition de l'espace de recherche

La recherche est initialisée dans un seul rectangle qui couvre l'espace de dimension (p-1). En effet, si le problème est limité, il est alors possible de définir les premières limites de l'espace de recherche pour chaque dimension en résolvant d'abord des sous-problèmes. On peut obtenir **la borne inférieure** pour chaque dimension en minimisant chaque fonction objectives sur l'ensemble réalisable  $\chi$  :

$$y_j^I = \min f_j(x) \quad (3)$$

$$s.t. \quad x \in \chi \quad (4)$$

Cette formulation renvoie le j-ème élément du point idéal ( $y^I$ ). De manière similaire, **la borne supérieure** ( $y_j^U$ ) est obtenue en maximisant chaque fonction objectives de  $\chi$  pour tout  $j \in 1, \dots, p$  :

$$y_j^U = \max f_j(x) \quad (5)$$

$$s.t. \quad x \in \chi \quad (6)$$

$y^U$  définit une limite supérieure pour chaque solution efficace dans  $\chi_E$ , i.e.  $f_j(x) \leq y_j^U$  pour  $j = 1, \dots, p$  pour tout  $x \in \chi_E$ . De plus,  $y^U$  est un surestimateur du point nadir  $y^N$  où  $y_j^N = \max_{x \in \chi_E} f_j(x)$  (Ehrgott and Tenfelde-Podehl, 2003 [6]). Puisque la recherche est gérée sur  $\varepsilon \in R^{P-1}$ , on définit  $\bar{y}^I$  et  $\bar{y}^U \in R^{P-1}$  où  $\bar{y}^I = (y_2^I, \dots, y_p^I)$  et  $\bar{y}^U = (y_2^U, \dots, y_p^U)$

Tout au long de la procédure d'obtention d'une solution de Pareto stricte, les rectangles sont partitionnés en plus petits rectangles disjoints. Lorsqu'un rectangle est sélectionné, la formulation en deux étapes (cf **Section 3.4**) est toujours résolue en utilisant son sommet supérieur. Par conséquent, certaines des régions possibles dans  $P_1(u_i)$  avec différents rectangles peuvent se superposer. Afin de faciliter la sélection de rectangles conduisant à de plus grands ensembles réalisables dans  $P_1(u_i)$ , nous définissons une mesure de volume associée au rectangle  $R_i$  :

$$V_i = \prod_{j=1}^{p-1} (u_{ij} - \bar{y}_j^I) \quad (7)$$

Le rectangle possédant le plus grand volume est sélectionné en priorité lors de la recherche de solutions.

### 3.4 L'approche par contrainte $\epsilon$ en 2 étapes

L'approche par contrainte  $\epsilon$  en 2 étapes,  $P_k(\epsilon)$  et  $Q_k(\epsilon)$  permet de trouver une solution non-dominée dans le rectangle sélectionné.

Le principe de cette approche consiste à minimiser une fonction objective, fixer la valeur trouvée pour cette fonction puis minimiser la somme des autres fonctions objectives du problème.

Pour tout  $\epsilon \in R^{P-1}$  et pour  $k \in 1, \dots, p$ ,  $P_k(\epsilon)$  est définie par :

$$P_k(\epsilon) : z = \min f_k(x) \quad (8)$$

$$s.t. \quad f_j(x) \leq \epsilon_j \quad j = 1, \dots, p \text{ and } j \neq k \quad (9)$$

$$x \in \chi \quad (10)$$

Soit  $z$  la valeur objective optimale du sous-problème  $P_k(\epsilon)$ ,  $Q_k(\epsilon)$  est définie par :

$$Q_k(\epsilon) : z = \min \sum_{j=1}^p f_j(x) \quad (11)$$

$$s.t. \quad f_j(x) \leq \epsilon_j \quad j = 1, \dots, p \text{ and } j \neq k \quad (12)$$

$$f_k(x) = z^* \quad (13)$$

$$x \in \chi \quad (14)$$

Soit  $x^*$  une solution optimale de la formulation en deux étapes  $P_k(\epsilon)$  et  $Q_k(\epsilon)$ . Alors  $x^*$  est toujours une solution efficace pour tout  $\epsilon \in R^{P-1}$ , et toute solution efficace au problème **MODO** peut être obtenue en utilisant l'algorithme à deux étapes.

### 3.5 Analyse des résultats

Selon la solution donnée par la formulation en deux étapes sur un rectangle de recherche, trois cas peuvent apparaître :

- une nouvelle solution optimale peut être obtenue. Dans ce cas, une procédure de mise à jour et suppression de rectangles est mise en place pour mettre à jour la liste des rectangles  $L$ ;
- une solution optimale  $x^*$  déjà trouvée peut être obtenue. Alors la recherche sur ce rectangle est arrêtée;
- la formulation en 2 étapes est infaisable. Ce cas arrive lorsque la liste des rectangles  $L$  est vide.

## 4 Outils du projet

### 4.1 Fico Xpress

Le solveur FICO Xpress est un solveur mathématique qui utilise le langage de modélisation et de programmation Xpress-Mosel ainsi que l'environnement de développement Xpress Workbench. Ce solveur permet de résoudre des problèmes mathématiques.

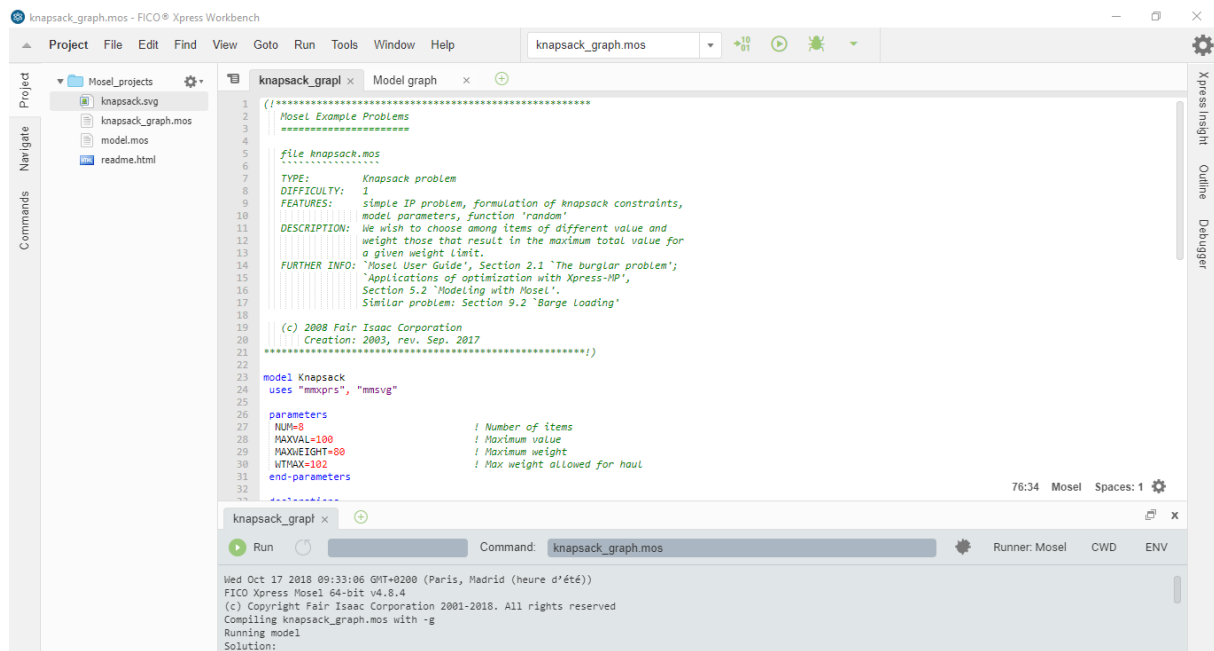


Figure 5 – Vue de l'environnement Xpress Workbench

L'environnement comporte les caractéristiques classiques d'un environnement de développement (debugger, console...).

Une documentation de ce solveur est disponible dans les sources du rapport ([Chapitre 8](#))

## 4.2 Mosel

Le langage Mosel est un langage de programmation et de modélisation permettant de définir le problème à traiter pour le solveur Xpress.

Le fichier (extension : .mos) définissant le problème à traiter est composé de plusieurs blocs. Les plus importants sont les suivants :

- *parameters* / *end-parameters* : ce bloc permet de définir les variables du projet ;
- *declarations* / *end-declarations* : ce bloc permet de donner un nom, un type et une structure aux entités que la partie "traitement du programme" utilisera ;
- *requirements* / *end-requirements* : ce bloc permet de déclarer les fonctions et les procédures ;
- *function* / *end-function* : ce bloc permet de définir les fonctions en incluant obligatoirement un type de retour ;
- *procedure* / *end-procedure* : ce bloc permet de définir un ensemble d'actions qui doivent être exécutées sans type de retour.

Une documentation plus détaillée de ce langage est disponible dans les sources du rapport ([Chapitre 8](#))



# 4

## Analyse et conception

### 1 Analyse de la fonction de l'algorithme de Kirlik et Sayin

La fonction de Kirlik et Sayin codée à l'aide de Mosel (*"mmoint minimise kirliksayin"*) est codée en plusieurs étapes reprenant les caractéristiques de l'algorithme.

#### 1.1 Etape 0 : initialisation

La fonction possède en paramètre la liste des fonctions objectives à minimiser (*ctx*).

Le code débute en définissant les variables permettant la résolution du problème :

- *bnds* de type *"box"* : elle correspond à une "boîte" permettant de stocker des coordonnées ;
- *Sols* de type *"set of string"* : elle correspond aux solutions non-dominées trouvées ;
- *p* de type *"point"* : elle correspond à un point défini par des coordonnées ;
- *pool* de type *"sols"* : cette variable répertorie l'ensemble des solutions non-dominées trouvées ;
- *objs* de type *"array(Sols,Objs) of real"* : cette variable répertorie la valeur de la fonction objective associée à la solution trouvée ;
- *boxes* de type *"list of box"* : une liste de boîte permettant de définir l'espace de recherche des solutions ;
- *optimal* de type *"boolean"* : un booléen permettant de savoir si le problème est réalisable ;
- *k* de type *"integer"* : un entier correspondant au numéro de la contrainte qui sera minimisé en premier ;
- *nit* de type *"integer"* : cette variable permet de compter le nombre d'itération de l'algorithme de Kirlik et Sayin ;
- *rowfmt* de type *"string"* : cette variable permet d'afficher les informations sur la console ;

```

procedure mmooint_minimise_kirliksayin(ctx: moctx)
  declarations
    bnds: box ! objective function bounds
    Sols: set of string
    p: point
    pool: array(Sols) of point
    objs: array(Sols,Objs) of real
    boxes: list of box
    optimal: boolean
    k: integer ! The objective function that is boxed-constraint
    nit: integer
    rowfmt: string
  end-declarations

```

Figure 1 – Code Mosel de l'étape 0

## 1.2 Etape 1 : définition des bornes de la solution

L'algorithme définit d'abord les bornes du problème : pour chaque fonctions objectives, on minimise et maximise la fonction afin de trouver les valeurs extrêmes des contraintes.

Deux cas peuvent survenir dans cette étape :

- Le problème est faisable et borné pour toutes les fonctions objectives : les bornes sont définis et la fonction se poursuit ;
- Le problème est infaisable ou non-borné : la fonction s'arrête.

```

k := ctx.iterates.first
! Get bounds on all objective functions by calling
! mmooint_minimise_nadir. We will also create the initial
! rectangle using the provided bounds, and populate
! the rectangle pool
forall(it in ctx.iterates, f:=ctx.objs(it), i in [1,-1]) do
  if i>0 then
    optimal := cb_minimise(f)
    bnds.l.coord(it) := getobjval
  else
    optimal := cb_maximise(f)
    bnds.u.coord(it) := getobjval
  end-if
  if optimal then
    infolog("Problem is feasible and bounded for all objective functions")
  else
    errorlog("Problem is infeasible or unbounded")
    return
  end-if
end-do
boxes += [bnds]

```

Figure 2 – Code Mosel de l'étape 1

### 1.3 Etape 2 : Recherche de la solution non-dominée

Après avoir défini et affiché les bornes, la recherche commence sur la première boîte définie dans la variable "boxes" en vérifiant qu'elle n'a pas déjà été analysé.

```
! Update Logarea for all boxes
forall(b in boxes) do
    updlogarea(b)
end-do

rowfmt := "%3i | %20s | %4i"
logging(1,formattext("%3s | %20s | %4s","it","searching","open"))
logging(1,formattext("%3s+-%20s+-%4s","-*3","-*20","-*4"))
while (boxes.size>0) do
    nit += 1
    if nit>MAXITER then
        break
    end-if
    with b = boxes.first do
        if b.gc then
            ! We have exhausted all boxes
            break 2
        end-if
        logging(1,formattext(rowfmt,nit,text(tolist(b.l))+ " - "+text(tolist(b.u)),count(bb in boxes | not bb.gc)))
    end-do
end-while
```

Figure 3 – Code Mosel de l'étape 2 (partie 1)

On commence par fixer la valeur d'épsilon pour chaque fonction sauf pour celle qui sera minimisée lors de la première étape de la méthode par contrainte  $\epsilon$

Ensuite, on exécute la première étape de la méthode par contrainte  $\epsilon$  en calculant  $P_k(\epsilon)$ . On fixe la valeur objective trouvée sur l'épsilon de la fonction minimisée afin de garder la valeur  $P_k(\epsilon)$  trouvée puis on calcule, dans une deuxième étape, la valeur  $Q_k(\epsilon)$  en minimisant la somme de toutes les fonctions objectives. (cf Section 3.4 (Chapitre 3))

Ces deux valeurs permettent de trouver le point correspondant à la solution de Pareto stricte trouvée que l'on sauvegarde dans les variables *pool* et *objs* et affiche si elle n'a pas déjà été trouvée.

```
! For each rectangle in the pool, find a non dominated
! solution. Use weak bounds for epsilon constraint to
! let the optimizer pick a non dominated solution that
! is outside the rectangle in case there isn't any
! non dominated solution in the defined rectangle.
forall(it in ctx.iterates | it<>k) do
    ! Fix epsilon. Only required for lower bound as we are
    ! only minimizing objective functions
    cb_epssetval(ctx.eps(it,CSTR_EPS_UB),b.u.coord(it))
end-do
cb_epsreset(ctx.eps(k,CSTR_EPS_UB))
if cb_minimise(ctx.objs(k)) then
    ! Fix the value of the 'k'th objective function and solve the second
    ! problem to get an efficient point that is not dominated by any
    ! other point in the box
    cb_epssetval(ctx.eps(k,CSTR_EPS_UB),getobjval)
    cb_minimise(sum(f in getobjs(ctx)) f)
    savepoint(ctx,p)
    if p.id not in Sols then
        logging(1,formattext("%3i | %20s | Found new non dominated solution : %s",nit,"",text(tolist(p))))
        ! Add the new non dominated solution
        pool(p.id) := copy(p)
        forall(it in ctx.iterates) do
            objs(p.id,it) := ctx.objs(it).sol
        end-do
        ! Call the new solution callback
        cb_storesol(getobjs(ctx))
    end-if
end-if
```

Figure 4 – Code Mosel de l'étape 2 (partie 2)

### 1.4 Etape 3 : Redéfinition des rectangles de recherche

La dimension  $k$  est initialisée à 0 pour ne pas être pris en compte lors de la redéfinition des boîtes de recherche.

Si la solution trouvée à l'étape 2 est située dans les limites du rectangle de recherche, alors la méthode "mmoint kirliksayin split" sera appelée pour découper la boîte selon le point trouvé. De plus, la fonction "mmoint kirliksayin remove" est appelée pour enlever les boîtes non intéressantes pour la recherche de solutions de Pareto faibles strictes ainsi que la boîte initiale. Si l'étape 2 n'a trouvée aucune solution, alors la boîte est enlevée avec la méthode "mmoint kirliksayin remove".

On finit par définir la boîte qui va être analysée en réorganisant la priorité de recherche (selon le calcul du volume) avec la méthode "mmoint kirliksayin reorderq".

```

! Delete the 'k'th dimension information because it should not be
! taken into account during the box split/remove action.
p.coord(k) := 0
! If the non dominated solutions is in the rectangle,
! then use the point to split the rectangles.
mmoint_kirliksayin_split(ctx,boxes,p,k) ! updateList
end-if
! Let's remove all boxes that will only contain points dominated by the current
! efficient solution. For a two dimensional space and for minimization this
! correspond to all boxes on the right/upper part of the "cadran" centered at the
! current solution.
mmoint_kirliksayin_remove(boxes,p,b.u) ! removeRect
else
! If problem is infeasible then remove the rectangles
! that do not need to be searched
mmoint_kirliksayin_remove(boxes,bnds.l,b.u) ! removeRect
end-if
end-do
mmoint_kirliksayin_reorderq(boxes)
end-do

```

Figure 5 – Code Mosel de l'étape 3

## 1.5 Etape 4 : Affichage de la solution

Lorsque la recherche est finie, on réinitialise les limites de la solution et on affiche les solutions et points trouvés.

```

! Reset bounds
forall(it in ctx.iterates) do
  cb_epsreset(ctx.eps(k,CSTR_EPS_UB))
end-do

logging(1,"Solutions : ")
forall(s in Sols | exists(pool(s))) do
  logging(1," - "+text(tolist(pool(s))))
  forall(i in ctx.iterates) do
    logging(1,"   f("+i+") = "+objs(s,i))
  end-do
end-do
end-procedure

```

Figure 6 – Code Mosel de l'étape 4

Un exemple d'affichage est disponible à la Figure 2.

## 2 Analyse du fichier test de Sébastien LANEZ

Le code fournit par la MOA possède un fichier test exécutant plusieurs algorithmes à partir d'un problème de type TOY<sup>1</sup> :

$$3 * x(1) + x(2) \geq 4 \quad (1)$$

$$x(1) + 3 * x(2) \geq 4 \quad (2)$$

$$x(1) \leq 10 \quad (3)$$

$$x(2) \leq 10 \quad (4)$$

Les algorithmes utilisés sont :

- L'algorithme d'exploration lexicographique. Cet algorithme permet de connaître les coordonnées du point idéal<sup>2</sup>(Section 2.3.9 (Chapitre 3));
- L'algorithme "Nadir". Cet algorithme permet de connaître les coordonnées des points extrêmes du premier rectangle de recherche en minimisant et maximisant chaque fonction objectives ;
- L'algorithme des sommes pondérées en fixant une importance égale aux différents critères(Section 2.3.5 (Chapitre 3));
- L'algorithme de Kirlik et Sayin(cf Section 1 pour savoir comment l'algorithme est programmé).

Au niveau des résultats sortis, l'algorithme des sommes pondérées donne en sortie une seule solution de Pareto stricte.

L'algorithme de Kirlik et Sayin ressort trois solutions de Pareto faibles strictes, dont la solution trouvée par l'algorithme des sommes pondérées.

```
Running model
0.032 | it | searching | open
0.032 | ----+-----+-----
0.032 | 1 | [0,0] - [10,10] | 1
0.036 | 1 | | Found new non dominated solution : [0,4]
0.036 | 2 | [0,0] - [10,4] | 1
0.040 | 2 | | Found new non dominated solution : [1,1]
0.044 | 3 | [0,0] - [10,1] | 1
0.048 | 3 | | Found new non dominated solution : [4,0]
0.048 | Solutions :
0.048 | - [0,4]
0.048 | f(1) = 0
0.048 | f(2) = 4
0.052 | - [1,1]
0.052 | f(1) = 1
0.052 | f(2) = 1
0.052 | - [4,0]
0.052 | f(1) = 4
0.052 | f(2) = 0
Process exited with code: 0
```

Figure 7 – Affichage console du résultat avec l'algorithme de Kirlik et Sayin

Ce fichier de test montre que l'algorithme a pu ressortir toutes les solutions de Pareto faibles strictes.

1. Cas simpliste d'un problème complexe utilisé pour étudier, mettre au point un prototype ou tester des algorithmes pour un problème réel

2. point obtenu en minimisant tous les critères

### 3 Analyse des performances de l'algorithme de Kirlik et Sayin

Une analyse des performances de l'algorithme de Kirlik et Sayin a été faite pour le problème d'ordonnancement bi-objectif (cf [Section 1.1](#) (Chapitre 5) pour la présentation du problème et [Section 1.4](#) (Chapitre 5) pour l'analyse des performances de résolution).

# 5

## Mise en œuvre

L'ensemble des problèmes utilisés pour analyser les performances de l'algorithme de Kirlik et Sayin se situe en annexes (cf [Section 8](#) (Chapitre 7)).

### 1 Problème d'ordonnancement bi-objectif

Ce problème est énoncé dans l'article de Federico Della Croce, Christos Koulamas et Vincent T'kindt ([\[3\]](#)), dans la section 2.1.

#### 1.1 Présentation du problème

Soit un ensemble de  $n$  travaux disponibles. Chaque travail  $j$  doit être traité sans un ordre pré-défini sur deux machines disponibles en permanence  $M1, M2$  avec des temps de traitement connus, respectivement  $a_j, b_j$ .

Chaque machine peut traiter au plus un travail à la fois et les opérations de chaque travaux ne peuvent pas se chevaucher.

L'objectif est de minimiser la date de fin des travaux  $d$  ainsi que le nombre de travaux rejetés  $\epsilon$ , où les travaux sont indexés et classés selon les critères de Johnson ([\[12\]](#)) :

On planifie d'abord les travaux dont  $a_i \leq b_i$  dans l'ordre croissant des  $a_i$  puis les travaux dont  $a_i > b_i$  dans l'ordre décroissant des  $b_i$ .

Le problème peut être modélisé de la façon suivante :

$$\min d \quad (1)$$

$$\sum_{i=1}^n x_i = n - \epsilon \quad (2)$$

$$a_1 x_1 + \sum_{i=1}^n b_i x_i \leq d \quad (3)$$

$$\sum_{i=1}^n a_i x_i + b_n x_n \leq d \quad (4)$$

$$\sum_{i=1}^j a_i x_i + \sum_{i=j}^n b_i x_i \leq d \quad \forall j = 2, \dots, n-1 \quad (5)$$

$$x_i \in \{0, 1\} \quad \forall i \in 1 \dots n \quad (6)$$

avec :

- $n$  : nombre de travaux sélectionnés ;
- $\epsilon$  : nombre de travaux rejetés ;
- $a_j$  : temps de traitement de la machine 1 ;
- $b_j$  : temps de traitement de la machine 2.

en introduisant la variable suivante :

$$y_j = \sum_{i=1}^j a_i x_i + \sum_{i=j}^n b_i x_i$$

Le problème peut être modélisé de la manière suivante :

$$\min d \quad (7)$$

$$\sum_{i=1}^n x_i = n - \epsilon \quad (8)$$

$$y_1 = a_1 x_1 + \sum_{i=1}^n b_i x_i \quad (9)$$

$$y_i = y_{i-1} - b_{i-1} x_{i-1} + a_i x_i, \forall i = 2, \dots, n \quad (10)$$

$$y_i \leq d, \forall i = 1, \dots, n \quad (11)$$

$$x_i \in \{0, 1\} \quad \forall i \in 1 \dots n \quad (12)$$

$$y_i \geq 0 \quad \forall i \in 1 \dots n \quad (13)$$

Cette modélisation améliore le temps d'exécution ainsi que les résultats ressortis : il sera donc utilisé lors de la modélisation sous Mosel (cf [Section 1.2](#)).

## 1.2 Modélisation du problème : fichier "Flowshop-problem.mos"

L'utilisation des fichiers Mosel est expliquée dans la documentation développeur (cf [Section 6](#) (Chapitre 7)) et utilisateur (cf [Section 7](#) (Chapitre 7)).



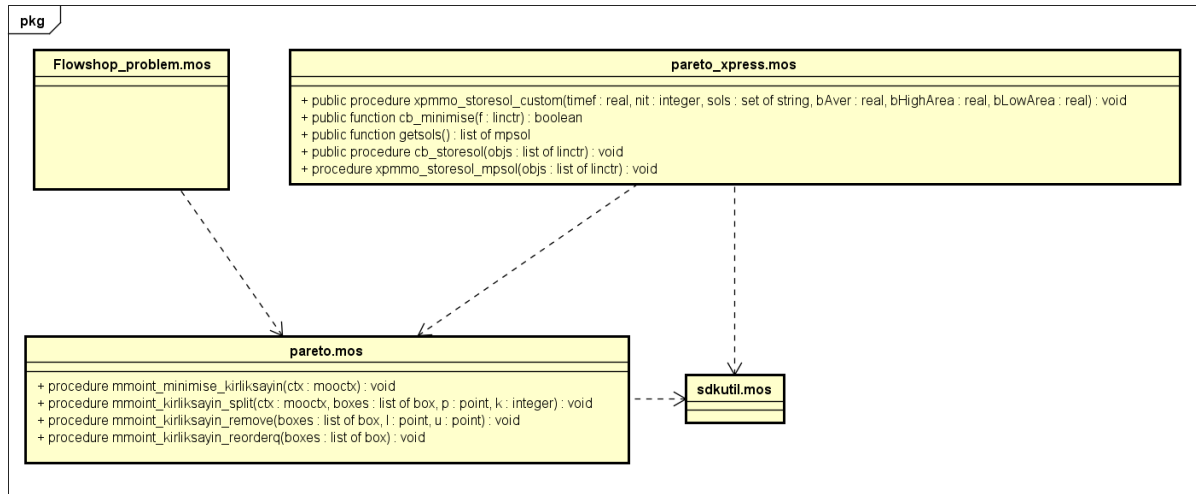


Figure 1 – Diagramme de classe de l'implémentation sous Xpress

Le problème a été modélisé par la suite dans un fichier Mosel ("Flowshop problem.mos"). Le fichier utilise les données de temps de travaux créés par le fichier "Johnson.c" (cf [Section 1.3.1](#)) ainsi que le fichier "pareto.mos" contenant l'algorithme.

```

forall(i in 1..n)
  x(i) is_binary
forall(i in 1..n)
  z(i) >= 0

y(1) is_semint 0 !d criteria (end date of work)
y(2) = n-(sum(i in 1..n)(x(i)))! number of rejected job criteria
Cstr(1) := y(1)
Cstr(2) := y(2)

Cstr(3) := z(1) = a(1)*x(1)+sum(i in 1..n)(b(i)*x(i))

forall(i in 2..n)
  Cstr(3+i-1) := z(i) = z(i-1) - b(i-1)*x(i-1) + a(i)*x(i)

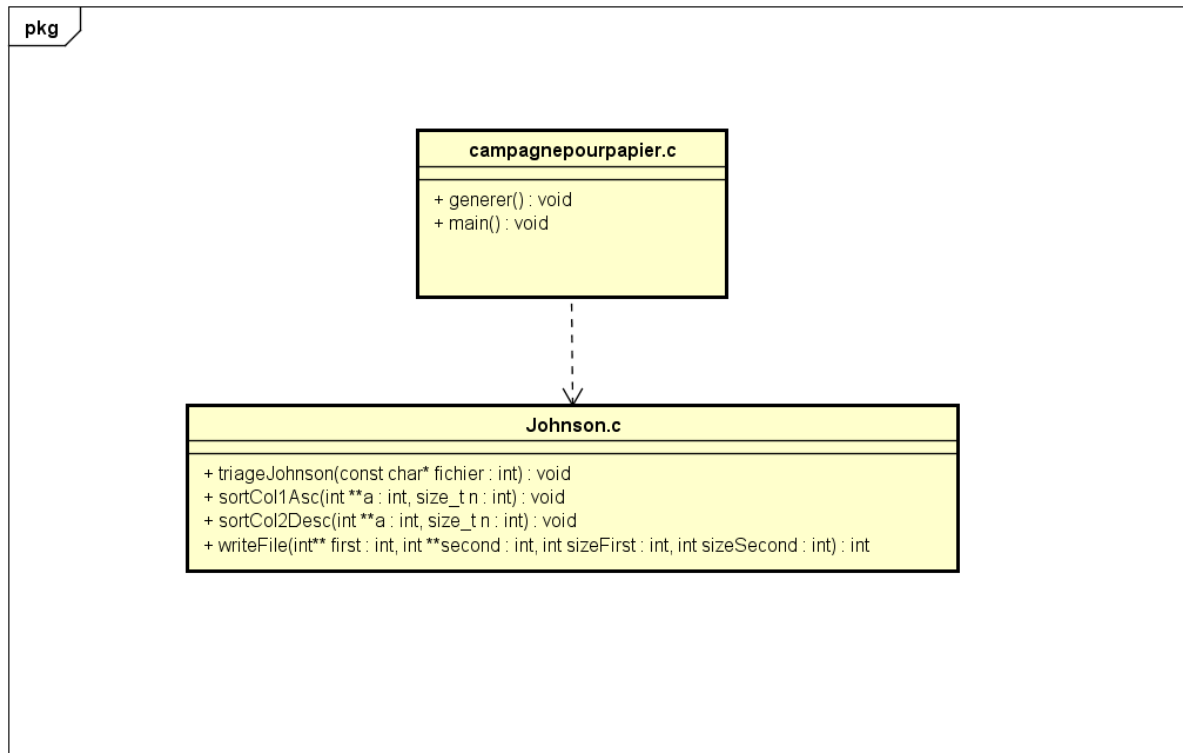
forall(i in 1..n)
  Cstr(3+n-1+i) := z(i) <= y(1)

Fobjs := [Cstr(1),Cstr(2)]
  
```

Figure 2 – "Conversion" Mosel du modèle mathématique

Après exécution de l'algorithme, un fichier de résultat générique est ressortit (cf [Section 3.4](#)) ainsi qu'un fichier de résultat spécifique ("XpressResults.dat") pour faciliter la comparaison avec l'algorithme dédié.

### 1.3 Comparaison des résultats



**Figure 3** – Diagramme de classe du projet "campagnepourpapier"

La comparaison a été réalisée à l'aide d'un projet Visual Studio. L'utilisation de ce projet est expliquée dans la documentation développeur (cf [Section 6](#) (Chapitre 7)) et utilisateur (cf [Section 7](#) (Chapitre 7)). La documentation développeur donne de plus les informations sur les bibliothèques utilisées.

Ce projet comporte la classe "Johnson.c" (cf [Section 1.3.1](#)) ainsi que le fichier "campagnepourpapier.c" de lancement du test (cf [Section 1.3.2](#)).

L'algorithme de Kirlik et Sayin sera ici comparé avec l'algorithme "branch and bound".

#### 1.3.1 Pré-traitement selon les critères de Johnson : fichier "Johnson.c"

Afin de pouvoir exécuter l'algorithme de Kirlik et Sayin, une étape de pré-traitement des données doivent être réaliser selon les critères de Johnson (cf [Section 1.1](#)). Cette étape a été réalisée à l'aide du langage C où la fonction principale du fichier "Johnson.c" exécutent les phases suivantes :

- 1) lecture du fichier de données initiale "donnees.don". Ce fichier est composé de 4 colonnes : le numéro d'ordonnancement, le temps de travail de la première machine, le temps de travail de la deuxième machine et une colonne ne contenant que des 0. Les colonnes 2 et 3 sont ainsi récupérées ;

- 2) tri des colonnes selon les critères de Johnson (cf [Section 1.1](#) ;

- 3) écriture des résultats dans un fichier texte "donneesJohnson.dat". L'écriture respecte un certain modèle afin d'être réutilisable par la suite dans le fichier Mosel de modélisation du problème.

Des tests unitaires ont été réalisés afin de vérifier la fonctionnalité des fonctions créées (cf [Section 5](#) (Chapitre 7)).

### 1.3.2 Programme de tests : "campagnepourpapier.c"

Le fichier "campagnepourpapier.c" permet de résoudre le problème multi-critère avec l'algorithme souhaité.

Avant le lancement, certains paramètres peuvent être modifiés selon les tests souhaités :

- le nombre de données utilisées pour le problème (i.e. le nombre de travaux) ;
- le nombre de fois que les algorithmes sont exécutés ;
- l'augmentation du nombre de données à la fin de toutes les itérations.

Le fichier s'exécute de la manière suivante :

- 1) Le fichier crée un fichier de données ("donnees.don") correspondant aux temps de travaux des 2 machines.
- 2) La ou les algorithmes(s) choisi(s) s'exécutent un certain nombre de fois pour résoudre le problème et sont comparés (cf [Section 1.3.3](#)). Les critères de comparaison utilisés pour ce problème sont le nombre de solutions trouvées et le temps de calcul.
- 3) A la fin de toutes les itérations, Un fichier statistique est créé à la fin de l'exécution de la fonction pour ainsi connaître les performances de l'algorithme. Il génère notamment, pour le comparatif des performances, les temps de calculs minimaux, maximaux et moyens.

```

This file contains the results for the constraint generation approach with preprocessing
-----
The tests have been conducted on a PC Pentium IV 2.6 Ghz
The time limit set is of 90000000.000000

Size n | Minimum Total CPU time (s) | Average Total CPU time (s) | Maximum Total CPU time (s) |
-----|-----|-----|-----|
50 | 1.719 | 2.031 | 2.500
100 | 5.454 | 6.356 | 6.953
150 | 14.533 | 15.496 | 17.096
200 | 27.316 | 28.888 | 30.393

```

**Figure 4** – Exemple de fichier Statistiques générés

- 4) Le nombre de données est incrémenté et décidera si le programme doit se relancer ou s'arrêter selon le nombre de travaux marquant l'arrêt renseigné.

Le fichier a été modifié afin d'inclure l'algorithme de Kirlik et Sayin et d'effectuer la comparaison des résultats. L'algorithme est lancé à partir d'une commande Mosel qui exécute le fichier au format .bim du problème.

### 1.3.3 Fichier de sortie de comparaison "Comparaison.txt"

Lorsque les deux algorithmes sont exécutés, un fichier texte est créé afin de comparer les résultats obtenus. Pour chaque itération, ce fichier indique :

- Le nombre de données initiaux ;
- si les deux algorithmes ont trouvé le même nombre de solution ;
- quel algorithme a été le plus rapide.

```

longueur :      50
====> Le nombre de solutions trouvees est similaire : 51
====> L'algorithme de Kirlik et Sayin s'est execute plus rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 1.875000
-----
longueur :      100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.660000
-----
longueur :      100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.953000
-----

```

Figure 5 – Exemple de fichier "Comparaison.txt"

Le critère de validité des résultats n'a pas été pris en compte.

En effet, les solutions trouvés par les 2 algorithmes peuvent être différentes suite à une erreur d'arrondi d'ordre + ou - 1.

Cette erreur ne remet cependant pas en cause la validité des résultats et c'est pour cela que l'on considère par la suite que les résultats ressortis par les 2 algorithmes sont justes.

## 1.4 Analyse des résultats

Les tests ont été réalisés selon les paramètres suivants :

- Nombre de données minimal : 50 ;
- Nombre de données maximal : 1100 ;
- Nombre d'itérations : 3 ;
- Incrémentation : 50/100 ;
- Ordinateur : PC Asus Intel Core i3-5005U 2.0 GHz.

Les résultats de la campagne de test ont été résumés dans un fichier Excel (cf [Section 9](#) (Chapitre 7)).

### 1.4.1 Analyse du temps d'exécution

En comparant les différences de temps des 2 algorithmes, l'analyse suivante peut être faite :

- l'algorithme fournit un temps d'exécution correcte entre 50 et 300 données. En effet, l'écart de temps ne dépasse pas la minute ;
- l'algorithme fournit un temps d'exécution "raisonnable" entre 300 et 800. Nous avons en effet un écart de 10 minutes pour 800 données ;
- l'algorithme connaît des premiers signe de latence au-delà. Il y a en effet un écart de 12 minutes à partir de 900 et nous dépassons les 15 minutes à partir de 1000 données, ce qui peut paraître long sur plusieurs itérations.

### 1.4.2 Analyse du nombre de solutions trouvées

Au niveau du nombre de solutions trouvées, le test nous aura permis de voir qu'il existe actuellement une limite de données pour la résolution du problème pour l'algorithme de Kirlik et Sayin. Ce problème apparaît à partir de 1000 données.

En relançant le problème à partir du solveur Xpress, l'erreur d'exécution apparaît lors de l'appel de la fonction minimise du solveur lors de la résolution de la 2ème étape de l'approche par contrainte  $\epsilon$ .

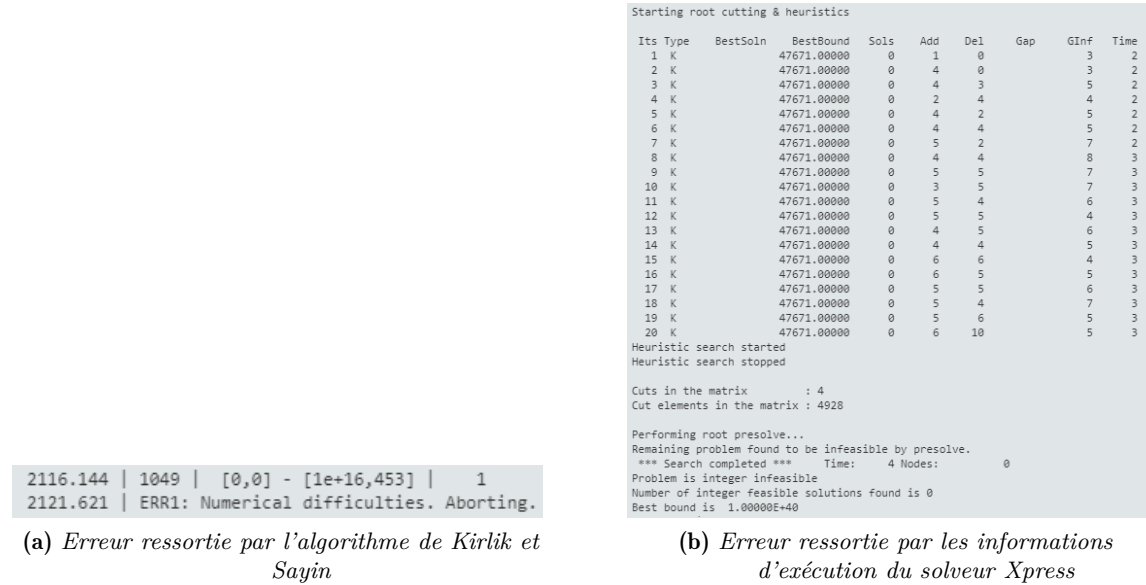


Figure 6 – Exemple de sortie d'exécution pour 1500 données

Le problème pourrait venir de 2 situations :

- lors de la redéfinition des boîtes de recherche (cf Section 1.4 (Chapitre 4)) où une mauvaise redéfinition des coordonnées de la boîte pourrait induire en erreur le solveur. Ce cas est plus plausible ;
- lors de l'étape de minimisation du solveur où ce dernier n'arrive pas à trouver une solution de Pareto malgré la bonne redéfinition des bornes du problème. Ce cas est moins plausible.

## 2 Problème d'ordonnancement à machines parallèles uniformes

Ce problème est énoncé dans la thèse de Karima Bouibede-Hocine ([1]), dans la section 2.1.1

### 2.1 Présentation et modélisation du problème

Soit un problème à machines parallèles défini de la façon suivante : un ensemble de  $\underline{n}$  travaux doivent être exécutés sur  $\underline{m}$  machines parallèles uniformes. Chaque travail  $J_i$  est défini par un temps opératoire absolu  $\underline{p}_i$ , une date de début au plus tôt  $\underline{r}_i$ , une date due  $\underline{d}_i$  et doit être totalement exécuté par une des machines. Les machines qui exécutent un travail à la fois ne sont pas nécessairement identiques mais uniformes, ce qui implique que chaque machine  $\underline{M}_j$  possède

une vitesse  $V_j$ . Le temps opératoire d'un travail dépend de la vitesse de la machine sur laquelle il est ordonnancé, le temps opératoire du travail  $J_i$  sera ainsi défini par  $p_{ij} = p_i/V_j$  lorsqu'il est exécuté sur  $M_j$ .

L'objectif est de déterminer un ou plusieurs ordonnancements des travaux sur les machines, c'est à dire déterminer pour chaque travail  $J_i$  la machine sur laquelle il sera affecté et sa date de fin  $C_i$ . Cet ordonnancement est effectué dans le but de minimiser deux critères :

- le **makespan**, qui correspond à la date de fin de tous les travaux et qui est défini par  $C_{max} = \max_{i=1..n} C_i$ .

- le **retard algébrique**, défini par  $L_{max} = \max_{i=1..n} C_i - d_i$ .

En utilisant la notation standard à trois-champs ([20]), ce problème se note  $Q/r_i, di/C_{max}, L_{max}$ .

Le problème peut être modélisé de la façon suivante :

$$\min C_{max} \quad (14)$$

$$\min L_{max} \quad (15)$$

$$\sum_{k=1}^n \sum_{j=1}^m x_{i,k}^j = 1, \forall i = 1, \dots, n \quad (16)$$

$$\sum_{i=1}^n x_{i,k}^j, \forall k = 1, \dots, n, \forall j = 1, \dots, m \quad (17)$$

$$t_k^j \geq t_{k-1}^j + \sum_{i=1}^n x_{i,k-1}^j P_i/V_j, \forall j = 1, \dots, m, \forall k = 2, \dots, n \quad (18)$$

$$t_k^j \geq \sum_{i=1}^n x_{i,k}^j r_i, \forall j = 1, \dots, m, \forall k = 1, \dots, n \quad (19)$$

$$L_{max} \geq t_k^j + \sum_{i=1}^n x_{i,k}^j P_i/V_j - \sum_{i=1}^n x_{i,k}^j d_i, \forall j = 1, \dots, m, \forall k = 1, \dots, n \quad (20)$$

$$C_{max} \geq t_k^j + \sum_{i=1}^n x_{i,k}^j P_i/V_j, \forall j = 1, \dots, m, \forall k = 1, \dots, n \quad (21)$$

$$(22)$$

## 2.2 Modélisation du problème : fichier "Scheduling problem.mos"

Une première implémentation du problème sous Mosel a été faite mais n'est pas complète.

## 3 Amélioration de la fonction de l'algorithme de Kirlik et Sayin

Durant la phase de développement, des améliorations ont été apportées sur la fonction de l'algorithme de Kirlik et Sayin afin de la rendre optimale.

### 3.1 Définition des limites de la solution

Lors de la définition des bornes initiales de la solution (cf [Section 1.2](#) (Chapitre 4)), le solveur pouvait retourner une erreur du au fait qu'il ne trouvait pas une valeur réel permettant de définir la première boîte de recherche.

Des valeurs "limites" ont ainsi été définies afin de poursuivre l'algorithme ("HV" et "LV").

HV = 1000000000000000000 ! High value  
 LV = 0 ! Low Value

Figure 7 – Définition des variables "HV" et "LV"

### 3.2 Modification du calcul du volume de la boîte

Lors du calcul du volume de la boîte de recherche, une erreur apparaissait lors du calcul du volume des boîtes de recherche (méthode "updlogarea").

En effet, la fonction logarithme pouvait retourner une erreur mathématique (calcul du logarithme d'un nombre négatif).

Cette erreur a été résolue en modifiant cette fonction.

```
procedure updlogarea(b: box)
  ! Log Logarea
  forall(o in Obj's | exists(b.u.coord(o)))
    if (b.u.coord(o)-b.l.coord(o)>0) then
      b.logarea += log(b.u.coord(o)-b.l.coord(o))
    end-if
  end-procedure
```

Figure 8 – Fonction "updlogarea" modifiée

### 3.3 Modification du découpage de la boîte

Lors du découpage de la boîte de recherche afin de trouver de nouvelles solutions (méthode "mmoint kirliksayin split"), une erreur pouvait apparaître sur les points situés appartenant à une frontière (extrémité de la boîte).

Cette erreur a été résolue en modifiant cette fonction.

```
procedure mmoint_kirliksayin_split(ctx: mmoctx, boxes: list of box, p: point, k: integer)
  declarations
    tboxes, tpboxes, ok: list of box
    l, u: box
    v: real
    tol: array(range) of real
  end-declarations

  forall(it in ctx.iterates) do
    ! Take reduced cost for integer variables for
    ! the smallest incremental steps, and take
    ! the solver tolerance for continuous
    tol(it) := cb_getfeastol(ctx.objs(it))
  end-do

  forall(b in boxes | not b.gc) do
    tboxes := [b]
    ok := [b] ! This box has not been searched yet
    forall(it in ctx.iterates | it < k) do
      v := p.coord(it)
      tpboxes := []
      if not b.l.coord(it) <= v and not v <= b.u.coord(it) then ! v is inside the box
        break
      end-if
      if not b.l.coord(it) < b.u.coord(it) then ! the box is not 'thin'
        break
      end-if
      if b.l.coord(it) < v and v < b.u.coord(it) then
        ! v is an interior point of the box. We know it does not
        ! lie on any boundaries of the box. It means that the
        ! split method will create two boxes.
        forall(t in tboxes) do
          new(l); new(u)
          l := copy(b)
          u := copy(b)
          setcoordlower(l, it, v+tol(it))
          setcoordupper(u, it, v-tol(it))
          tpboxes += [l, u]
        end-do
        if b.l.coord(it) = v then
          ! v lies on the boundary of the box. We won't split the box, just
          ! update the boundary to exclude v.
          forall(t in tboxes) do
            new(l); l := copy(b)
            setcoordlower(l, it, v+tol(it))
            tpboxes += [l]
          end-do
        end-if
        if b.u.coord(it) = v then
          ! v lies on the boundary of the box. We won't split the box, just
          ! update the boundary to exclude v.
          forall(t in tboxes) do
            new(u); u := copy(b)
            setcoordupper(u, it, v-tol(it))
            tpboxes += [u]
          end-do
        end-if
        tboxes += tpboxes
        ok += tpboxes
      end-do
      b.gc := true
    end-do

    if ok.size > 0 then
      boxes := ok
    end-if
    ! This is "too aggressive". It causes some troubles with previously set pointers
    ! that would reference dead items
    ! !! boxes := sum(b in ok | not b.gc) [b] ! too aggressive
  end-procedure
```

(a) Partie 1

(b) Partie 2

Figure 9 – Fonction "mmoint kirliksayin split" modifiée

### 3.4 Fichier de sortie Mosel : "XpressResults.txt"

Pour pouvoir analyser les performances d'exécution de l'algorithme, un fichier générique a été créé à l'aide de la fonction "xpmmo storesol custom" contenu dans le fichier "pareto xpress.mos". Cette fonction a été conçue pour être réutilisable pour tout problème multicritère implémenté. Ainsi, en appelant cette fonction, un fichier texte est créé : "XpressResults.txt".

```
Temps : 1.694
Nombre de solutions de Pareto trouvés : 51
Nombre de boîte explorée: 51
Surface de la plus grande boîte: 882.182
Surface de la plus petite boîte: 17.699
Surface moyenne des boîtes: 454.856
=====
- Solution 1 : |0|50|
- Solution 2 : |30|49|
- Solution 3 : |57|48|
- Solution 4 : |77|47|
- Solution 5 : |100|46|
- Solution 6 : |129|45|
- Solution 7 : |161|44|
- Solution 8 : |189|43|
```

Figure 10 – Exemple d'affichage de "XpressResults.txt"

Ce fichier contient les informations suivantes :

- le temps d'exécution ;
- le nombre de solutions de Pareto trouvé ;
- le nombre de boîte de recherche exploré ;
- la surface de la plus grande boîte ;
- la surface de la plus petite boîte ;
- la surface moyenne des boîtes ;
- la liste de toutes les solutions de Pareto trouvées.



# 6

## Bilan et conclusion

### 1 État d'avancement

#### 1.1 Semestre 1

Il n'y a actuellement pas de retard sur le projet.

Le code de l'algorithme de Kirlik et Sayin a été analysé et respecte bien les étapes définies dans la [Section 3](#) (Chapitre 3).

L'analyse et la comparaison des algorithmes avec des problèmes multi-objectifs pourront être traités lors de la deuxième partie du projet.

#### 1.2 Semestre 2

##### 1.2.1 Tâches effectuées

Au cours de ce projet, j'ai pu :

- implémenter un problème bi-critère à travers le langage Mosel ;
- analyser l'efficacité de l'algorithme de Kirlik et Sayin ;
- apporter des correctifs au niveau du code implémenté par monsieur Lannez, améliorant ainsi les résultats données ;
- créer un fichier de résultat général permettant d'obtenir plus d'informations sur l'exécution de l'algorithme.

##### 1.2.2 Tâches à réaliser

Certaines tâches restent cependant à faire :

- Résolution du problème de "limite des données" (cf [Section 1.4.2](#) (Chapitre 5)) ;
- Amélioration du fichier de résultat générique (information sur le nombre total de nœuds explorés par Xpress lors de la résolution de  $P_k(\epsilon)$  et  $Q_k(\epsilon)$ ) ;
- Analyse sur d'autres problèmes multicritères (cf [Section 8](#) (Chapitre 7)).

Bien qu'un début d'implémentation sous Mosel a été réalisée, l'analyse des performances sur le deuxième problème n'a pas été faite.

### 1.2.3 Retard

Les tâches suivantes ont pris plus de temps que prévu et ont par conséquent fait ralentir l'avancement de mon projet (cf [Section 4.3](#) (Chapitre 7)) :

- Retranscription du problème 1 sous Mosel : des erreurs d'implémentation ont été faites et ont rendu la tâche plus longue.
- Intégration de l'algorithme sous Visual Studio : la compréhension du fichier "campagnepour-papier.c" ainsi que la recherche d'une méthode pour lancer l'algorithme de Kirlik et Sayin directement dans ce fichier ont rendu la tâche plus longue. De plus, un temps a été perdu en voulant lancer le projet sur les ordinateurs de Polytech, dont leur configuration spécifique ne permettait pas de lancer le programme de test.

## 2 Planning

### 2.1 Semestre 1

Le diagramme de Gantt indiquant le planning prévisionnel du semestre 2 est disponible ici : [Section 4.2](#) (Chapitre 7).

Selon les résultats obtenus, La durée de la tâche "Amélioration du code de l'algorithme de Kirlik et Sayin" de la partie "Conception" peut varier.

Deux problèmes multi-objectifs vont normalement être traités durant ce projet. Ce nombre peut cependant être modifié selon l'avancement du projet.

Un diagramme de ressources a de plus été créé pour montrer notamment l'aide dans certaines tâches (cf [Section 4.4](#) (Chapitre 7)).

### 2.2 Semestre 2

Le diagramme de Gantt correspondant à ce qui a été réalisé au semestre 2 est disponible ici : [Section 4.3](#) (Chapitre 7).

La tâche "intégration de l'algorithme sous Visual Studio" n'a pas été renseignée lors du Gantt initial et aurait pu pour ainsi gérer le retard pris(cf [Section 1.2.3](#)).

Un diagramme de ressources a de plus été créé pour montrer notamment l'aide dans certaines tâches (cf [Section 4.5](#) (Chapitre 7)).

## 3 Bilan qualité

Au niveau de la qualité du code implémentée, des tests unitaires et fonctionnels ont été réalisés afin de contrôler l'exactitude des spécifications de mon projet (cf [Section 5](#) (Chapitre 7)).

En ce qui concerne la réutilisabilité de mon projet, en plus de la documentation développeur et utilisateur (cf [Section 6](#) (Chapitre 7) et [Section 7](#) (Chapitre 7)), des commentaires à l'intérieur des fonctions ainsi que la génération du Doxygen ont été mis en place.

Le lancement des tests ainsi que la comparaison des résultats sur le problème d'ordonnancement bi-objectif [Section 1.3.2](#) (Chapitre 5) auraient pu cependant être amélioré :

- Organisation des fichiers Johnson : les fichiers de pré-traitement "Johnson.c" et "Johnson.h" aurait pu être placé dans un dossier spécifique.
- Automatisation du "build" : pour changer les paramètres d'initialisation, il faut modifier

directement le fichier "campagnepourpapier.c". Cette modification aurait pu se faire en ligne de commande au lancement du programme.

- Sortie des résultats : après exécution du programme de tests, un fichier de comparaison et de statistique est créé pour analyser les résultats. L'ensemble des résultats auraient pu être regroupés directement dans un fichier csv (cf [Section 9](#) (Chapitre 7) montrant le fichier voulu).

## 4 Bilan gestion de projet

Ce projet m'aura permis d'apprendre à savoir gérer partiellement les différents risques d'un projet conséquent.

En effet, tout au long de mon projet, j'ai pris des rendez-vous et posé de nombreuses questions à mes encadrants afin de faire suivre mon avancement et de me "débloquer" sur certaines tâches. De plus, une étape de diminution du périmètre a été faite au vu du temps qu'il me restait : 2 problèmes ; il s'est avéré au final que je n'ai pu traiter qu'un seul problème. Néanmoins, le travail que j'ai effectué aura permis de repérer certaines erreurs dans l'implémentation de l'algorithme de Kirlik et Sayin (cf [Section 3](#) (Chapitre 5)) que je n'aurais peut-être pas pu voir si je m'étais concentré sur un deuxième problème en parallèle.

D'un point vu personnel, j'ai une satisfaction mitigée de ce que j'ai réalisé. Même si je pense m'être donné les moyens pour effectuer ce que je pouvais faire, je pense que certaines erreurs auraient pu être évitées pour ne pas perdre plus de temps et ainsi réaliser plus de choses.

# 7

## Annexes

### 1 Description des interfaces externes du système

#### 1.1 Interfaces matériel/logiciel

Le solveur Xpress est implanté localement sur l'ordinateur.  
Les données étant écrites sur le solveur, il n'existe pas d'interface matériel/logiciel.

#### 1.2 Interface homme/machine

L'utilisateur interagit ici uniquement avec le solveur Xpress en modélisant les différents problèmes multi-objectifs.

#### 1.3 Interface logicielle/logicielle

Le solveur Xpress n'interagit pas avec une base de données.

### 2 Spécifications fonctionnelles

#### 2.1 Initialiser le problème

##### 2.1.1 Identification de la fonction « Initialiser le problème »

**Nom** : Initialiser le problème.

**Rôle** : Initialiser le problème multi-objectif sous le langage Mosel.

**Priorité** : Primordiale.

### 2.1.2 Description de la fonction « Initialiser le problème »

**Entrées** : Problème multi-objectif.

**Préconditions** : Utiliser le langage Mosel.

**Sorties** : Problème multi-objectif modélisé sous Xpress.

**Postconditions** : La modélisation du problème doit s'adapter aux paramètres de lancement de l'algorithme de Kirlik et Sayin.

**Composants extérieurs** : Sans objet.

## 2.2 Exécuter l'algorithme de Kirlik et Sayin

### 2.2.1 Identification de la fonction « Exécuter l'algorithme de Kirlik et Sayin »

**Nom** : Exécuter l'algorithme de Kirlik et Sayin

**Rôle** : Utiliser l'algorithme de Kirlik et Sayin sur un problème multi-objectif et obtenir les résultats.

**Priorité** : Primordiale.

### 2.2.2 Description de la fonction « Exécuter l'algorithme de Kirlik et Sayin »

**Entrées** : Problème multi-objectif modélisé.

**Préconditions** : La modélisation du problème doit s'adapter aux paramètres de lancement de l'algorithme de Kirlik et Sayin.

**Sorties** : Affichage des résultats donnés par l'algorithme.

**Postconditions** : Sans objet.

**Composants extérieurs** : Sans objet.

## 2.3 Comparer les résultats

### 2.3.1 Identification de la fonction « Comparer les résultats »

**Nom** : Comparer les résultats.

**Rôle** : Comparer les résultats obtenus entre l'algorithme de Kirlik et Sayin et l'algorithme X "dédié" au problème multi-objectif.

**Priorité** : Primordiale.

### 2.3.2 Description de la fonction « Comparer les résultats »

**Entrées** : Résultats de l'algorithme de Kirlik et Sayin, résultats de l'algorithme X "dédié".

**Préconditions** : Sans objet.

**Sorties** : Fichier texte de comparaison.

**Postconditions** : Sans objet.

**Composants extérieurs** : Sans objet.

## 2.4 Améliorer la fonction de l'algorithme de Kirlik et Sayin

### 2.4.1 Identification de la fonction « Améliorer la fonction de l'algorithme de Kirlik et Sayin »

**Nom** : Améliorer la fonction de l'algorithme de Kirlik et Sayin.

**Rôle** : Modifier éventuellement la fonction Mosel programmée de l'algorithme de Kirlik et Sayin pour améliorer les résultats donnés.

**Priorité** : Importante.

### 2.4.2 Description de la fonction « Améliorer la fonction de l'algorithme de Kirlik et Sayin »

**Entrées** : Fonction de l'algorithme de Kirlik et Sayin.

**Préconditions** : Sans objet.

**Sorties** : Fonction de l'algorithme de Kirlik et Sayin améliorée.

**Postconditions** : Vérifier l'amélioration des résultats de l'algorithme de Kirlik et Sayin.

**Composants extérieurs** : Sans objet.

## 2.5 Informations complémentaires

- La fonction "Initialiser le problème" et "Exécuter l'algorithme" se feront dans un fichier Mosel ;
- la fonction "Améliorer la fonction de l'algorithme de Kirlik et Sayin" dépendra des résultats de la fonction " Comparer les résultats ".

## 3 Spécifications non fonctionnelles

### 3.1 Contraintes de développement et conception

Le projet reposera sur l'utilisation du solveur Xpress ainsi que du langage de programmation et de modélisation Mosel.

Le code de l'algorithme de Kirlik et Sayin est fourni par la [MOA](#).

### 3.2 Contraintes de fonctionnement et d'exploitation

#### 3.2.1 Performances

L'algorithme de Kirlik et Sayin devra fournir les bons résultats aux problèmes testés.

De plus, le temps d'exécution devra être raisonnable (plus ou moins 10 secondes d'écart) en comparaison aux algorithmes dédiés à ces problèmes.

#### 3.2.2 Capacité

Il n'existe pas de limite de stockage. Elles seront fixées selon l'efficacité de l'algorithme. il serait ainsi intéressant de pouvoir appliquer l'algorithme sur un grand nombre de contraintes.

### 3.2.3 Contrôlabilité

L'affichage des résultats de l'algorithme de Kirlik et Sayin se fait sur la console du solveur Xpress. Les informations de recherche des solutions sont aussi affichées, ce qui permettra de vérifier que le code s'exécute correctement.

### 3.2.4 Maintenance et évolution du système

Selon les résultats produits, le code de l'algorithme de Kirlik et Sayin pourra évoluer pour pouvoir améliorer les performances.

## 4 Gestion de projet

### 4.1 Outils

Comme énoncé dans la [Section 4](#) (Chapitre 1), les outils utilisés pour mener ce projet sont :

- La méthodologie du "cycle en V" permettant de réaliser de manière efficace le projet ;
- Le diagramme de Gantt avec l'outil "GanttProject" permettant de définir et de suivre nos tâches.

A cela, nous pouvons rajouter les outils "Skype" et "Microsoft Teams" pour pouvoir organiser des entretiens téléphoniques avec la MOA.

Nous utilisons de plus les messageries électroniques pour pouvoir échanger nos informations de manière écrite.

L'outil "Github" a enfin été utilisé afin de garder une version de mon projet.

### 4.2 Diagramme de Gantt semestre 1

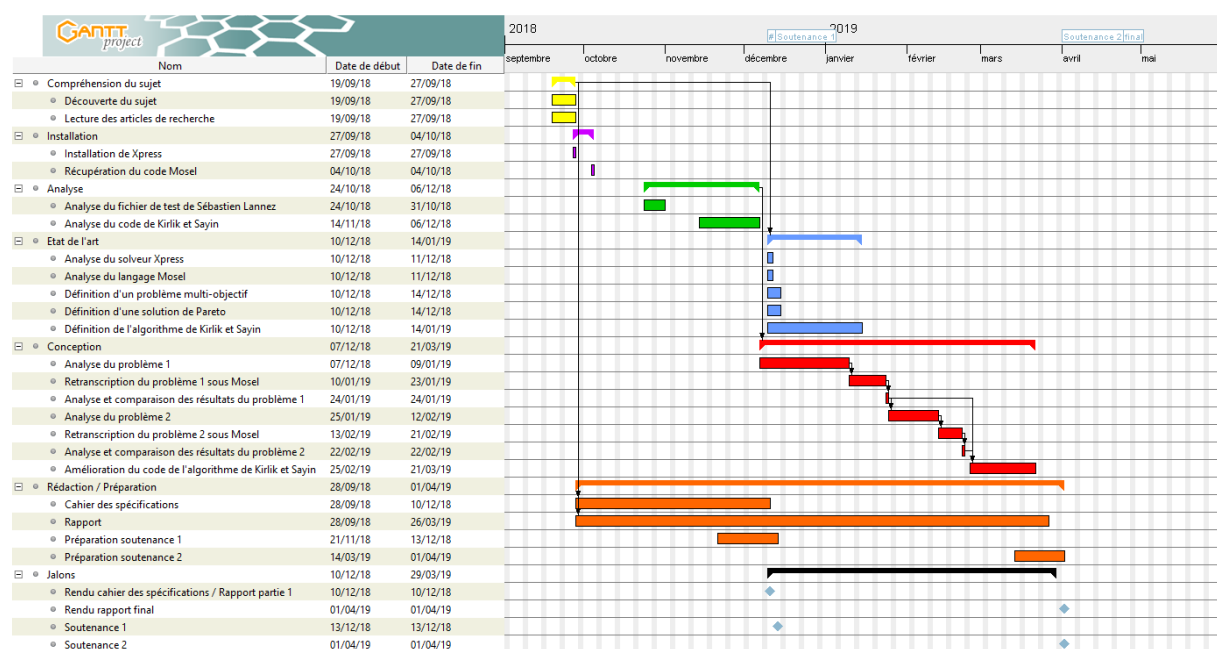


Figure 1 – Diagramme de Gantt début du semestre 2

### 4.3 Diagramme de Gantt final

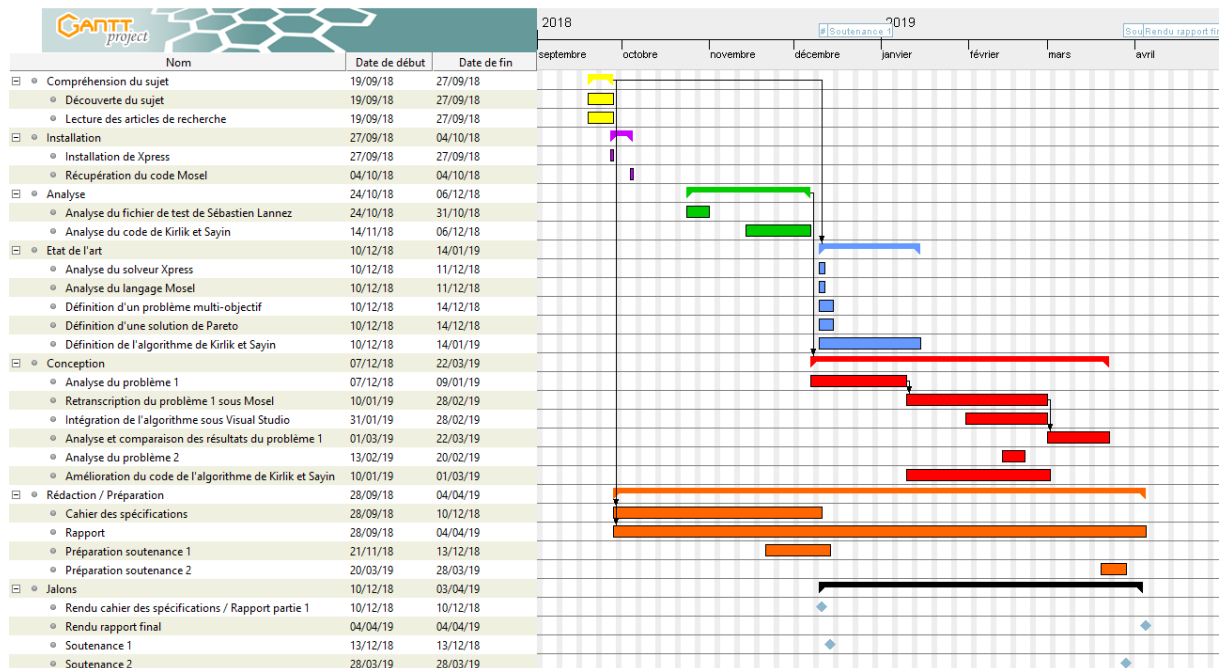


Figure 2 – Diagramme de Gantt fin du semestre 2

### 4.4 Diagramme de ressources

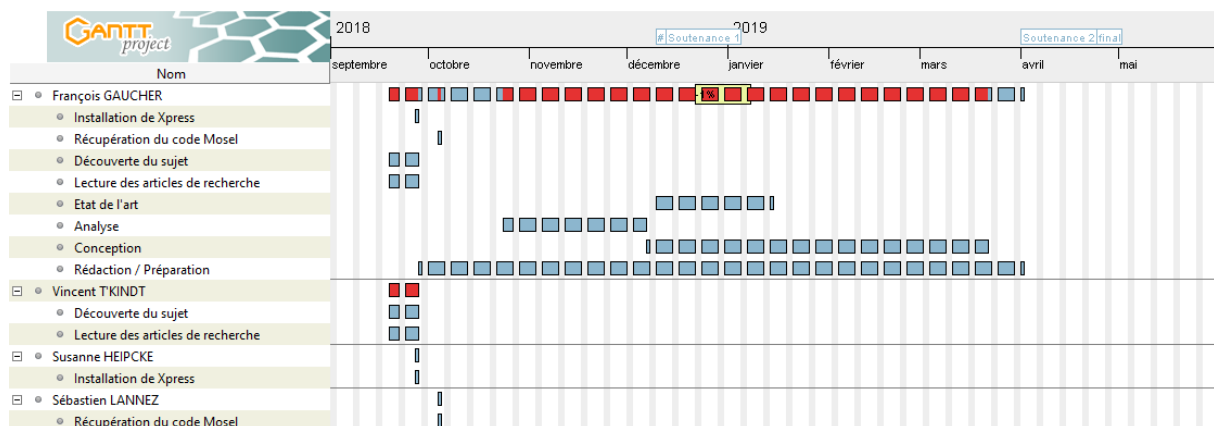


Figure 3 – Diagramme de Ressources



## 4.5 Diagramme de ressources final

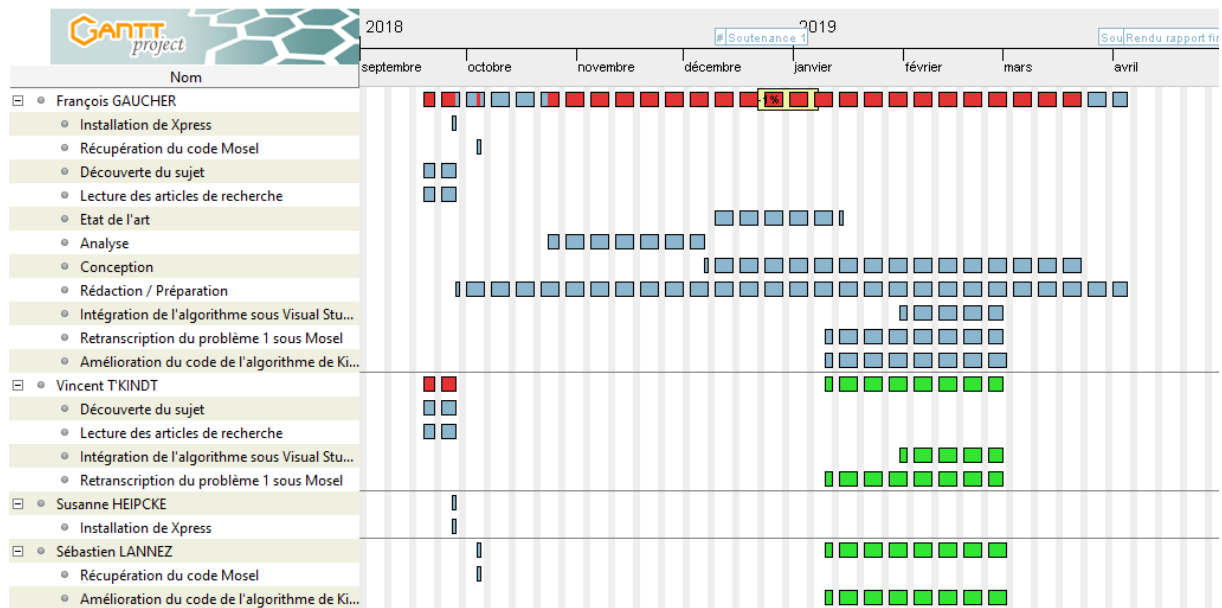


Figure 4 – Diagramme de Ressources

## 5 Documentation de tests

# Document de tests : Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO

# Sommaire

Tests unitaires .....	3
Johnson.c.....	3
Fiche de test « triage1 » .....	3
Fiche de test « triage2 » .....	4
Fiche de test « write » .....	5
Fiche de test « mainMethod » .....	6
Informations de pourcentage de couverture des tests.....	7
Tests fonctionnels.....	8
Problème « Flowshop ».....	8
Initialiser le problème .....	8
Exécuter l'algorithme de Kirlik et Sayin.....	9
Comparer les résultats .....	10
Améliorer la fonction de l'algorithme de Kirlik et Sayin.....	11

# Tests unitaires

## Johnson.c

Fiche de test « triage1 »

CODE PROJET	
Dossier « JohnsonTests »	Fiche de test unitaire

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
JohnsonTest.cpp	1	GAUCHER François	trriage1
Date du test :			
03/03/2019			

IDENTIFICATION DU CAS
La classe « Johnson.c » contient une méthode de tri d'un tableau 2D par ordre croissant selon la première colonne

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que la fonction « sortCol1Asc » trie bien le tableau donné

RESULTATS ATTENDUS
Résultats justes pour « Assert::AreEqual »

RESULTATS OBTENUS
<b>Test réussi</b>

### Fiche de test « triage2 »

CODE PROJET	
Dossier « JohnsonTests »	Fiche de test unitaire

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
JohnsonTest.cpp	1	GAUCHER François	trriage2
Date du test :			
03/03/2019			

IDENTIFICATION DU CAS
La classe « Johnson.c » contient une méthode de tri d'un tableau 2D par ordre décroissant selon la deuxième colonne
L'objectif est de tester cette méthode

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que la fonction « sortCol2Desc » trie bien le tableau donné

RESULTATS ATTENDUS
Résultats justes pour « Assert::AreEqual »

RESULTATS OBTENUS
<b>Test réussi</b>

### Fiche de test « write »

CODE PROJET	
Dossier « JohnsonTests »	Fiche de test unitaire

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
JohnsonTest.cpp	1	GAUCHER François	writeTest
Date du test :			
03/03/2019			

IDENTIFICATION DU CAS
La classe « Johnson.c » contient une méthode d'écriture des résultats du tri des jobs en format adapté pour le langage Mosel
L'objectif est de tester cette méthode en vérifiant que le fichier.dat s'est correctement créé

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que la fonction « int writeFile (int** first,int **second, int sizeFirst, int sizeSecond);» crée bien le fichier demandé

RESULTATS ATTENDUS
Résultats justes pour « Assert::AreEqual » Création du fichier « donneesJohnson.dat »

RESULTATS OBTENUS
<b>Test réussi</b>

### Fiche de test « mainMethod »

CODE PROJET	
Dossier « JohnsonTests »	Fiche de test unitaire

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
JohnsonTest.cpp	1	GAUCHER François	mainMethod
Date du test :			
03/03/2019			

IDENTIFICATION DU CAS
La classe « Johnson.c » contient une méthode principale utilisée dans le fichier « campagnepourpapier.c »
L'objectif est de tester cette méthode en vérifiant notamment qu'il lit bien le fichier « donnees.don » et que le fichier.dat s'est correctement créé

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que la fonction « triageJohnson() » crée bien le fichier demandé

RESULTATS ATTENDUS
Création du fichier « donneesJohnson.dat » correctement trié avec les bonnes valeurs

RESULTATS OBTENUS
<b>Test réussi</b>

## Informations de pourcentage de couverture des tests

Hiérarchie	Non couverts (blocs)	Non couverts (% blocs)	Couverts (blocs)	Couverts (% blocs)
▲ 🏠 johnsonstests.dll	8	4,12 %	186	95,88 %
▲ {} Classes globales	8	5,59 %	135	94,41 %
▲ 🧩 Fonctions globales	8	5,59 %	135	94,41 %
🔗 sortCol1Asc	0	0,00 %	11	100,00 %
🔗 sortCol2Desc	0	0,00 %	11	100,00 %
🔗 triageJohnson	4	5,00 %	76	95,00 %
🔗 writeFile	4	9,76 %	37	90,24 %



# Tests fonctionnels

## Problème « Flowshop »

*Initialiser le problème*

CODE PROJET	
Dossier « Flowshop »	Fiche de test fonctionnel

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
Flowshop_problem.mos	1	GAUCHER François	Initialiser le problème
Date du test :			
Février 2019			

IDENTIFICATION DU CAS
Initialiser le problème multi-objectif sous le langage Mosel.

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que le fichier « Flowshop_problem » compile correctement sous « Xpress Workbench »

RESULTATS ATTENDUS
Compilation réussie

RESULTATS OBTENUS
<b>Test réussi</b>

### Exécuter l'algorithme de Kirlik et Sayin

CODE PROJET	
Dossier « Flowshop »	Fiche de test fonctionnel

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
Flowshop_problem.mos XpressResults.txt	2	GAUCHER François	Exécuter l'algorithme de Kirlik et Sayin
Date du test :			
Février 2019			

IDENTIFICATION DU CAS
Utiliser l'algorithme de Kirlik et Sayin sur un problème multi-objectif et obtenir les résultats.

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que le fichier de modélisation s'exécute correctement avec « Xpress Workbench » et que le fichier « XpressResults.txt » ressort les informations sur les résultats trouvés.

RESULTATS ATTENDUS
Exécution de l'algorithme sans erreur. Création du fichier « XpressResults.txt » avec informations sur les résultats trouvés.

RESULTATS OBTENUS
<b>Test réussi</b>

## Comparer les résultats

CODE PROJET	
Dossier « TesteurEnumeration »	Fiche de test fonctionnel

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
campagnepourpapier.c	3	GAUCHER François	Comparer les résultats
Date du test :			
Février 2019			

IDENTIFICATION DU CAS
Comparer les résultats obtenus entre l'algorithme de Kirlik et Sayin et l'algorithme X "dédié" au problème multi-objectif.

DESCRIPTION DU TEST (valeurs, actions)
Vérifier que le fichier « campagnepourpapier.c » s'exécute correctement et crée le fichier « Comparaison.txt » de comparaison des résultats des 2 algorithmes

RESULTATS ATTENDUS
Exécution de la méthode « main() » de « campagnepourpapier.c » sans erreur. Création du fichier « Comparaison.txt » avec les informations de comparaison dans le dossier « ComparisonFlowshopXpress »

RESULTATS OBTENUS
<b>Test réussi</b>

### Améliorer la fonction de l'algorithme de Kirlik et Sayin

CODE PROJET	
Dossier « Flowshop »	Fiche de test fonctionnel

DESCRIPTION			
Projet :	N° de fiche :	Auteur :	Référence séquence de test :
pareto_xpress.mos pareto.mos	4	GAUCHER François	Améliorer la fonction de l'algorithme de Kirlik et Sayin
Date du test :			
Février / Mars 2019			

IDENTIFICATION DU CAS
Modifier éventuellement la fonction Mosel programmée de l'algorithme de Kirlik et Sayin pour améliorer les résultats donnés.

DESCRIPTION DU TEST (valeurs, actions)
Vérifier les résultats donnés par le fichier « Comparaison.txt » pour améliorer l'exécution de l'algorithme de Kirlik & Sayin

RESULTATS ATTENDUS
Résultats améliorés

RESULTATS OBTENUS
<b>Amélioration des résultats</b>

## 6 Documentation développeur

# Document développeur : Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO

# Sommaire

Windows 10.....	3
Installation de Windows 10.....	3
Xpress .....	4
Installation de Xpress .....	4
Diagramme de classe du problème « Flowshop » .....	4
Génération du fichier .bim .....	4
Visual Studio.....	6
Installation de Visual Studio .....	6
Installation des librairies « campagnepourpapier.c » .....	6
Diagramme de classe de l'application de test du problème « Flowshop ».....	6
Génération de l'exécutable « campagnepourpapier.exe ».....	7
Lancement du fichier « campagnepourpapier.c » sous Visual Studio .....	7
Lancement du fichier « campagnepourpapier.exe » .....	7
Modification du programme de tests « campagnepourpapier.c » .....	8
Variables globales utilisées.....	8
Variables locales utilisées .....	8
Algorithme de Kirlik et Sayin .....	8
Algorithme dédié.....	8

# Windows 10

## Installation de Windows 10

L'application est compilée pour fonctionner sous **Windows 10 64 bits**

Lien du site de téléchargement de Windows 10 : <https://www.microsoft.com/fr-fr/software-download/windows10>



# Xpress

## Installation de Xpress

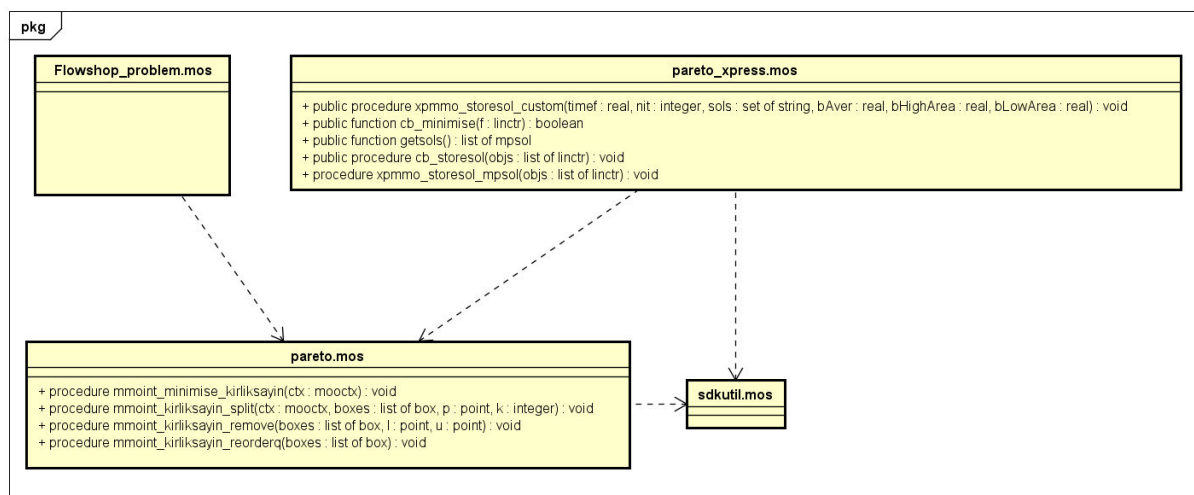
**Xpress Fico 64 bits** doit être installé afin de lancer l'algorithme de Kirlik et Sayin.

L'application fonctionne sous la **version 8.5**

**Xpress Workbench** doit de plus être installé afin de configurer les fichiers de type « .mos » et de générer le fichier.bim. Il est normalement inclus dans l'installation.

Lien du site de téléchargement de Xpress : <https://www.fico.com/en/products/fico-xpress-optimization>

## Diagramme de classe du problème « Flowshop »



**Flowshop\_problem.mos** : fichier contenant le modèle mathématique du problème et lançant sa résolution ;

**pareto.mos** : fichier contenant l'algorithme de Kirlik & Sayin ;

**pareto\_xpress.mos** : fichier contenant les fonctions de minimisation et de sauvegarde des solutions (« xpmmo\_storesol\_custom ») ;

**sdkutil.mos** : fichier contenant les outils nécessaires à l'exécution de l'algorithme.

## Génération du fichier .bim

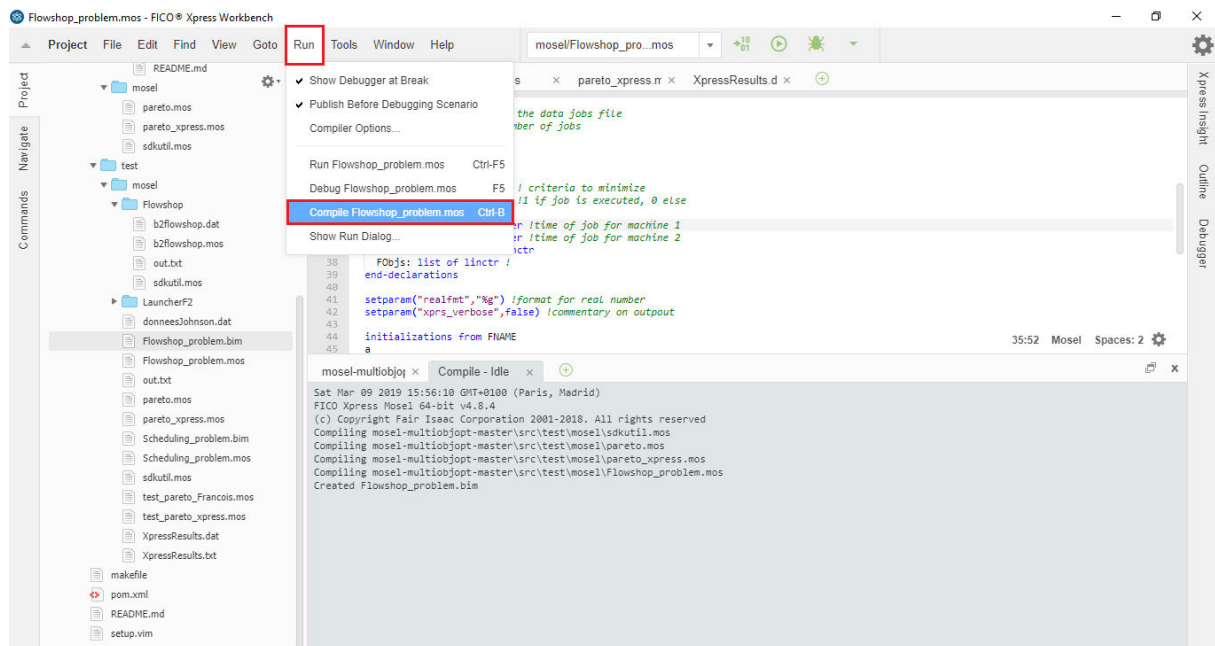
Certains fichiers doivent être renseignés afin de générer le fichier.bim du modèle mathématique :

- Pareto.mos ;
- Pareto\_xpress.mos ;

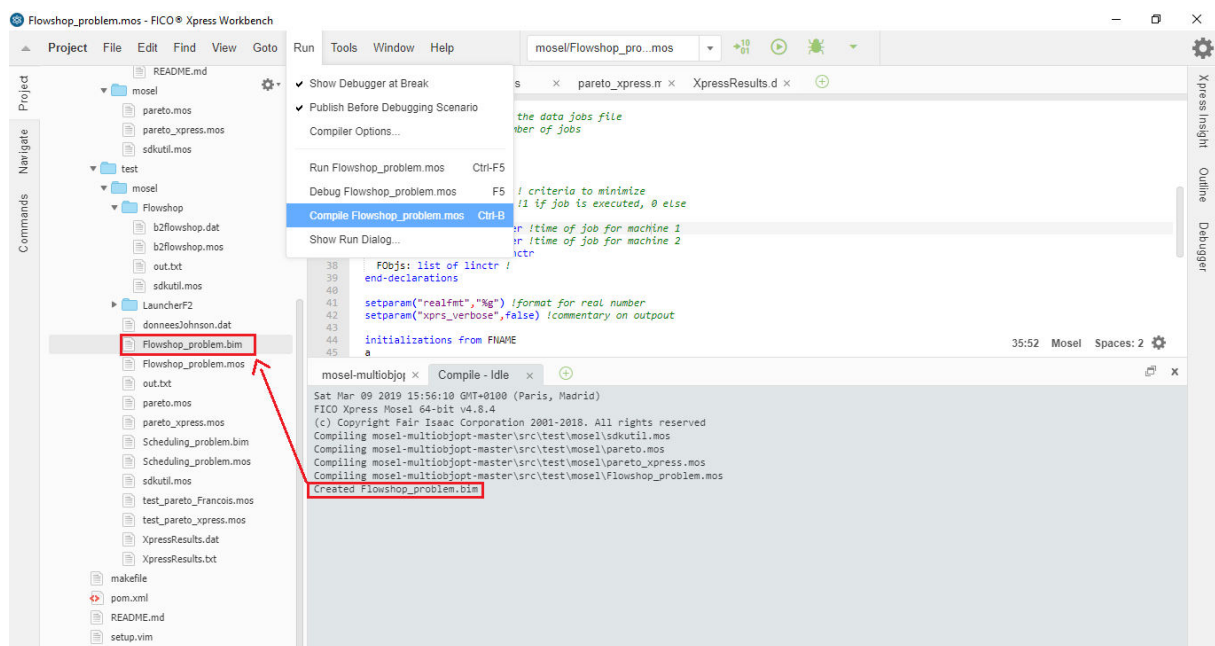
- sdkutil.mos ;

Dans l'interface Xpress Workbench, sur le fichier souhaité :

Run -> Compile <fichier.mos>



Si la compilation a fonctionné, un fichier .bim est généré



# Visual Studio

## Installation de Visual Studio

Lien du site de téléchargement de Visual Studio : <https://visualstudio.microsoft.com/fr/downloads/>

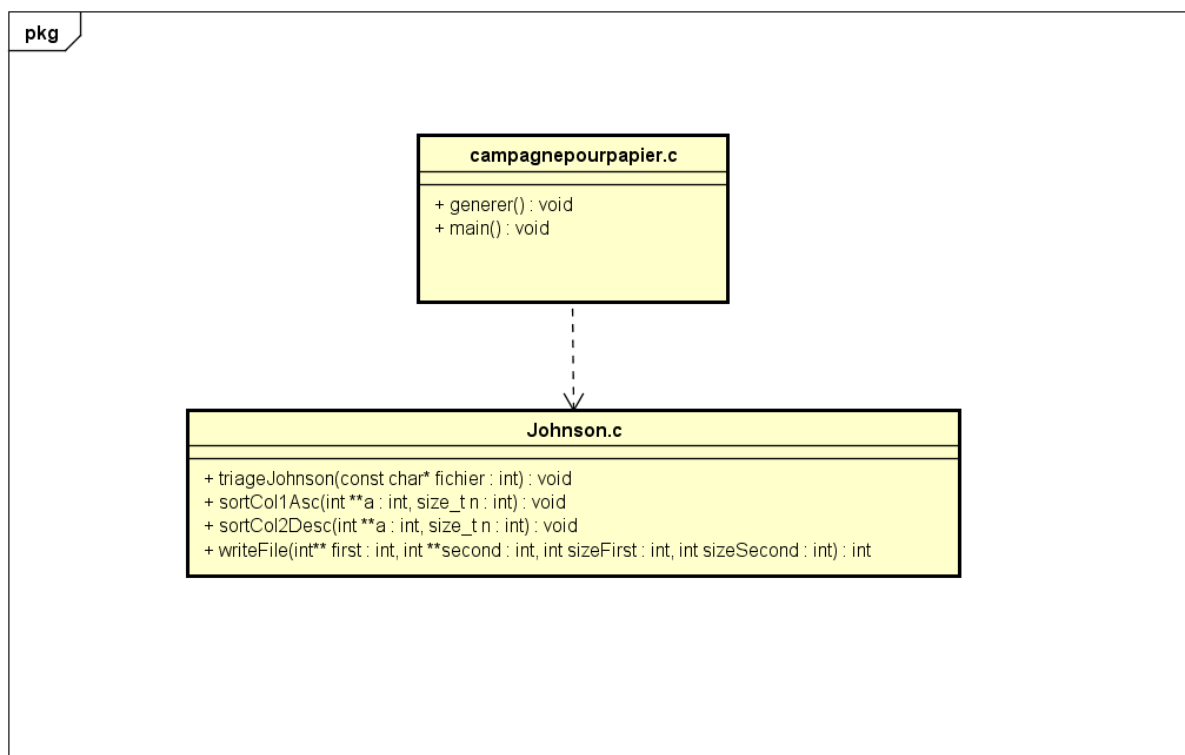
Le projet est compatible sous **Visual Studio 2012.**

## Installation des librairies « campagnepourpapier.c »

Les librairies utilisées pour le lancement du programme doivent être compatibles pour un **système d'exploitation 64 bits.**

Pour le lancement de l'algorithme dédié du problème « Flowshop », la librairie nécessaire est **cplex1280.dll.**

## Diagramme de classe de l'application de test du problème « Flowshop »

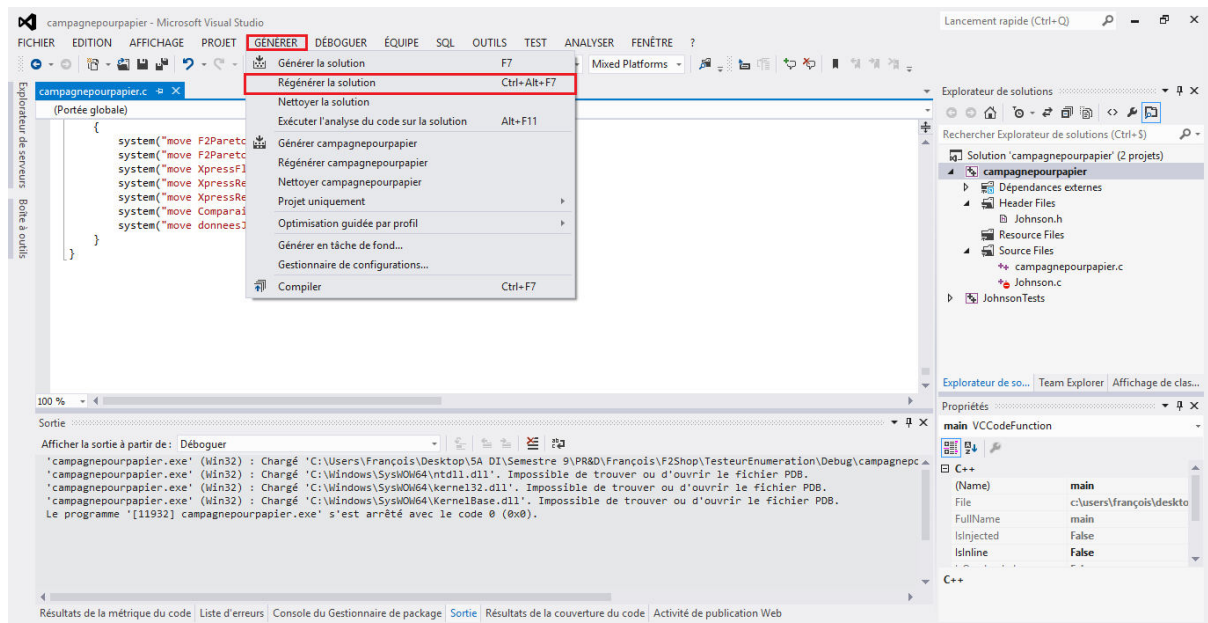


**campagnepourpapier.c** : fichier contenant la méthode d'exécution du test de comparaison des algorithmes.

**Johnson.c** : fichier contenant les fonctions de pré-traitement du problème pour l'algorithme de Kirlik & Sayin

## Génération de l'exécutable « campagnepourpapier.exe »

Pour générer l'exécutable : « Générer » -> « Régénérer la solution »



Selon la configuration sélectionnée, l'exécutable se situe dans le dossier « Debug » ou « Release » du projet.

## Lancement du fichier « campagnepourpapier.c » sous Visual Studio

Pour pouvoir exécuter la méthode main() du fichier « campagnepourpapier.c », il faut s'assurer que les fichiers et dossier suivants se situent **dans le même emplacement que le « campagnepourpapier.c »** :

- cplex1280.dll
- F2ParetoEnum.exe
- Flowshop\_problem.bim
- dossier « ComparisonFlowshopXpress »

## Lancement du fichier « campagnepourpapier.exe »

Pour pouvoir lancé l'exécutable, il doit être situé **dans le même emplacement que le fichier « campagnepourpapier.c »**.

# Modification du programme de tests

## « campagnepourpapier.c »

### Variables globales utilisées

- Nombre d'itération fait sur le fichier de données : **nbiter**
- Nombre de données de départ : **StartSize**
- Nombre de données marquant l'arrêt du programme : **EndSize**
- Nombre de données marquant un changement dans l'incrémentation du nombre de données : **MidSize**
- Valeur de l'incrémentation sur le nombre de données effectué après la fin de toutes les itérations : **FirstInc**
- Valeur de l'incrémentation sur le nombre de données effectué après la fin de toutes les itérations lorsque la valeur « **MidSize** » est dépassée : **SecondInc**

### Variables locales utilisées

#### *Algorithme de Kirlik et Sayin*

- **timeFlowXpress** : temps d'exécution
- **maxTimeFlowXpress** : temps d'exécution maximal
- **avgTimeFlowXpress** : temps d'exécution moyen
- **minTimeFlowXpress** : temps d'exécution minimal
- **nbSolsFlowXpress** : nombre de solutions de Pareto trouvé
- **solutionsF2FlowXpress** : tableau de valeur des critères des solutions de Pareto de l'algorithme de Kirlik & Sayin
- **playFlowXpress** : variable indiquant si l'algorithme de Kirlik & Sayin est lancé (1) ou non (0)

#### *Algorithme dédié*

- **TimeBaBcover** : temps d'exécution
- **MaxTimeBaBcover** : temps d'exécution maximal

- **AvgTimeBaBcover** : temps d'exécution moyen
- **MinTimeBaBcover** : temps d'exécution minimal
- **nbSolsBabcover** : nombre de solutions de Pareto trouvé
- **solutionsF2Pareto** : tableau de valeur des critères des solutions de Pareto de l'algorithme dédié
- **playmipACprepro** : variable indiquant si l'algorithme dédié est lancé (1) ou non (0)

## 7 Documentation utilisateur

# Document utilisateur : Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO



# Sommaire

Etape 1 : installation de Windows 10.....	3
Etape 2 : installation de Xpress .....	3
Etape 3 : lancement du fichier « campagnepourpapier.exe » .....	3
Etape 4 : visualisation du dossier « ComparisonFlowshopXpress ».....	4

# Etape 1 : installation de Windows 10

L'application est compilée pour fonctionner sous **Windows 10 64 bits**

Lien du site de téléchargement de Windows 10 : <https://www.microsoft.com/fr-fr/software-download/windows10>

# Etape 2 : installation de Xpress

**Xpress Fico 64 bits** doit être installé afin de lancer l'algorithme de Kirlik et Sayin.

L'application fonctionne sous la **version 8.5**

Lien du site de téléchargement de Xpress Fico : <https://www.fico.com/en/products/fico-xpress-optimization>

# Etape 3 : lancement du fichier « campagnepourpapier.exe »

Le lancement de l'exécutable « campagnepourpapier.exe » (double click) permet de lancer le programme de test.

Backup	30/01/2019 15:14	Dossier de fichiers	
ComparisonFlowshopXpress	21/03/2019 11:31	Dossier de fichiers	
Debug	21/03/2019 11:28	Dossier de fichiers	
JohnsonTests	21/03/2019 11:27	Dossier de fichiers	
campagnepourpapier.c	21/03/2019 11:28	C source file	35 Ko
campagnepourpapier.dsp	26/01/2004 13:01	VC++ 6 Project	5 Ko
campagnepourpapier.dsw	26/01/2004 12:53	VC++ 6 Workspace	1 Ko
campagnepourpapier.exe	21/03/2019 11:28	Application	637 Ko
campagnepourpapier.ncb	12/04/2013 22:01	VC++ Intellisense ...	475 Ko
campagnepourpapier.opensdf	21/03/2019 11:26	Fichier OPENSDF	0 Ko
campagnepourpapier.plg	22/10/2007 16:50	Fichier PLG	1 Ko
campagnepourpapier.sdf	21/03/2019 11:29	SQL Server Comp...	9 152 Ko
campagnepourpapier.sln	13/03/2019 09:00	Microsoft Visual S...	3 Ko
campagnepourpapier.suo	12/04/2013 22:01	Visual Studio Solu...	11 Ko
campagnepourpapier.v11.suo	21/03/2019 11:05	Visual Studio Solu...	77 Ko
campagnepourpapier.vcproj	29/06/2011 16:36	VC++ Project	6 Ko
campagnepourpapier.vcproj.POLYTECH...	12/04/2013 22:01	Visual Studio Proj...	2 Ko
campagnepourpapier.vcxproj	14/03/2019 10:59	VC++ Project	7 Ko
campagnepourpapier.vcxproj.filters	13/02/2019 14:46	VC++ Project Filte...	2 Ko
campagnepourpapier.vcxproj.user	06/03/2019 15:04	Visual Studio Proj...	1 Ko
cplex1280.dll	09/01/2018 12:27	Extension de l'app...	29 701 Ko
donnees.don	21/03/2019 11:31	Fichier DON	2 Ko
F2Cplex.cpp	19/12/2018 15:03	C++ source file	86 Ko
F2ParetoEnum.exe	19/12/2018 14:50	Application	603 Ko
Flowshop_problem.bim	16/03/2019 13:05	Fichier BIM	74 Ko
initpse.txt	21/03/2019 11:31	Document texte	1 Ko
initvalidation.txt	11/06/2003 17:18	Document texte	1 Ko
Johnson.c	20/03/2019 08:29	C source file	4 Ko

## Etape 4 : visualisation du dossier « ComparisonFlowshopXpress »

Après exécution du fichier « campagnepourpapier.c », un certain nombre de fichiers sont créés et déplacés dans le dossier « ComparisonFlowshopXpress » :

- **Comparaison.txt** : fichier comparatif des résultats entre l'algorithme de Kirlik et Sayin et l'algorithme dédié.
- **donneesJohnson.dat** : fichier utilisé pour le l'initialisation du problème sous Mosel
- **F2ParetoEnumeration.txt** : fichier de résultat de l'algorithme dédié
- **F2ParetoEnumerationStats.txt** : fichier statistique de l'algorithme dédié
- **XpressFlowshopStats.txt** : fichier statistique de l'algorithme de Kirlik et Sayin
- **XpressResults.dat** : fichier de résultat de l'algorithme de Kirlik et Sayin (utilisé uniquement pour comparer les résultats avec l'algorithme dédié)
- **XpressResults.txt** : fichier de résultat de l'algorithme de Kirlik et Sayin (pour toutes les itérations)

## 8 Campagne de tests

# 1) Comparaison avec des algorithmes dédiés

1.1) Problème d'ordonnancement de type flowshop à deux machines :  $F2|d_i=d, d \text{ unknown}| d, u$

Critères évalués : nombre de solutions trouvées, correspondance des solutions, temps d'exécution.

1.2) Problème d'ordonnancement à machines parallèles uniformes :  $Q|r_i, d_i| C_{\max}, L_{\max}$

# 2) Evaluation sur des instances de référence

2.1) Bi-Objective Set Packing Problem

2.2) Multi-Objective Assignment Problem

2.3) Multiobjective Knapsack Problem

2.4) Multiobjective ILP

## 9 Résultats des tests de performance du problème d'ordonnancement bi-objectif

Tests réalisés sur un PC Asus Intel Core i3-5005U 2.0 GHz						
Nombre d'itérations	Nombre de données	Nombre de solutions de Pareto trouvées	Temps moyen (s)	Temps minimal (s)	Temps maximal (s)	
3	50	51	Algorithme "branch and bound" : 1.667 Algorithme de Kirlik et Sayin : 2.031 Algorithme "branch and bound" : 2 Algorithme de Kirlik et Sayin : 6.356	Algorithme "branch and bound" : 1 Algorithme de Kirlik et Sayin : 1.719 Algorithme "branch and bound" : 2 Algorithme de Kirlik et Sayin : 5.454	Algorithme "branch and bound" : 2 Algorithme de Kirlik et Sayin : 2.5 Algorithme "branch and bound" : 2 Algorithme de Kirlik et Sayin : 6.953	
3	100	101	Algorithme "branch and bound" : 3.333 Algorithme de Kirlik et Sayin : 15.496	Algorithme "branch and bound" : 3 Algorithme de Kirlik et Sayin : 14.533	Algorithme "branch and bound" : 4 Algorithme de Kirlik et Sayin : 17.096	
3	150	151	Algorithme "branch and bound" : 5 Algorithme de Kirlik et Sayin : 28.888 Algorithme "branch and bound" : 7 Algorithme de Kirlik et Sayin : 53.473	Algorithme "branch and bound" : 5 Algorithme de Kirlik et Sayin : 27.316 Algorithme "branch and bound" : 6 Algorithme de Kirlik et Sayin : 49.426	Algorithme "branch and bound" : 5 Algorithme de Kirlik et Sayin : 30.393 Algorithme "branch and bound" : 8 Algorithme de Kirlik et Sayin : 57.879	
3	200	201	Algorithme "branch and bound" : 11 Algorithme de Kirlik et Sayin : 74.427	Algorithme "branch and bound" : 9 Algorithme de Kirlik et Sayin : 71.630	Algorithme "branch and bound" : 11 Algorithme de Kirlik et Sayin : 76.474	
3	250	251	Algorithme "branch and bound" : 12.667 Algorithme de Kirlik et Sayin : 144.017	Algorithme "branch and bound" : 12 Algorithme de Kirlik et Sayin : 140.917	Algorithme "branch and bound" : 14 Algorithme de Kirlik et Sayin : 149.248	
3	300	301	Algorithme "branch and bound" : 16.667 Algorithme de Kirlik et Sayin : 232.832	Algorithme "branch and bound" : 15 Algorithme de Kirlik et Sayin : 227.581	Algorithme "branch and bound" : 18 Algorithme de Kirlik et Sayin : 235.679	
3	400	401	Algorithme "branch and bound" : 26.667 Algorithme de Kirlik et Sayin : 347.742 Algorithme "branch and bound" : 35.667 Algorithme de Kirlik et Sayin : 497.779 Algorithme "branch and bound" : 52.667 Algorithme de Kirlik et Sayin : 635.234 Algorithme "branch and bound" : 67.333 Algorithme de Kirlik et Sayin : 807.960	Algorithme "branch and bound" : 23 Algorithme de Kirlik et Sayin : 339.636 Algorithme "branch and bound" : 30 Algorithme de Kirlik et Sayin : 478.229 Algorithme "branch and bound" : 37 Algorithme de Kirlik et Sayin : 589.673 Algorithme "branch and bound" : 47 Algorithme de Kirlik et Sayin : 799.423	Algorithme "branch and bound" : 33 Algorithme de Kirlik et Sayin : 354.857 Algorithme "branch and bound" : 44 Algorithme de Kirlik et Sayin : 508.135 Algorithme "branch and bound" : 71 Algorithme de Kirlik et Sayin : 679.902 Algorithme "branch and bound" : 91 Algorithme de Kirlik et Sayin : 825.035	
3	500	501	Algorithme "branch and bound" : 75 Algorithme de Kirlik et Sayin : 1026.74	Algorithme "branch and bound" : 55 Algorithme de Kirlik et Sayin : 1019.48	Algorithme "branch and bound" : 95 Algorithme de Kirlik et Sayin : 1034	*Statistiques faites sur les 2 itérations ayant le bon nombre de solutions de Pareto
3	600	601	Algorithme "branch and bound" : 94.33 Algorithme de Kirlik et Sayin : 1246.807	Algorithme "branch and bound" : 66 Algorithme de Kirlik et Sayin : 1168	Algorithme "branch and bound" : 109 Algorithme de Kirlik et Sayin : 1286.210	
3	700	701				
3	800	801				
3	900	901				
3	1000*	Algorithme "branch and bound" : 1001,1001,1001 Algorithme de Kirlik et Sayin : 901,1001,1001				
3	1100	1101				

# 8

## Sources

Problèmes multi-objectifs : <http://blog.wikimemoires.com/2014/02/optimisation-multiobjectif-et-problemes-doptimisation-mono-objectifs/>

Documentation Mosel : [http://home.deib.polimi.it/malucell/didattica/appunti/mosel/mosel-language\\_reference.pdf](http://home.deib.polimi.it/malucell/didattica/appunti/mosel/mosel-language_reference.pdf)

Documentation Xpress : <https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html>



# 9

## Comptes rendus hebdomadaires

### Compte rendu n°1 du 27/09/2018

Comme suggéré lors de notre Skype, j'ai effectué des recherches qui me permettront de faire mon état de l'art (et mettre en place le plan de mon rapport) concernant les optimum de Pareto et la notion d'optimisation multi-objectif.

J'ai de plus commencé à étudier l'algorithme de Kirlik.

Je devrais normalement recevoir le code source de Mosel la semaine prochaine.

### Compte rendu n°2 du 04/10/2018

J'ai commencé à rédiger mon rapport, notamment la description de l'algorithme de Kirlik.

Je vais essayé de terminer la partie "Etat de l'art" pour mi-octobre (comme il a été conseillé durant notre Skype) pour ensuite commencer à programmer sur Mosel.

Pour information, j'ai récupéré le code de Sébastien.

### Compte rendu n°3 du 11/10/2018

J'ai globalement fini la partie "Etat de l'art" de mon projet.

Je commencerais la semaine prochaine à utiliser le solveur XPRESS.

### Compte rendu n°4 du 18/10/2018

J'ai commencé à me familiariser avec le solveur Xpress en regardant l'exemple de projet "sac à dos".

J'ai de plus regardé de manière générale le code développé par Sébastien.

La semaine prochaine, je vais essayé de tester le code afin de voir les résultats générés et réfléchir sur les spécifications fonctionnelles que je pourrais mettre en place.

### Compte rendu n°5 du 25/10/2018

Je me suis familiarisé avec le solveur Xpress en essayant de reproduire un problème de sac à dos multi-dimensionnel (je n'ai pas encore réussi à le modéliser)

Nous disposons d'une semaine de vacance la semaine prochaine : je pense faire essentiellement de la rédaction du rapport cette semaine là.

### Compte rendu n°6 du 07/11/2018

[Compte-rendu non transmis]

J'ai pris contact avec Sébastien LANNEZ pour des explications sur le code de l'algorithme. J'ai ensuite rédigé essentiellement du rapport.

### Compte rendu n°7 du 15/11/2018

J'ai principalement fait de la rédaction du cahier des spécifications et de l'état de l'art. Je vous présenterais mercredi comme convenu ce que j'ai compris de l'algorithme de Kirlik.

### Compte rendu n°8 du 22/11/2018

[Compte-rendu non transmis]

Nous avons discuté avec Vincent T'KINDT sur la compréhension de l'algorithme de Kirlik et Sayin.

Ce rendez-vous m'a permis de mieux comprendre son fonctionnement et de bien rédiger mon état de l'art.

### Compte rendu n°9 du 29/11/2018

[Compte-rendu non transmis]

J'ai principalement fait de la rédaction de rapport ainsi que de la préparation à la soutenance. J'ai de plus commencé à analyser les deux premiers problèmes que je devais implémenter

### Compte rendu n°10 du 06/12/2018

[Compte-rendu non transmis]

J'ai principalement fait de la rédaction de rapport ainsi que de la préparation à la soutenance.

### Compte rendu n°11 du 13/12/2018

J'ai principalement fait de la rédaction de rapport et j'ai effectué ma soutenance auprès de mes encadrants, permettant ainsi de faire un rapport de ce que j'ai actuellement fait

### Compte rendu n°12 du 20/12/2018

J'ai commencé à faire une ébauche de la modélisation du problème sous Mosel.

J'ai de plus commencé à réfléchir à comment récupérer et afficher les différentes informations que l'on veut dans le fichier texte.

### Compte rendu n°13 du 10/01/2019

J'ai effectué le pré-traitement des données selon la règle de Johnson.

J'ai continué à effectuer la modélisation du problème "d'ordonnancement bi-objectif" sous Mosel que j'espère finir la semaine prochaine.

### Compte rendu n°14 du 17/01/2019

J'ai modélisé le problème "d'ordonnancement bi-objectif" sous Mosel mais l'algorithme ne ressort pas de résultats (erreur lors du calcul de la première borne supérieure).

Je peux vous montrer comment j'ai modélisé le problème la semaine prochaine si vous le souhaitez pour être sûr que je n'ai pas fait d'erreurs.

### Compte rendu n°15 du 24/01/2019

Je n'ai pas pu tester le code que j'ai programmé sur une machine virtuelle de l'école. La cause est du (je pense) à la configuration des PC.

Dites-moi l'heure de notre rendez-vous la semaine prochaine pour que je vous montre mon code, sachant que je ne suis pas disponible mercredi 30 janvier de 10h30 à 11h30.

### Compte rendu n°16 du 31/01/2019

J'ai corrigé le problème concernant la contrainte du problème mathématique. L'algorithme ne donne cependant pas l'ensemble des solutions de Pareto. L'erreur vient du fait que la valeur des  $x(i)$  ne change jamais et ne permet pas d'avoir l'ensemble des solutions réalisables. Je n'ai pas encore réussi à résoudre ce problème et je compte poser la question à Sébastien si je ne trouve pas de solutions avant la fin de la semaine.

Je suis de plus en train de coder la fonction de Johnson en C, que je vais essayé de finir avant la fin de la semaine.

### Compte rendu n°17 du 07/02/2019

Après avoir fait fonctionner l'algorithme, Je suis actuellement en train d'essayé d'appeler le fichier Mosel dans un programme en C. Je continuerais cette partie ce week-end.

Au vu du retard sur mon projet, je tenais à vous informer que je ne pense pas que j'aurais le temps de traiter intégralement la partie benchmark sur des instances de référence. Je ferais tout mon possible pour réaliser au minimum la comparaison des deux problèmes multi-critères avec les algorithmes dédiés.

Nous pouvons programmer un rendez-vous la semaine prochaine si vous le souhaitez pour que je vous montre mon avancement.

### Compte rendu n°18 du 14/02/2019

Je suis toujours sur la modification du fichier de campagne : j'ai encore quelque soucis de compilation que je vais bientôt résoudre.

Je vous ferais un compte-rendu durant la pause pédagogique de mon avancée.

### Compte rendu n°19 du 28/02/2019

J'ai résolu le problème concernant le formatage des données et j'ai inclus les informations concernant la surface de la plus petite boîte explorée ainsi que le temps minimale d'exécution pour les 2 algorithmes.

Un nouveau problème est parvenu lors de la comparaison des résultats donnés sur les 2 algorithmes : à partir de 200 vecteurs de jobs, des erreurs apparaissent sur la valeur du critère  $d$  (cf fichier texte donné en pièce jointes).

Je vais essayé de résoudre le problème ce week-end en analysant l'implémentation Mosel de l'algorithme de Kirlik et Sayin.

### Compte rendu n°20 du 07/03/2019

J'ai exposé mon problème à Sébastien et l'erreur semblerait venir de la modélisation du problème. Je vais continuer à regarder ce week-end.

### Compte rendu n°21 du 14/03/2019

J'ai essayé d'exécuter la campagne de test sur un des ordinateurs de Polytech, mais je n'ai pas pu le faire avec les problèmes d'administrateur ainsi que les fichiers dll n'étant pas à jour sur les VM. Je lancerais ainsi les tests sur mon ordinateur ce week-end.

J'ai de plus préparé les documents nécessaires pour mon oral de qualité de code de la semaine prochaine.

**Compte rendu n°22 du 21/03/2019**

J'ai lancé la campagne de tests et l'algorithme de Kirlik et Sayin a une erreur d'exécution à partir de 1200 (et non 4000) jeu de données.

L'erreur apparaît lors de la minimisation de la 2ème contrainte ("problème  $Q_k(e)$ ") sur l'appel de la fonction de minimisation : le problème ne semble ainsi pas venir de l'implémentation de l'algorithme.

Je vous donnerais avant la soutenance les résultats au niveau du temps de l'algorithme pour un jeu de données compris entre 50 et 1100.

**Compte rendu n°23 du 22/03/2019**

Après meilleur analyse, il semblerait que certaines erreurs puisse apparaître à partir de 1000 jeu de données.

# 10

## Bibliographie

- [1] Karima BOUIBEDE-HOCINE. « Enumération en ordonnancement multicritère : application à un problème bicritère à machines parallèles ». 2007TOUR4005. Thèse de doct. 2007, 1 vol. (138 f.) URL : <http://www.theses.fr/2007TOUR4005>.
- [2] V. Joseph BOWMAN. « On the Relationship of the Tchebycheff Norm and the Efficient Frontier of Multiple-Criteria Objectives ». In : *Multiple Criteria Decision Making*. Sous la dir. d'Hervé THIRIEZ et Stanley ZIONTS. Berlin, Heidelberg : Springer Berlin Heidelberg, 1976, p. 76-86. ISBN : 978-3-642-87563-2.
- [3] Federico Della CROCE, Christos KOULAMAS et Vincent T'KINDT. « A constraint generation approach for two-machine shop problems with jobs selection ». In : *European Journal of Operational Research* 259.3 (2017), p. 898-905. ISSN : 0377-2217. DOI : <https://doi.org/10.1016/j.ejor.2016.11.036>. URL : <http://www.sciencedirect.com/science/article/pii/S0377221716309614>.
- [4] C. DHAENENS, J. LEMESRE et E.G. TALBI. « K-PPM : A new exact method to solve multi-objective combinatorial optimization problems ». In : *European Journal of Operational Research* 200.1 (2010), p. 45-53. ISSN : 0377-2217. DOI : <https://doi.org/10.1016/j.ejor.2008.12.034>. URL : <http://www.sciencedirect.com/science/article/pii/S0377221708010709>.
- [5] Matthias EHRGOTT. « Multicriteria Optimization ». In : (2005). URL : [https://books.google.fr/books?hl=fr&lr=&id=8wGyB5Sa2CUC&oi=fnd&pg=PA1&dq=Ehrgott,+M.+\(2005\).Multicriteria+optimization.+Berlin:+Springer&ots=af\\_OFV0miY&sig=X1jRyPL0oLYfLK7YUblabVZlN6U#v=onepage&q&f=false](https://books.google.fr/books?hl=fr&lr=&id=8wGyB5Sa2CUC&oi=fnd&pg=PA1&dq=Ehrgott,+M.+(2005).Multicriteria+optimization.+Berlin:+Springer&ots=af_OFV0miY&sig=X1jRyPL0oLYfLK7YUblabVZlN6U#v=onepage&q&f=false).
- [6] Matthias EHRGOTT et Dagmar TENFELDE-PODEHL. « Computation of ideal and Nadir values and implications for their use in MCDM methods ». In : *European Journal of Operational Research* 151.1 (2003), p. 119-139. ISSN : 0377-2217. DOI : [https://doi.org/10.1016/S0377-2217\(02\)00595-7](https://doi.org/10.1016/S0377-2217(02)00595-7). URL : <http://www.sciencedirect.com/science/article/pii/S0377221702005957>.
- [7] P. K. ESWARAN, A. RAVINDRAN et H. MOSKOWITZ. « Algorithms for nonlinear integer bicriterion problems ». In : *Journal of Optimization Theory and Applications* 63.2 (nov. 1989), p. 261-279. ISSN : 1573-2878. DOI : [10.1007/BF00939577](https://doi.org/10.1007/BF00939577). URL : <https://doi.org/10.1007/BF00939577>.

- [8] F. W. GEMBICKI. « Vector optimization for control with performance and parameter sensitivity indices ». In : *Ph. D. thesis, Case Western Reserve Univ.* (1974). URL : <https://ci.nii.ac.jp/naid/10015595604/en/>.
- [9] Arthur M GEOFFRION. « Proper efficiency and the theory of vector maximization ». In : *Journal of Mathematical Analysis and Applications* 22.3 (1968), p. 618-630. ISSN : 0022-247X. DOI : [https://doi.org/10.1016/0022-247X\(68\)90201-1](https://doi.org/10.1016/0022-247X(68)90201-1). URL : <http://www.sciencedirect.com/science/article/pii/0022247X68902011>.
- [10] Y. HAIMES. « On a bicriterion formulation of the problems of integrated system identification and system optimization ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 1.3 (1971), p. 296-297. DOI : [10.1109/TSMC.1971.4308298](https://doi.org/10.1109/TSMC.1971.4308298). URL : <https://ci.nii.ac.jp/naid/30038849409/en/>.
- [11] Yacov Y. HAIMES et David A. WISMER. « A computational approach to the combined problem of optimization and parameter identification ». In : *Automatica* 8.3 (1972), p. 337-347. ISSN : 0005-1098. DOI : [https://doi.org/10.1016/0005-1098\(72\)90052-0](https://doi.org/10.1016/0005-1098(72)90052-0). URL : <http://www.sciencedirect.com/science/article/pii/0005109872900520>.
- [12] S. M. JOHNSON. « Optimal two- and three-stage production schedules with setup times included ». In : *Naval Research Logistics (NRL)* 1.1 (mar. 1954), p. 61-68. ISSN : 1931-9193. DOI : [10.1002/nav.3800010110](https://doi.org/10.1002/nav.3800010110). URL : <https://doi.org/10.1002/nav.3800010110>.
- [13] Gokhan KIRLIK et Serpil SAYIN. « A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems ». In : *European Journal of Operational Research* 232.3 (2014), p. 479-488. ISSN : 0377-2217. DOI : <https://doi.org/10.1016/j.ejor.2013.08.001>. URL : <http://www.sciencedirect.com/science/article/pii/S0377221713006474>.
- [14] Marco LAUMANN, Lothar THIELE et Eckart ZITZLER. « An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method ». In : *European Journal of Operational Research* 169.3 (2006), p. 932-942. ISSN : 0377-2217. DOI : <https://doi.org/10.1016/j.ejor.2004.08.029>. URL : <http://www.sciencedirect.com/science/article/pii/S0377221704005715>.
- [15] Melih ÖZLEN et Meral AZIZOĞLU. « Multi-objective integer programming : A general approach for generating all non-dominated solutions ». In : *European Journal of Operational Research* 199.1 (2009), p. 25-35. ISSN : 0377-2217. DOI : <https://doi.org/10.1016/j.ejor.2008.10.023>. URL : <http://www.sciencedirect.com/science/article/pii/S0377221708009624>.
- [16] Hoang Tuy REINER HORST. « Global Optimization : Deterministic Approaches ». In : (1995). URL : [https://books.google.fr/books?id=usFjGFvuBDEC&printsec=frontcover&dq=Horst,+R.,+%26+Tuy,+H.+\(1995\).Global+optimization&hl=fr&sa=X&ved=0ahUKEwjVnvam9ezdAhUFyIUkHTP6AwQ6AEIKDAA#v=onepage&q&f=false](https://books.google.fr/books?id=usFjGFvuBDEC&printsec=frontcover&dq=Horst,+R.,+%26+Tuy,+H.+(1995).Global+optimization&hl=fr&sa=X&ved=0ahUKEwjVnvam9ezdAhUFyIUkHTP6AwQ6AEIKDAA#v=onepage&q&f=false).
- [17] Richard M. SOLAND. « MULTICRITERIA OPTIMIZATION : A GENERAL CHARACTERIZATION OF EFFICIENT SOLUTIONS\* ». In : *Decision Sciences* 10.1 (), p. 26-38. DOI : [10.1111/j.1540-5915.1979.tb00004.x](https://doi.org/10.1111/j.1540-5915.1979.tb00004.x). eprint : <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-5915.1979.tb00004.x>. URL : <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5915.1979.tb00004.x>.
- [18] R. E. STEUER. « Multiple Criteria Optimization ». In : *Theory, Computation and Applications* (1986). URL : <https://ci.nii.ac.jp/naid/10011751184/en/>.

- [19] John SYLVA et Alejandro CREMA. « A method for finding the set of non-dominated vectors for multiple objective integer linear programs ». In : *European Journal of Operational Research* 158.1 (2004), p. 46-55. ISSN : 0377-2217. DOI : [https://doi.org/10.1016/S0377-2217\(03\)00255-8](https://doi.org/10.1016/S0377-2217(03)00255-8). URL : <http://www.sciencedirect.com/science/article/pii/S0377221703002558>.
- [20] V. T'KINDT, H. SCOTT et J.C. BILLAUT. *Multicriteria Scheduling : Theory, Models and Algorithms*. Springer Berlin Heidelberg, 2006. ISBN : 9783540247890. URL : [https://books.google.fr/books?id=BHj4%5C\\_XdncLsC](https://books.google.fr/books?id=BHj4%5C_XdncLsC).
- [21] D. TENFELDE-PODEHL. « A recursive algorithm for multiobjective combinatorial optimization problems with Q criteria. Unpublished. » In : (2003).
- [22] A. WIERZBICKI. « The use of reference objectives in multiobjective optimization, pages 468–486 ». In : *Fandel, G. and Gal, T. (1997). Multiple Criteria Decision Making. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin.* (1990).
- [23] L. ZADEH. « Optimality and non-scalar-valued performance criteria ». In : *IEEE Transactions on Automatic Control* 8.1 (jan. 1963), p. 59-60. ISSN : 0018-9286. DOI : [10.1109/TAC.1963.1105511](https://doi.org/10.1109/TAC.1963.1105511).

# 11

## Acronymes

**MOA** maîtrise d'ouvrage. 1, 19, 30, 31

**MODO** optimisation discret multi-objectif. 11, 13

**MOE** maîtrise d'œuvre. 1

**MOP** problème d'optimisation multi-objectif. 6



# Optimisation multicritère au sein du solveur Xpress/-Mosel de la société FICO

François GAUCHER

Encadrement : Vincent T'KINDT



En collaboration avec FICO

## Objectifs

- Implémentation des problèmes multicritères sous le solveur mathématique Xpress
- Analyse des performances de l'algorithme de Kirlik et Sayin avec un algorithme dédié
- Amélioration du code Mosel de l'algorithme



Technologies utilisés

## Problème traité

Soit un ensemble de  $n$  travaux disponibles. Chaque travail  $j$  doit être traité sans un ordre défini sur deux machines disponibles en permanence M1, M2 avec des temps de traitement connus respectivement  $a_j, b_j$ . L'objectif est de minimiser la date de fin des travaux  $d$  ainsi que le nombre de travaux rejetés  $\epsilon$ .

$$\begin{aligned} \min \quad & d \\ \sum_{i=1}^n x_i &= n - \epsilon \\ a_1 x_1 + \sum_{i=1}^n b_i x_i &\leq d \\ \sum_{i=1}^n a_i x_i + b_n x_n &\leq d \\ \sum_{i=1}^j a_i x_i + \sum_{i=j}^n b_i x_i &\leq d \quad \forall j = 2, \dots, n-1 \\ x_i &\in \{0, 1\} \quad \forall i = 1 \dots n \end{aligned}$$

Modèle mathématique du problème implémenté

## Mise en œuvre

- Implémentation du problème bi-critère sous le langage Mosel
- Lancement des deux algorithmes sous Visual Studio
- Comparatif des résultats
- Amélioration du code de l'algorithme de Kirlik et Sayin selon les résultats donnés

```
longueur :      50
====> Le nombre de solutions trouvees est similaire : 51
====> L'algorithme de Kirlik et Sayin s'est execute plus rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 1.875000
-----
longueur :      100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.660000
-----
longueur :      100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.953000
-----
```

Exemple de fichier de comparaison des résultats



# Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO

François GAUCHER

Encadrement : Vincent T’KINDT

## Objectifs

- Implémentation des problèmes multicritères sous le solveur mathématique Xpress
- Analyse des performances de l’algorithme de Kirlik et Sayin avec un algorithme dédié
- Amélioration du code Mosel de l’algorithme



Technologies utilisés



En collaboration avec FICO

## Problème traité

Soit un ensemble de  $n$  travaux disponibles. Chaque travail  $j$  doit être traitée sans un ordre défini sur deux machines disponibles en permanence M1, M2 avec des temps de traitement connus respectivement  $a_j, b_j$ . L’objectif est de minimiser la date de fin des travaux  $d$  ainsi que le nombre de travaux rejetés  $\epsilon$ .

$$\begin{aligned} \min \quad & d \\ \text{s.t.} \quad & \sum_{i=1}^n x_i = n - \epsilon \\ & a_1 x_1 + \sum_{i=1}^n b_i x_i \leq d \\ & \sum_{i=1}^n a_i x_i + b_n x_n \leq d \\ & \sum_{i=1}^j a_i x_i + \sum_{i=j}^n b_i x_i \leq d \quad \forall j = 2, \dots, n-1 \\ & x_i \in \{0, 1\} \quad \forall i \in 1 \dots n \end{aligned}$$

Modèle mathématique du problème implémenté

Ecole Polytechnique de l’Université François Rabelais de Tours  
Département Informatique  
64 avenue Jean Portalis, 37200 Tours, France  
[polytech.univ-tours.fr](http://polytech.univ-tours.fr)



POLYTECH<sup>®</sup>  
TOURS

Informatique

## Mise en œuvre

- Implémentation du problème bicritère sous le langage Mosel
- Lancement des deux algorithmes sous Visual Studio
- Comparatif des résultats
- Amélioration du code de l’algorithme de Kirlik et Sayin selon les résultats donnés

```
longueur : 50
====> Le nombre de solutions trouvees est similaire : 51
====> L'algorithme de Kirlik et Sayin s'est execute plus rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 1.875000
-----
longueur : 100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.660000
-----
longueur : 100
====> Le nombre de solutions trouvees est similaire : 101
====> L'algorithme de Kirlik et Sayin s'est execute moins rapidement
Temps d'execution de l'algorithme dedie : 2.000000
Temps d'execution de l'algorithme de Kirlik et Sayin : 6.953000
-----
```

Exemple de fichier de comparaison des résultats

# Optimisation multicritère au sein du solveur Xpress/Mosel de la société FICO

## Résumé

Ce rapport présente les résultats de recherche concernant l'efficacité de l'algorithme de Kirlik et Sayin dans la résolution de problèmes multi-objectifs. Cet algorithme est implémenté dans le solveur Xpress à l'aide du langage Mosel

## Mots-clés

Algorithme de Kirlik et Sayin, problème multi-objectif, Optimum de Pareto, FICO, Xpress, Mosel

## Abstract

This report presents research results regarding the effectiveness of Kirlik and Sayin's algorithm in solving multi-objective problems. This algorithm is implemented in the Xpress solver using the Mosel language

## Keywords

Kirlik and Sayin's algorithm, multi-objective problems, optimum of Pareto, FICO, Xpress, Mosel

Entreprise



Tuteur entreprise

Sebastien LANNÉZ (Ingénieur)

Étudiant

François GAUCHER (DI5)

Tuteur académique

Vincent T'KINDT