

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement 2016-2017

Nouveaux problèmes d'ordonnancement

Tuteurs académiques

Jean-Charles BILLAUT

Thanh Thuy Tien TA

Étudiant

Kivin RINGARD (DI5)



Liste des intervenants

Nom	Email	Qualité
Kivin RINGARD	kivin.ringard@etu.univ_tours.fr	Étudiant DI5
Jean-Charles BILLAUT	jean-charles.billaut@univ-tours.fr	Tuteur académique, Département Informatique
Thanh Thuy Tien TA	ttttien@gmail.com	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Kivin Ringard susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Jean-Charles Billaut et Thanh Thuy Tien TA susnommés les tuteurs académiques.

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

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

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

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

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

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

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



Pour citer ce document

Kivin Ringard, *Nouveaux problèmes d'ordonnancement*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2016-2017.

```
@mastersthesis{
  author={Ringard, Kivin},
  title={Nouveaux problèmes d'ordonnancement},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2016-2017}
}
```



Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Liste des tableaux	vi
Remerciements	1
Introduction	2
I Partie 1 : Objectif, Recherche et État de l’art	3
1 Contexte de réalisation	4
1 Contexte.....	4
2 Description du problème	4
3 Objectif	5
2 Description générale	7
1 Environnement du projet	7
2 Caractéristiques des utilisateurs	7
3 Fonctionnalités et structure générale du système de génération et de résolution....	7
4 Contrainte de développement, d’exploitation et de maintenance.....	8

5	Description des interfaces externes du logiciel	8
5.1	Interfaces homme/machine.....	8
5.2	Interfaces logiciel/logiciel	8
6	Conditions de fonctionnement.....	9
6.1	Performances.....	9
6.2	Capacités.....	9
6.3	Contrôlabilité.....	9
6.4	Sécurité	9
6.5	Facteurs de qualité.....	9
3	État de l'art	10
1	Contexte et caractéristiques de l'ordonnancement.....	10
1.1	Qu'est-ce que l'ordonnancement ?.....	10
1.2	Vocabulaire	10
1.3	Domaines d'application.....	10
1.4	Définition d'un problème.....	11
2	Description générale des éléments de recherches	12
2.1	Les problèmes à une machine	12
2.2	Théorie de la complexité	14
2.3	Treillis des permutations	14
2.4	Problème d'affectation : méthode hongroise.....	15
3	Les approches de résolution du problème	17
3.1	La méthode arborescente Branch and Bound	17
3.2	Le programme MILP.....	17
3.3	Le programme DP.....	18
II	Partie 2 : Développement	19
4	Analyse et Mise en œuvre	20
1	Architecture générale des systèmes	20
2	Implémentation	20
2.1	Générateur d'instances facile et difficile.....	21
2.2	Solveur CPLEX.....	22
2.3	Solveur BB.....	23
5	Application des tests expérimentaux et résultat	25
1	Conditions d'expérimentation	25
2	Déroulement des tests expérimentaux.....	25
2.1	Génération des instances tests.....	25
2.2	Lancement des tests	25
2.3	Résultats des tests expérimentaux	26

6	Reproductibilité	28
1	Générateur d'instance	28
1.1	Fichier de configuration	28
1.2	Lancement du programme	29
1.3	Exemple	29
2	Solveur	31
2.1	Fichier d'instance.....	31
2.2	Lancement du programme	31
2.3	Fichier de résultat	31
2.4	Spécification.....	32
	Conclusion	33
	Bibliographie	34
	Webographie	35
III	Annexe	36
	Description des fonctionnalités du programme de résolution et de génération	37
	Gestion de projet	38
	Cahier de tests	45
	Guide d'installation	48
	Guide d'utilisation du Programme	56
	Comptes rendus hebdomadaires	57



Table des figures

1 Contexte de réalisation	
1 Treillis des permutations de taille 3 et 4.....	5
2 Description générale	
1 Diagramme de cas d'utilisation du générateur.....	7
2 Diagramme de cas d'utilisation du solveur	8
3 État de l'art	
1 Treillis des permutations.....	15
4 Analyse et Mise en œuvre	
1 Diagramme de classe.....	21
5 Application des tests expérimentaux et résultat	
1 Script d'exécution.....	26
2 Utilisation CPU avec Cplex.....	27
3 Utilisation CPU avec Branch and Bound	27
6 Reproductibilité	
1 Fichier de configuration	28
2 Mode Releasee	29
3 Emplacement de l'exécutable.....	29
4 Exemple d'instance facile pour n=30	30

5	Exemple d'instance difficile pour $n=30$	30
6	Exemple de fichier de résultat.....	31
Gestion de projet		
7	Diagramme de Gant prévisionnel	39
8	Diagramme de Gant final : S10	39
Guide d'installation		
9	Téléchargement de Cplex	48
10	Choix de Cplex	49
11	Début de l'installation.....	49
12	Introduction.....	50
13	Contrat de licence.....	50
14	Include des répertoires.....	51
15	Représentation : Include des répertoires	51
16	Définition de préprocesseur.....	52
17	Représentation : Définition des préprocesseur.....	52
18	Bibliothèques Cplex	53
19	Représentation : Bibliothèques Cplex.....	53
20	Dépendances Cplex	54
21	Représentation : Dépendances Cplex	54
22	Ordre bibliothèques Cplex.....	55
Guide d'utilisation du Programme		
23	Répertoire du programme.....	56

Liste des tableaux

1 Contexte de réalisation

- 1 Exemple d'un problème d'ordonnancement 4
- 2 Plusieurs problèmes possibles 5

3 État de l'art

- 1 Problème d'ordonnancement 1 machine 12
- 2 Ordonnancement sans retard..... 12
- 3 Séquence associé à WSPT 12
- 4 Séquencement par WSPT 12
- 5 Séquencement par SPT 13
- 6 Séquence associé à EDD 13
- 7 Séquencement par EDD 13
- 8 Séquencement par Moore-Hodgson..... 13
- 9 Matrice des coûts 15
- 10 1ère étape de la méthode Hongroise 15
- 11 2ème étape de la méthode Hongroise 16
- 12 3ème étape de la méthode Hongroise 16
- 13 4ème étape de la méthode Hongroise 16
- 14 Suite de la 4ème étape de la méthode Hongroise 16
- 15 5ème étape de la méthode Hongroise 16
- 16 Résultat obtenu via la méthode Hongroise 17

4 Analyse et Mise en œuvre

- 1 Représentation de la séquence 24



Remerciements

Je tiens à remercier Jean-Charles Billaut et Thanh Thuy Tien TA pour avoir proposé ce sujet de Recherche et Développement, ainsi que pour leurs disponibilités tout au long du projet et pour leurs aides face à mes interrogations et mes difficultés.



Introduction

Ce projet a été réalisé dans le cadre du projet Recherche et Développement.

Ce sujet est proposé par l'équipe "Recherche Opérationnelle - Ordonnancement Transport"(Root) et il est encadré par Jean-Charles Billaut et Thanh Tuy Tien Ta qui représente la MOA.

Il s'agit de réaliser un générateur d'instance aléatoire (facile est difficile) ainsi qu'une application de résolution (axé sur deux méthodes) pour le problème : $1||\tilde{d}_j|\sum w_j.p_j$ (pour le cas d'ordonnancement d'atelier à une machine) afin d'obtenir une ensemble de solution valable.

Ce rapport se divise en deux parties, une première sur la "Recherche" et la deuxième sur le "Développement".

Dans la première partie je vais présenter le cahier de spécification (contexte, objectif, but) puis toutes mes recherches et analyses liées à l'état de l'art (Présentation, Définition, Solution existantes).

Dans la deuxième partie je présenterais en détail l'implémentation des générateurs ainsi que du programme de résolution.

En annexe vous trouverez : Un rapport sur l'organisation de ce projet, un guide d'installation pour la librairie CPLEX ainsi qu'un cahier de tests.

Première partie

Partie 1 : Objectif, Recherche et État de l'art

1

Contexte de réalisation

1 Contexte

Les problèmes d'ordonnancement consistent à organiser la réalisation de tâches en prenant en compte plusieurs contraintes : temporelles (délais, contraintes d'enchaînements, ...) et de ressources (nombre de machine disponible, ...). Exemple : n tâches à ordonnancer sur 2 machines, le critère d'optimisation est de minimiser le maximum des dates de fin d'exécutions des tâches : $F2||C_{max}$

Dans le cadre de recherche au sein de l'équipe ROOT nous allons nous intéresser aux problèmes liés à l'utilisation d'une seule ressource.

2 Description du problème

Un problème d'ordonnancement sur une machine est caractéristique des problèmes d'ordonnancement d'atelier pour lesquels il n'y a qu'une seule ressource (machine disponible). Exemple : n tâches à effectuer sur 1 seule machine, le critère d'optimisation est de minimiser l'écart maximum par rapport aux délais : $1||L_{max}$

Dans ce projet nous nous intéressons aux problèmes d'ordonnancement à 1 machines avec contraintes de date de fin impérative, en d'autres termes, une tâche ne peut pas terminer son exécution après une date donnée.

Dans cette situation, la méthode de résolution optimale consiste à trier les tâches dans l'ordre des dates de fins impératives croissantes. Cette résolution peut se faire en temps polynomiale en $O(n \log(n))$.

Table 1 – Exemple d'un problème d'ordonnancement

Job (i)	1	2	3	4	5
Processing time (P_i)	2	1	4	7	3
Due date (d_i)	3	4	8	9	20
Completion time (C_i)	2	3	7	14	17
$L_i = C_i - d_i$	-1	-1	-1	5	-3
Deadline $\tilde{d}_i = d_i + L_{max}$	8	9	13	14	25

Les date de fin impératives (\tilde{d}_i) sont calculés pour chaque job par la somme de leur date de fin (d) et du plus haut retard algébrique (L_{\max}).

Voici plusieurs problèmes possibles avec W_j et P_j non défini :

Table 2 – Plusieurs problèmes possibles

Problème	Nature du problème
$1 \tilde{d}_j \sum w_j P_j$	problème NP-difficile
$1 \tilde{d}_j, p_j = 1 \sum w_j P_j$	problème non déterminé
$1 \tilde{d}_j \sum w_j H_j$	distance de Hamming (avec H_j non défini), problème non déterminé
$1 \tilde{d}_j \sum j P_j$	problème non déterminé

La difficulté étant de trouver un problème qui ne soit pas NP-difficile, toutefois la recherche de la complexité n'est pas l'objectif de ce PRD.

3 Objectif

L'objectif est de déterminer un ensemble de solutions optimales pour le problème : $1 | \tilde{d}_j | \sum w_j p_j$ où il faut minimiser $\sum w_j p_j$, avec \tilde{d}_j qui correspond à la deadline, W_j qui correspond au poids et p_j qui correspond à la position du job j .

Pour cela plusieurs méthodes algorithmiques seront implémentées dont le critère à optimiser est le « level » dans la recherche de l'arbre. Ainsi nous partons d'un treillis des permutations dont chaque nœud correspond à une solution possible.

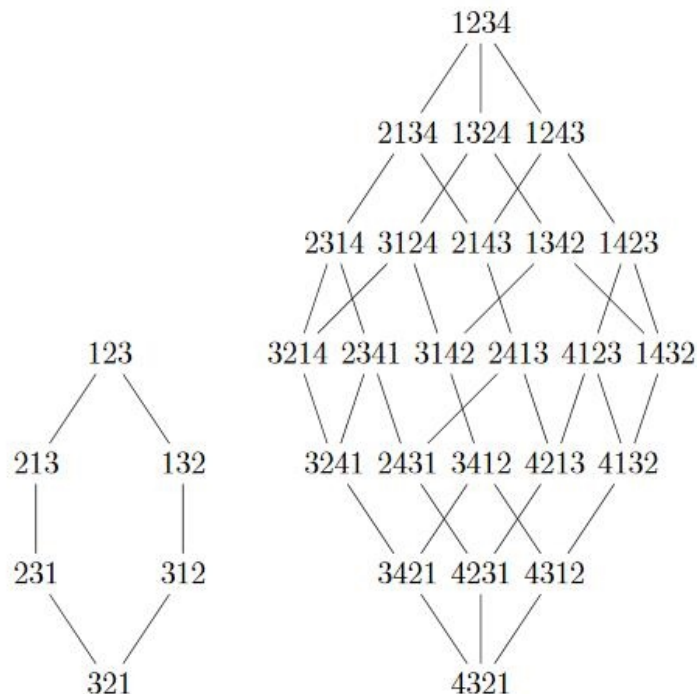


Figure 1 – Treillis des permutations de taille 3 et 4

L'une des fonctions à développer est une méthode arborescente branch and bound, qui consiste, à partir de la dernière tâche (celle avec la date de fin impérative la plus élevée), d'obtenir les tâches antérieures afin d'obtenir une solution valable.

La deuxième consiste à développer un programme MILP (Mixed-Integer Linear Programming) via les librairies CPLEX. Ceci est axé sur un modèle contenant des variables, méthodes ainsi que des contraintes.

La dernière consiste à implémenter un programme dynamique (DP : Dynamic Programming). Afin d'obtenir des instances pour y appliquer ces méthodes, un « générateur » est également implémenté permettant la génération de 30 instances aléatoires pour chaque valeur de n , suivant un fichier de configuration.

2

Description générale

1 Environnement du projet

Ce projet ne dépend d'aucun environnement matériel ou logiciel. Il n'est donc pas nécessaire de l'intégrer dans un environnement spécifique.

2 Caractéristiques des utilisateurs

La MOA sera l'utilisatrice de ces algorithmes, de par leur connaissance en informatique et en ordonnancement.

3 Fonctionnalités et structure générale du système de génération et de résolution

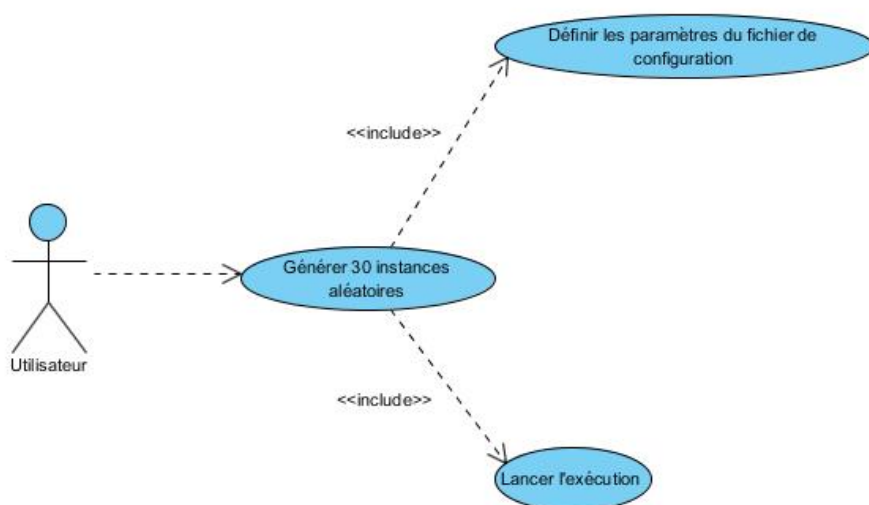


Figure 1 – Diagramme de cas d'utilisation du générateur

Le fonctionnement du système est le suivant : A travers un fichier de configuration, 30 instances (30 fichiers txt) aléatoires (de taille n définie) sont générées via un générateur.

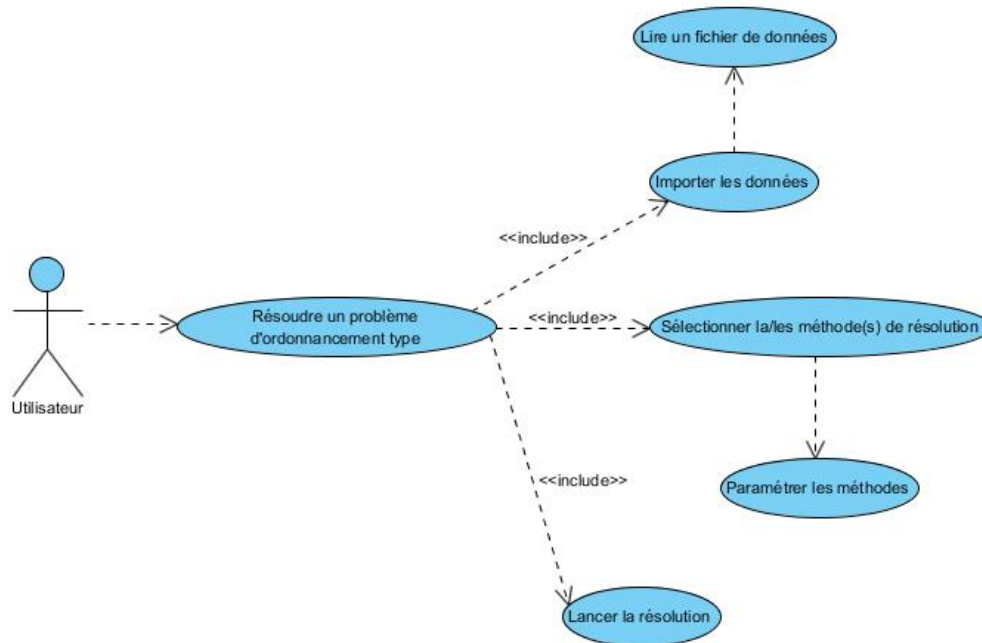


Figure 2 – Diagramme de cas d'utilisation du solveur

Ces 30 fichiers sont ensuite lus par le programme de résolution qui appliquera les 3 méthodes de résolution pour chaque instance. A la fin du traitement un fichier xls est généré afin de permettre une comparaison simple des résultats. Ce fichier contiendra le niveau (« level ») ainsi que le temps CPU pour chaque instance et chaque méthode.

4 Contrainte de développement, d'exploitation et de maintenance

Il n'y a pas de contrainte liée au matériel.

La contrainte imposée sur le langage est le C++, l'IDE utilisé sera donc Visual Studio professionnel 2010. Les libraires CPLEX seront utilisées. Les résultats devront être écrits dans un fichier csv.

5 Description des interfaces externes du logiciel

5.1 Interfaces homme/machine

Le programme de résolution s'exécutera via un programme exécutable qui prendra en argument un fichier txt. Un script batch sera créé pour l'exécution des différentes commandes.

5.2 Interfaces logiciel/logiciel

Le programme de résolution devra interagir avec Excel par la génération d'un fichier csv contenant tous les résultats.

6 Conditions de fonctionnement

6.1 Performances

Les méthodes devront pouvoir donner un résultat valable, tout en minimisant le niveau avec un temps CPU correct.

6.2 Capacités

Il n'y a pas de limite, cependant dans le cas de tests d'exécution de l'application, 30 instances (donc 30 fichiers textes) par taille de problème (valeur de n) seront utilisées lors de son exécution.

6.3 Contrôlabilité

Afin de pouvoir suivre le programme (en particulier dans les phases de tests), il sera possible d'afficher des messages en cours d'exécution (par exemple le numéro de l'instance exécutée).

6.4 Sécurité

Aucune demande particulière n'a été faite de la part de la MOA pour la sécurité.

6.5 Facteurs de qualité

Ce PRD pouvant être repris pour des améliorations/reprises futures, il est nécessaire de fournir un support de qualité en particulier sur les rapports, la documentation et les commentaires au sein du code. Afin de fournir un code opérationnel et optimisé, des tests seront effectués et des rendez-vous seront pris régulièrement.

3

État de l'art

1 Contexte et caractéristiques de l'ordonnancement

1.1 Qu'est-ce que l'ordonnancement ?

Les problèmes d'ordonnancement sont présents dans de nombreux domaines (informatique, milieu industriel, ...). On les retrouve dans l'organisation des machines d'une usine, l'organisation d'activités de services ou même l'allocation dynamique des ressources (pour les systèmes d'exploitation par exemple). Ces problèmes consistent à organiser la réalisation de tâches en prenant en compte plusieurs contraintes : temporelles (délais, contraintes d'enchaînements, ...) et aussi sur l'utilisation et la disponibilité des ressources requises. L'ordonnancement est donc lié à des impératifs de productivité, de flexibilité et de réactivité.

1.2 Vocabulaire

Afin de bien comprendre ces problèmes, plusieurs mots de vocabulaire sont à connaître :

- Il s'agit de la réalisation de **tâches**.
- Ces tâches utilisent des **ressources** (exemple : machine).
- Sur ces tâches et les ressources il y a des **contraintes**.
- Enfin il y a toujours un **critère d'optimisation** à respecter (exemple : réduire au maximum le retard).

1.3 Domaines d'application

Les problèmes d'ordonnancement peuvent être divisés en deux domaines principaux d'application :

1. Si les tâches concourent à la réalisation d'un nombre réduit de projets (caractère unique et très peu récurrent) mobilisant ainsi l'ensemble des ressources sur une période assez grande (construction navale, tunnel sous la manche, ...). On parle ainsi **d'ordonnancement de projet**.

- (a) Si les ressources sont partageables mais disponible en quantités limitée (personnes polyvalentes, espace limité de traitement, de stockage, de déplacement) : Ordonnement sous contraintes de ressources cumulatives.
 - (b) Dans cette configuration, le chef de projet doit veiller au respect des engagements auprès du client. Il est responsable d'une équipe spécialisée et doit gérer les ressources humaines et techniques.
2. Le deuxième domaine est celui de l'ordonnement de la production des unités spécialisées ou **Ordonnement d'atelier**. Il s'agit de la réalisation de produit (variabilité au niveau de la demande, et au niveau des processus de fabrication). Une commande de produit = un projet différent.
- (a) Cependant il est possible d'avoir un problème/complexité au niveau de la coordination des projets (**Une unité de ressource = une machine**, elle ne peut donc servir que pour un seul projet à la fois), on parle donc de Problématique multi-projets.
 - (b) L'activité de service est une problématique identique à celle de l'ordonnement d'atelier avec une activité standardisé et des ressources non partageables.

L'ordonnement a lieu après la planification et avant le lancement.

1.4 Définition d'un problème

Les problèmes d'ordonnement possèdent leur propre notation qui sera utilisés par la suite dans ce rapport.

Voici les termes utilisés pour la définition d'un problème :

- Tache : j
- Date de début : t_j
- Date de fin : c_j
- Date de fin au plus tard : d_j
- Date de fin impérative : \tilde{d}_j
- Durée : $p_j = c_j - t_j$
- Moyen : k avec une intensité : a_{jk}
- Poids : W_j
- Retard algébrique : $L_j = C_j - d_j$
- Date de fin de la dernière tâche : C_{\max}
- L'écart maximum à la date de fin : L_{\max}

Voici la notation :

$\alpha|\beta|\gamma$ avec :

α : décrit les caractéristiques des machines, représente l'environnement. Il peut valoir : 1 pour une machine, P pour machines parallèles, F pour flow shop et J pour job shop.

β : est l'ensemble des contraintes.

γ : le (ou les) critère(s) à optimiser.

2 Description générale des éléments de recherches

2.1 Les problèmes à une machine

Les problèmes d'ordonnancement à 1 machine proviennent du domaine de l'ordonnancement d'atelier. Dans cette situation il n'y a qu'une seule machine comme ressource pour la réalisation des tâches. Il s'agit donc d'un problème où $\alpha = 1$.

Dans cette situation, une règle est implémentée qui consiste à donner une priorité à chaque tâche suivant un critère (plus petit temps opératoire, plus petite date de fin, ...). Généralement il y a un ordre de priorité par rapport aux opérations prêtes (date de début au plus tôt), ce qui permet un ordonnancement sans retard. Toutefois suivant les problèmes, certains peuvent être NP-difficiles.

Table 1 – Problème d'ordonnancement 1 machine

Tâches j	1	2	3	4	5
Durée p_j	2	3	6	5	1
Poids w_j	1	3	2	1	2
Date échue d_j	8	11	7	15	1

Pour la résolution de ces problèmes, 2 cas de figures sont à prendre en compte :

— **Sans considération des dates de fin au plus tard :**

- Critère de minimisation de la durée totale. Son calcul s'obtient par la somme des durées : $C_{\max} = \sum p_j$

Table 2 – Ordonnancement sans retard



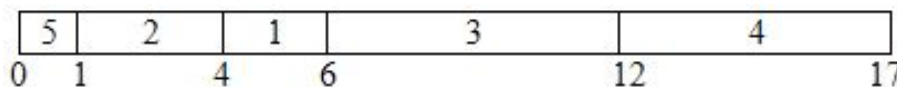
La durée totale s'élève à 17.

- Temps de cycle : séquencement des opérations dans l'ordre croissant des durées/poids. Il s'agit de la règle WSPT (Weighted Shortest Processing Time first) ou règle de Smith. Ceci permet de minimiser le temps moyen pondéré de séjour.

Table 3 – Séquence associé à WSPT

p_j/w_j	0.5	1	2	3	5
j	5	2	1	3	4

Table 4 – Séquencement par WSPT



Si les pondérations disparaissent, la règle WSPT se simplifie en règle SPT qui minimise le temps moyen de séjour dans l'atelier ou la durée moyenne de réalisation.

— **Avec considération des dates de fin au plus tard :**

- Retard algébrique moyen : on trie les tâches par ordre croissant des durées (règle SPT) afin de minimiser L (écart moyen par rapport aux délais : date échue).

Table 5 – Séquencement par SPT

5	1	2	4	3	
0	1	3	6	11	17

Délai des tâches : la tâche 5 vaut 1, la tâche 3 vaut 7, la tâche 1 vaut 8, la tâche 2 vaut 11 et la tâche 4 vaut 15. Donc L vaut :

$$L = ((1-1) + (3-8) + (6-11) + (11-15) + (17-7))/5 = -0.8$$

- Plus grand retard : ordre croissant des dates échues, ce qui permet de minimiser L_{\max} (écart maximum par rapport aux délais) et T_{\max} (retard maximum).

Utilisation de la règle EDD (Earliest Due Date first) ou règle de Jackson.

On donne la priorité aux tâches les plus "pressées", ce qui permet de diminuer le retard sauf dans certains cas :

- Plusieurs tâches sont en retard, cette règle va donner la priorité à des tâches déjà en retard, créant encore plus de retard.
- Suivant les durées opératoires, on peut donner priorité à une tâche longue pouvant ainsi entraîner du retard sur des tâches courtes.

Table 6 – Séquence associée à EDD

d_j	1	7	8	11	15
j	5	3	1	2	4
C_j	1	7	9	12	17
$L_j = C_j - d_j$	0	0	1	1	2

Table 7 – Séquencement par EDD

5	3			1	2	4	
0	1	7			9	12	17

Ici $L_{\max} = T_4 = 2$

- Nombre d'opérations en retard. Règle de More-Hodgson.

L'objectif de cette règle est de minimiser le nombre d'opération en retard. Le but est de construire un ordonnancement basé sur EDD, et dès qu'un retard est détecté la tâche de plus grande durée déjà placée est reportée en fin d'ordonnancement.

Si nous prenons l'exemple de la table 1, la tâche 5 est de plus petite date de fin (1), elle est donc placée en première position, ensuite arrive la tâche 3 (date de fin = 7) pour laquelle il n'y a toujours pas de retard. Puis c'est la tâche 1 qui est placée (date de fin = 8), cependant un retard est détecté ($9 > 8$: retard de 1). Par conséquent parmi les tâches déjà placée, on rejette (placée à la fin de l'ordonnancement) celle de plus grande durée (ici la 3) afin de ne plus avoir de retard. On continue la procédure en plaçant les tâches 2 puis 4. Au final nous obtenons la séquence suivante :

Table 8 – Séquencement par Moore-Hodgson

5	1	2	4	3	
0	1	3	6	11	17

C'est dans ce contexte que tournera la PRD, avec la résolution d'un problème d'ordonnancement lié à une machine sous contrainte de date de fin impérative (\tilde{d}_i). Le but étant de déterminer un ensemble de solutions opérationnelles.

2.2 Théorie de la complexité

Il est important de faire un point sur la théorie de la complexité. En effet il est utile de pouvoir définir la difficulté de résolution d'un problème d'ordonnancement. Ces problèmes se regroupent principalement en 2 classes : polynomial et NP-difficile (que l'on cherche à éviter).

Voici les principales classes de difficultés existantes :

Classe L : c'est quand un problème de décision peut être résolu par un algorithme déterministe en espace logarithmique par rapport à la taille de l'instance.

Classe NL : identique à L mais pour un algorithme non-déterministe.

Classe P : correspond à un problème de décision qui peut être décidé par un algorithme déterministe en un temps polynomial par rapport à la taille de l'instance. On qualifie alors le problème de polynomial.

Classe NP : c'est la classe des problèmes de décision pour lesquels la réponse oui peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille de l'instance.

Classe Co-NP : l'équivalent de la classe NP avec la réponse non.

Classe PSPACE : les problèmes décidables par un algorithme déterministe en espace polynomial par rapport à la taille de son instance.

Classe NSPACE ou NPSPACE : les problèmes décidables par un algorithme non-déterministe en espace polynomial par rapport à la taille de son instance.

Classe EXPTIME : les problèmes décidables par un algorithme déterministe en temps exponentiel par rapport à la taille de son instance.

Il existe quelques propriétés : $P \subset NP$ et par conséquent $P \subset Co-NP$, ainsi que $NP \subset PSPACE = NPSPACE$ et $Co-NP \subset PSPACE$.

2.3 Treillis des permutations

Qu'est-ce qu'une permutation ?

Une permutation est un « mot » de taille n dans lequel chaque « lettre » n'apparaît qu'une seule fois (exemple : 1234, 241, 873159).

Dans le cas d'un problème d'ordonnancement, un mot correspond à une solution possible et les lettres correspondent à un job.

Dans le cas d'un treillis de permutation, on applique un « ordre faible » droit sur les permutations, ce qui veut dire qu'à chaque étape on échange deux valeurs consécutives croissantes.

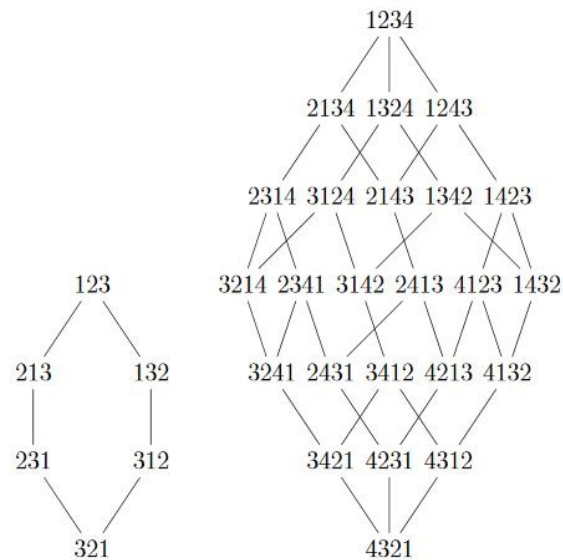


Figure 1 – Treillis des permutations

Prenons le mot 123, à cette étape on peut échanger le 1 avec le 2 (car $1 < 2$ et sont consécutifs) ou le 2 avec le 3 (car $2 < 3$ et sont consécutifs) pour obtenir 2 nouveaux mots à savoir 213 ou 132. Cependant il n'est pas possible d'échanger le 1 avec le 3 car non consécutif.

Au niveau de l'ordonnancement, on peut considérer ce treillis comme un ensemble de solution possible respectant la contrainte imposé par le problème. L'objectif d'un programme de résolution est donc de déterminer le plus bas « level » (le plus bas niveau possible dans l'arbre) afin d'obtenir un maximum de solution.

2.4 Problème d'affectation : méthode hongroise

La méthode hongroise est utilisée pour « simplifier » la résolution des problèmes d'affectation, elle se décompose en 5 étapes.

Exemple d'une matrice des coûts :

Table 9 – Matrice des coûts

VP	1	2	3	4
A	24	10	21	11
B	14	22	10	15
C	15	17	20	19
D	11	19	14	13

La **1ère étape** : consiste à réduire les lignes : On va créer une nouvelle matrice des coûts en choisissant le cout minimal sur chaque ligne et en la soustrayant de chaque coût sur la ligne.

Table 10 – 1ère étape de la méthode Hongroise

VP	1	2	3	4	REDUIT DE :
A	14	0	11	1	10
B	4	12	0	5	10
C	0	2	5	4	15
D	0	8	3	2	11

La **2ème étape** : consiste à réduire les colonnes : on effectue la même chose que pour l'étape 1 sauf que l'on regarde les colonnes et non les lignes.

Table 11 – 2ème étape de la méthode Hongroise

VP	1	2	3	4
A	14	0	11	0
B	4	12	0	4
C	0	2	5	3
D	0	8	3	1
REDUIT DE :	0	0	0	1

La **3ème étape** : consiste à déterminer le nombre minimal de « lignes » nécessaires sur les lignes et colonnes de la matrice pour couvrir tous les zéros. Si ce nombre est égal au nombre de colonnes (ou de ligne), la matrice est réduite et on peut passer à l'étape 5 sinon on passe à l'étape 4.

Table 12 – 3ème étape de la méthode Hongroise

VP	1	2	3	4
A	14	0	11	0
B	4	12	0	4
C	0	2	5	3
D	0	8	3	1

Ici le nombre de ligne = 3 < 4 on passe donc à l'étape 4.

La **4ème étape** : consiste à trouver la cellule contenant la valeur minimum non-couverte par une ligne (ici la valeur vaut 1). Ensuite on soustrait cette valeur de toutes les cellules non-couvertes et on ajoute cette valeur aux cellules situées à l'intersection de 2 lignes. Puis enfin on retourne à l'étape 3.

Table 13 – 4ème étape de la méthode Hongroise

VP	1	2	3	4
A	13	0	10	0
B	3	11	0	3
C	0	1	4	2
D	0	7	2	0

Table 14 – Suite de la 4ème étape de la méthode Hongroise

VP	1	2	3	4
A	13	0	10	0
B	3	11	0	3
C	0	1	4	2
D	0	7	2	0

Ici le nombre de ligne = 4. On passe donc à l'étape 5.

La **5ème et dernière étape** : consiste à déterminer la solution optimale.

Table 15 – 5ème étape de la méthode Hongroise

VP	1	2	3	4
A	15	0	12	0
B	4	11	0	3
C	0	1	5	2
D	0	7	3	0

Table 16 – Résultat obtenu via la méthode Hongroise

VP	USINE	COUT
A	2	10
B	3	10
C	1	15
D	4	13

Le cout total est de 48.

3 Les approches de résolution du problème

3.1 La méthode arborescente Branch and Bound

La méthode arborescente Branch and Bound est une procédure par évaluation et séparation progressive qui consiste à énumérer des solutions suivant certaines propriétés par rapport au problème énoncé. Cette technique permet d'éliminer des solutions partielles qui ne mènent pas à la solution recherchée. On arrive ainsi souvent à obtenir la solution recherchée en des temps raisonnables.

La performance de cette méthode dépend de la fonction permettant de définir une borne (LB et UB) sur ces solution pour soit les exclure soit les maintenir comme solution potentielles.

Généralement cette méthode est basée sur une arborescence dont la racine représente l'ensemble des solutions du problème.

Pour appliquer cette méthode il faut donc :

- Un moyen de calcul d'une borne inférieure
- Une stratégie pour subdiviser l'espace de recherche
- Un moyen de calcul d'une borne supérieure

Le cheminement est le suivant :

On définit la borne inférieure et supérieure sur la racine, si les deux bornes sont égales alors une solution optimale est trouvée sinon on divise l'ensemble du problème suivant la stratégie employée (les sous-problèmes deviennent les enfants de la racine).

A chacun de ses sous-problème on recalcule la borne inférieure (ou les deux). Si une solution optimale est trouvée pour un sous-problème, elle est réalisable, mais pas nécessairement optimale pour le problème de départ. Toutefois comme elle est réalisable, elle peut être utilisée pour éliminer toute sa descendance : si la borne inférieure d'un nœud est supérieure à la valeur d'une solution déjà connue alors on peut dire que la solution optimale globale ne peut être contenue dans le sous-ensemble de solution représenté par ce nœud. Donc on « coupe » le nœud.

La recherche continue jusqu'à ce que tous les nœuds soient explorés ou éliminés.

3.2 Le programme MILP

Cplex est un solveur, utilisé pour la résolution de programmes linéaires mixtes mais aussi de problèmes quadratiques.

Il peut être directement exécuté via son IDE, mais il est également possible d'incorporer un ensemble de bibliothèques dans un autre IDE afin de pouvoir accéder à ses composantes (pour des applications C++, Java ou .NET).

Il fonctionne suivant des données, des variables et des contraintes.

Dans le cadre de ce PRD les données sont :

- n : le nombre de job
- P_j : La position du job j
- W_j : le poids du job j
- D_j : la date de fin du job j

Les variables sont :

- $X_{jk} = 1$ si « j » en position « k », 0 sinon

Les contraintes sont :

- $\sum_{j=1}^n X_{jk} = 1$
- $\sum_{k=1}^n X_{jk} = 1$
- $\sum_{h=1}^k \sum_{j=1}^n X_{jh} \leq \sum_{j=1}^n D_j \cdot X_{jk}$
- $\text{MIN} \sum_{j=1}^n W_j \sum_{k=1}^n k \cdot X_{jk}$

3.3 Le programme DP

La programmation dynamique est une méthodologie pour concevoir des algorithmes permettant de résoudre certains problèmes d'optimisation.

Sa conception se décompose en quatre étapes :

- Caractérisation de la structure d'une solution optimale.
- Définition récursive de la valeur de la solution optimale.
- Calcul ascendant de la valeur de la solution optimale.
- Construction de la solution optimale à partir des informations obtenues à l'étape précédente.

Axée sur deux stratégies :

- Décomposer le problème en une séquence de problèmes.
- Établir une relation de récurrence entre les solutions optimales des problèmes.

Elle peut donc être employée pour résoudre en temps polynomial des problèmes de type : parcours optimal dans un atelier de montage, calcul d'un arbre binaire de recherche optimale.

Deuxième partie

Partie 2 : Développement

4

Analyse et Mise en œuvre

1 Architecture générale des systèmes

Les programmes sont lancés via différents exécutables. L'ensemble des commandes (exécutable, génération du csv) pourront être exécutées via un script batch.

Le programme de génération des instances (« Générateur ») contiendra :

- Une classe abstraite « generator » qui va contenir la méthode de lecture fichier ainsi que la méthode virtuelle de génération d'instance.
- Une classe « Easygenerator » (hérite de generator) qui va contenir la méthode de génération d'instance facile.
- Une classe « Hardgenerator » (hérite de generator) qui va contenir la méthode de génération d'instance difficile.

Le programme de résolution est composé de deux classes :

- Une classe « solveurBB » contenant les informations et méthodes permettant d'obtenir une solution via la méthode arborescente Branch and Bound.
- Une classe « solveurMILP » contenant les informations et méthodes permettant d'obtenir une solution via un programme MILP.
- Une classe « approachDP » contenant les informations et méthodes permettant d'obtenir une solution via un programme DP (cette classe n'est pas implémentée car elle n'est pas pertinente).

2 Implémentation

Le programme (Générateur et Solveur) est caractérisé par le diagramme de classe suivant :

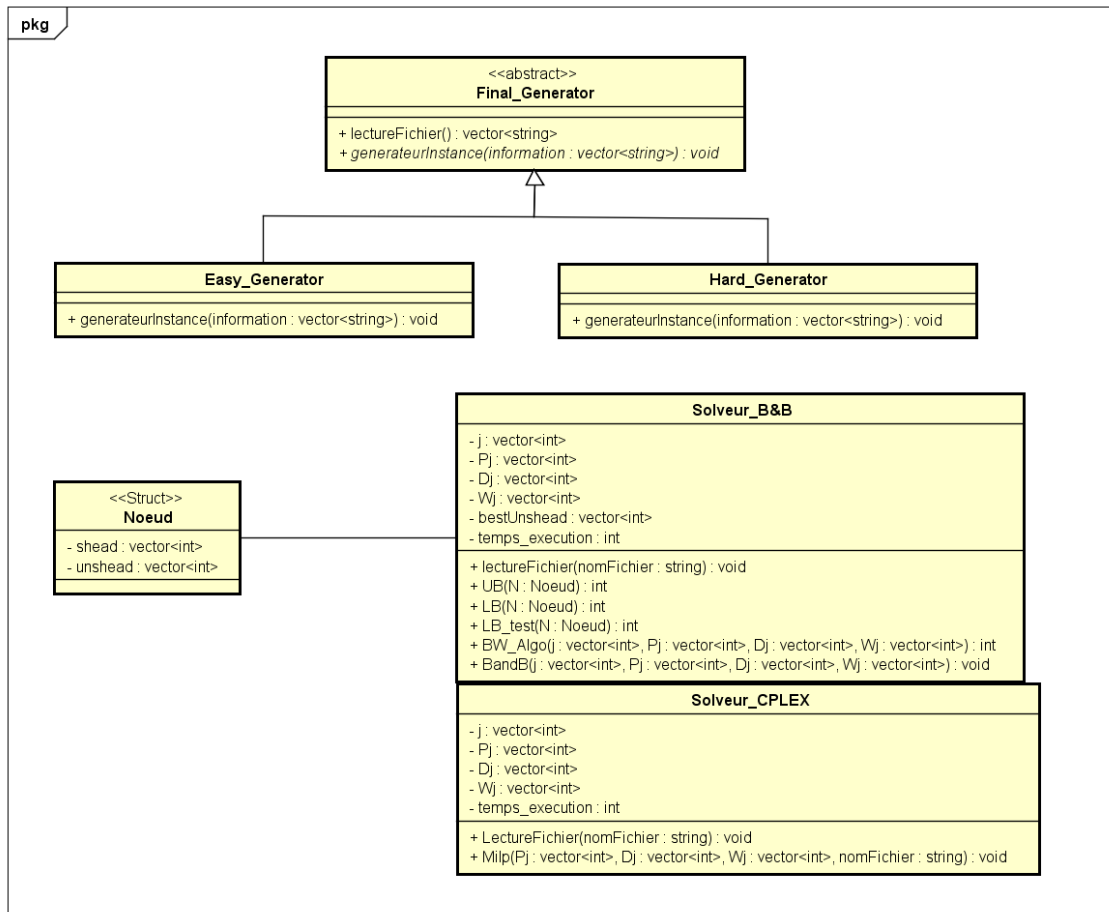


Figure 1 – Diagramme de classe

Le projet est composé de trois programmes(classes) distinct(e)s :

- La classe abstraite **FinalGenerator** dont héritent Easy et HardGenerator permet l'implémentation du générateur d'instance facile et difficile.
la méthode "LectureFichier()" est implémentée dans la classe abstraite, tandis que la méthode de génération est implémentée dans les classes filles. Ceci permet de faciliter l'ajout de générateur.
- La classe **SolveurBB** permet l'implémentation du programme de résolution par la méthode Branch and Bound.
- La classe **SolveurCplex** permet l'implémentation du programme de résolution par la méthode MILP.

Les deux programmes de résolutions sont dans des programmes séparés afin d'éviter toute interaction entre eux au niveau du code et aussi dans le but d'avoir deux exécutables distincts.

2.1 Générateur d'instances facile et difficile

Le générateur est composé d'une classe abstraite "FinalGenerator" contenant la méthode "lectureFichier()" et la méthode virtuelle "generateurInstance(information : vector<string>)".

Ainsi que de deux classes fille : "EasyGenerator" et "HardGenerator" qui vont chacune implémenter la méthode "generateurInstance".

- La méthode "lectureFichier()" va récupérer les informations en provenance d'un fichier de configuration (config.txt) et va les stocker dans un vecteur qui sera lu par les générateurs d'instances. Les informations récupérées sont : le nombre de job (**n**), la valeur de **PMAX**, la valeur de **WMAX**, la valeur d'**alpha**, la valeur de **beta** et le nombre d'instance à générer (**NBINST**).
- La méthode de génération d'instance est implémentée différemment suivant la génération d'instance facile ou difficile. Dans les deux cas, elle récupère le vecteur généré par la méthode de lecture fichier afin d'avoir toutes les informations nécessaires à la génération d'instance aléatoire.
 - Pour le générateur d'instance facile les valeurs sont générées de la manière suivante :

$$P_j \in [1, PMAX] \text{ random et } P = \sum P_j$$

$$W_j \in [1, WMAX] \text{ random}$$

$$D_j \in [(\alpha - \beta/2)P, (\alpha + \beta/2)P] \text{ random}$$
 - Pour le générateur d'instance difficile les valeurs sont générées de la manière suivante :
 - Pour 75% des jobs :

$$P_j \in [1, PMAX] \text{ random et } P = \sum P_j$$

$$W_j = P_j + P$$

$$D_j \in [(\alpha - \beta/2)P', (\alpha + \beta/2)P'] \text{ random avec } P' = 0.25*n$$
 - Pour 25% des jobs :

$$P'_j = 1$$

$$W'_j = 0$$

$$D'_j = j*(P'/n') \text{ avec } n' = 0.25*n$$

Dans les deux cas, une fois la génération des valeurs terminées, on applique la règle EDD sur la sequence (tri par D_j décroissant).

Ensuite on calcule la valeur de L_{max} .

Par la suite on effectue une renumérotation par EDD.

Et pour finir on modifie la valeur des D_j par $D_j + L_{max}$.

Une fois que tout est terminé un fichier instance est créé contenant les **n** valeurs de **J**, **P_j**, **D_j** et **W_j**.

Un exécutable est généré permettant son exécution sur n'importe quelle machine Windows.

2.2 Solveur CPLEX

Le solveur MILP, utilise Cplex pour résoudre un problème d'ordonnancement.

Il est composé d'une seule classe contenant deux méthodes :

- la méthode "lectureFichier()" va lire un fichier d'instance et va stocker les informations du fichier (**j**, **P_j**, **D_j** et **W_j**) dans des vecteurs qui seront lus par la suite dans la méthode de résolution.
- la méthode "MILP" qui va prendre en paramètre les différents vecteurs qui ont été remplis par la méthode de lecture de fichier.
Cette méthode utilisant la bibliothèque Cplex, plusieurs étapes ont lieu :

- Tout d'abord on définit plusieurs variables à savoir : l'**environnement** de type *"IloEnv"*, le **modèle** de type *"IloModel"* qui est associé à l'environnement et le **solveur** de type *"IloCplex"* qui est associé au modèle.
A travers la variable solveur, il est possible de définir des paramètres, comme le temps d'exécution qui est ici défini à 180 secondes via la méthode *setParam(cplex.TiLim,180)*.
- Ensuite on va créer un modèle qui sera à résoudre contenant : des **variables**, des **contraintes** et un **objectif**.
 - les **variables** : leurs déclarations se font grâce à des objets de type **IloArray** et **IloNumVarArray**.
Ici on indique à cplex que la variable ne peut contenir que des 0 ou 1 suivant l'emplacement d'un job par rapport à une position.
 - les **contraintes** : sous cplex elles sont représentées par des expressions (des objets de type *IloExpr*) pour lesquelles on indique un résultat à avoir, à respecter.
On y retrouve les contraintes de positionnement uniques, ainsi que les contraintes liées aux temps aux dates de fin.
 - la fonction **objective** : comme pour les contraintes il s'agit d'une expression qu'il faudra minimiser (avec *IloMinimize*).
Ici on cherche à minimiser : $\sum W_j P_j$.
- Une fois la construction du modèle terminé, il ne reste plus qu'à le résoudre avec la méthode *solve()* propre à Cplex.
- Une fois l'instance résolue, il est possible de récupérer la valeur objective obtenue via *getObjValue*.
On peut aussi savoir si la résolution est optimale ou non en récupérant le statut (avec *getStatus()*). Si la résolution se termine dans le temps imparti alors le statut est optimal sinon il ne l'est pas.
- A la fin on écrit ces informations (le statut, la valeur objective et le temps d'exécution) dans un fichier csv.

Un exécutable est généré permettant son exécution sur toute machine Windows ayant les librairies Cplex.

2.3 Solveur BB

Le solveur Branch and Bound est représenté par une classe contenant six méthodes.

Étant une méthode arborescente, elle est basée sur l'utilisation de **Noeud** qui est une structure contenant deux vecteurs (*shead* et *unshead*).

Le vecteur **shead** contient les jobs qui doivent être placés dans la séquence (**unshead**). A chaque ajout de job dans la séquence on fait un test vérifiant si la séquence est réalisable ou non. Si elle est réalisable on calcule sa borne inférieure (LB).

Le vecteur **unshead** qui est vide par défaut, va contenir à la fin la séquence correspondant au problème recherché.

- La méthode *"lectureFichier()"* va lire un fichier d'instance et va stocker les informations du fichier (j , P_j , D_j et W_j) dans des vecteurs qui seront lus par la suite dans la méthode de résolution.
- La méthode *"UB()"* prend en paramètre un **Noeud** et y retourne la borne supérieure.
Cette borne est calculée en faisant : $\sum W_j P_j$ avec P_j la position du job, sur le vecteur *Unshead* (lors de l'appel de cette méthode le vecteur *Shead* est toujours vide) du **Noeud**.

- La méthode "LB()" prend en paramètre un Noeud et y retourne la borne inférieure. Elle permet également de vérifier que la séquence est correcte (faisable).

La séquence est correcte si : $D_j \geq C_j$ pour les valeurs du vecteur Unshead. Si ça ne l'est pas, alors la borne vaut : $+\infty$.

Cette borne est calculée en faisant : $\sum W_j P_j$ sur la séquence contenant les jobs du Shead et du Unshead.

Table 1 – Représentation de la séquence

shead	unshead
-------	---------

Les jobs du vecteur shead sont triés par W_j décroissant.

- La méthode "LBtest()" prend en paramètre un Noeud et initialise la valeur de lb à 0. L'objectif de cette méthode est de forcer l'algorithme Branch and Bound à créer toute l'arborescente (aucun Noeud n'est coupé). Elle permet ainsi de vérifier le bon fonctionnement de la méthode au niveau de la création de l'arbre.

- La méthode "BWAlgo()" prend en paramètre les vecteurs contenant les informations obtenues via la méthode "lectureFichier()". Elle définit une séquence qui deviendra le Noeud initial et y calcule la borne UB.

L'objectif est simple : à partir du dernier job on recherche tous les jobs possibles ($D_j \geq C_j$) qui peuvent le précéder et on sélectionne celui avec le plus petit W_j . On crée donc une séquence en partant de la fin.

- La méthode "BandB()" prend en paramètre les vecteurs contenant les informations obtenues via la méthode "lectureFichier()". Il s'agit du programme principal.

Au début il y a une liste de Noeud (vide) qui est par la suite remplie avec un premier Noeud obtenu par l'algorithme BW (il s'agit du Noeud initial dont le shead va contenir la séquence obtenue via la méthode BW).

A partir de ce Noeud nous allons tester toutes les séquences possibles (création de l'arborescence en ajoutant des jobs du shead vers l'unshead afin d'obtenir les enfants), chaque séquence est contrôlée lors du calcul de la borne LB, si la séquence est valide et que sa borne LB est inférieure à UB alors on ajoute le Noeud dans la liste (afin de pouvoir rechercher ses enfants plus tard), sinon on coupe le Noeud (on ne cherche plus d'enfants pour ce Noeud).

Si on arrive à une feuille (Noeud avec le vecteur shead de vide) alors on calcule sa borne supérieure et si elle est inférieure à celle déjà existante (au départ celle du Noeud initial) alors on la change.

L'objectif étant d'avoir la valeur de UB optimal correspondant au problème.

Tout comme pour Cplex le temps d'exécution est limité à 180 secondes.

A la fin les données sont écrites dans un fichier csv (le même que pour la méthode CPLEX) afin de pouvoir faire des comparaisons.

Un exécutable est généré permettant son exécution sur toute machine Windows.

5

Application des tests expérimentaux et résultat

1 Conditions d'expérimentation

Les tests ont été réalisés sur un ordinateur Portable Windows 7 64 bit, avec un processeur Intel Core i7 2.3 GHz (turbo-boost 3.3 GHz) et 8 Go de RAM

2 Déroulement des tests expérimentaux

2.1 Génération des instances tests

Afin de tester correctement l'ensemble du programme, des tests sont effectués sur des instances faciles et difficiles générées par les générateurs.

Sont ainsi testées trente instances pour chaque valeur de $n = 10, 40$ et 90 pour un total de cent quatre-vingts instances.

Chaque instance a un nom de fichier qui lui est propre :

instance facile : **I-nombreJob-numeroInstance**

instance difficile : **IN-nombreJob-numeroInstance**

Les fichiers se trouvent directement dans le répertoire contenant les exécutables.

2.2 Lancement des tests

Afin de pouvoir lancer des tests automatiquement (Générateur + Solveur) j'utilise un script batch.

```

@echo off
start Final_Generator.exe

echo n;#inst;CPLEX : som(WjPj);CPLEX : opt; CPLEX : CPU(s);BandB : som(WjPj); BandB : opt; BandB : CPU(s) > result.csv

timeout /t 1

for %%f in (%~dp0\I*.txt) do call :process %%~nxf
goto :eof

:process

start /wait Solveur_CPLEX.exe %1

start /wait Solveur_BB.exe %1

```

Figure 1 – Script d'exécution

Le script effectue trois étapes :

- Il appelle le générateur d'instance pour générer les instances qui seront testées par la suite.
- Ensuite il crée le fichier csv qui va contenir les résultats.
- Puis pour chaque fichier instance du dossier il appelle les deux générateurs qui vont résoudre le problème et écrire le résultat dans le fichier csv.

2.3 Résultats des tests expérimentaux

La génération d'instance facile et difficile est rapide et ne pose aucun problème.

Pour la résolution (cas de $n = 10$) :

Pour les instances faciles et difficiles Cplex et B&B arrivent à trouver le même résultat en moins d'une seconde.

Pour la résolution (cas de $n = 40$) :

Pour les instances faciles, Cplex arrive à trouver le résultat principalement en moins d'une seconde, certains cas durant dans les deux secondes. B&B arrive à trouver les résultats en moins d'une seconde.

Pour les instances difficiles, Cplex trouve le résultat dans la quasi-totalité des cas (vingt-huit cas sur trente) avec un temps d'exécution entre quinze et quatre-vingt-quatre secondes. B&B n'arrive pas à trouver le résultat en moins de cent quatre-vingts secondes.

Pour la résolution (cas de $n = 90$) :

Pour les instances faciles, Cplex arrive à trouver le résultat pour vingt-deux instances sur trente dans des temps variant entre vingt et cent-vingts secondes. B&B ne trouve le résultat que pour dix-huit instances mais en des temps variant entre une et quarante-huit secondes.

Pour les instances difficiles, ni Cplex, ni B&B n'arrivent à trouver le bon résultat en moins de cent-quatre-vingts secondes.

Dans le cas de 90 jobs, j'ai également mesuré l'utilisation du CPU lors de l'exécution de Cplex et de B&B.

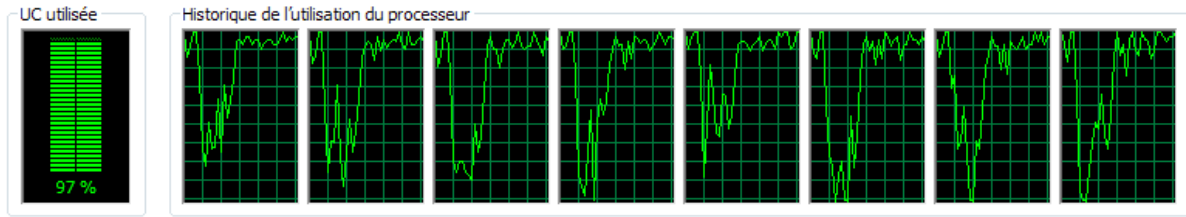


Figure 2 – Utilisation CPU avec Cplex

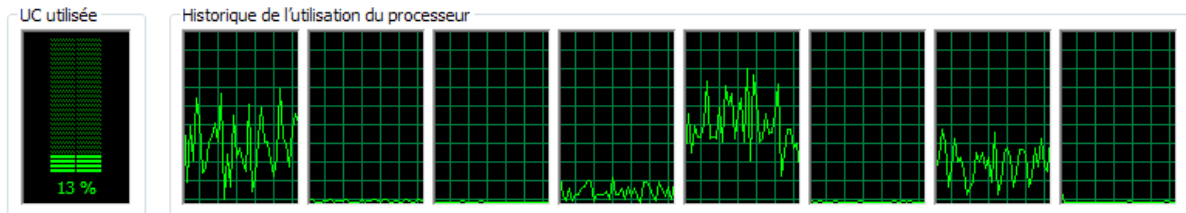


Figure 3 – Utilisation CPU avec Branch and Bound

On peut constater une grande différence sur l'utilisation du CPU entre les deux méthodes avec la méthode CPLEX qui utilise jusqu'à 100% du CPU contrairement à B&B qui utilise entre 15 à 30%.

Dans le meilleur des cas pour les instances faciles, B&B arrive à trouver plus rapidement le résultat et avec une consommation CPU moindre mais Cplex en trouve plus.

Pour les instances difficiles, Cplex arrive à trouver plus de résultats pour des valeurs de n élevées contrairement à B&B.

6

Reproductibilité

Voici les différentes informations nécessaires pour une reprise du projet.

1 Générateur d'instance

1.1 Fichier de configuration

Comme vu précédemment, le programme de génération d'instance a besoin d'un fichier de configuration en entrée portant le nom : config.txt (attention le nom a une importance).

Bien qu'il y ait deux types de générateur, le fichier de configuration reste le même du point de vue syntaxique.

```
n = 30
PMAx = 100
alpha = 1
beta = 0
WMAx = 100
NBINST = 30
```

Figure 1 – Fichier de configuration

La première ligne : la valeur n permettant d'indiquer le nombre de job.

La deuxième ligne : la valeur de $PMAx$ permettant d'indiquer la limite pour la génération aléatoire des valeurs de P_j .

La troisième ligne : la valeur α utilisée pour la génération aléatoire de D_j .

La quatrième ligne : la valeur β utilisée également pour la génération aléatoire de D_j .

La cinquième ligne : la valeur $WMAx$ permettant d'indiquer la limite pour la génération aléatoire des valeurs W_j .

La sixième et dernière ligne : la valeur $NBINST$ pour indiquer le nombre d'instances aléatoires qui seront générées pour la valeur de n .

Le format du fichier doit être impérativement respecté sinon le générateur ne pourra pas le lire. C'est à travers ce fichier que le programme du générateur peut générer les instances.

1.2 Lancement du programme

Afin de pouvoir lancer le générateur sur différentes machines Windows, un exécutable portable doit être créé.

Pour cela sur Visual Studio il suffit de compiler le projet en mode Release



Figure 2 – Mode Release

Afin d'obtenir un exécutable portable dans le dossier Release du projet.

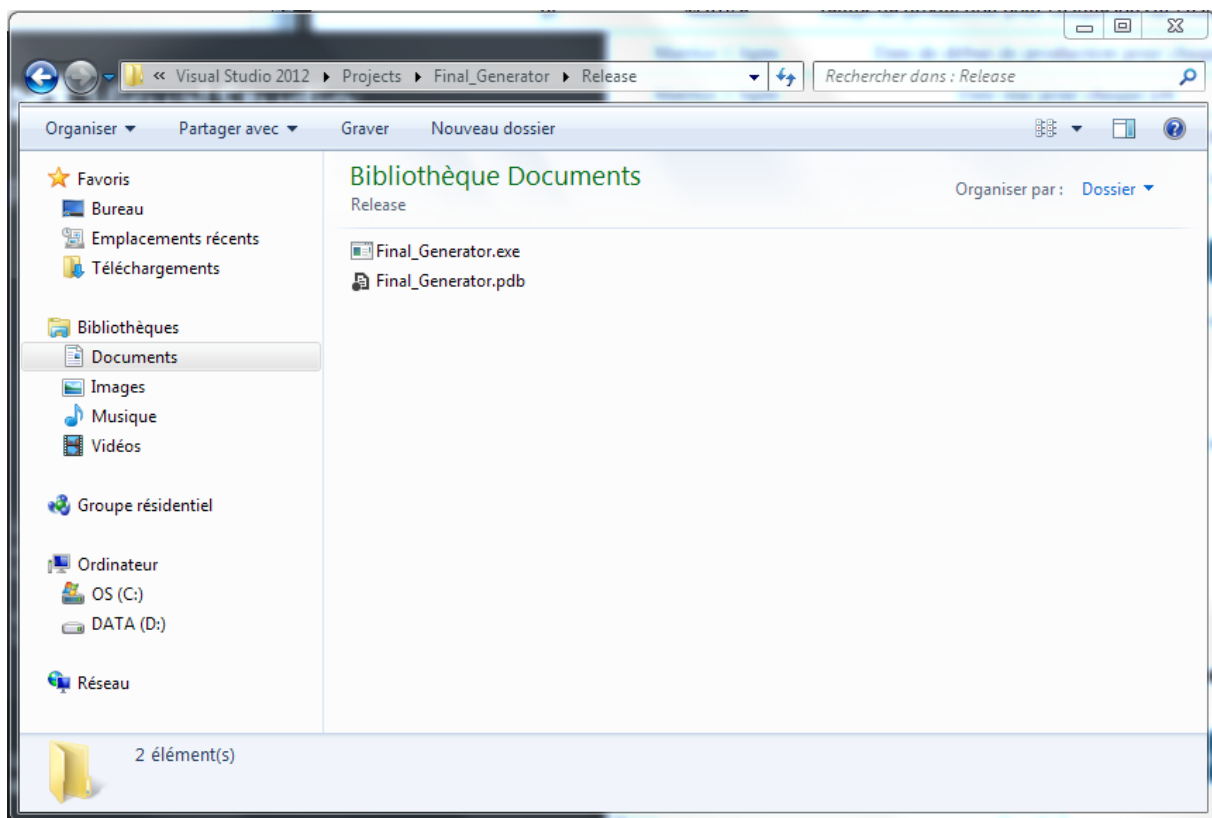


Figure 3 – Emplacement de l'exécutable

Pour l'exécuter il suffit de double-cliquer dessus. Attention à vérifier que le fichier de configuration soit présent dans le répertoire de l'exécutable.

1.3 Exemple

Voici des exemples d'instances faciles et difficiles générées par le générateur.

Instance facile :

1	90	1323	9
2	72	1371	32
3	54	1390	87
4	49	1403	28
5	11	1403	82
6	44	1408	92
7	6	1438	23
8	68	1444	1
9	83	1457	37
10	42	1506	69
11	52	1527	21
12	82	1528	8
13	93	1557	18
14	77	1567	41
15	60	1570	76
16	88	1577	31
17	73	1588	23
18	83	1592	67
19	5	1597	71
20	81	1602	23
21	88	1628	91
22	10	1640	51
23	39	1647	22
24	36	1651	99
25	100	1656	49
26	68	1663	66
27	45	1666	75
28	89	1732	78
29	41	1742	26
30	21	1750	39

Figure 4 – Exemple d'instance facile pour $n=30$

Instance difficile :

1	1	131	0
2	1	262	0
3	1	393	0
4	1	524	0
5	1	655	0
6	1	786	0
7	1	917	0
8	1	1048	0
9	99	1054	1145
10	8	1054	1054
11	21	1054	1067
12	43	1054	1089
13	47	1054	1093
14	54	1054	1100
15	59	1054	1105
16	70	1054	1116
17	71	1054	1117
18	30	1054	1076
19	92	1054	1138
20	76	1054	1122
21	15	1054	1061
22	1	1054	1047
23	86	1054	1132
24	22	1054	1068
25	50	1054	1096
26	29	1054	1075
27	48	1054	1094
28	53	1054	1099
29	59	1054	1105
30	13	1054	1059

Figure 5 – Exemple d'instance difficile pour $n=30$

2 Solveur

2.1 Fichier d'instance

Afin de pouvoir fonctionner, le solveur (Cplex et BB) a besoin d'une instance sous la forme d'un fichier txt générée par le générateur.

Ce fichier contient 4 données en colonne à savoir :

- Première colonne : les valeurs de j
- Deuxième colonne : les valeurs de P_j
- Troisième colonne : les valeurs de D_j
- Quatrième colonne : les valeurs de W_j

Le format du fichier est automatiquement géré par le générateur.

2.2 Lancement du programme

Comme pour le générateur, un exécutable est généré en compilant en mode release.

Attention, le solveur Cplex utilisant une librairie externe, l'exécutable n'est pas portable contrairement à celui du solveur B&B.

Un guide d'installation est présent en annexe de ce rapport pour l'installation des librairies Cplex sous Visual Studio.

2.3 Fichier de résultat

Les résultats sont écrits dans un fichier csv (Excel), portant le nom de "result.csv", permettant ainsi de faire des comparaisons entre Cplex et B&B.

	A	B	C	D	E	F	G	H
1	n	#inst	Cplex : som(WjPj)	Cplex : opt	Cplex : CPU(s)	BandB : som(WjPj)	BandB : opt	BandB : CPU(s)
2	40	I_40_01.txt	29653 O	0.69		29653 O	0.1	
3	40	I_40_02.txt	28762 O	2.166		28762 O	0.33	
4	40	I_40_03.txt	23665 O	2.075		23665 O	0.02	
5	40	I_40_04.txt	29169 O	0.44		29169 O	0.08	
6	40	I_40_05.txt	29853 O	0.58		29853 O	0.62	
7	40	I_40_06.txt	30484 O	0.7		30484 O	0.13	
8	40	I_40_07.txt	21114 O	0.691		21114 O	0.04	
9	40	I_40_08.txt	32451 O	0.48		32451 O	0.66	
10	40	I_40_09.txt	28727 O	0.48		28727 O	0.04	
11	40	I_40_10.txt	25809 O	0.65		25809 O	0.1	
12	40	I_40_11.txt	27751 O	1.523		27751 O	0.22	
13	40	I_40_12.txt	26347 O	1.33		26347 O	0.62	
14	40	I_40_13.txt	29132 O	0.76		29132 O	0.08	
15	40	I_40_14.txt	29406 O	0.3		29406 O	0.03	
16	40	I_40_15.txt	27011 O	0.48		27011 O	0.06	
17	40	I_40_16.txt	28057 O	0.8		28057 O	0.03	
18	40	I_40_17.txt	32715 O	0.58		32715 O	0.05	
19	40	I_40_18.txt	29791 O	0.31		29791 O	0.06	
20	40	I_40_19.txt	29017 O	0.23		29017 O	0.04	
21	40	I_40_20.txt	27340 O	2.68		27340 O	0.17	
22	40	I_40_21.txt	31085 O	0.49		31085 O	0.02	
23	40	I_40_22.txt	29598 O	1.02		29598 O	0.38	
24	40	I_40_23.txt	27987 O	0.72		27987 O	0.2	
25	40	I_40_24.txt	24982 O	0.694		24982 O	0.11	
26	40	I_40_25.txt	35218 O	0.74		35218 O	0.23	
27	40	I_40_26.txt	31652 O	0.34		31652 O	0.02	
28	40	I_40_27.txt	34857 O	1.857		34857 O	0.05	
29	40	I_40_28.txt	29310 O	0.46		29310 O	0.12	
30	40	I_40_29.txt	27947 O	0.86		27947 O	0.25	

Figure 6 – Exemple de fichier de résultat

On y retrouve : le nombre de job, le nom de l'instance, le résultat obtenu pour Cplex et pour BB, si c'est optimal ou non (terminé en moins de 180 secondes par exemple) et le temps d'exécution CPU.

2.4 Spécification

Un problème peut survenir sur l'utilisation des librairies CPLEX suivant la version utilisée (32 ou 64 bits). Dans le cadre de ce projet, travaillant sur un OS 64 bit, ce sont les librairies 64 bits qui ont été téléchargées

Lors de la compilation il faut donc choisir le mode 64 bit en Release et en Debug pour éviter les erreurs de versions.



Conclusion

Ce projet m'a permis de travailler sur le domaine de l'ordonnancement à travers le développement d'un générateur d'instance facile et difficile et d'un solveur axé sur deux méthodes.

Pendant le premier semestre de ce projet j'ai réalisé un état de l'art et une analyse sur les problèmes d'ordonnancement à une machine avec leurs différentes méthodes de résolution possible, sur la théorie de la complexité et sur la méthode hongroise. J'ai également commencé la modélisation de l'algorithme de résolution et du générateur.

Durant le second semestre, j'ai poursuivi la modélisation de l'algorithme puis j'ai effectué son implémentation. A la suite de l'implémentation, j'ai effectué différents tests pour vérifier de leur bon fonctionnement. Un guide utilisateur est également créé pour l'installation des librairies externes.

Ce fut un projet très intéressant car il m'a permis d'améliorer mes compétences dans le domaine de la recherche (recherche bibliographique, veille technologie) et voir le déroulement d'un projet lié au domaine de la recherche. J'ai également pu approfondir mes connaissances en développement C++ pour le générateur et le solveur et ainsi pu découvrir la librairie Cplex et son utilisation dans un langage de développement. Le fait de travailler seul sur un projet complet, que j'ai réalisé de A à Z (modélisation, développement, tests) est pour moi une expérience enrichissante.

Au niveau organisationnel, mon encadrant était toujours disponible ce qui m'a permis de faire le point régulièrement, de résoudre rapidement les problèmes rencontrés et de valider chaque étape du projet.

Ce projet pourrait être repris dans le but de développer de nouvelle méthode de résolution pour le même type de problème et ainsi obtenir de meilleurs comparatifs.



Bibliographie

[1] Esquirol, PATRICK. et Lopez, PIERRE. (1999). "L'ordonnancement". Paris : Économica. 141 p.
- (Collection Gestion. Série Production et techniques quantitatives appliquées).



Webographie

[WWW1] Bernard Fortz. PDF - "Modèles mathématiques et algorithmes pour l'ordonnancement". URL : <http://homepages.ulb.ac.be/~bfortz/ordo.pdf>

[WWW2] Philippe Chrétienne. PDF - "Ordonnancement sur une machine". URL : <https://www-master.ufr-info-p6.jussieu.fr/2005/IMG/pdf/MOP2-2.pdf>

[WWW3] Mohamed Ali ALOULOU. PDF - "Introduction aux problèmes d'ordonnancement". URL : <http://www.lamsade.dauphine.fr/~aloulou/cours/partie4.pdf>

[WWW4] Christophe RAPINE. PDF - "Ordonnancement sur une ressource". URL : http://idmme06.inpg.fr/~rapinec/Ordonnancement/Documents/cours_2.pdf

[WWW5] UQAC-Cours hiver 2005. PDF - "La méthode de branch and bound". URL : <http://www.uqac.ca/rebaine/8INF806/techniquedebranchandboundcourshiver2005.pdf>

Troisième partie

Annexe



Description des fonctionnalités du programme de résolution et de génération

Définition de la première fonction : LireFichier

Identification de la fonction :

Cette fonction permet de récupérer les données du fichier txt.

Description de la fonction :

Cette fonction prend en paramètre le fichier et retourne dans un vecteur les données du fichier.

Définition de la deuxième fonction (solveur) : SolveurNomMethode

Identification de la fonction :

Cette fonction généralisée dans chaque classe, va exécuter le programme de résolution.

Description de la fonction :

Elle prend en paramètre toutes les variables nécessaires à son exécution.

Elle écrit dans un fichier csv : la valeur de $\sum W_j P_j$, le temps CPU et si la solution est optimale ou non.

Définition de la troisième fonction (Générateur) : GeneratorInstance

Identification de la fonction :

Cette fonction permet de générer des instances aléatoires, facile ou difficile.

Description de la fonction :

Cette fonction va prendre en entrée un fichier txt contenant la configuration nécessaire à la génération d'instances aléatoires.

Le fichier de configuration contient le nombre de job (n), P_{\max} , α , β , W_{\max} et NBinst.



Gestion de projet

Méthode de gestion de projet

Le but de ce projet intitulé : Nouveau problème d'ordonnancement, est le développement du générateur d'instance (facile et difficile) ainsi que le développement d'un solveur avec deux méthodes de résolution (CPLEX et B&B), le tout codé en C++.

L'exécution des programmes doit se faire via un script batch et à la fin un fichier Excel doit être généré contenant le résultat de la résolution avec le temps d'exécution CPU.

Doivent ainsi être livrés : les codes sources du générateur et du solveur, ainsi que le programme contenant : les exécutables (générateur.exe, solveurCPLEX.exe, solveurBB.exe), le fichier de configuration (config.txt), le script (Execution.bat).

Une méthodologie Agile a été choisie comme méthode de gestion de projet. Tout d'abord des rendez-vous ont eu lieu régulièrement pour montrer l'avancement du projet, ensuite des livrables ont dû être donnés durant le PRD2 (Générateur d'instance facile, Générateur d'instance difficile, instances aléatoires et les résultats obtenus lors des tests).

TexPortable (un outil regroupant MikTex, TexMaker et SumatraPDF) est utilisé pour la création de ce rapport.

Plusieurs versions du rapport et du code sont sauvegardées en local sur ma machine personnelle.

Planning

Les diagrammes de gant ont été générés avec MSProject.

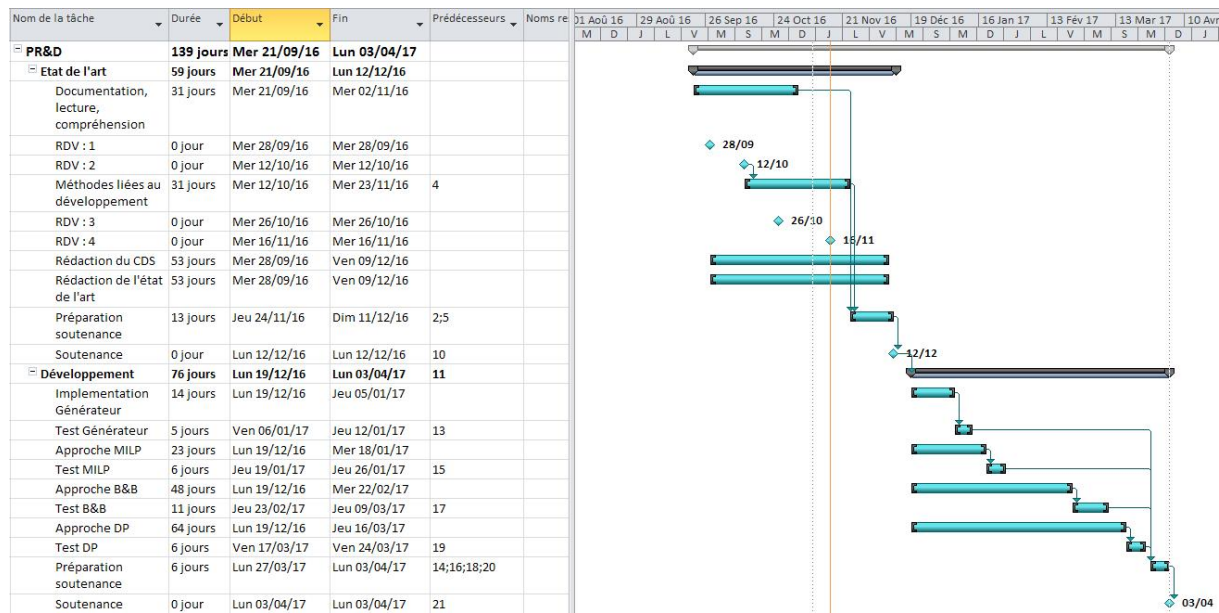


Figure 7 – Diagramme de Gant prévisionnel

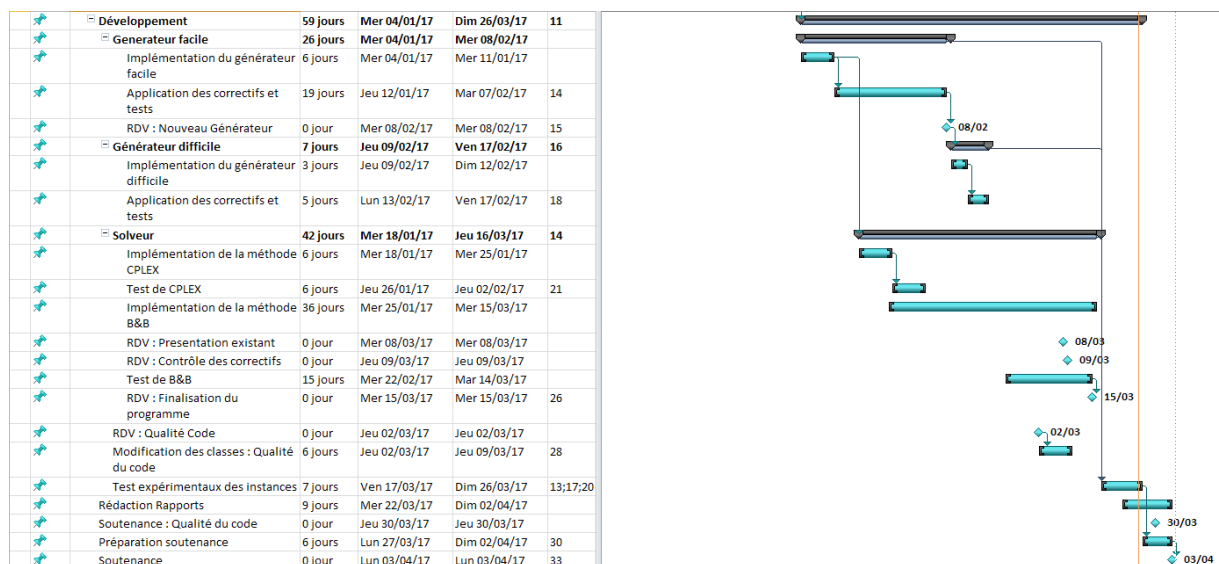


Figure 8 – Diagramme de Gant final : S10

De grandes modifications ont eu lieu dans la partie développement :

- La méthode de résolution DP (programme dynamique) a été supprimée, car elle est moins performante que les deux autres.
- Le générateur est en deux parties :
 - Un générateur d'instance « facile ».
 - Un générateur d'instance « difficile ».
- Le temps nécessaire au développement du générateur est plus faible que prévu. Toutefois des correctifs ont dû être appliqués (norme pour les fichiers instances par exemple) en plus des tests.
- Le développement de la méthode B&B a pris plus de temps que prévu, suite à des problèmes rencontrés durant la phase de développement.

— Le temps nécessaire à la partie développement est plus faible que ce qui était prévu.

Découpage du projet en tâches

Tâche 1 : Documentation, lecture et compréhension générale

Description de la tâche :

Cette tâche consiste à comprendre le projet dans son contexte, l'objectif fixé. Elle sera valable tout le long du projet via la recherche d'informations nouvelles (méthodes existantes, connaissances à acquérir pour une bonne compréhension du sujet). Elle est composée de recherches documentaires (internet et/ou article), d'informations fournies par la MOA et de réunion entre la MOA et la MOE.

Cycle de vie et contrainte temporelle :

Cette tâche doit être validée en priorité avant la phase de recherche sur les fonctionnalités et donc avant le début des phases de développement.

Estimation de charge :

Cette tâche est estimée à 3 jours/homme.

Tâche 2 : Rédaction du cahier des spécifications et de l'état de l'art

Description de la tâche :

Cette tâche consiste à la rédaction du cahier de spécification avec la définition du contexte, de l'objectif, des fonctionnalités attendues ainsi que du déroulement du projet.

Livrable :

Un rapport contenant l'état de l'art, le cahier de spécification et le plan de développement seront à rendre

Cycle de vie et contrainte temporelle :

Ces rapports sont écrits durant toute la durée du PRD 1 (jusqu'au 9 décembre).

Estimation de charge :

Cette tâche est estimée à 4 jours/homme.

Tâche 3 : Recherche sur les méthodes spécifiques à la partie développement

Description de la tâche :

Cette tâche consiste sur un état de l'art lié aux méthodes qui seront utilisées dans la partie de développement.

Cycle de vie et contrainte temporelle :

Cette tâche s'effectue après avoir compris le contexte général du projet, et doit être validée avant la phase de développement.

Estimation de charge :

Cette tâche est estimée à 2 jours/homme.

Tâche 4 : Préparation de la soutenance de PRD 1

Description de la tâche :

Préparation pour la soutenance du PRD 1 : État de l'art et déroulement du projet.

Livrable :

Le diaporama de la soutenance.

Cycle de vie et contrainte temporelle :

La soutenance a lieu le 12 décembre.

Estimation de charge :

Cette tâche est estimée à 0.5 jour/homme.

Tâche 5 : Implémentation du générateur "facile"

Description de la tâche :

Après validation par la MOA, cette tâche consiste à implémenter le programme permettant la génération de fichiers txt correspondant à des instances faciles qui seront par la suite utilisées par le programme de résolution.

Livrable :

Code source et exécutable du programme.

Cycle de vie et contrainte temporelle :

Ce programme est à implémenter en premier puisqu'il est nécessaire au fonctionnement du programme de résolution.

Estimation de charge :

Cette tâche est estimée à 1 jour/homme.

Tâche 6 : Test du générateur facile

Description de la tâche :

Vérification du générateur via des tests assurant la cohésion des données en sortie.

Estimation de charge :

Cette tâche est estimée à 1.5 jour/homme.

Tâche 7 : Implémentation du générateur "difficile"

Description de la tâche :

Après validation par la MOA, cette tâche consiste à implémenter le programme permettant la génération de fichiers txt correspondant à des instances difficiles qui seront par la suite utilisées par le programme de résolution.

Livrable :

Code source et exécutable du programme.

Cycle de vie et contrainte temporelle :

Ce programme est à implémenter après l'implémentation du générateur facile.

Estimation de charge :

Cette tâche est estimée à 0.5 jour/homme.

Tâche 8 : Test du générateur difficile

Description de la tâche :

Vérification du générateur via des tests assurant la cohésion des données en sortie.

Estimation de charge :

Cette tâche est estimée à 1 jour/homme.

Tâche 9 : Implémentation de la méthode Branch and Bound

Description de la tâche :

Après validation de la méthode par la MOA (avec toutes les données nécessaires), cette tâche consistera à implémenter cette méthode.

Livrable :

Code source.

Cycle de vie et contrainte temporelle :

Il s'agit d'une méthode prioritaire.

Estimation de charge :

Cette tâche est estimée à 5 jours/homme.

Tâche 10 : Tests et correction de la méthode Branch and Bound

Description de la tâche :

Vérification du bon fonctionnement de cet algorithme via des tests.

Estimation de charge :

Cette tâche est estimée à 3.5 jours/homme.

Tâche 11 : Implémentation de la méthode MILP

Description de la tâche :

Après validation de la méthode par la MOA (avec toutes les données nécessaires), cette tâche consistera à implémenter la méthode MILP.

Livrable :

Code source.

Cycle de vie et contrainte temporelle :

Il s'agit d'une méthode prioritaire.

Estimation de charge :

Cette tâche est estimée à 2 jours/homme.

Tâche 12 : Test et correction de la méthode MILP

Description de la tâche :

Vérification du bon fonctionnement de l'algorithme MILP via des tests.

Estimation de charge :

Cette tâche est estimée à 1.5 jours/homme par semaine.

Tâche 13 : Rédaction du rapport final

Description de la tâche :

Rédaction du rapport final contenant l'état de l'art du PRD1 avec le cahier de spécification ainsi que l'explication de la partie développement, les guides utilisateurs, guide d'installation.

Livrable :

Le rapport en lui-même.

Cycle de vie et contrainte temporelle :

Le rapport de développement et les guides d'utilisations seront écrits durant le PRD 2.

Estimation de charge :

Cette tâche est estimée à 4 jours/homme.

Tâche 14 : Préparation de la soutenance PRD 2

Description de la tâche :

Préparation de la soutenance finale du PRD qui sera axée sur la partie développement avec diaporama et exécution de l'application à l'appui.

Livrable :

Code source, exécutable du projet et le diaporama de la soutenance.

Cycle de vie et contrainte temporelle :

La soutenance a lieu début avril.

Estimation de charge :

Cette tâche est estimée à 1 jours/homme.

Analyse de faisabilité

Il n'y a pas de contrainte matériel. Le temps alloué à la partie développement suffit pour produire le livrable final (même avec les modifications/ajouts du client : ici ajout du générateur difficile).

La troisième méthode (programme dynamique) a été supprimée (méthode non pertinente).

Analyse de risque

Deux risques sont identifiés dans le cadre de ce projet :

- Le premier vient de l'interdépendance entre la tâche du développement du générateur et les tâches de développement du solveur.
 - En effet afin de pouvoir tester et exécuter les méthodes de résolution il faut obligatoirement avoir des instances générées aléatoirement afin de pouvoir analyser les résultats.
 - Ces instances sont générées par le générateur, par conséquent si le générateur n'est pas opérationnel, alors il ne sera pas possible de tester le solveur.
 - De plus le générateur est un programme important qui est demandé en début de phase de développement.
- Le deuxième est lié au temps de développement des méthodes de résolution.
 - En effet deux méthodes sont à implémenter afin de pouvoir faire des comparaisons de résultat pour une instance.
 - Si le développement d'une méthode dure trop longtemps, il ne sera donc pas possible de développer et de tester la deuxième méthode (et donc de faire des comparaisons).

Afin d'éviter ces risques, il est important de voir régulièrement le client afin de valider ce qui a été fait, vérifier que l'on a bien compris ce qui était demandé pour éviter toute perte de temps inutile.

Suivi de projet

Afin de valider les différentes étapes du projet et d'en suivre l'avancement, des rendez-vous ont eu lieu régulièrement entre Mr Billaut et moi-même.

A travers ces rendez-vous il a été possible de valider les résultats obtenus par mes solveurs en les comparant avec les résultats obtenus par Mr Billaut. Des démonstrations ont également eu lieu afin de montrer l'exécution du programme et ainsi valider ou non le code.

Ceci m'a également permis de résoudre mon problème lié à l'implémentation de la méthode de résolution B&B.

Cahier de tests

Dans le cadre de ce PR&D, plusieurs tests ont été effectués pour s'assurer du bon fonctionnement du programme.

Tests de portabilité :

Des tests de portabilités ont été effectués sur les différents exécutables du programme, plus précisément sur une machine ne possédant ni d'IDE C++ ni de librairies CPLEX.

Test de l'exécutable du générateur :

- Condition : Exécution du générateur (facile et difficile) sur une machine ne possédant pas l'IDE Visual Studio.
- Résultat : Succès

Test de l'exécutable du solveur B&B :

- Condition : Exécution du solveur sur une machine ne possédant pas d'IDE C++
- Résultat : Succès

Test de l'exécutable du solveur CPLEX :

- Condition : Exécution du solveur sur une machine ne possédant pas les libraires CPLEX (n'ayant pas IBM CPLEX sur la machine)
- Résultat : Échec
 - Raison : les libraires CPLEX étant externes, malgré la compilation en release du programme, il n'est pas possible d'exécuter le programme sur une machine ne possédant pas les librairies.

Tests d'exécution :

Ces tests ont pour objectif de vérifier le bon fonctionnement du programme (exécutable et script) suivant divers cas de figure.

Test d'exécution du script pour le générateur :

- Condition : Lancement du script pour l'exécution du générateur (facile et difficile)
 - Cas 1 : le fichier de configuration est présent mais la syntaxe n'est pas correcte, le programme retourne un message d'erreur terminant l'exécution du programme.
 - Résultat : Succès

- Cas 2 : le fichier de configuration n'est pas présent, le programme retourne un message terminant l'exécution du programme.
 - Résultat : Succès
- Cas 3 : le fichier de configuration est présent, le programme s'exécute correctement.
 - Résultat : Succès

Test d'exécution du script pour le solveur (B&B et Cplex) :

- Condition : Lancement du script pour l'exécution du solveur.
- Cas 1 : Modification de l'emplacement du programme
 - Résultat : Échec
 - Raison : Utilisation d'un chemin absolu et non relatif.
 - Correctif : Mettre un chemin relatif dans le fichier.
 - Note : Éviter les noms de dossier avec un espace (mettre « Undossier » au lieu de « Un dossier »).
- Cas 2 : les fichiers d'instances ne sont pas présents, le programme retourne un message d'erreur terminant l'exécution du programme.
 - Résultat : Succès
- Cas 3 : les fichiers d'instances sont présents, le programme s'exécute correctement.
 - Résultat : Succès

Tests de contrôle :

Ces tests ont pour objectif de contrôler le bon déroulement d'une méthode.

Tests de contrôle des fichiers d'entrées :

Les fichiers d'entrées sont analysés.

- Condition : Exécution des méthodes de lecture de fichier.
 - Cas 1 : Syntaxe du fichier incorrect, le programme retourne un message d'erreur et se termine.
 - Résultat : Succès
 - Cas 2 : Nom du fichier incorrect, le programme retourne un message d'erreur et se termine.
 - Résultat : Succès

Tests de contrôle sur les valeurs obtenues (générateur) :

S'agissant d'un projet d'ordonnancement, les calculs effectués durant la génération des instances et durant leurs résolutions sont analysés.

Les valeurs sont affichées durant l'exécution du programme.

Le générateur d'instance difficile ayant une particularité qui lui est propre, la valeur de Lmax est contrôlée.

- Condition : Exécution de la méthode de génération.
 - Cas 1 : Les Valeurs du fichier de configuration sont correctes.
 - Résultat : Succès
 - Cas 2 : Valeur du fichier de configuration incorrecte, la valeur de Lmax ne vaut pas 0, le programme retourne un message d'erreur et se termine.
 - Résultat : Succès

Tests de contrôle sur les valeurs obtenues (solveur) :

Les valeurs obtenues durant la résolution de l'instance (valeurs des bornes, valeur finale) sont affichées durant l'exécution du programme.

Le résultat obtenu par CPLEX est comparé avec le résultat obtenu par Mr Billaut, permettant ainsi de le valider ou non. Ainsi ce résultat peut être comparé par celui obtenu par le solveur B&B.

Guide d'installation

Ce guide va expliquer comment installer les bibliothèques CPLEX sur l'IDE Visual Studio pour le fonctionnement de la méthode de résolution CPLEX.

1ère étape : Télécharger et installer Cplex Studio

Il est possible de télécharger la version « Community Edition » dans le cadre du projet sous l'adresse suivante : <https://www-01.ibm.com/software/websphere/products/optimization/cplex-studio-community-edition/>

Try and learn optimization and constraint programming modeling for planning and scheduling with IBM ILOG CPLEX Optimization Studio. Choose the method that best suits your needs.	
Community Edition Problem size limited to 1000 variables and 1000 constraints. All features included. Available on the most popular supported platforms.	Download
Academic Initiative A global program that facilitates the collaboration between IBM and educators to teach students the information technology skills they need to be competitive and keep pace with changes in the workplace. Faculty members, research professionals at accredited institutions, and qualifying members of standards organizations can join and can get full versions of a large selection of IBM software, including IBM ILOG CPLEX Optimization Studio at no charge.	Learn more
Extended trial The extended trial does not have problem size limitations. Your request will be reviewed for appropriateness, as determined by IBM, and you may be contacted by an IBM sales representative. If your request is approved, you will be sent an e-mail containing an evaluation license for your review and acceptance, and once accepted and returned, you will be sent the necessary instructions for downloading the software. Available for all supported platforms.	Request trial
Students and faculty of colleges and universities are not eligible for this program. Use the Community Edition or Academic Initiative.	

Figure 9 – Téléchargement de Cplex

Il est également possible de télécharger la version « academic » grâce à l'adresse mail étudiante : <https://ibm.onthehub.com/WebStore/OfferingDetails.aspx?o=9b4eadea-9776-e611-9421-b8ca3a5db7a1>

Pour cela il suffit juste de créer un compte (avec l'adresse email @etu.univ-tours.fr) et de

télécharger gratuitement IBM ILOG CPLEX Optimization Studio 12.7 – Student.

Attention au choix de la version (32 ou 64 bit), en effet si vous choisissez la version 64 bit vous devrez compiler votre projet sous une plateforme x64 (par défaut : 32 bit).

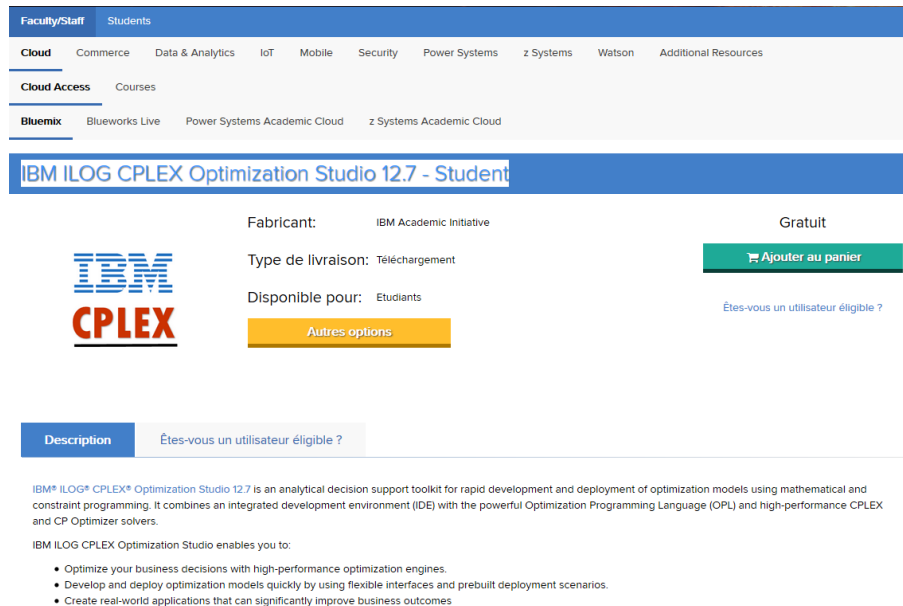


Figure 10 – Choix de Cplex

Une fois téléchargé, il vous suffit de lancer l'exécutable pour procéder à l'installation.

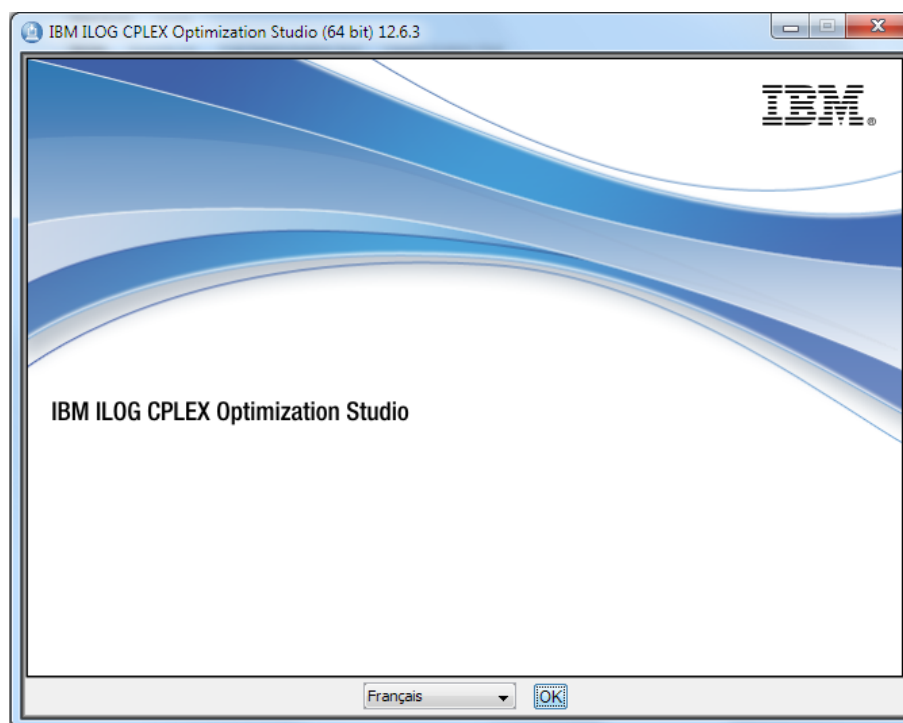


Figure 11 – Début de l'installation

Choisissez Français et cliquer sur OK

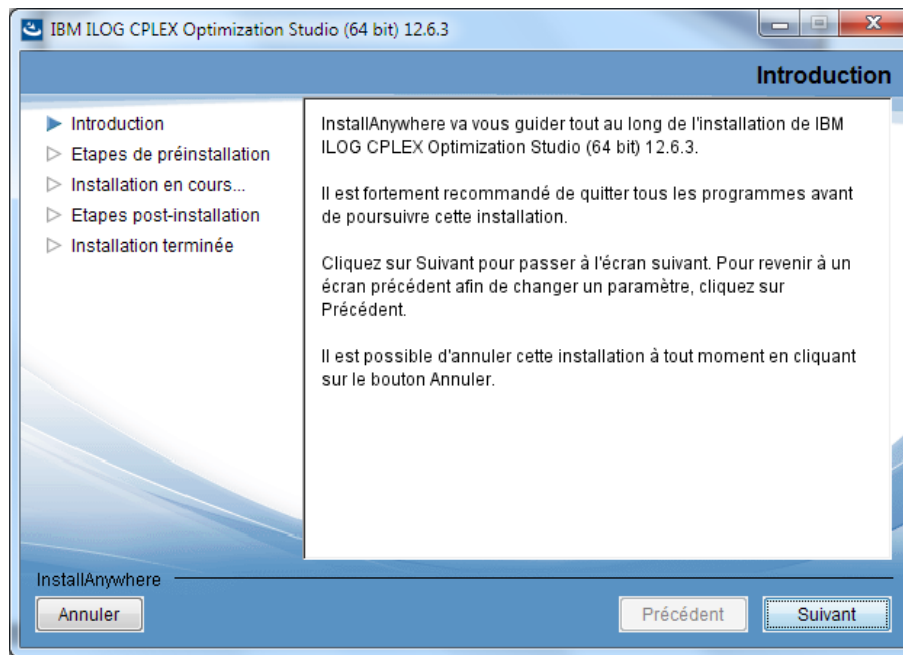


Figure 12 – Introduction

Cliquer sur Suivant

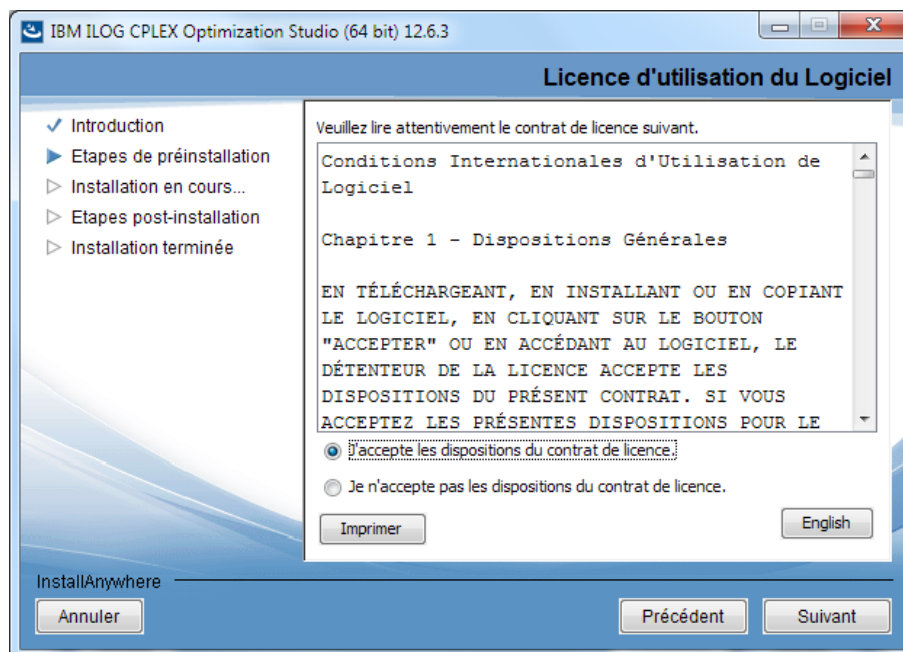


Figure 13 – Contrat de licence

Accepter le contrat de Licence et cliquer sur "Suivant" pour lancer l'installation.

2eme étape : Installation des libraires dans l'IDE Visual Studio

L'installation diffère légèrement suivant le mode d'exécution (Release ou Debug).

Vous devez paramétrer le projet pour prendre en compte les librairies externes propres à CPLEX. Pour cela dans votre projet cliquer sur "Projet", Propriétés de «nomDuProjet»

Mode Debug

Tout d'abord aller dans Propriétés de configuration, C/C++, Général

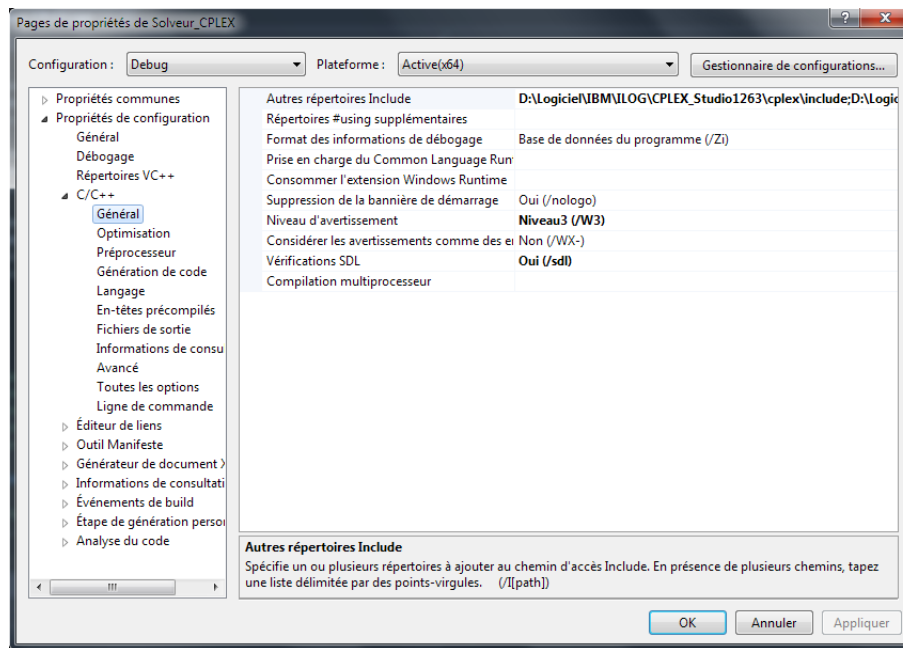


Figure 14 – Include des répertoires

Dans « Autres répertoires Include » vous devez inclure les répertoires cplex include et concert include.

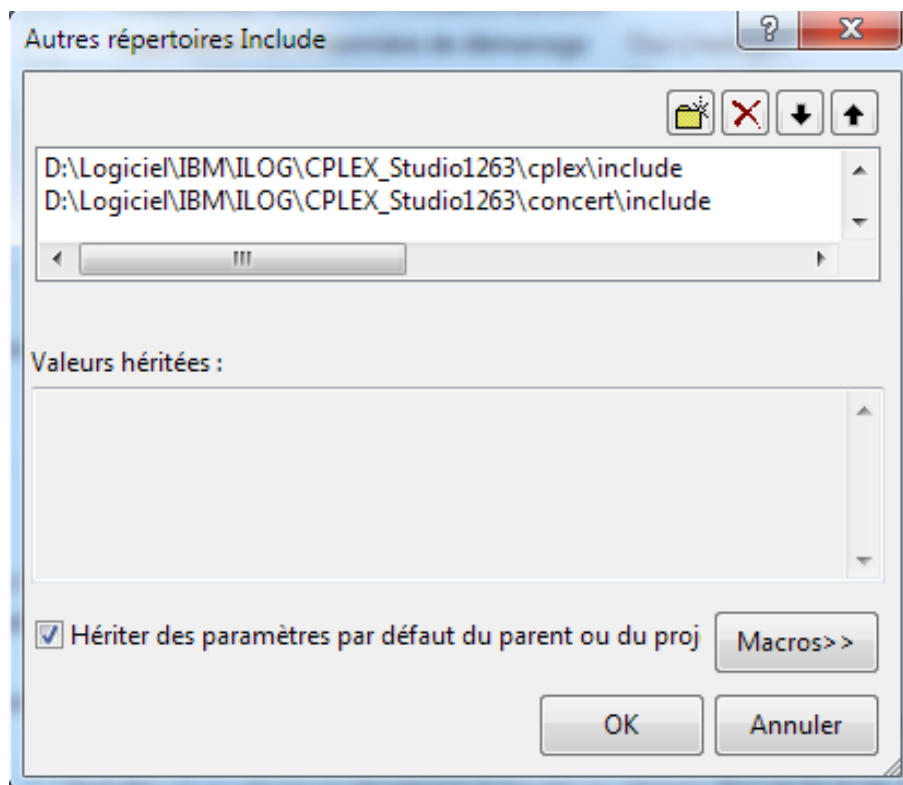


Figure 15 – Représentation : Include des répertoires

Ensuite aller dans Propriétés de configuration, C/C++, Préprocesseur

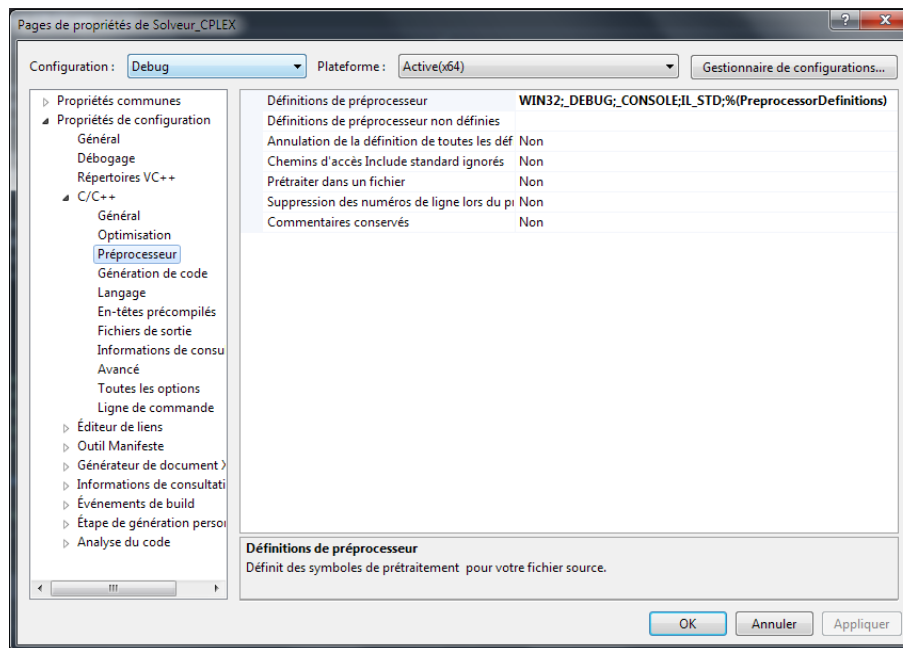


Figure 16 – Définition de préprocesseur

Pour ajouter « IL_STD » dans les « définitions de préprocesseur »

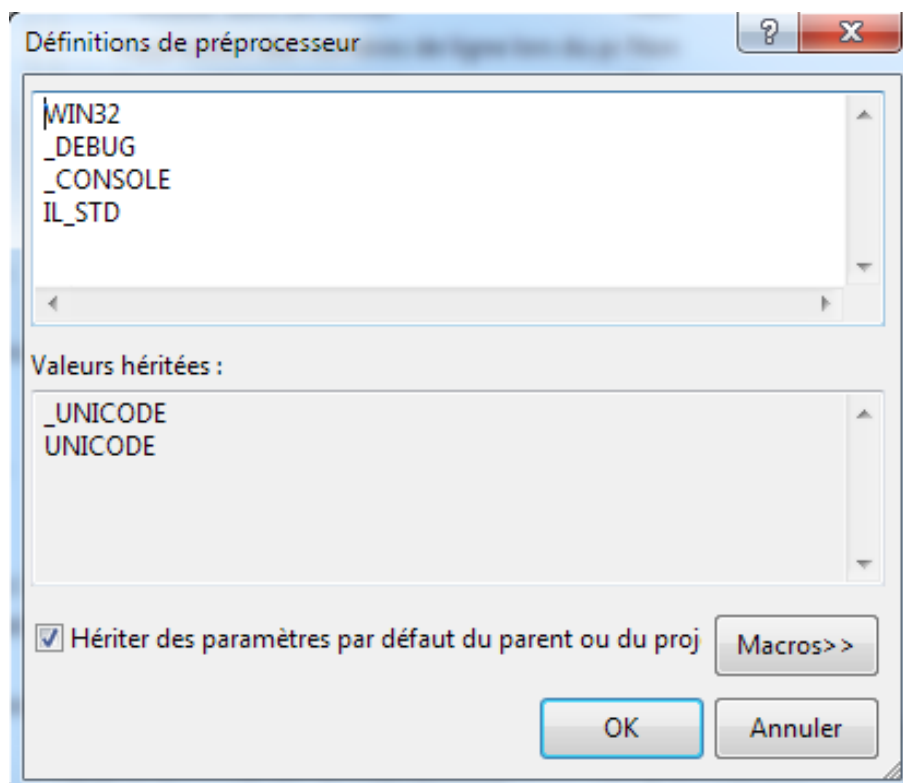


Figure 17 – Représentation : Définition des préprocesseur

Ensuite aller dans Propriétés de configuration, Éditeurs de liens, Général
Afin d'indiquer les répertoires de bibliothèques supplémentaires.

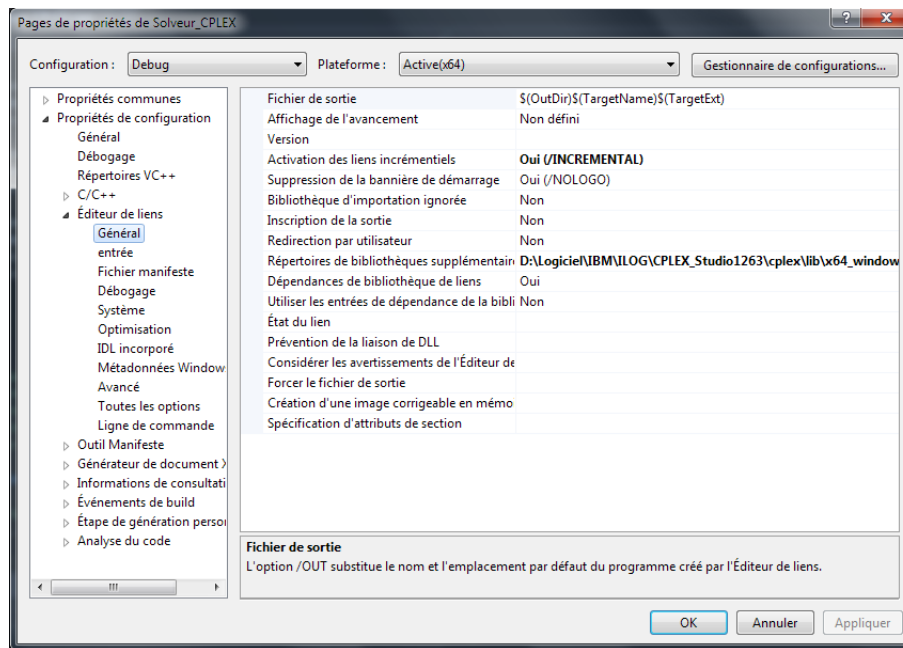


Figure 18 – Bibliothèques Cplex

IBM propose des librairies suivant la version de Visual studio, étant ici en version 64 bit professionnel 2012, nous prenons les librairies du dossier x64_windows_vs2012

Attention à bien respecter l'ordre des répertoires

En haut les stat_mdd et en bas les stat_mda.

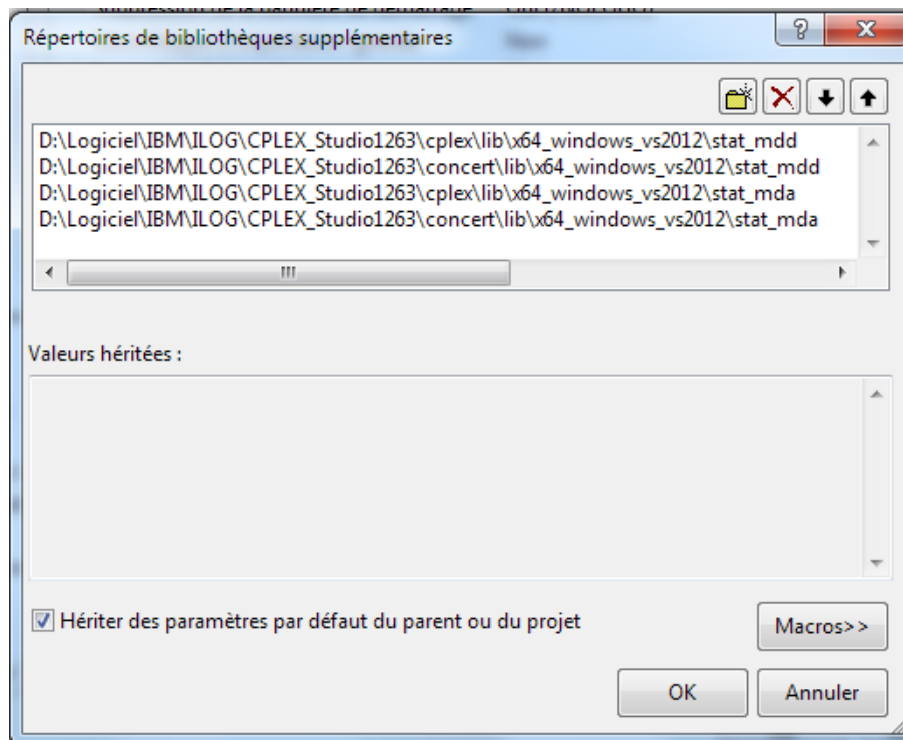


Figure 19 – Représentation : Bibliothèques Cplex

Pour finir aller dans Propriétés de configuration, Éditeur de liens, entrée, afin d'inclure les dépendances supplémentaires à savoir cplex1263.lib, concert.lib et ilocplex.lib.

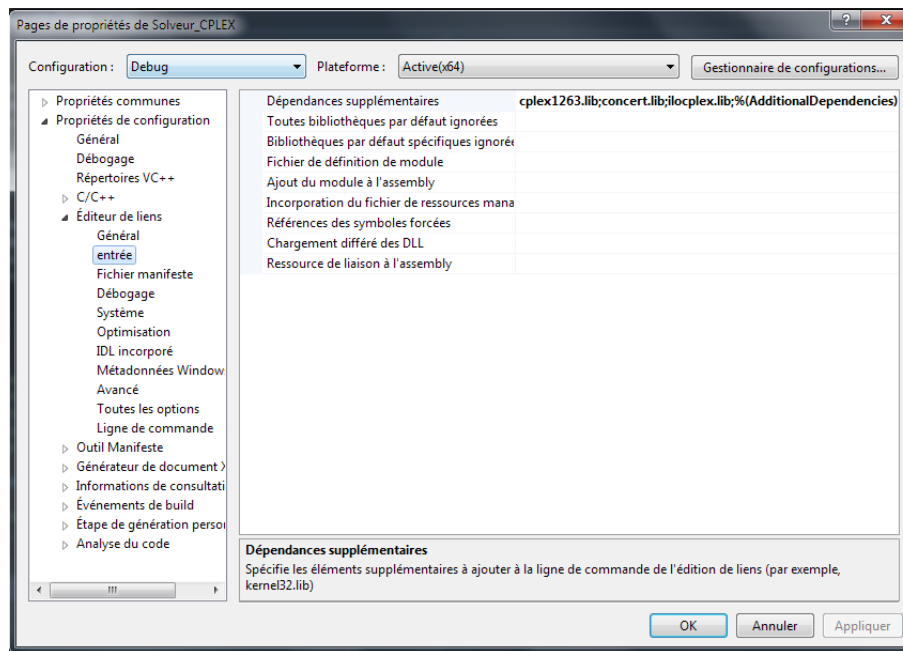


Figure 20 – Dépendances Cplex

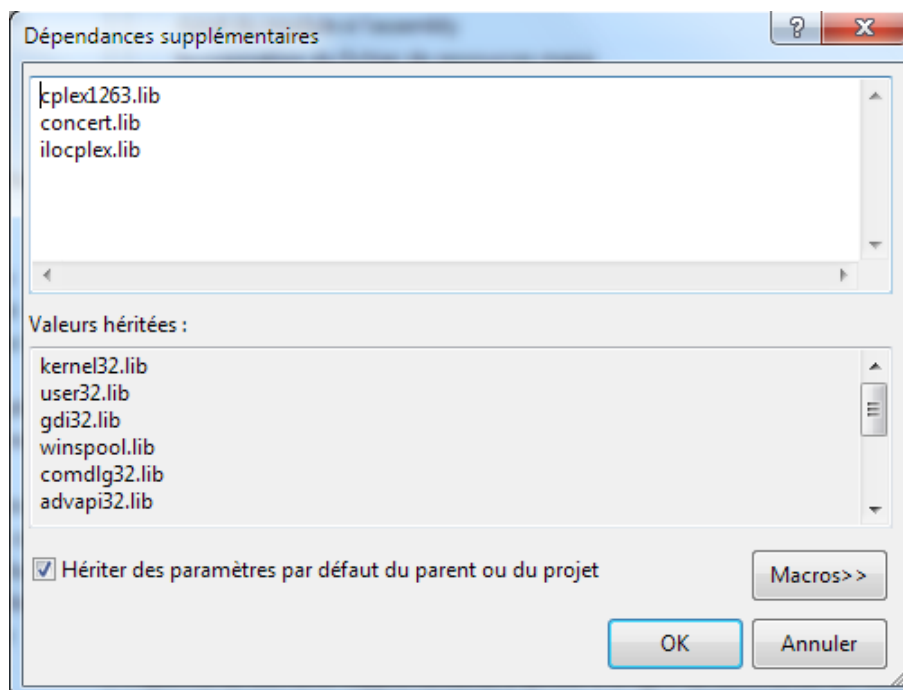


Figure 21 – Représentation : Dépendances Cplex

Le projet est maintenant configuré pour le mode Debug.

Mode Release

La configuration du mode Release est identique à celle du mode Debug à l'exception des répertoires de bibliothèques supplémentaires.

En effet l'ordre est inversé avec en premier les répertoires stat_mda

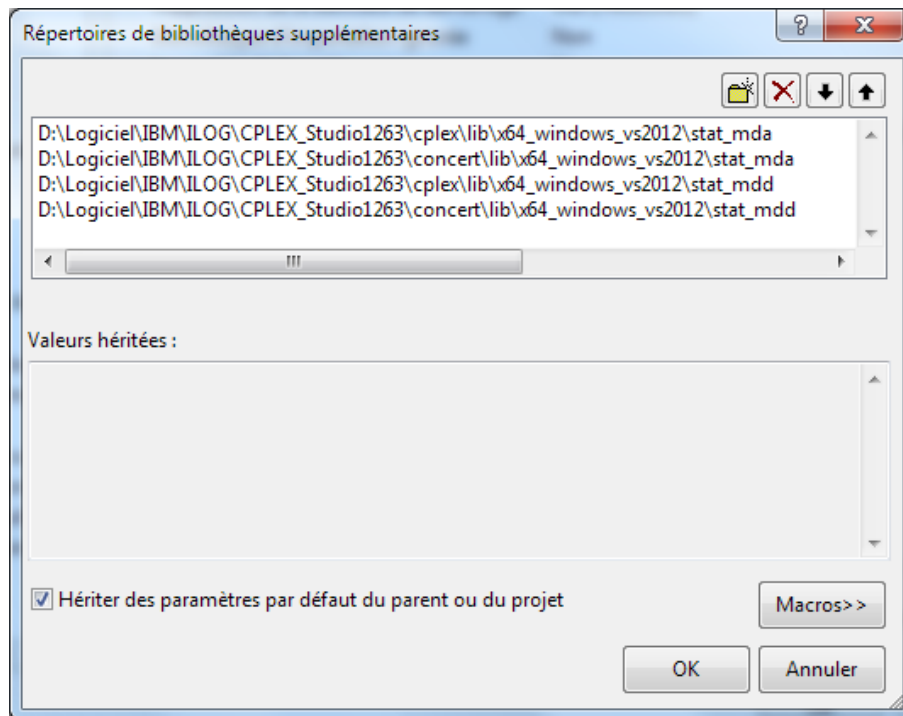


Figure 22 – Ordre bibliothèques Cplex

Maintenant le projet est configuré en mode Debug et Release.

Il est maintenant possible de compiler un projet utilisant CPLEX.

Guide d'utilisation

L'utilisation du programme est très simple, puisqu'il suffit juste d'exécuter le script Execution.bat se trouvant dans le répertoire afin de lancer les solveurs.

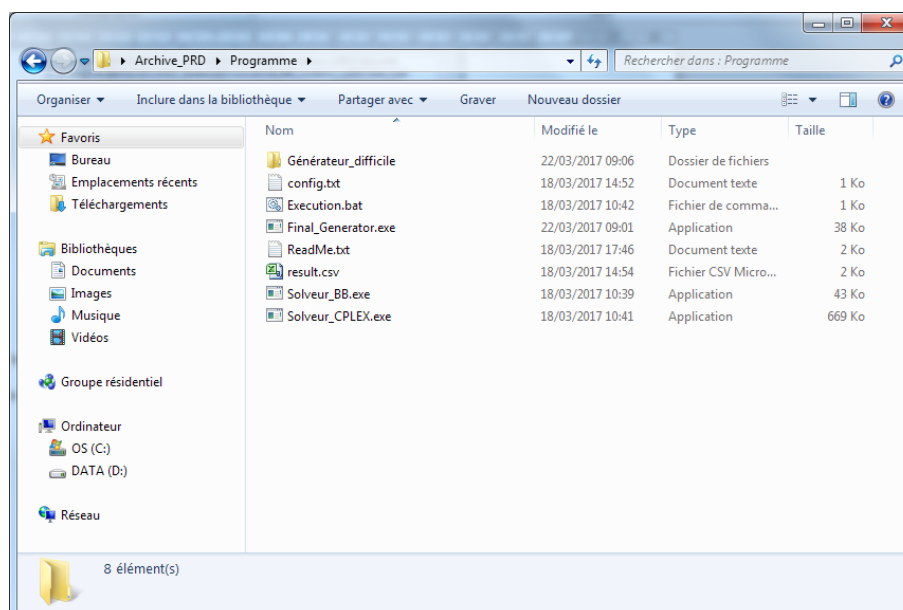


Figure 23 – Répertoire du programme

Le fonctionnement du programme est le suivant :

Tout d'abord il faut créer des instances via le générateur en exécutant « Final_Generator.exe » qui va lire le fichier « config.txt » pour pouvoir générer des instances sous format txt (celui dans le dossier "Générateur_difficile" permet la génération d'instances difficiles).

Ensuite, en exécutant le script le programme va pour chaque fichier txt du dossier (commençant par I ou IN), lancer les exécutables « Solveur_Cplex » puis « Solveur_BB » qui vont chacun résoudre l'instance et écrire dans le fichier « result.csv » le résultat obtenu (avec le nom de l'instance, la valeur n, le temps d'exécution CPU et si le programme a résolu l'instance dans un laps de temps déterminé).

La génération d'instance et leurs résolutions se font séparément.



Comptes rendus hebdomadaires

Compte rendu n°1 du 25/09/2016

Phase de recherche sur : l'ordonnancement (éléments fondamentaux, problèmes liés à une machine, éléments de complexité), la règle EDD, la théorie de la complexité et le treillis des permutations.

Prise de rendez-vous avec Mr Billaut pour le 28/09

Compte rendu n°2 du 2/10/2016

Poursuite des recherches sur les problèmes existants d'ordonnancement avec leurs solutions, les problèmes de type NP-difficile et la méthode Hongroise.

Début de rédaction des rapports.

Prise de rendez-vous avec Mr Billaut pour le 12/10

Compte rendu n°3 du 9/10/2016

Poursuite de la rédaction des rapports (Etat de l'art et CDS).

Poursuite des recherches sur les problèmes d'ordonnancement.

Compte rendu n°4 du 16/10/2016

Recherche sur la méthode arborescente Branch and Bound et préparation du matériel pour la phase de développement (CPLEX)

Prise de rendez-vous avec Mr Billaut pour le 26/10

Compte rendu n°5 du 23/10/2016

Finalisation de la première version du cahier de spécification.

Début de la rédaction de la deuxième partie.

Poursuite de l'état de l'art.

Analyse des différentes méthodes de développement possible (MILP et DP).

Compte rendu n°6 du 6/11/2016

Cahier de spécification et plan de développement en version 1.0

État de l'art en version 0.7

Prise de rendez-vous avec Mr Billaut pour le 16/11

Compte rendu n°7 du 13/11/2016

Poursuite des rapports et des recherches sur la partie développement.

Compte rendu n°8 du 20/11/2016

Modification de l'existant sur les rapports.

Compte rendu n°9 du 27/11/2016

Finalisation des rapports.

Compte rendu n°10 du 04/12/2016

Fin du rapport de PRD1 (État de l'art, cahier de spécification et plan de développement).

Compte rendu n°11 du 11/12/2016

Préparation pour la soutenance de PRD 1.

Compte rendu n°12 du 08/07/2017

Développement du générateur en version 1.0 avec deux méthodes : "lectureFichier()" et "générateurInstance()".

Les données sont affichées en lignes.

Configuration de l'IDE Visual Studio avec l'ajout des librairies CPLEX.

Début de développement du solveur.

Compte rendu n°13 du 15/01/2017

Poursuite du développement du solveur Cplex et B&B

Recherche sur Concert Technology pour le C++

Compte rendu n°14 du 22/01/2017

Fin du développement de la méthode MILP

Des problèmes ont été détectés durant l'exécution à cause de conflit entre concert Technology et le système de compilation.

Les paramètres du projet ont été modifiés permettant son fonctionnement en x64 et en mode debug.

Compte rendu n°15 du 29/01/2017

Mises à jours mineures sur l'existant.

Début d'implémentation de B&B.

Compte rendu n°16 du 05/02/2017

Le générateur fonctionne en Debug et en Release.

Le solveur Cplex fonctionne en Debug et Release (exécutable fonctionnel).
Début d'implémentation du script batch.

Compte rendu n°17 du 12/02/2017

Implémentation du générateur difficile.
Modification du code pour cplex, optimisation et résolution d'erreur.

Compte rendu n°18 du 19/02/2017

Modification des commentaires du code.
Test de portabilité sur les exécutables du générateur.
Ajout de contrôle sur les fichiers d'entrée.

Compte rendu n°19 du 26/02/2017

Poursuite du développement de la méthode B&B.
Pris de de RDV pour la Qualité du code.

Compte rendu n°20 du 05/03/2017

Exécution de tests, poursuite de développement.
Des difficultés ont été rencontrés durant la phase de développement.
Prise de RDV pour la présentation de l'existant et contrôle des correctifs.

Compte rendu n°21 du 12/03/2017

Exécution des tests sur la méthode B&B.
Modification de l'existant.
Prise de RDV pour la finalisation du programme.

Compte rendu n°22 du 19/03/2017

Obtention des résultats pour les instances générées au début du PRD2 et tests de contrôle et de portabilité.

Compte rendu n°23 du 26/03/2017

Tests expérimentaux.

Compte rendu n°24 du 02/04/2017

Rédaction des rapports et préparation pour la soutenance.

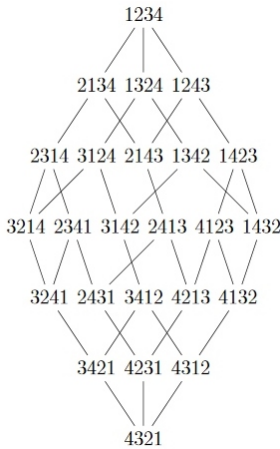
Nouveaux problèmes d'ordonnancement

Kivin Ringard

Encadrement : Jean-Charles Billaut et Thanh Thuy Tien TA

Constat

Les problèmes d'ordonnements sont courants lors de l'organisation de tâche dans le cas d'un projet. L'objectif de ce PRD est de permettre d'obtenir un ensemble de solutions possibles via la recherche dans un treillis pour le problème : $1 | \tilde{d}_j | \sum w_j, p_j$.



Treillis de permutation

Solution

Il y a plusieurs méthodes qui sont implémentées dans le solveur pour la résolution du problème :

- Méthode Branch and Bound
- Méthode MILP : Mixed Integer Linear Programming
- Méthode DP : Dynamic Programming

Problème	Nature du problème
$1 d^{\sim}_j \sum w_j P_j$	problème NP-difficile
$1 d^{\sim}_j, p_j = 1 \sum w_j P_j$	problème non déterminé
$1 d^{\sim}_j \sum w_j H_j$	distance de Hamming (avec H_j non défini), problème non déterminé
$1 d^{\sim}_j \sum j P_j$	problème non déterminé

Problèmes existants

Conclusion

A travers un générateur d'instance et d'un solveur, il est possible d'obtenir le "level" dans l'arbre avec le temps CPU pour obtenir un ensemble de solutions.

Taches j	1	2	3	4	5
Durée p_j	2	3	6	5	1
Poids w_j	1	3	2	1	2
Date échue d_j	8	11	7	15	1

Ordonnancement pour 1 machine

Nouveaux problèmes d’ordonnancement

Kivin Ringard

Encadrement : Jean-Charles Billaut et Thanh Thuy Tien TA

Constat

Les problèmes d’ordonnancements sont courants lors de l’organisation de tâche dans le cas d’un projet. L’objectif de ce PRD est de permettre d’obtenir un ensemble de solutions possibles via la recherche dans un treillis pour le problème : $1|\tilde{d}_i|\sum w_i.p_i$.

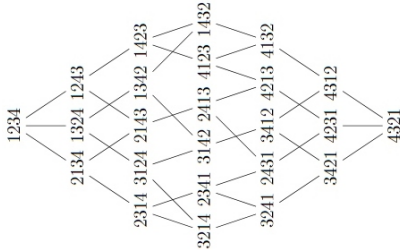
Solution

Il y a plusieurs méthodes qui sont implémentées dans le solveur pour la résolution du problème :

- Méthode Branch and Bound
- Méthode MILP : Mixed Integer Linear Programming
- Méthode DP : Dynamic Programming

Conclusion

A travers un générateur d’instance et d’un solveur, il est possible d’obtenir le "level" dans l’arbre avec le temps CPU pour obtenir un ensemble de solutions.



Treillis de permutation

Problème	Nature du problème
$1 d^{*j} \sum w_j p_j$	problème NP-difficile
$1 d^{*j}, p_j = 1 \sum w_j p_j$	problème non déterminé
$1 d^{*j} \sum w_j H_j$	distance de Hamming (avec H_j non défini), problème non déterminé
$1 d^{*j} \sum j p_j$	problème non déterminé

Problèmes existants

Taches j	1	2	3	4	5
Durée p_j	2	3	6	5	1
Poids w_j	1	3	2	1	2
Date écheue d_j	8	11	7	15	1

Ordonnement pour 1 machine



Nouveaux problèmes d'ordonnancement

Résumé

Ce projet de recherche et développement a pour objectif d'obtenir un ensemble de solutions à travers un treillis via plusieurs méthodes de résolution, dans le cas d'un problème d'ordonnement à une machine.

Dans ce projet, un générateur d'instance aléatoire est développé, ainsi que 3 méthodes de résolutions : MILP, DP et Branch and Bound.

Mots-clés

PRD, Ordonnancement, Treillis de permutation, CPLEX, Branch and Bound

Abstract

This research and development project aims to obtain a set of solutions through several methods of resolution, in the case of problem's scheduling with one machine.

In this project, a random instance generator is developed, as well as 3 resolving methods: MILP, DP and Branch and Bound.

Keywords

PRD, Scheduling, Treillis of permutation, CPLEX, Branch and Bound

Tuteurs académiques

Jean-Charles BILLAUT
Thanh Thuy Tien TA

Étudiant

Kivin RINGARD (DI5)