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

Département Informatique
5e année
2012 - 2013

Rapport de projet de fin d'étude

Etude d'un flowshop pour minimiser le C_{max} sous diverses contraintes de blocage

Encadrants
Christophe LENTÉ
christophe.lente@univ-tours.fr
Nhat Vinh VO
nhat.vo@univ-tours.fr

Étudiante
Mélanie MAUGEAIS
melanie.maugeais@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours

Version du 16 mai 2013

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier mes encadrants, Christophe LENTÉ et Nhat Vinh VO, enseignant chercheur et doctorant à Polytech'Tours, pour leur disponibilité et leur aide.

Un grand merci à Pauline FOUILLET, étudiante et amie, qui a travaillé sur un projet de fin d'étude similaire au mien, et avec qui l'entraide était de mise.

Je remercie également mes collègues de promo, pour leur aide ponctuelle.

Introduction

Ce projet de fin d'étude (PFE) s'inscrit dans le cadre de la dernière année de formation d'ingénieur en informatique à Polytech'Tours. Il s'agit d'étudier un problème d'ordonnancement d'atelier du type flowshop, soumis à différentes contraintes de blocage, dont le critère à minimiser est le C_{max}, qui correspond à la date de fin d'exécution du dernier travail sur le dernier atelier. Dans cette étude, c'est l'absence de zone de stockage entre deux machines qui vient créer des blocages.

L'objectif de l'étude est de s'intéresser plus particulièrement à un nouveau type de blocage : la contrainte RCb (Release when Completing Blocking), qui intervient dans de nombreux domaines industriels tels que le traitement de déchets industriels et le transport ferroviaire. Nous verrons également les contraintes RSb (Release when Starting Blocking) et RCb* qui est une variante de la contrainte RCb.

Afin de mener à bien ce projet, il faut étudier deux articles présentant des travaux menés sur cette nouvelle contrainte de blocage, et adapter une procédure par séparation et évaluation (PSE) créée en 2005 par Vincent AUGUSTO, résolvant un problème de flowshop avec délais minimaux et maximaux, pour la minimisation du C_{max}.

Après avoir fait le point sur quelques prérequis nécessaires à la compréhension du travail effectué, ce rapport présente les modélisations en algèbre Max-Plus établies ainsi que les bornes inférieures et supérieures mises en place pour adapter la PSE existante à la résolution de ce problème de flowshop spécifique.

Enfin, un bilan et une conclusion détaillés permettront d'avoir une vue globale sur l'avancement et les perspectives d'avenir du projet. Vous trouverez également en annexes quelques exemples, plusieurs fichiers résultats et divers rapports complémentaires dont le cahier de spécifications de ce projet.

Prérequis

3.1 Notations utilisées

Dans ce rapport, j'utilise quelques notations qu'il est bon de connaître pour comprendre les formules ou les schémas proposés. En voici une courte liste :

- S_{ij} désigne la date de début d'exécution du job i sur la machine j
- C_{ij} indique la date de fin d'exécution du job i sur la machine j
- P_{ij} correspond à la durée d'exécution du job i sur la machine j
- t_j désigne la date de disponibilité initiale de la machine j, c'est à dire avant ordonnancement d'une séquence ou d'un job.
- D_j correspond à la date de disponibilité finale de la machine j, c'est à dire après l'ordonnancement d'une séquence ou d'un job.
  On a donc $\forall j, t_j \leq D_j$
- n désigne le nombre de jobs et m le nombre de machines du problème.

3.2 Le flowshop

Le flowshop est un problème d'ordonnancement d'atelier dans lequel n jobs vont s'exécuter successivement sur m machines. Les contraintes d'un flowshop sont de deux types :

- Contrainte de gamme : toutes les tâches doivent passer sur toutes les machines (de la machine 1 à la machine m).
- Contrainte de ressource : une machine ne peut traiter qu'une seule tâche à la fois.

On peut distinguer également deux types d'ordonnancement, selon que les tâches peuvent ou non se doubler et ainsi modifier leur ordre de passage d'une machine à l'autre.

Si l'ordre dans lequel les travaux passent sur les machines reste le même jusqu'au bout, alors on parle de flowshop de permutation, sinon, il s'agit d'un flowshop de non-permutation.

3.2.1 Min-delay

Le min-delay est une appellation personnalisée pour désigner le $Fm|a_{ij}|C_{max}$. Il s'agit d'un problème de flowshop particulier de min-max-delay, où on considère le délai maximum comme étant infini.

Le délai minimum correspond à un temps d'attente minimum qui vient en quelque sorte gonfler la durée d'exécution du job sur une machine, l'empêchant de commencer son exécution sur la suivante dès qu'il est normalement terminé.

Le délai d'attente peut donc varier entre la valeur minimale a_{ij} et l'infini.

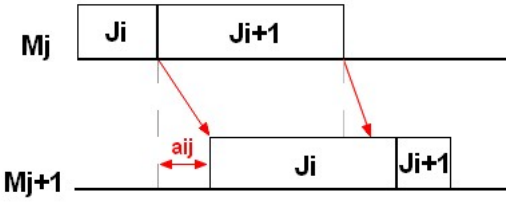


FIGURE 3.1 – Schéma d'illustration d'un flowshop min-delay

Nous pouvons constater sur le schéma précédent qu'un job J_i peut commencer sur la machine suivante M_{j+1} soit à la fin de son exécution sur la machine M_j plus le délai a_{ij}, soit dès la fin d'exécution du job précédent sur la machine M_{j+1}.

La matrice d'un tel flowshop, pour deux machines par exemple, se distingue par le fait que l'élément le plus en bas à gauche de la matrice vaut $-\infty$.

3.2.2 A-wait

L'appellation a-wait désigne ici un flowshop de type min-max-délai particulier, pour lequel les délais minimaux et maximaux sont égaux : $min_{ij} = max_{ij} = a_{ij}$. Dans ce cas, le délai est donc fixe, un job J_i ne peut donc commencer sur la machine M_{j+1} ni avant ni après sa fin d'exécution sur la machine M_j plus le délai a_{ij}. Ce flowshop est ainsi bien plus restrictif que le min-max-delay, et un ordonnancement de flowshop min-max-delay est la plupart du temps plus court que celui d'un flowshop de type a-wait.

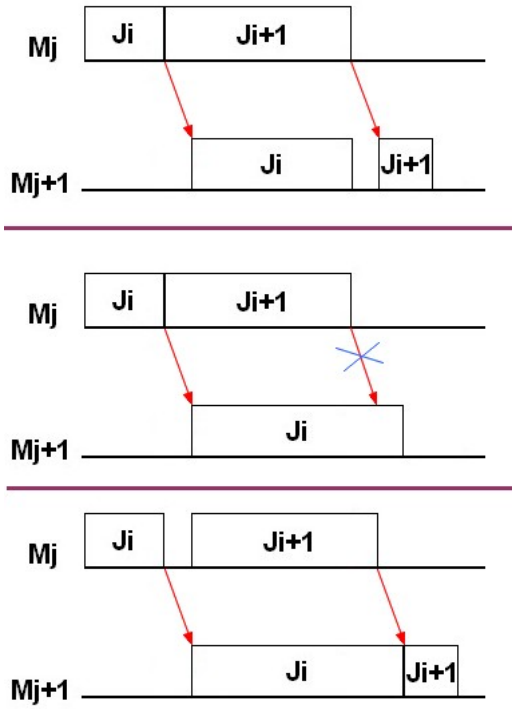


FIGURE 3.2 – Schéma d'illustration d'un flowshop a-wait

Ce schéma présente 3 situations différentes : la première est un cas où tout se passe bien, le délai a_{ij} est respecté. Les deux autres schémas montrent une situation non réalisable, pour laquelle le délai a_{ij} serait dépassé pour le job J_{i+1} et la manière par laquelle on résout le problème : On insère un temps mort entre les jobs J_i et J_{i+1} sur la machine M_j pour pouvoir respecter le délai a_{ij}.

Dans le cas particulier où le délai est nul pour tous les jobs sur toutes les machines, on appelle le problème "no-wait", et il n'y a donc aucune attente entre deux opérations successives d'un même job.

La matrice d'un flowshop de type a-wait sur deux machines se reconnaît par le fait que les additions de ses termes diagonaux sont égales.

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}$$

On a un flowshop de type a-wait si $M_{11} + M_{22} = M_{21} + M_{12}$.

3.3 Les contraintes de blocage

Lorsqu'une usine ne possède pas d'espace de stockage intermédiaire entre deux ateliers, on peut rencontrer des blocages, qui ont pour conséquence principale de ralentir la production.

En effet, un travail ayant fini son exécution sur une machine ne peut commencer son exécution sur la machine suivante seulement si cette dernière est disponible. Sans zone de stockage entre les machines, on obtient un blocage sur la première, qui peut ensuite en engendrer d'autres en cascade. Afin d'éviter cela, on fait intervenir l'ordonnancement, de sorte à réduire au plus les temps de blocage.

Nous allons nous intéresser ici à trois types de blocage : RSb, RCb et RCb*.

3.3.1 RSb

Cette première contrainte, Release when Starting Blocking, est une contrainte de blocage classique dans un problème de flowshop.

Dans ce cas, un job J_i ne peut commencer son exécution sur la machine M_j que si le job J_{i-1} a commencé son exécution sur la machine M_{j+1}.

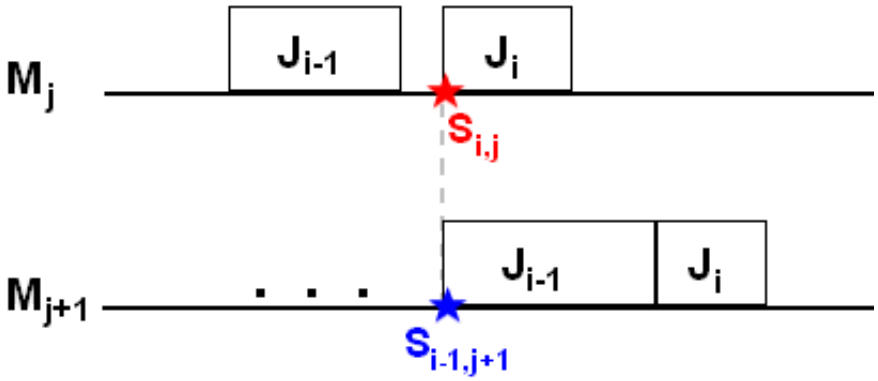


FIGURE 3.3 – Schéma explicatif de la contrainte RSb

Des blocages interviennent alors entre les jobs J_{i-1} et J_i sur la machine M_j dès que la date de début d'exécution du job J_{i-1} sur la machine M_{j+1} est supérieure à sa date de fin d'exécution sur la machine M_j.

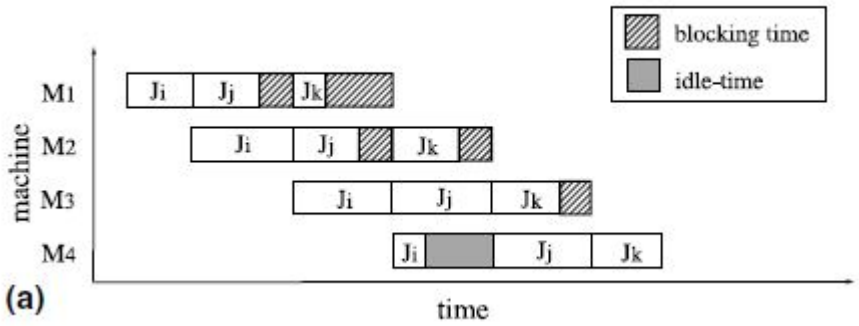


FIGURE 3.4 – Illustration de la contrainte RSb

3.3.2 RCb

La contrainte RCb, Release when Completing Blocking, est le nouveau blocage qui a été très peu étudié jusqu'à présent, et sur lequel porte cette étude.

Cette contrainte entraîne qu'un job J_i ne peut commencer sur la machine M_j seulement lorsque le job précédent, J_{i-1}, a libéré la machine M_{j+1}, c'est à dire s'il a commencé son exécution sur la machine M_{j+2}.

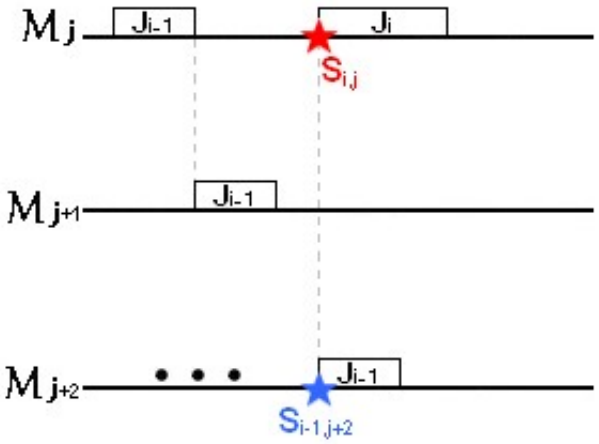


FIGURE 3.5 – Schéma explicatif de la contrainte RCb

Avec cette contrainte, les blocages peuvent être nombreux si l'ordonnancement des tâches n'est pas mûrement réfléchi.

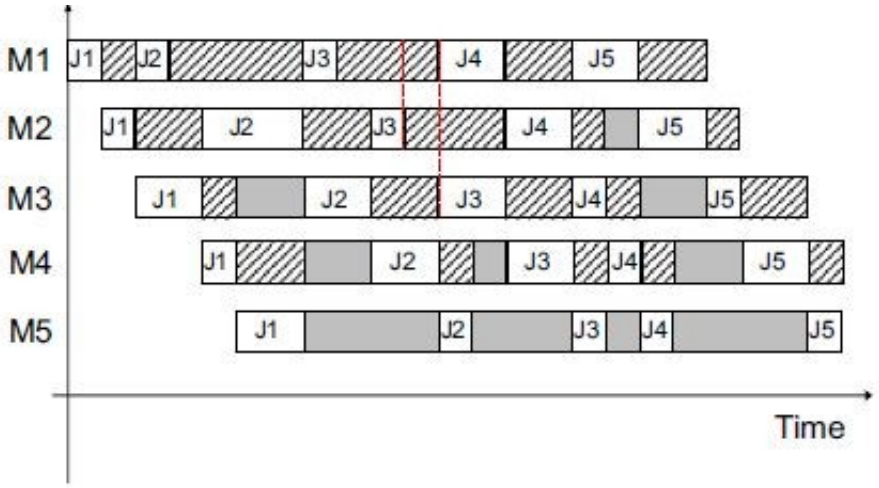


FIGURE 3.6 – Illustration de la contrainte RCb

3.3.3 RCb*

La contrainte RCb* est une nuance de la contrainte précédente. Dans ce cas, au lieu d'attendre la libération de la machine suivante par le job précédent, le job J_i va pouvoir commencer son exécution sur la machine M_j dès que le job J_{i-1} aura **terminé** son exécution sur la machine M_{j+1}.

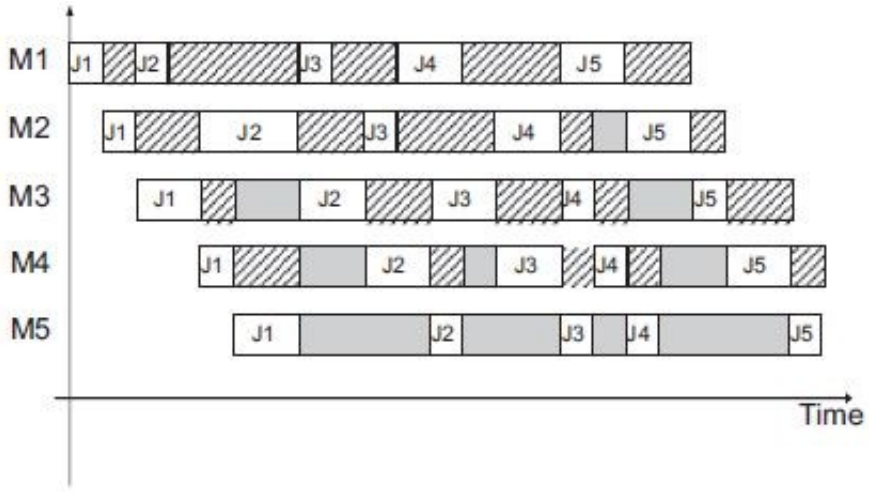


FIGURE 3.7 – Illustration de la contrainte RCb*

3.4 L'algèbre Max-Plus

3.4.1 Quelques notions

L'algèbre Max-Plus est un autre nom donné aux mathématiques tropicales, elle est définie de la manière suivante : $(\mathbf{R} \cup \{-\infty\}, \oplus, \otimes)$.

L'addition et la multiplication sont redéfinies telles que :

- L'addition devient un maximum. Nous appelons cette nouvelle opération "o-plus" : $\oplus \Leftrightarrow \max$
- La multiplication devient une addition. Nous appelons alors cette opération "o-fois" : $\otimes \Leftrightarrow +$

Chacune de ces opérations possède un élément neutre :

- L'élément neutre du o-plus $\oplus$ est : $\mathbb{0} = -\infty$
- L'élément neutre du o-fois $\otimes$ est : $\mathbb{1} = 0$

Le système ainsi défini possède plusieurs propriétés, dont quelques unes sont citées ci-dessous :

- Demi-corps idempotent : $\forall a, a \oplus a = a$.
- $\nexists b$ tel que $a \oplus b = \mathbb{0}$, sauf si $a = \mathbb{0}$.
- $a \oplus b = a \oplus c \not\Rightarrow b = c$. Voici un exemple : $max(4,1) = max(4,3) \not\Rightarrow 1 = 3$.
- $a \otimes (-a) = \mathbb{1}$, on note $(-a) = a^{-1}$.
- $a \otimes \mathbb{0} = \mathbb{0}$.

Il est possible d'effectuer des calculs matriciels avec de telles notations, dont l'addition et la multiplication se font ainsi :

L'addition se fait terme à terme, avec l'opérateur o-plus.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \oplus \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a \oplus e & b \oplus f \\ c \oplus g & d \oplus h \end{pmatrix}$$

La multiplication s'effectue de manière classique, sauf que l'on utilise les opérateurs o-fois et o-plus.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \otimes \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a \otimes e \oplus b \otimes g & a \otimes f \oplus b \otimes h \\ c \otimes e \oplus d \otimes g & c \otimes f \oplus d \otimes h \end{pmatrix}$$

Il est important de noter que les matrices ne sont pas toujours inversables.

Notons également la forme de la matrice identité : $\begin{pmatrix} \mathbb{1} & \mathbb{0} & \mathbb{0} & \mathbb{0}\dots & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} & \mathbb{0}\dots & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} & \mathbb{0}\dots & \mathbb{0} \\ . & . & . & . & . \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0}\dots & \mathbb{1} \end{pmatrix}$

NB : L'écriture $a \otimes b$ est souvent simplifiée en $a.b$ ou ab.

3.4.2 Intérêt

La modélisation des problèmes d'ordonnancement avec l'algèbre Max-Plus permet une analyse plus aisée de ces problèmes.

En effet, dans l'étude d'un flowshop par exemple, elle permet de faire ressortir des caractéristiques précises telles que le type de flowshop : a-wait/no-wait, min-delay, ...

Toutefois, cette méthode présente un inconvénient : l'algèbre Max-Plus est utile pour mieux comprendre le problème, mais elle n'est pas pratique pour les calculs, il faut donc repasser en notations usuelles.

3.5 Procédure par Séparation et Evaluation

Une procédure par séparation et évaluation, aussi appelée branch-and-bound en anglais, est une méthode de résolution exacte basée sur une énumération de toutes les solutions possibles.

L'énumération des solutions se fait par construction d'un arbre, qui contient à chaque noeud la séquence partielle des jobs ordonnancés, et la séquence des travaux restant.

Voici un exemple d'arbre, dans lequel est inscrit en chaque noeud la séquence partielle ordonnancée :

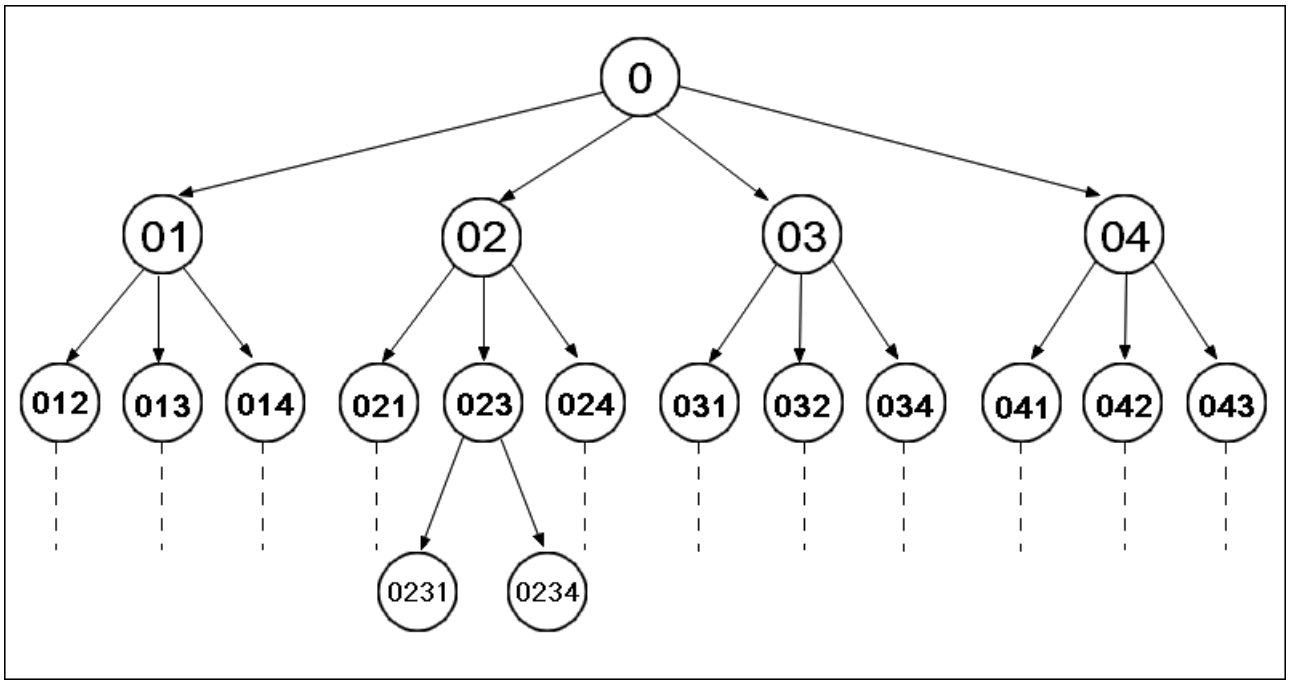


FIGURE 3.8 – Arbre des solutions d'une PSE

Une telle procédure se distingue d'une force brute par le fait qu'elle dispose d'une borne inférieure et d'une borne supérieure à chaque noeud, encadrant ainsi la meilleure solution qu'il sera possible d'atteindre en parcourant cette branche. Ainsi, en gardant en mémoire à chaque instant la meilleure valeur de borne supérieure calculée lors de son parcours, la PSE est en mesure de couper une branche qui ne pourra pas aboutir à l'optimum, dès lors que la borne inférieure en un noeud devient supérieure ou égale à cette meilleure borne supérieure.

Une borne inférieure est d'autant plus efficace que sa valeur est haute, contrairement à la borne supérieure, qui est meilleure lorsque sa valeur est la plus basse possible ; l'idéal étant que la borne inférieure et la borne supérieure deviennent égales, puisqu'elles désigneraient alors toutes les deux l'optimum cherché.

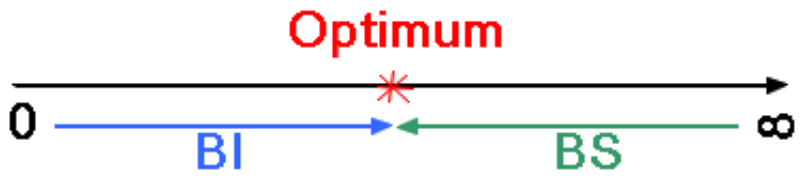


FIGURE 3.9 – Encadrement de l'optimum par les bornes inférieure et supérieure

De cette manière, on réduit considérablement la durée de résolution de grandes instances de problèmes NP-difficiles.

L'avantage principal d'une PSE est qu'elle permet de trouver la valeur optimale du critère évalué, en un temps réduit comparé à une exploration de toutes les solutions possibles, et, si jamais la durée d'exécution de la méthode venait à s'allonger de trop, il sera tout de même possible d'avoir un encadrement de la valeur optimale, grâce aux bornes inférieure et supérieure calculées en chaque noeud.

Travail réalisé

Le problème étudié dans le cadre de ce projet de fin d'étude est un flowshop dans lequel il n'y a pas de zone de stockage intermédiaire entre deux machines. Le critère à minimiser est la date de fin d'exécution du dernier job sur la dernière machine, appelé le C_{max}, en prenant en compte une contrainte de blocage unique sur chacune des machines de l'atelier.

C'est à ce stade les seules spécificités connues sur le problème.

4.1 Modélisations

J'ai établi les modélisations, en algèbre Max-Plus, des trois problèmes étudiés : un flowshop soumis à chacune des trois contraintes RSb, RCb et RCb*, afin de pouvoir les analyser.

4.1.1 Etude préalable

Dans un premier temps, ma réflexion s'est portée sur papier. J'ai commencé par étudier les graphes de précédence pour chaque contrainte, afin de trouver d'autres spécificités au problème, et peut être trouver une similitude entre ces trois blocages.

Les graphes n'ont été faits que pour un flowshop à quatre machines, et pour un seul job, mais ils reflètent des caractéristiques générales pour le problème.

Notez également que les notations sont simplifiées dans ces schémas : afin de simplifier, on n'indique pas l'indice du job considéré puisqu'il n'y en a qu'un seul.

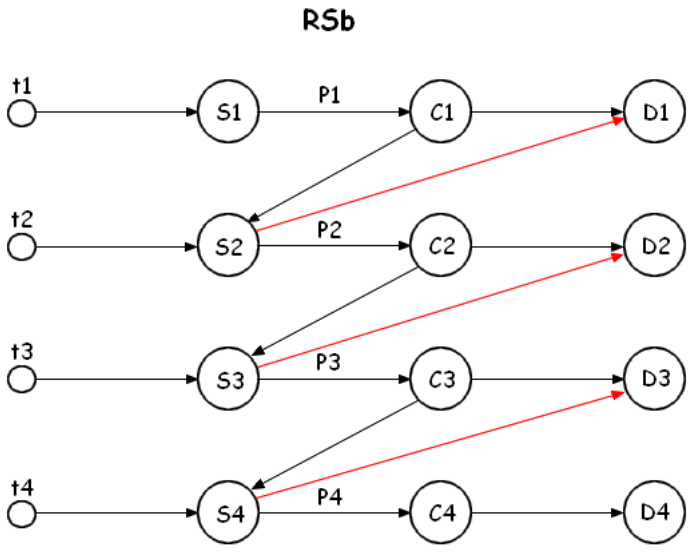


FIGURE 4.1 – Graphe de précédence : contrainte RSb

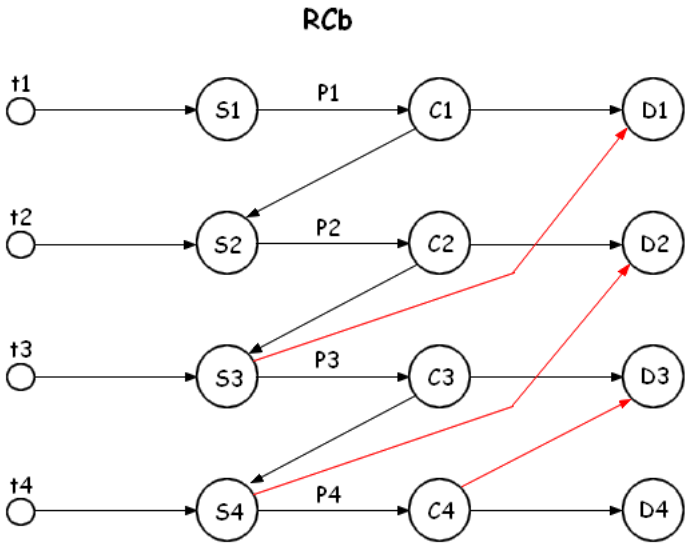


FIGURE 4.2 – Graphe de précédence : contrainte RCb

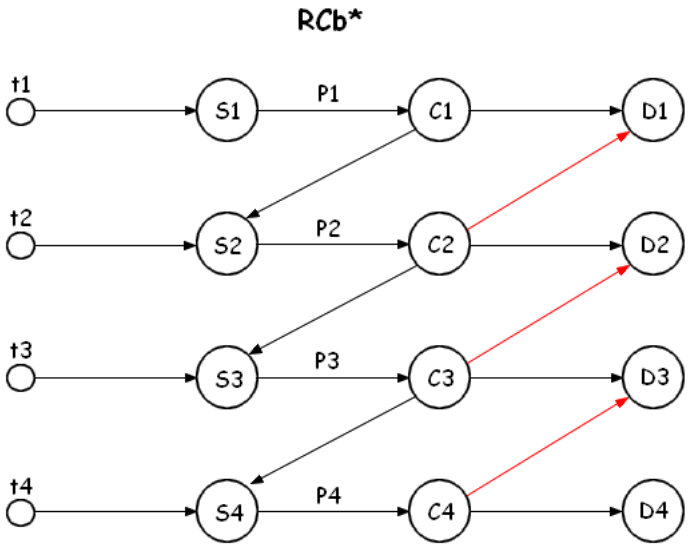


FIGURE 4.3 – Graphe de précédence : contrainte RCb*

Grâce à ces graphes de précédence, qui sont à la base de mes modélisations, j'ai pu constater que le flowshop était forcément de **permutation** [1] pour chacun des blocages. Nous avons donc ici mis en évidence une nouvelle spécificité de notre problème.

1. Un flowshop est dit de permutation si les jobs s'exécutent toujours dans le même ordre sur toutes les machines.

4.1.2 Modélisations établies

Ensuite, toujours en me basant sur les graphes de précédence précédents, j'ai relevé différentes inégalités mettant en jeu les dates de début et de fin d'exécution d'un job sur les quatre machines prises pour base, ainsi que les dates de disponibilité t et D de chaque machine et les durées d'exécution du job considéré sur les machines.

RSb	RCb*	RCb
$\begin{cases} D_1 \geq C_1 \oplus t_2 \\ D_2 \geq C_2 \oplus t_3 \\ D_3 \geq C_3 \oplus t_4 \\ D_4 \geq C_4 \end{cases}$	$\begin{cases} D_1 \geq C_1 \oplus C_2 \\ D_2 \geq C_2 \oplus C_3 \\ D_3 \geq C_3 \oplus C_4 \\ D_4 \geq C_4 \end{cases} \Leftrightarrow \begin{cases} D_1 \geq C_2 \\ D_2 \geq C_3 \\ D_3 \geq C_4 \\ D_4 \geq C_4 \end{cases}$	$\begin{cases} D_1 \geq C_1 \oplus S_3 \\ D_2 \geq C_2 \oplus S_4 \\ D_3 \geq C_3 \oplus C_4 \\ D_4 \geq C_4 \end{cases} \Leftrightarrow \begin{cases} D_1 \geq C_1 \oplus C_2 \oplus t_3 \\ D_2 \geq C_2 \oplus C_3 \oplus t_4 \\ D_3 \geq C_3 \oplus C_4 \\ D_4 \geq C_4 \end{cases}$

TABLE 4.1 – Modélisations : inégalités initiales

Notons également les contraintes issues du problème de flowshop suivantes :

$$\begin{cases} C_1 \geq t_1.P_1 \\ C_2 \geq t_2.P_2 \oplus C_1.P_2 \\ C_3 \geq t_3.P_3 \oplus C_2.P_3 \\ C_4 \geq t_4.P_4 \oplus C_3.P_4 \end{cases} \Leftrightarrow \begin{cases} C_1 = t_1.P_1 \\ C_2 = t_2.P_2 \oplus t_1.P_1.P_2 \\ C_3 = t_3.P_3 \oplus t_2.P_2.P_3 \oplus t_1.P_1.P_2.P_3 \\ C_4 = t_4.P_4 \oplus t_3.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_1.P_1.P_2.P_3.P_4 \end{cases}$$

Puis, en détaillant ces inégalités grâces aux contraintes issues du problème de flowshop, on peut les simplifier et exprimer les dates suivantes de disponibilité des machines D, en fonction de leurs dates de disponibilité initiales t et des durées d'exécution du job :

RSb	RCb*
$\begin{cases} D_1 \geq t_1.P_1 \oplus t_2 \\ D_2 \geq t_1.P_1.P_2 \oplus t_2.P_2 \oplus t_3 \\ D_3 \geq t_1.P_1.P_2.P_3 \oplus t_2.P_2.P_3 \oplus t_3.P_3 \oplus t_4 \\ D_4 \geq t_1.P_1.P_2.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_3.P_3.P_4 \oplus t_4.P_4 \end{cases}$	$\begin{cases} D_1 \geq t_1.P_1.P_2 \oplus t_2.P_2 \\ D_2 \geq t_1.P_1.P_2.P_3 \oplus t_2.P_2.P_3 \oplus t_3.P_3 \\ D_3 \geq t_1.P_1.P_2.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_3.P_3.P_4 \oplus t_4.P_4 \\ D_4 \geq t_1.P_1.P_2.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_3.P_3.P_4 \oplus t_4.P_4 \end{cases}$
RCb	
$\begin{cases} D_1 \geq t_1.P_1.P_2 \oplus t_2.P_2 \oplus t_3 \\ D_2 \geq t_1.P_1.P_2.P_3 \oplus t_2.P_2.P_3 \oplus t_3.P_3 \oplus t_4 \\ D_3 \geq t_1.P_1.P_2.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_3.P_3.P_4 \oplus t_4.P_4 \\ D_4 \geq t_1.P_1.P_2.P_3.P_4 \oplus t_2.P_2.P_3.P_4 \oplus t_3.P_3.P_4 \oplus t_4.P_4 \end{cases}$	

TABLE 4.2 – Modélisations : inégalités détaillées

Enfin, en écrivant sous forme matricielle ces dernières inégalités, nous obtenons la forme de la matrice d'un travail, pour chacune des trois contraintes ; il s'agit des matrices mxm situées à droite des inégalités :

RSb
$\begin{pmatrix} D_1 & D_2 & D_3 & D_4 \end{pmatrix} \geq \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \end{pmatrix} \otimes \begin{pmatrix} P_1 & P_1.P_2 & P_1.P_2.P_3 & P_1.P_2.P_3.P_4 \\ \mathbb{1} & P_2 & P_2.P_3 & P_2.P_3.P_4 \\ \mathbb{0} & \mathbb{1} & P_3 & P_3.P_4 \\ \mathbb{0} & \mathbb{0} & \mathbb{1} & P_4 \end{pmatrix}$
RCb*
$\begin{pmatrix} D_1 & D_2 & D_3 & D_4 \end{pmatrix} \geq \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \end{pmatrix} \otimes \begin{pmatrix} P_1.P_2 & P_1.P_2.P_3 & P_1.P_2.P_3.P_4 & P_1.P_2.P_3.P_4 \\ P_2 & P_2.P_3 & P_2.P_3.P_4 & P_2.P_3.P_4 \\ \mathbb{0} & P_3 & P_3.P_4 & P_3.P_4 \\ \mathbb{0} & \mathbb{0} & P_4 & P_4 \end{pmatrix}$
RCb
$\begin{pmatrix} D_1 & D_2 & D_3 & D_4 \end{pmatrix} \geq \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \end{pmatrix} \otimes \begin{pmatrix} P_1.P_2 & P_1.P_2.P_3 & P_1.P_2.P_3.P_4 & P_1.P_2.P_3.P_4 \\ P_2 & P_2.P_3 & P_2.P_3.P_4 & P_2.P_3.P_4 \\ \mathbb{1} & P_3 & P_3.P_4 & P_3.P_4 \\ \mathbb{0} & \mathbb{1} & P_4 & P_4 \end{pmatrix}$

TABLE 4.3 – Modélisations : forme matricielle des inégalités et matrice d'un travail

En observant bien ces matrices, on remarque qu'elles sont généralisables à m machines de la manière suivante, avec aij l'élément de la matrice à la ligne i et à la colonne j :

RSb
$a_{ij} = \begin{cases} \bigotimes_{k=i}^{j}(P_k) & , \quad \text{si } i \leq j \\ \begin{cases} \mathbb{1} & , \quad \text{si } (i > j)\&(i = j+1) \\ \mathbb{0} & , \quad \text{sinon} \end{cases} \end{cases}$
RCb*
$\begin{cases} \forall j \in [1, m-1] & a_{ij} = \begin{cases} \bigotimes_{k=i}^{j+1}(P_k) & , \quad \text{si } i \leq j \\ P_i & , \quad \text{si } (i > j)\&(i = j+1) \\ \mathbb{0} & , \quad \text{sinon} \end{cases} \\ \forall i \in [1, m] & a_{im} = a_{im-1} \end{cases}$
RCb
$\begin{cases} \forall j \in [1, m-1] & a_{ij} = \begin{cases} \bigotimes_{k=i}^{j+1}(P_k) & , \quad \text{si } i \leq j \\ P_i & , \quad \text{si } (i > j)\&(i = j+1) \\ \mathbb{1} & , \quad \text{si } (i > j)\&(i = j+2) \\ \mathbb{0} & , \quad \text{sinon} \end{cases} \\ \forall i \in [1, m] & a_{im} = a_{im-1} \end{cases}$

TABLE 4.4 – Modélisations : généralisation de la matrice d'un travail à m machines

Ce sont donc ces dernières équations, donnant la forme généralisée de la matrice d'un travail qui vont nous être utiles pour établir une borne inférieure à notre problème.

Les inégalités, exprimant les D_j en fonction des t_j, vont également servir à plusieurs reprises dans la PSE : pour le calcul du C_{max} d'un flowshop soumis à chacune des contraintes de blocage étudiées, et tout simplement pour mettre à jour les dates de disponibilité des machines après chaque ordonnancement partiel.

4.1.3 Tests

Afin de tester les modélisations établies, j'ai réalisé deux fonctions : *CalculerCmaxInegalites* et *CalculerCmaxMatrices*.

La première permet de tester la justesse des modélisations à partir d'une mise à jour successive des dates de disponibilité des machines, après l'ordonnancement de chacun des jobs de la séquence. Le C_{max} de la séquence correspond à la date de disponibilité finale de la dernière machine.

La seconde permet de calculer le C_{max} d'une séquence à partir de multiplications successives des matrices des jobs. Le C_{max} correspond alors à l'élément le plus en haut à droite de la matrice finale.

Chacune de ses deux fonctions a été testée sur des exemples plus ou moins simples, pour lesquels je calculais à la main le C_{max} afin de connaître la valeur à obtenir.

4.2 Bornes supérieures

L'établissement d'une borne supérieure pour ma PSE repose sur le mini-projet de GPF (Gestion des Flux et de la Production) que j'ai encadré, dans lequel les étudiants de 4ème année Maxime SERRA et Fabien FARIN étaient chargés d'étudier, d'implémenter et de tester les algorithmes fournis dans l'article "Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems", de W. TRABELSI, C. SAUVEY et N. SAUER.

De ce fait, je ne donnerais pas les algorithmes ni d'explication poussée, mais le rapport du projet de GPF étant fourni en annexe, vous aurez tous les compléments d'information souhaités.

NB : L'illustration donnée pour une meilleure compréhension de l'algorithme NEH ne tient pas compte d'une contrainte de blocage. La différence de résultat est visible uniquement dans la valeur du C_{max} calculé, et influence ainsi la séquence finale obtenue, mais ceci n'interfère pas dans la compréhension du principe des heuristiques.

4.2.1 NEH

NEH (Nawaz-Enscore-Ham) est une heuristique très connue pour la résolution des problèmes d'ordonnancement de type flowshop.

Le principe est le suivant :

- Trier les jobs par $\sum_{j=1}^{m}(P_{ij})$ décroissant
- Ordonnancer les deux premiers travaux dans l'ordre minimisant le C_{max}
- Ensuite, prendre chaque autre travail dans la liste triée et l'insérer à la meilleure position pour la minimisation du C_{max} dans l'ordonnancement partiel.

Afin d'illustrer cette heuristique, voici un exemple d'application :

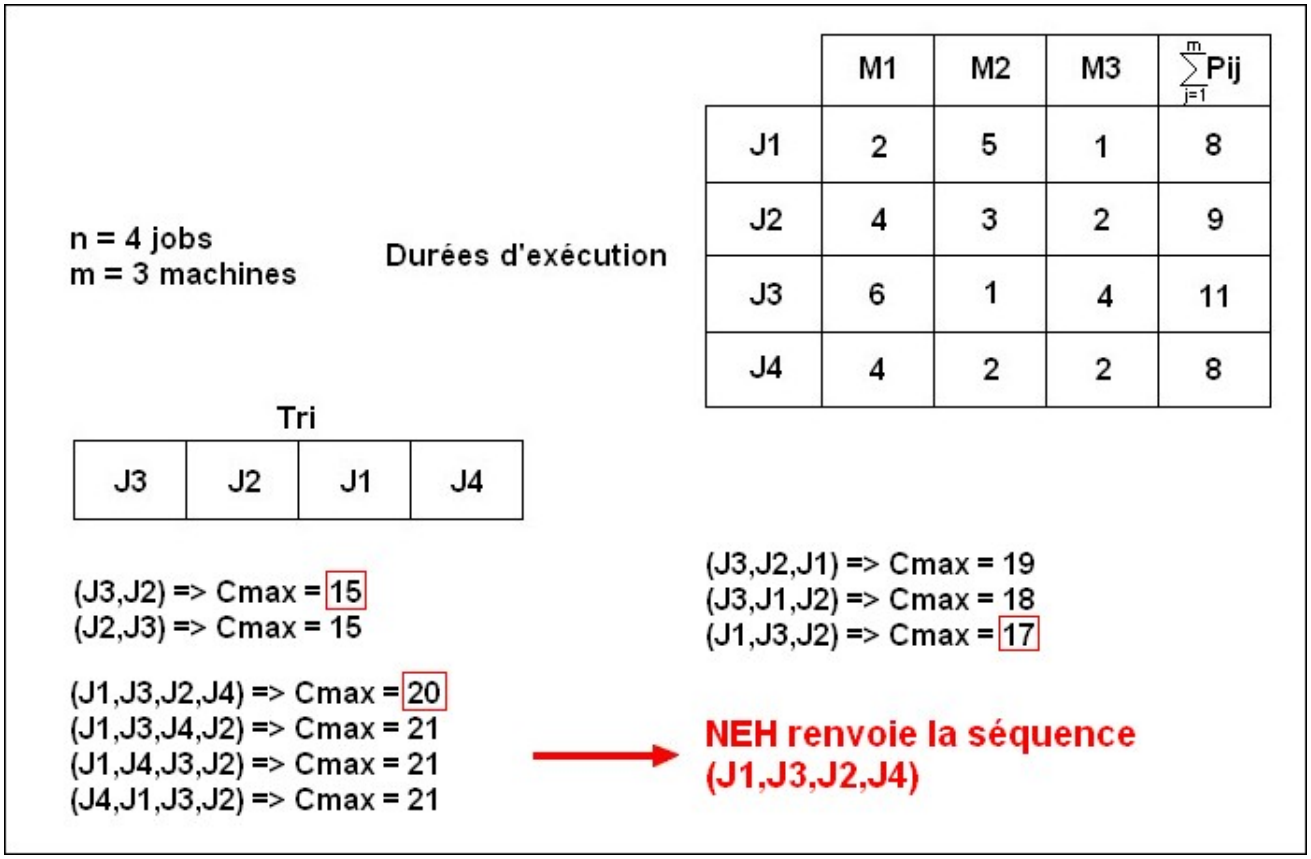


FIGURE 4.4 – Exemple d'application de l'heuristique NEH

4.2.2 TSS

Cette heuristique est nouvelle dans le monde de l'ordonnancement. Elle porte le nom de ses auteurs, Trabelsi, Sauvey et Sauer, et a été créée dans l'objectif de résoudre les problèmes d'ordonnancement de type flowshop, soumis aux diverses contraintes de blocage étudiées dans ce projet.

Pour leur heuristique, les auteurs introduisent trois variables et un critère :
- Cp_{max} : C_{max} de la séquence partielle,
- $SomTpsIn$: somme des durées d'inactivité (temps de blocage + temps d'attente),
- $SomTpsEx$: somme des durées d'exécution des opérations placées, et
- $Cr = \min(Cp_{max} + SomTpsIn - SomTpsEx)$: critère de choix pour la tâche à ordonnancer à la suite de la séquence partielle déjà ordonnancée.

Le principe est le suivant :
- Placer successivement chaque job J_i en première position dans la séquence.
- A chaque fois, il faut calculer, pour chacun des jobs J_t restant à placer, Cp_{max}, $SomTpsIn$, $SomTpsEx$ et Cr le critère après placement du job J_t. On place à la suite de la séquence partielle le job J_t qui minimise le critère.
- On calcule ensuite pour chaque job J_i, $C_{max,i}$, le temps d'exécution final avec J_i en première position.
- Enfin, on choisi la séquence qui minimise $C_{max,i}$.

Voici un exemple d'application, sur la contrainte RCb, pour mieux comprendre le déroulement de la méthode ; un autre exemple est donné dans l'article présentant l'algorithme, mixant les contraintes de blocage :

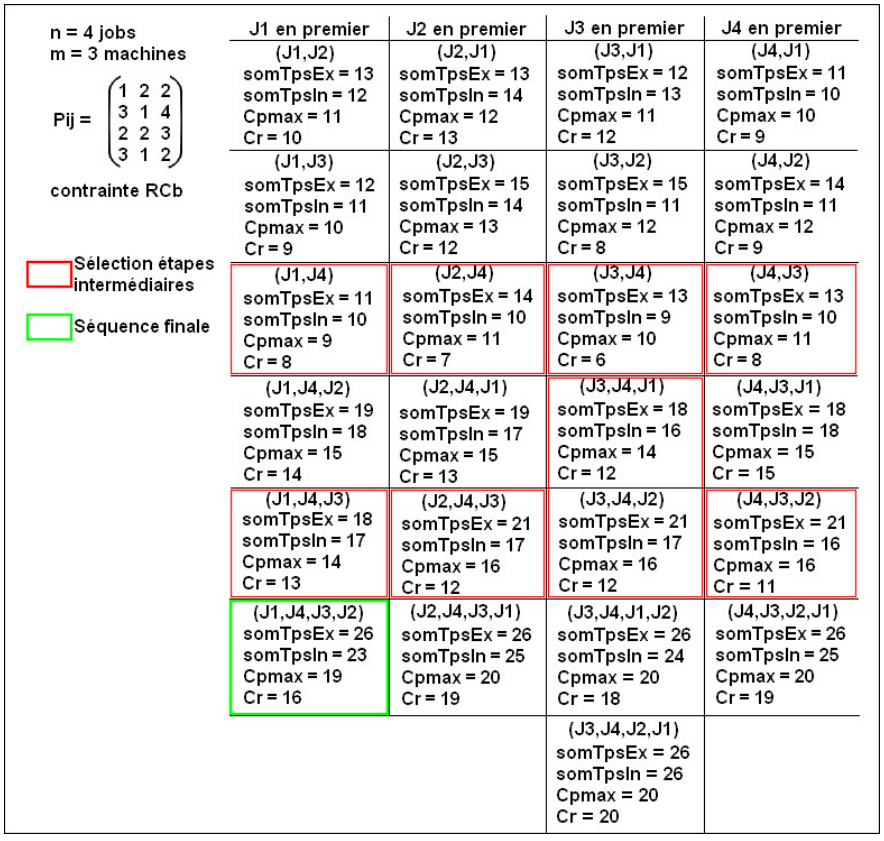


FIGURE 4.5 – Exemple d'application de l'heuristique TSS

4.2.3 NEH improvement

Pour améliorer la solution obtenue après application de la méthode NEH sur un problème de flowshop, on peut s'appuyer sur la méthode NEH itérative. Le principe est simple : on applique la méthode NEH à partir de la séquence précédemment obtenue avec l'algorithme, au lieu de partir d'une séquence composée des jobs triés par temps d'exécution décroissants.

L'algorithme pour cette amélioration est donné dans l'article, il s'agit de l'algorithme 3 ; il est également présent dans le rapport du projet de GPF.

4.2.4 TSS improvement

L'amélioration de la solution obtenue avec l'algorithme TSS est établie à partir d'un nouveau critère, qui nous indiquera comment modifier la séquence résultat de TSS.

Soit S la solution retournée par TSS.

On calcule les temps différentiels de blocage avant et après un job J_i dans la solution d'ordonnancement S.

Ensuite, on prend le job ayant la plus grande somme de temps différentiels de blocage (temps différentiel avant + temps différentiel après), et on le permute, c'est-à-dire qu'on échange sa position, avec tous les autres jobs de la séquence, pour trouver à quelle position il convient le mieux pour minimiser le C_{max}.

Par exemple, si $S = (J_1, J_3, J_4, J_2)$, si on trouve que J_4 possède le plus grand temps différentiel total, alors on va regarder la valeur du C_{max} pour les séquences suivantes : S elle même, (J_4, J_3, J_1, J_2), (J_1, J_4, J_3, J_2) et (J_1, J_3, J_2, J_4).

Si la séquence nouvellement obtenue est différente de S, alors elle remplace S et on recommence. Sinon, on s'arrête.

Aucun algorithme n'est fourni dans l'article, toutefois, j'en ai fourni un aux étudiants en charge du projet de GPF, et il figure donc dans leur rapport.

4.2.5 Encapsulation des fonctions

Le projet de GPF ayant débuté très tôt dans l'avancement de mon PFE, je ne m'étais pas encore penchée sur le code de la PSE fournie, et n'avais par conséquent pas de contrainte particulière à imposer aux étudiants de DI4 pour le codage des algorithmes, si ce n'est que langage de programmation soit le langage C.

De ce fait, lorsque j'ai dû intégrer les fonctions de leur programme dans la PSE, je me suis aperçue que les structures de données qu'ils utilisaient étaient très différentes de celles de la PSE.

J'ai alors dû créer des fonctions encapsulantes, prenant en paramètres les données de la PSE et les transformant en données du programme de GPF pour appeler directement leurs fonctions. A l'inverse, à la fin de mes fonctions encapsulantes, je prévois la modification des données pour qu'elles puissent être retournées et utilisées dans la PSE.

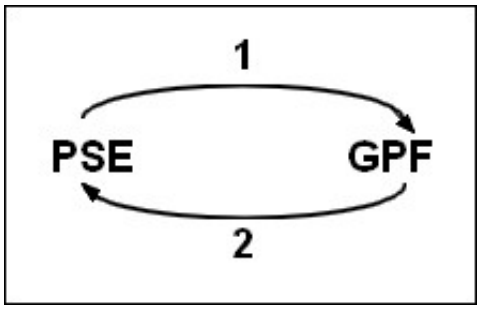


FIGURE 4.6 – Schéma du principe des encapsulations de fonctions

4.2.6 Différentes méthodes de calcul d'une borne supérieure

Dans ma fonction de calcul d'une borne supérieure à mon problème, "BorneSuperieure", j'ai ajouté un paramètre *methode* permettant de renseigner la méthode à appliquer pour obtenir la borne.

En effet, à partir des différentes heuristiques présentées ci-dessus j'ai mis en place 9 méthodes différentes permettant d'obtenir une borne supérieure :

1. Simple calcul du C_{max} de la séquence passée en paramètre à la fonction.
2. On applique l'heuristique NEH sur la séquence, et on calcule le C_{max} de cette nouvelle séquence.
3. On applique l'heuristique TSS sur la séquence, et on calcule le C_{max} de cette nouvelle séquence.
4. Cette méthode se base sur l'amélioration de NEH, suivie d'un calcul du C_{max} de la séquence obtenue.
5. Sur le même principe, nous appliquons l'amélioration de TSS et retournons le C_{max} de la séquence.

   Dans la suite, nous appliquons une combinaison des heuristiques :

6. On applique l'heuristique NEH puis son amélioration, et on récupère le C_{max} de la séquence obtenue.
7. On applique NEH puis l'amélioration de TSS, et on termine à nouveau par un calcul du C_{max}.
8. On calcule le C_{max} de la séquence donnée par l'heuristique TSS, suivie de TSS improvement.
9. Enfin, la dernière méthode possible consiste à utiliser TSS puis l'amélioration de NEH.

L'énumération de ces 9 méthodes est rappelée au début de la fonction de calcul de la borne supérieure, dans le code de la PSE.

4.3 Borne inférieure

Une PSE est d'autant plus efficace que sa borne inférieure est proche de l'optimum. La borne inférieure établie repose sur un principe commun et deux méthodes différentes.

Le principe est le suivant : On considère un flowshop à *n* jobs et *m* machines. On va créer $\frac{m*(m-1)}{2}$ sous-problèmes à n jobs et $m = 2$ machines. Ensuite, en résolvant chacun de ses sous-problèmes de manière optimale, nous obtenons une borne inférieure pour le problème initial.

- On considère la matrice de chaque travail (cette matrice étant donnée par les modélisations de chaque contrainte de blocage).
- L'extraction de chaque sous-matrice 2x2 centrée sur la diagonale de la matrice de chaque travail nous donne les $\frac{m*(m-1)}{2}$ sous problèmes à deux machines.
- Ensuite, selon le type de flowshop correspondant à la forme des sous-matrices extraites (a-wait ou min-delay), nous utilisons la méthode de résolution optimale adéquate : Johnson ou Gilmore et Gomory.

4.3.1 Johnson

L'algorithme de Johnson permet de résoudre de manière optimale un problème de flowshop min-delay à deux machines pour la minimisation du C_{max}.

On note P_{i1} et P_{i2} les durées d'exécution du job J_i sur les machines M_1 et M_2. Le principe de la méthode est le suivant :

- Soient deux ensembles de tâches U et V. On sépare les n jobs à ordonnancer tel que :
$$U = \{J_i / P_{i1} < P_{i2}\} \text{ et } V = \{J_i / P_{i1} \geq P_{i2}\}.$$
- Trier les jobs de l'ensemble U dans l'ordre des P_{i1} croissant.
- Trier les jobs contenus dans l'ensemble V par P_{i2} décroissant.
- La séquence optimale est donnée par la concaténation des ensembles U et V ainsi triés.

Voici un exemple simple, à titre d'illustration :

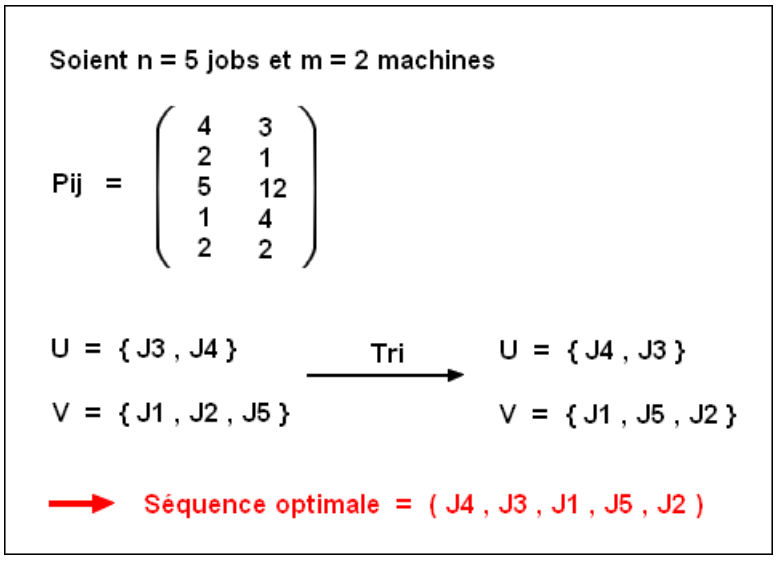


FIGURE 4.7 – Exemple d'application de l'algorithme de Johnson

4.3.2 Gilmore et Gomory

Gilmore et Gomory ont publié en 1964 un article dans lequel ils décrivent une méthode de résolution exacte pour un cas particulier du problème du voyageur de commerce (PVC).

Cette méthode permet de trouver l'ordonnancement optimal pour $n+1$ jobs (ajout d'un job fictif supplémentaire) sur une seule machine, en connaissant leurs dates de début et de fin d'exécution.

Dans ce projet, l'algorithme sera utilisé pour résoudre un problème de flowshop de type a-wait à n jobs et deux machines. Pour utiliser Gilmore et Gomory sur un tel problème, il nous faut constituer un problème fictif à une machine équivalent au premier, en utilisant les données du problème à deux machines.

Le principe de la méthode de Gilmore et Gomory est le suivant : à partir des dates de début d'exécution A_i et de fin d'exécution B_i de chaque job J_i, la méthode calcule le cycle de coût minimal reliant chacun des jobs. En retirant de ce cycle le job fictif ajouté, nous obtenons la séquence optimale cherchée.

Voici un algorithme simplifié de cette méthode de résolution ; l'algorithme complet et les formules de calculs figurent dans l'article de 1964, dont la référence est donnée dans la bibliographie de ce rapport :

- On dispose des dates de début d'exécution A_i et de fin d'exécution B_i de chaque job J_i, dont le job fictif.
- On initialise une permutation ϕ des jobs à partir de quelques manipulations sur les tableaux A_i et B_i.
- On calcule les coûts de chaque transposition $\alpha_{i,i+1}$ entre les jobs J_i et J_{i+1}. On note ce coût $c_\phi(\alpha_{i,i+1})$.
- On forme ensuite un graphe indirect, reliant chaque job J_i au job en position $J_{\phi(i)}$.
- Tant que le graphe possède plus d'une composante, on sélectionne le plus petit coût $c_\phi(\alpha_{i,i+1})$, de telle sorte que les jobs J_i et J_{i+1} soient dans deux composantes différentes, et on les relie par un arc $R_{i,i+1}$.
- On range chaque arc $R_{i,i+1}$ créé dans les groupes suivants :
  - Groupe 1 : contient les arcs $R_{i,i+1}$ tels que $A_{\phi(i)} \geq B_i$
  - Groupe 2 : contient les arcs $R_{i,i+1}$ tels que $B_i > A_{\phi(i)}$
- On trie les arcs contenus dans le groupe 1 dans l'ordre des indices décroissants, et ceux du groupe 2 dans l'ordre des indices croissants.
- La permutation finale est obtenue par composition de la permutation ϕ avec les transpositions désignées par les indices du groupe 1, puis celles du groupe 2 : Si on considère que le groupe 1 possède l éléments et que le groupe 2 en possède m on a alors :

$$\Phi^*(i) = \phi.\alpha_{i_1,i_{1+1}}.\alpha_{i_2,i_{2+1}}...\alpha_{i_l,i_{l+1}}.\alpha_{j_1,j_{1+1}}.\alpha_{j_2,j_{2+1}}...\alpha_{j_m,j_{m+1}}(i)$$

Un exemple d'application complet et détaillé de l'algorithme est fourni dans l'article de Gilmore et Gomory.

4.3.3 Mise en oeuvre

La fonction de calcul d'une borne inférieure porte le nom "BorneInferieure" dans le programme. Elle contient deux portions de code distinctes, mettant en oeuvre deux méthodes : la première est un calcul de borne inférieure basé sur le principe des sous-problèmes à deux machines résolus par Johnson et Gilmore et Gomory selon le type de flowshop dont il s'agit ; et la deuxième est quasi-identique, sauf qu'on améliore la borne inférieure en la rehaussant de quelques valeurs judicieuses.

Nous verrons cette amélioration par la suite.

Dans l'implémentation de la borne inférieure, on commence par extraire la sous-matrice 2x2 correspondant aux machines d'indices k et l, pour chacun des jobs.

Ensuite, selon le type de flowshop auquel elles correspondent, on applique la méthode de résolution qu'il faut : Johnson pour un flowshop min-delay, Gilmore et Gomory pour un flowshop a-wait.

Pour utiliser la méthode de Johnson, nous formons un problème fictif en renseignant les durées d'exécution sur chacune des deux machines du problème et les délais minimaux, à partir des éléments de la matrice extraite :

En modélisation Max-Plus, la matrice extraite est de la forme : $\begin{pmatrix} t_1 & t_{12} \\ \mathbb{0} & t_2 \end{pmatrix}$

Les processing times de chaque job sont donc de la forme : $\begin{pmatrix} \frac{t_{12}}{t_2} & \frac{t_{12}}{t_1} \end{pmatrix}$

Et les délais valent, pour chaque job : $\frac{t_{12}}{(t_1.t_2)}$.

On applique ensuite l'algorithme de Johnson sur ce problème, et on calcule le C_{max} de la séquence obtenue, en considérant un flowshop min-delay.

On obtient ainsi une borne inférieure de notre problème global.

Pour utiliser la méthode de Gilmore et Gomory, il nous faut d'abord créer les tableaux des dates de début et de fin de chaque job, et rajouter une valeur pour le job fictif :

Toujours en modélisation Max-Plus, la matrice extraite est de la forme : $\begin{pmatrix} P_{i1} & P_{i1}.a_i.P_{i2} \\ \frac{\mathbb{1}}{a_i} & P_{i2} \end{pmatrix}$

On a alors pour chacun des n premiers jobs : $A_i = \frac{P_{i1}}{\frac{\mathbb{1}}{a_i}}$ et $B_i = \frac{P_{i2}}{\frac{\mathbb{1}}{a_i}}$.

Pour le job fictif, on définit $A_{n+1} = \min_{i=1}^{n}(\frac{P_{i1}.a_i.P_{i2}}{P_{i1}})$ et $B_{n+1} = \min_{i=1}^{n}(\frac{P_{i1}}{\frac{\mathbb{1}}{a_i}})$

Nous pouvons alors appliquer Gilmore et Gomory sur le problème ainsi créé, et récupérer la longueur du cycle de coût minimal calculé par la méthode. Nous appelons cette longueur "longueurGG".

Enfin, pour obtenir la borne inférieure de notre problème global, nous devons appliquer la formule suivante, donnant le C_{max} du sous-problème :

$$BI = \bigotimes_{i=1}^{n}(\frac{\mathbb{1}}{a_i}).t_k.c^*(\nu),$$

où, nous avons t_k la date de disponibilité de la machine M_k, et en **notation usuelle** cette fois :

$$c^*(\nu) = \frac{1}{2} \times (\sum_{i=1}^{n+1}(A_i + B_i) + longueurGG)$$

Comme nous avons $\frac{m*(m-1)}{2}$ sous-problèmes, nous obtenons autant de bornes inférieures. C'est donc la meilleure borne calculée qui est retournée par la fonction *BorneInferieure*, "meilleure" désignant ici la valeur la plus élevée obtenue.

4.3.4 Améliorations

Il est possible d'améliorer les $\frac{m*(m-1)}{2}$ bornes inférieures calculées de la manière suivante :

– Johnson : On peut améliorer les dates de disponibilité des machines M_k et M_l avant de calculer le C_{max} de la séquence de Johnson et la date de fin sur la dernière machine en utilisant celles des deux machines M_k et M_l, après ordonnancement de la séquence de Johnson. En effet, l'algorithme de Johnson permet de minimiser la longueur du bloc d'exécution des jobs sur la machine M_k, il en est de même sur la machine M_l, et il permet également de minimiser la diagonale joignant t_k à D_l (correspondant à la date de fin d'exécution des jobs sur la machine M_l), ce qui équivaut à minimiser le C_{max} de notre problème global.

1. **Amélioration des dates de disponibilité initiales** : Il va de soit que les jobs de la séquence donnée par Johnson ont été exécutés sur les machines précédant les machines M_k et M_l. De ce fait, il va être possible d'améliorer les dates de disponibilité initiales de ces machines, pour décaler tout le bloc ordonnancé par Johnson.

   Pour cela, on va procéder de la manière suivante :

   Pour chaque machine M_p précédant M_k, on ajoute à sa date de disponibilité initiale t_p le minimum des sommes des durées d'exécution de chaque job sur les machines $M_p \rightarrow M_k - 1$ incluses.
   Ensuite, on prend le maximum parmi ces $k-1$ valeurs obtenues, et on obtient ainsi une amélioration de t_k, que l'on note t'_k.
   Pour améliorer t_l, on procède exactement de la même manière, sauf que l'on considère les $l-1$ premières machines.
   Une fois cette première amélioration appliquée, on calcule les dates de disponibilité suivantes des machines, c'est-à-dire après ordonnancement de la séquence de Johnson pour un flowshop min-delay.

   NB : A ce stade, la date de disponibilité finale D_l donne une minoration du C_{max} du problème initial, correspondant à la valeur optimale du C_{max} du sous-problème à deux machines traité, améliorée par l'augmentation de la date de disponibilité initiale des machines M_k et M_l.

2. **Amélioration des dates de fin** : Il est également possible d'améliorer la borne inférieure à partir d'une augmentation de la date de fin sur chacune des deux machines M_k et M_l, puisque que chacun des jobs de la séquence va venir s'exécuter sur les machines suivantes jusqu'à M_m.
   Pour améliorer le C_{max} minorant à partir de D_k, la date de disponibilité finale de la machine M_k, on calcule la somme des durées d'exécution de chaque job sur les machines qui suivent M_k.
   On récupère le minimum de ces n sommes, et on l'ajoute à D_k, donnant ainsi D'_k, que nous pouvons considérer comme étant la date de fin sur M_m améliorée par D_k.
   On fait de même pour D_l, et le C_{max} du problème initial est alors minoré par le maximum entre D'_k et D'_l.

NB : Ces améliorations sont possibles pour Johnson parce qu'il minimise les dates de fin D_k **et** D_l, ainsi que la diagonale joignant t_k et D_l.

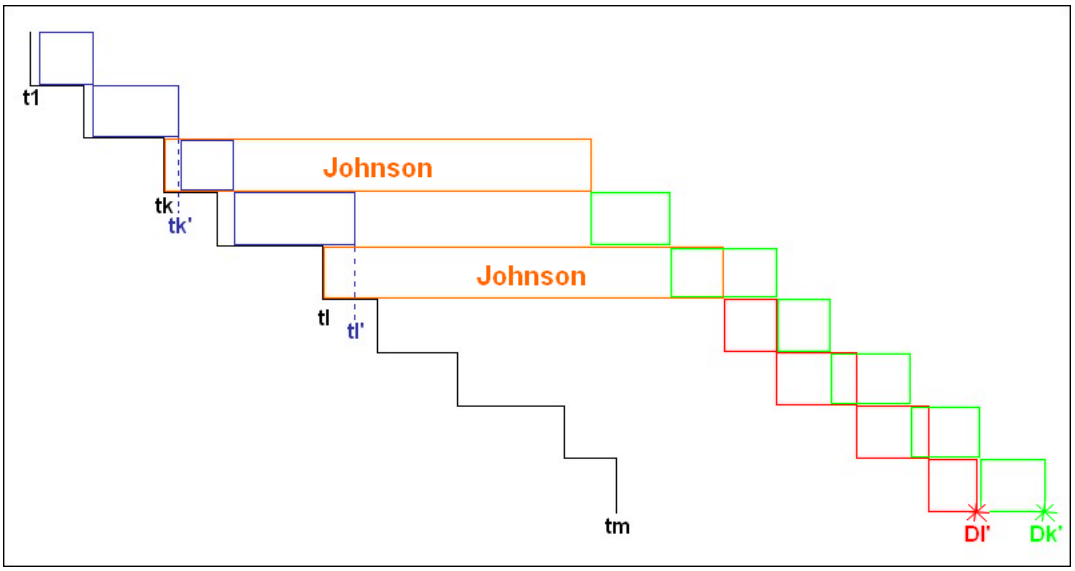


FIGURE 4.8 – Schéma explicatif de l'amélioration de la borne inférieure

– Gilmore et Gomory : On améliore la date de disponibilité t_k de la machine M_k avant de l'ajouter dans la formule pour obtenir la borne inférieure, et on améliore la date de fin sur la machine M_l uniquement.
  En effet, Gilmore et Gomory permet seulement de minimiser soit D_k, soit D_l, soit la diagonale. Comme l'utilisation que nous faisons de cet algorithme dans la PSE minimise la diagonale, c'est-à-dire le C_{max}, nous ne pouvons pas améliorer t_l ni D_k.
  NB : Les calculs sont les mêmes que pour Johnson.

Il est important de noter que l'amélioration de la date de fin des machines n'est justifiée à partir de la machine M_{k+1} ou M_{l+1} que pour la contrainte RSb. En effet, les contraintes de blocage RCb et RCb* prenant en compte à chaque instant la machine $j+2$, nous devons commencer l'amélioration à partir des machines M_{k+2} et M_{l+2}.

4.4 Adaptation de la PSE existante

Une personnalisation de la PSE fournie a été nécessaire pour plusieurs raisons :

- Une meilleure compréhension et lisibilité du code
- Une cohérence accrue entre le programme et le problème à résoudre
- Tout simplement pour pouvoir utiliser la PSE comme méthode de résolution d'un flowshop soumis à diverses contraintes de blocage pour la minimisation du C_{max}

4.4.1 Génération d'instances

Tout d'abord, la génération d'instances de la PSE initiale ne correspondait pas du tout au problème étudié dans ce PFE. En effet, les fichiers générés ne contenaient pas de contrainte spécifique, par contre ils contenaient des paramètres pour le min-delay, qui sont inutiles pour ce nouveau problème.

Du coup, j'ai modifié la génération d'instances de sorte à ce que ces dernières correspondent à mon problème.

Cette adaptation passe par la modification des paramètres passés à la commande d'exécution du programme de génération, ainsi que celle de l'écriture dans les fichiers.

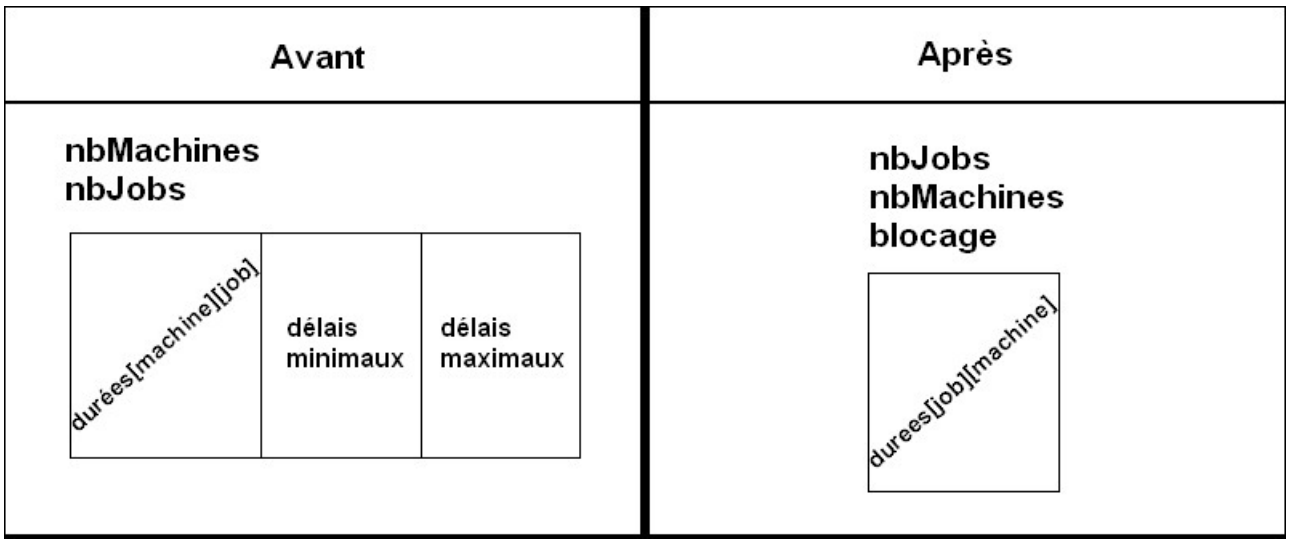


FIGURE 4.9 – Schéma des fichiers d'instance de la PSE initiale *vs* PSE modifiée

Pour connaître les paramètres à passer au programme, je vous renvoie vers la consultation de l'aide fournie par ce petit programme indépendant, qui est disponible en précisant la commande - *-help* lors du lancement, ou bien en lançant la génération sans argument.

4.4.2 Lecture des fichiers

Afin de pouvoir exploiter les données contenues dans les fichiers d'instances modifiés, il faut changer la fonction de lecture de fichier, qui s'occupe aussi de stocker les données récupérées dans les variables globales de la PSE.

Cette modification a été rapide et est simple à comprendre, je ne m'étalerais donc pas plus sur ce sujet.

4.4.3 Remise du code sous un IDE

La PSE initiale n'est pas structurée, que ce soit dans le code ou dans les fichiers qui le contiennent. En effet, les fichiers header (.h) contiennent l'implémentation de nombreuses fonctions, la fonction principale (main) est contenue dans un fichier avec bien d'autres, et il n'y a aucune séparation des fonctions par catégorie. De plus, le code n'est que peu commenté, et certaines fonctions portent un nom ne correspondant pas à leur action.

C'est pourquoi j'ai migré toute la PSE dans un nouveau projet sous Code : :Blocks, dès que cela m'a été possible.

En effet, le problème traité dans mon PFE me permet de supprimer la portion de code en Fortran utilisée dans la PSE pour résoudre un PVC (problème du voyageur de commerce), qui était la seule raison pour laquelle le code ne pouvait pas être mis sous un IDE, puisqu'il nécessitait alors un Makefile spécifique.

Cette migration est au premier abord une légère perte de temps, mais au final c'est un grand gain de temps puisqu'il est possible d'utiliser un debugger pour résoudre les problèmes d'exécution en pas à pas, plutôt que d'utiliser des *printf()* qui sont peu précis et avec lesquels le débogage reste laborieux.

4.4.4 Restructuration du code

En créant un projet Code : :Blocks, j'ai pu restructurer tout le code de la PSE, en retirant ce qui m'était inutile et en séparant correctement les fichiers source des fichiers header. J'ai également isolé la fonction *main()* dans un fichier source particulier.

Le code est maintenant cohérent, aéré, commenté et pratique.

Cette restructuration du code permet également, via l'IDE Code : :Blocks, d'accéder aux diverses fonctions du programme de manière très aisée : une liste déroulante affiche toutes les fonctions contenues dans le fichier source ouvert, et un clic droit sur le nom d'une fonction permet d'accéder à l'implémentation de la méthode, quelque soit le fichier du projet dans lequel elle se trouve.

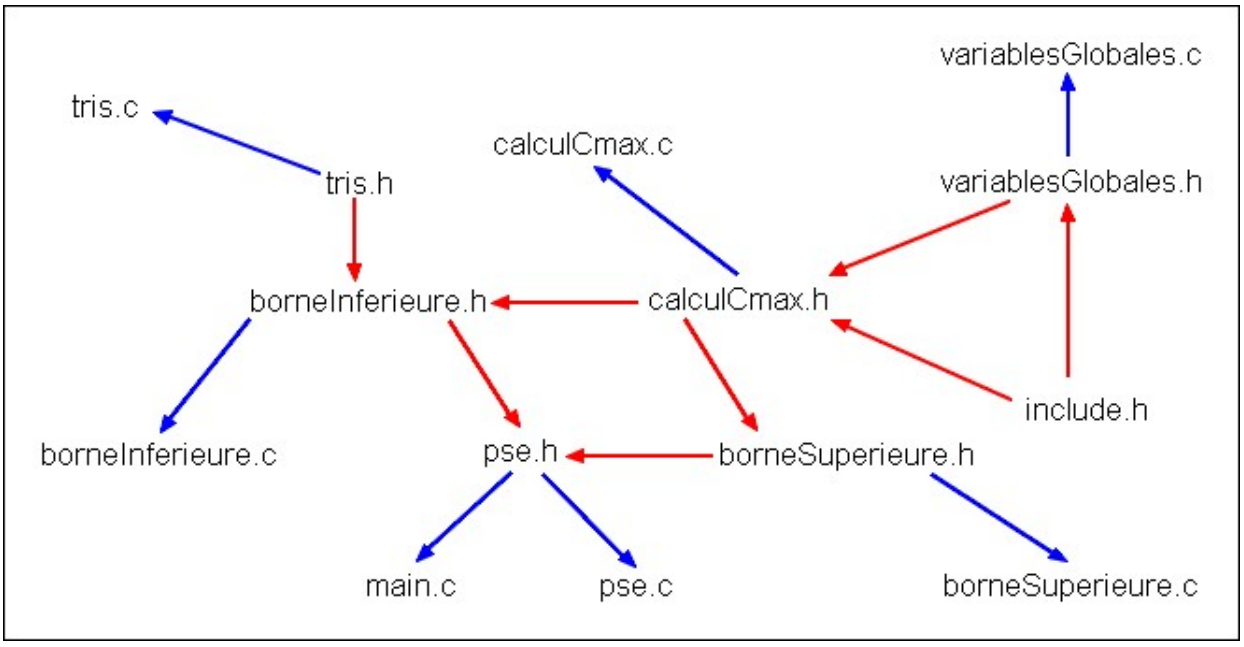


FIGURE 4.10 – Schéma de la structure du code

4.4.5 Renommage de certaines variables et fonctions

Enfin, pour rester cohérente avec le cahier de spécifications rédigé pour ce projet, j'ai renommé un bon nombre de variables et fonctions afin de respecter un maximum les conventions de nommage que je m'étais fixées.

Il reste toutefois des noms ne respectant pas cette convention initiale, mais ils n'interfèrent ni dans la compréhension, ni dans la lisibilité du code.

4.4.6 Comparaison avec la force brute

Afin de pouvoir, dans un premier temps, vérifier la justesse de ma PSE, puis de tester sa puissance, j'ai rajouté un argument permettant de créer un fichier CSV dans lequel sont contenus le C_{max} optimal de la force brute et celui de la PSE, ainsi que la durée d'exécution de chaque méthode, et quelques autres données supplémentaires.

L'argument à passer à la commande pour générer ce fichier de comparaison est **-c cheminDossier-Comparaison**.

Voici un extrait du fichier généré après application de la PSE sur un dossier contenant 10 instances de 9 jobs et 9 machines, pour la contrainte RSb, en utilisant la première méthode de borne inférieure et la première méthode de borne supérieure.

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	9	155	155
fd_9	9	9	RSb	1	9	167	167
fd_2	9	9	RSb	1	9	153	153
fd_1	9	9	RSb	1	9	156	156
fd_8	9	9	RSb	1	9	164	164
fd_10	9	9	RSb	1	9	158	158
fd_7	9	9	RSb	1	9	168	168
fd_5	9	9	RSb	1	9	165	165
fd_4	9	9	RSb	1	9	170	170
fd_6	9	9	RSb	1	9	169	169

FIGURE 4.11 – Extrait d'un fichier de comparaison généré par la PSE - 1ère partie

duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
3.800000	0.000000	131	0	2	0
3.450000	2.060000	144	5091	1	24
3.870000	3.780000	137	8990	105	23
3.450000	0.790000	135	1545	0	16
3.610000	0.350000	151	624	0	21
3.390000	0.200000	145	300	221	13
3.480000	1.880000	160	3748	1	13
3.390000	0.240000	147	338	422	8
3.580000	2.240000	158	3979	0	18
3.420000	1.280000	157	2792	0	12

FIGURE 4.12 – Extrait d'un fichier de comparaison généré par la PSE - 2ème partie

4.5 Lancement de la PSE

La PSE prend plusieurs arguments en paramètres. Si on lance la PSE via un invite de commandes sans préciser de paramètre, ou bien en indiquant - *-help*, l'aide est affichée. Cette aide correspond à l'énumération des arguments nécessaires et optionnels pour l'exécution du programme.

Nous retrouvons ainsi les arguments suivants, qui sont les plus pertinents dans le cadre de ce projet :

- *-f nomFichier* : lancement de la PSE sur un fichier d'instance
- *-m nomDossier* : lancement de la PSE sur un ensemble de fichiers d'instances, contenus dans le dossier spécifié

  Au moins un des arguments précédents est nécessaire pour que la PSE puisse être exécutée ; les arguments qui suivent sont optionnels.

- *-l nomDossierLog* : cet argument permet d'enregistrer des logs dans un fichier créé automatiquement dans le dossier spécifié. Ces logs correspondent à divers affichages permettant de suivre avec précision le déroulement de la PSE et la construction de l'arbre des solutions.
- *-c nomDossierComparaison* : cet argument permet d'indiquer au programme que l'on souhaite comparer les performances et les résultats de la PSE avec ceux de la force brute. Un fichier de comparaison est alors créé automatiquement dans le dossier passé en paramètre.

NB : Attention, si un fichier de logs existe déjà sous le même nom que celui qui va être créé dans le dossier donné en paramètre, alors il sera écrasé. Si un fichier de comparaison du même nom que celui créé existe déjà, alors les résultats seront ajoutés à la suite du contenu initial du fichier.

Sous Code : :Blocks, pour spécifier des arguments au programme, il faut aller dans "Project" > "Set programs' arguments...", et inscrire les commandes voulues parmi celles citées précédemment.

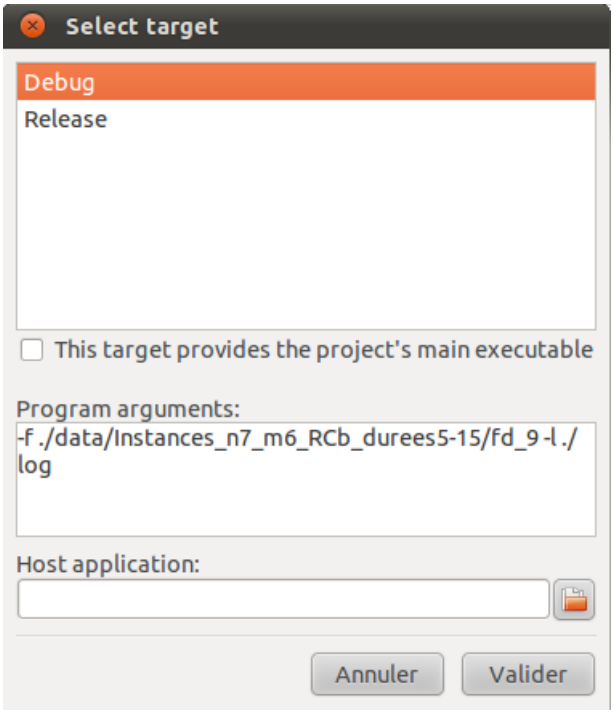


FIGURE 4.13 – Spécifier des arguments à un projet Code : :Blocks

NB : Pour la génération d'instances, qui nécessite également certains arguments, vous aurez toutes les indications nécessaires grâce à l'aide fournie, obtenue en ne passant aucun paramètre au programme ou bien en tapant - *-help*.

Il est également important de noter que les méthodes utilisées pour le calcul des bornes sont à renseigner directement dans le fichier "VariablesGlobales.c". Une amélioration simple à mettre en place serait de rajouter un paramètre à prendre en compte lors du lancement de la PSE pour renseigner automatiquement la méthode voulue, et donner une valeur par défaut en cas de non saisie de l'argument.

4.6 Résultats et tests de validité

4.6.1 Performances et résultats

En lançant la PSE sur diverses instances, grâce au mode de lancement pour des fichiers multiples, j'ai pu constater que, mises à part certaines instances qui sortent du lot, la PSE est bien plus rapide que la force brute.

Notamment, les étudiants en charge du projet de GPF m'ont informée que leur force brute commençait à devenir relativement longue dès 11 jobs, ce qui n'est pas le cas de la PSE.

Toutefois, la PSE commence à rencontrer des problèmes d'allocations mémoire dès $n = 12$ jobs et $m = 6$ machines. Les capacités mémoire sont limitées par l'utilisation d'une machine virtuelle relativement âgée et quelque peu mal configurée, il serait donc bon de migrer le projet sur une machine Ubuntu physique.

J'ai essayé de faire cette migration, mais la différence de version du compilateur GCC étant significative, des erreurs de segmentation surviennent en plus grand nombre à cause des fuites mémoire qui existent, et je n'ai pas eu le temps de me pencher plus sur ces problèmes, ni même de lancer Valgrind sur le projet.

Quoiqu'il en soit, la PSE a prouvé ses performances même sur de petites instances, en divisant les durées d'exécution de la force brute par 3, et bien plus encore sur de grandes instances.

Voici quelques extraits de fichiers de comparaison obtenus, pour chaque contrainte et vous en trouverez d'autres en annexe :

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	9	155	155
fd_9	9	9	RSb	1	9	167	167
fd_2	9	9	RSb	1	9	153	153
fd_1	9	9	RSb	1	9	156	156
fd_8	9	9	RSb	1	9	164	164
fd_10	9	9	RSb	1	9	158	158
fd_7	9	9	RSb	1	9	168	168
fd_5	9	9	RSb	1	9	165	165
fd_4	9	9	RSb	1	9	170	170
fd_6	9	9	RSb	1	9	169	169

FIGURE 4.14 – Extrait d'un fichier de comparaison pour la contrainte RSb - 1ère partie

duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
3.800000	0.000000	131	0	2	0
3.450000	2.060000	144	5091	1	24
3.870000	3.780000	137	8990	105	23
3.450000	0.790000	135	1545	0	16
3.610000	0.350000	151	624	0	21
3.390000	0.200000	145	300	221	13
3.480000	1.880000	160	3748	1	13
3.390000	0.240000	147	338	422	8
3.580000	2.240000	158	3979	0	18
3.420000	1.280000	157	2792	0	12

FIGURE 4.15 – Extrait d'un fichier de comparaison pour la contrainte RSb - 2ème partie

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	1	277	277
fd_9	9	9	RCb	1	1	319	319
fd_2	9	9	RCb	1	1	319	319
fd_1	9	9	RCb	1	1	297	297
fd_8	9	9	RCb	1	1	309	309
fd_10	9	9	RCb	1	1	303	303
fd_7	9	9	RCb	1	1	316	316
fd_5	9	9	RCb	1	1	332	332
fd_4	9	9	RCb	1	1	267	267
fd_6	9	9	RCb	1	1	295	295

FIGURE 4.16 – Extrait d'un fichier de comparaison pour la contrainte RCb - 1ère partie

duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
3.390000	0.000000	233	0	1765	0
4.050000	3.080000	253	11089	1195	44
3.390000	2.580000	268	9756	370	66
3.420000	0.290000	226	1178	1032	51
3.850000	4.050000	247	10973	1063	71
3.410000	1.880000	247	6040	350	62
3.860000	0.470000	270	1481	3529	56
3.410000	12.270000	271	48917	538	46
3.570000	1.470000	219	5238	425	61
3.450000	1.080000	243	4641	2855	48

FIGURE 4.17 – Extrait d'un fichier de comparaison pour la contrainte RCb - 2ème partie

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	7	357	357
fd_9	9	9	RCb*	1	7	355	355
fd_2	9	9	RCb*	1	7	369	369
fd_1	9	9	RCb*	1	7	345	345
fd_8	9	9	RCb*	1	7	404	404
fd_10	9	9	RCb*	1	7	372	372
fd_7	9	9	RCb*	1	7	366	366
fd_5	9	9	RCb*	1	7	353	353
fd_4	9	9	RCb*	1	7	367	367
fd_6	9	9	RCb*	1	7	398	398

FIGURE 4.18 – Extrait d'un fichier de comparaison pour la contrainte RCb* - 1ère partie

duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
3.260000	0.000000	281	0	0	0
3.240000	1.490000	264	2395	0	76
4.190000	0.870000	270	1433	658	91
3.300000	0.000000	251	0	9333	0
3.310000	0.000000	318	0	0	0
3.240000	0.000000	295	0	87	0
3.300000	0.000000	279	0	8238	0
3.320000	0.000000	290	0	1	0
3.340000	0.000000	280	0	176	0
3.250000	0.000000	303	0	401	0

FIGURE 4.19 – Extrait d'un fichier de comparaison pour la contrainte RCb* - 2ème partie

4.6.2 Tests de validité

Afin de tester la justesse des résultats retournés par la PSE, j'ai utilisé la force brute du programme de GPF, et dans les fichiers de comparaison générés, j'ai pu constater rapidement s'il y avait des différences entre les deux valeurs données.

Lors de mes tests sur de nombreuses instances, j'ai été confrontée plusieurs fois à des différences entre le C_{max} retourné par la PSE, et la valeur optimale réelle, donnée par la force brute. Ensuite, en utilisant le débugger de Code : :Blocks sur l'une des instances donnant un écart, je pouvais déterminer l'endroit exact d'où venait le problème, et ainsi le résoudre.

Dans ces expérimentations, il s'est avéré que la borne inférieure donnée par la méthode de Gilmore et Gomory venait dépasser le C_{max} optimal, et fausser ainsi le résultat final.

Pour tester un peu plus en profondeur Gilmore et Gomory, j'ai repris le code de ma PSE et je l'ai adapté de sorte à ce qu'il résolve un $F2|a - wait|C_{max}$. En appliquant également la méthode de Gilmore et Gomory, je pouvais ainsi comparer les deux résultats et juger des incohérences.

C'est ce second programme qui m'a permis de mettre le doigt sur un problème dans l'algorithme de tri utilisé : l'introsort, puisqu'avec l'utilisation d'un simple tri à bulle le problème était résolu.

Au final, j'ai une PSE fonctionnelle, donnant des résultats corrects et en un temps réduit.

Bilan

5.1 Comparaison avec le planning prévisionnel

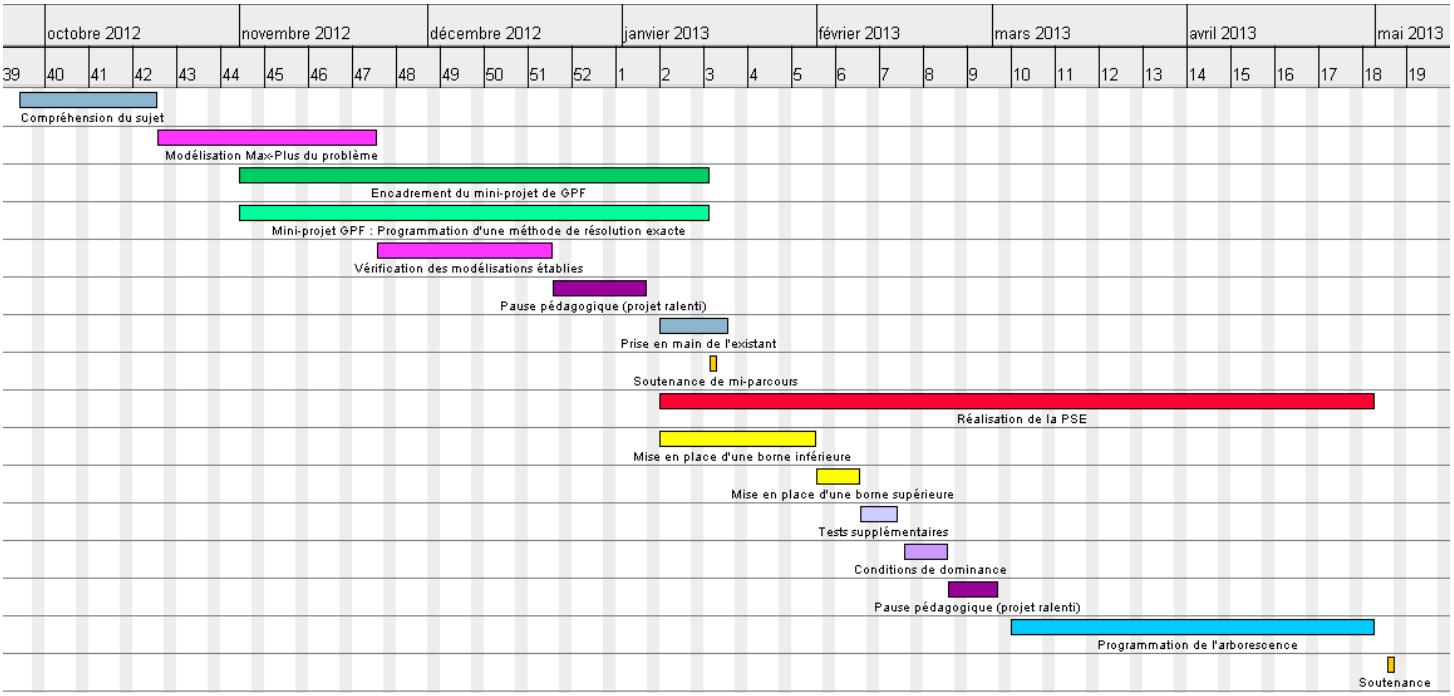


FIGURE 5.1 – Planning initial

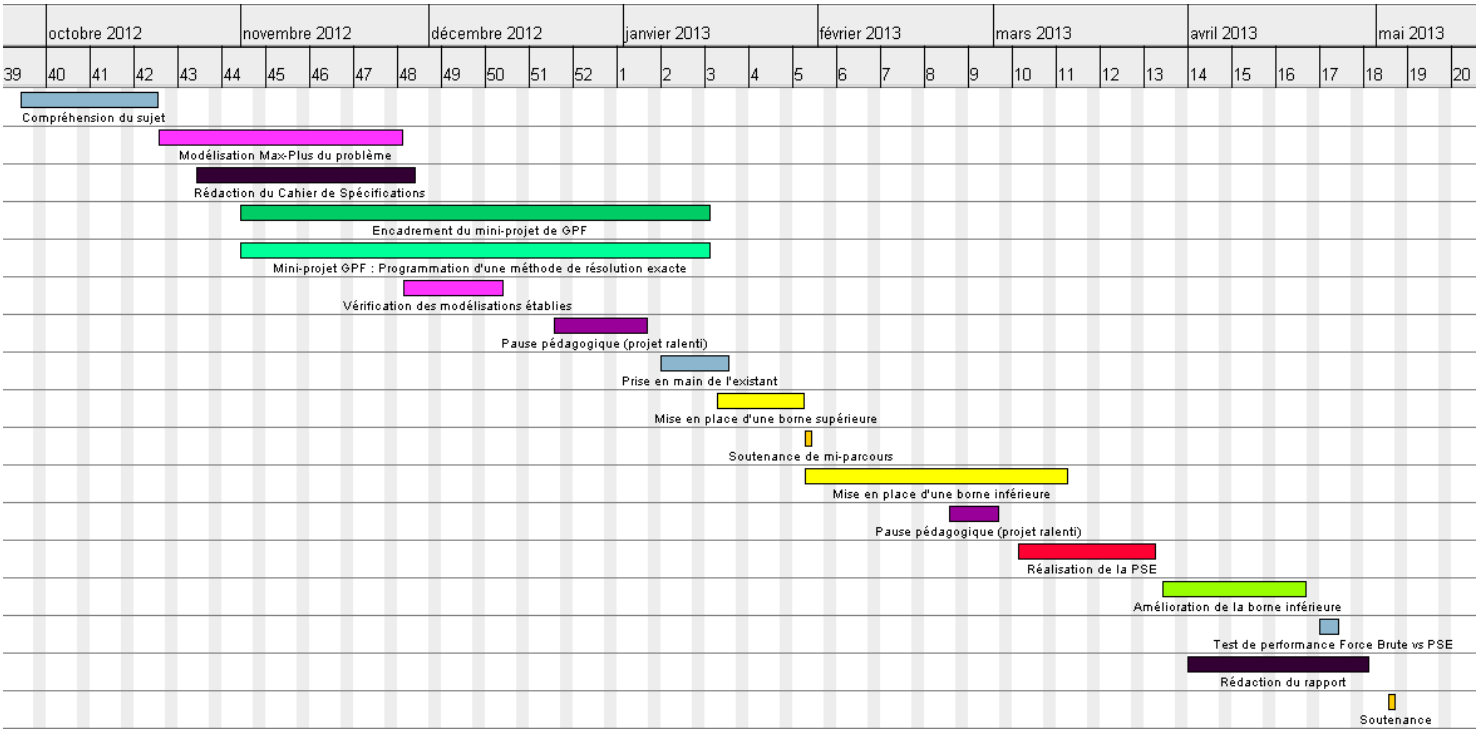


FIGURE 5.2 – Planning Réel

Les deux diagrammes de Gantt précédents montrent les plannings prévisionnel et réel du projet. Ils ne sont que peu lisibles (zoomez pour lire les écritures), mais on peut aisément constater que le planning prévisionnel a été quasiment respecté, hormis une inversion dans la définition d'une borne inférieure et d'une borne supérieure. En effet, il était prévu de commencer par la borne inférieure, mais dans la pratique, j'ai d'abord mis en place différentes méthodes de calcul d'une borne supérieure.

Dans sa globalité, l'avancement du projet a été lisse et en accord avec les prévisions. Il faut toutefois souligner que le temps consacré à l'établissement de conditions de dominance à été entièrement consumé et utilisé pour la gestion des difficultés, il a donc été réparti dans les diverses autres tâches.

5.2 Gestion des difficultés

Tout au long de ce projet, j'ai été confrontée à diverses difficultés qu'il m'a fallut surmonter, certaines de ces difficultés étant plus bloquantes que d'autres.

5.2.1 Mise en place des modélisations

La première difficulté rencontrée dans ce projet a été lors de l'établissement des modélisations Max-Plus du problème. En effet, dès que je pensais toucher du bout du doigt la bonne modélisation, il s'avérait qu'elle n'était pas correcte. J'ai donc dû à plusieurs reprises reprendre tout depuis le début pour repartir sur de bonnes bases, et cela faisait monter en moi une impression de "tourner en boucle", assez déplaisante.

Toutefois, il ne m'a pas fallu beaucoup de tentatives pour trouver la modélisation adéquate et pouvoir ainsi poursuivre sur les tests et la mise en place des bornes.

5.2.2 Compréhension de la PSE

La compréhension de la PSE a été compliquée par sa pauvreté en commentaires, et sa mauvaise structure du code. Nous avons décortiqué avec monsieur LENTÉ à deux reprises la PSE dans sa totalité, afin de comprendre son déroulement, l'utilité des variables et de cibler les parties à reprendre et à adapter au nouveau problème.

5.2.3 Mise en place d'une borne inférieure

Cette étape a été la plus difficile à mener, et les bogues rencontrés étaient des plus pénalisants.

En effet, la mise en place d'une borne inférieure m'a demandé d'implémenter les méthodes de Johnson et de Gilmore et Gomory, spécialement pour des problèmes à deux machines.

La méthode de Johnson ne m'a pas posé de problème, étant relativement simple à comprendre, à appliquer et à programmer.

Par contre, l'algorithme de Gilmore et Gomory a été à la fois compliqué à comprendre, à dérouler et à implémenter. Monsieur LENTÉ a été très présent dans la première phase de compréhension, ce qui m'a permis de partir assez rapidement sur la programmation. L'implémentation de cet algorithme m'a parut particulièrement longue puisqu'il me fallait comprendre de manière plus poussée le déroulement de l'algorithme, et également trouver comment le transcrire en langage de programmation. De plus, cette méthode utilise des tris, que j'ai voulu le moins complexe possible. J'ai donc choisi d'implémenter une méthode de tri en O(n.log(n)), qui s'appelle l'introspective sort, ou introsort.

Toutefois, ce tri m'a posé problème, et c'est dans la fonction mettant en oeuvre Gilmore et Gomory que les erreurs survenaient, notamment lors des tests de la PSE, mettant en jeu diverses instances. J'ai repris à 2 fois l'algorithme de tri sans trouver le problème final, et j'ai fini au bout du compte par utiliser un simple tri à bulle, peu performant, mais ayant le mérite d'être juste !

5.2.4 Amélioration de la borne inférieure

L'amélioration de la borne inférieure a également été subtile à mettre en place puisque j'avais par moment une borne inférieure améliorée qui venait dépasser mon optimum. Du coup, je ne trouvais pas le C_{max} optimal réel, mais une valeur plus ou moins surélevée.

Le débogage de cette portion de code a demandé de la patience puisqu'il a fallu principalement faire du pas à pas, et vérifier la théorie autant que la mise en pratique.

Jusqu'à la dernière semaine de projet, l'amélioration de la borne inférieure donnée par Gilmore et Gomory semblait obsolète (celle de Johnson était correcte). En effet, dans le cas des contraintes RCb et RCb* notamment, l'amélioration de la date de fin sur la machine M_m à partir de la date de libération de la machine M_l semblait ne pas être justifiée car en s'ajoutant au C_{max} minorant, ce dernier vennait dépasser la valeur de l'optimum. Dans le cas de la contrainte RSb, sur les instances testées, ce problème restait transparent : il ne modifie pas suffisamment la borne inférieure pour la rendre supérieure à l'optimum.

Pour tenter de comprendre et de résoudre ce problème, M. LENTÉ et moi même avons d'abord vérifié les valeurs de chaque donnée utilisée pour le calcul du C_{max} minorant, et testé la justesse de ce C_{max}, d'un flowshop de type a-wait, de deux manières différentes :

A partir de l'instance optimale donnée par la PSE de résolution d'un flowshop de type a-wait, nous avons retrouvé le C_{max} optimal en considérant le problème a-wait et en nous ramenant à un flowshop de type no-wait.

Dans les deux cas nous avons obtenu des résultats en accord avec ceux de la PSE.

Le passage d'un flowshop de type a-wait en un no-wait se fait simplement, pour une matrice 2x2, en mettant le délai en facteur de la matrice.

$$\begin{pmatrix} P_1 & P_1.a.P_2 \\ \frac{1}{a} & P_2 \end{pmatrix} \Leftrightarrow \frac{1}{a}.\begin{pmatrix} P_1.a & P_1.a^2.P_2 \\ \mathbb{1} & P_2.a \end{pmatrix}$$

Il s'avèrait donc que l'amélioration de la borne inférieure donnée par Gilmore et Gomory n'était pas correcte d'un point de vue théorique, puisque du côté pratique tout était bon.

Cette supposition d'erreur sur les formules d'amélioration du C_{max} d'un flowshop de type a-wait ne remettait pas en question la méthode de calcul d'une borne inférieure non-améliorée. Toutefois, même si l'amélioration semblait fonctionner pour la contrainte RSb, nous soupçonnions qu'elle soit fausse puisque la théorie sous-jacente n'était elle même pas correcte.

En fin de compte, l'amélioration de la borne inférieure est correcte telle quelle pour la contrainte RSb uniquement, et comme les contraintes RCb et RCb* prennent en compte la machine $j+2$ à tout instant, l'amélioration de la date de fin à partir des dates de disponibilité des machines M_k et M_l devient fausse. En effet, cette amélioration prend en compte les durées d'exécution des jobs à partir des machines M_{k+1} et M_{l+1}, donc dans le cas des contraintes RCb et RCb*, on prend alors deux fois en compte les durées des jobs sur ces deux machines. Il faut donc commencer l'amélioration à partir des machines M_{k+2} et M_{l+2} pour les contraintes RCb et RCb*.

La justification de cette affirmation n'a pas été établie à ce jour, mais elle a été validée par expérimentation.

5.2.5 Intégration du code dans la PSE

Tout le code que j'ai produit a été réalisé et testé sous Code : :Blocks sous Windows, puis sous Linux, pour enfin être inséré dans la PSE.

Les problèmes rencontrés ont tous été du même type : des erreurs d'indices dans la manipulation des divers tableaux et matrices. En effet, dans la PSE, on appelle les fonctions "rapportées" en un noeud de l'arbre, en utilisant la séquence restante et les dates de disponibilités des machines données par l'ordonnancement de la séquence partielle.

De ce fait, la séquence restante contient des indices allant de 0 à *nbjobs*, *nbjobs* étant une variable globale contenant le nombre de jobs de l'instance. Toutefois, les fonctions que j'ai réalisées prennent en paramètres, la plupart du temps, une séquence et un nombre de jobs donnant la taille de cette séquence.

Le problème est là : la taille de la séquence restante ne correspond pas à l'amplitude des indices qu'elle contient, donc lorsque j'utilise les éléments de la séquence dans mes fonctions pour accéder à d'autres données telles que la durée d'exécution d'un job sur une machine, je vais taper en dehors de la zone mémoire voulue.

Dans ma fonction je considérais que les éléments du tableau donnant la séquence étaient compris entre 0 et la taille du tableau exclue. Cette hypothèse devenant fausse dans la PSE, j'ai dû adapter mon code à plusieurs endroits, et parfois même gérer des re-numérotations.

Par exemple, si mon instance étudiée possède 5 jobs et que j'en arrive à un noeud ayant comme séquence partielle 0, 2, ma séquence restante contient les jobs 1, 3, 4 mais est de taille 3. Si je veux accéder au dernier élément de ma séquence, à l'indice 2, j'obtiens 4, ce qui ne correspond pas du tout à une valeur comprise entre 0 et 2, et fausse ainsi toute utilisation auxiliaire.

5.3 Le projet dans le futur

Les évolutions envisageables pour ce projet sont diverses et variées. Donnons simplement quelques exemples de projets possibles à soumettre aux futurs étudiants en 5ème année, dans la continuité de ce PFE.

Un projet très intéressant à mener pourrait porter sur l'étude de contraintes mixtes de blocage. C'est à dire que des contraintes de blocage différentes peuvent intervenir entre les divers couples de machines.

Un tel projet demanderait de revoir les modélisations établies, ou plus exactement de les affiner pour essayer d'établir un modèle commun aux différents mélanges possibles avec les contraintes RSb, RCb et RCb*. Il serait également nécessaire de revoir les méthodes de calcul d'une borne inférieure, afin d'en trouver une qui soit efficace sur ce problème spécifique.

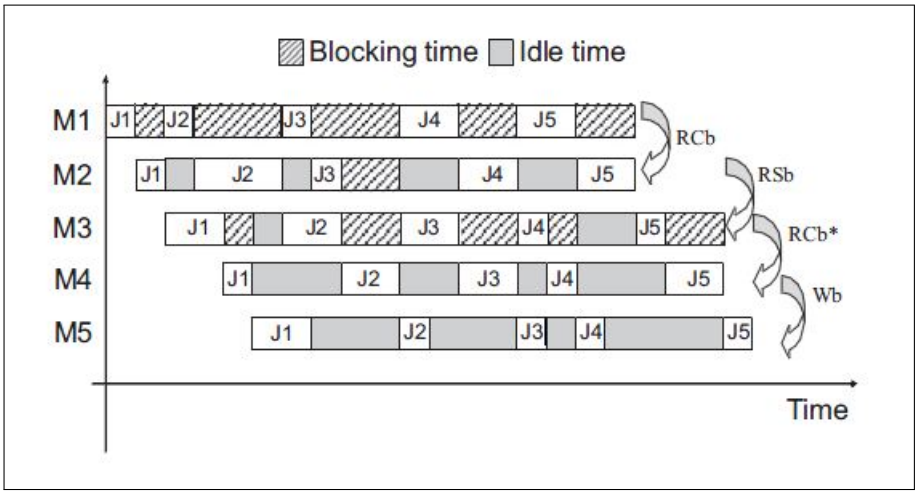


FIGURE 5.3 – Schéma d'illustration de contraintes mixtes – issu de l'article "Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems"

Une autre poursuite possible serait tout simplement d'améliorer le code, de l'optimiser au mieux, et de trouver de nouvelles bornes inférieures et supérieures pouvant être encore plus efficaces que celles mises en place actuellement, et réduire davantage la durée de résolution d'un flowshop soumis aux contraintes de blocage RSb, RCb et RCb*.

Conclusion

Ce projet de fin d'étude étant mon premier choix dans la liste des sujets proposés, mon intérêt en est d'autant plus prononcé. De plus, il s'inscrit dans le secteur de la recherche qui m'attire tout particulièrement, sans pour autant que je sache pour quelles raisons, puisque ce domaine m'était jusqu'alors inconnu.

Le fait de mener une étude complète sur un sujet nouveau donne un attrait supplémentaire à ce projet.

J'ai dû m'organiser afin de découvrir le problème en question et d'être en mesure de le traiter : trouver des documents existants, les étudier et les comprendre afin de pouvoir utiliser leurs concepts clés.

Une seule chose est à déplorer : que l'application contienne encore des fuites mémoire venant interrompre parfois l'exécution de la PSE sur plusieurs instances, obligeant alors de relancer le programme sur les fichiers ainsi délaissés.

Je suis toutefois pleinement ravie du travail réalisé. En effet, être partie de presque rien et avoir au final une application qui tourne, et résolve un problème d'ordonnancement nouveau de manière efficace, est une grande satisfaction.

Enfin, la PSE réalisée pourra évoluer de diverses manières, pour s'orienter soit vers une optimisation du code (gestion de la mémoire, optimisation des algorithmes, amélioration des bornes), soit vers un élargissement des problèmes possibles à résoudre (contraintes mixtes de blocage).

Exemples d'application des contraintes de blocage

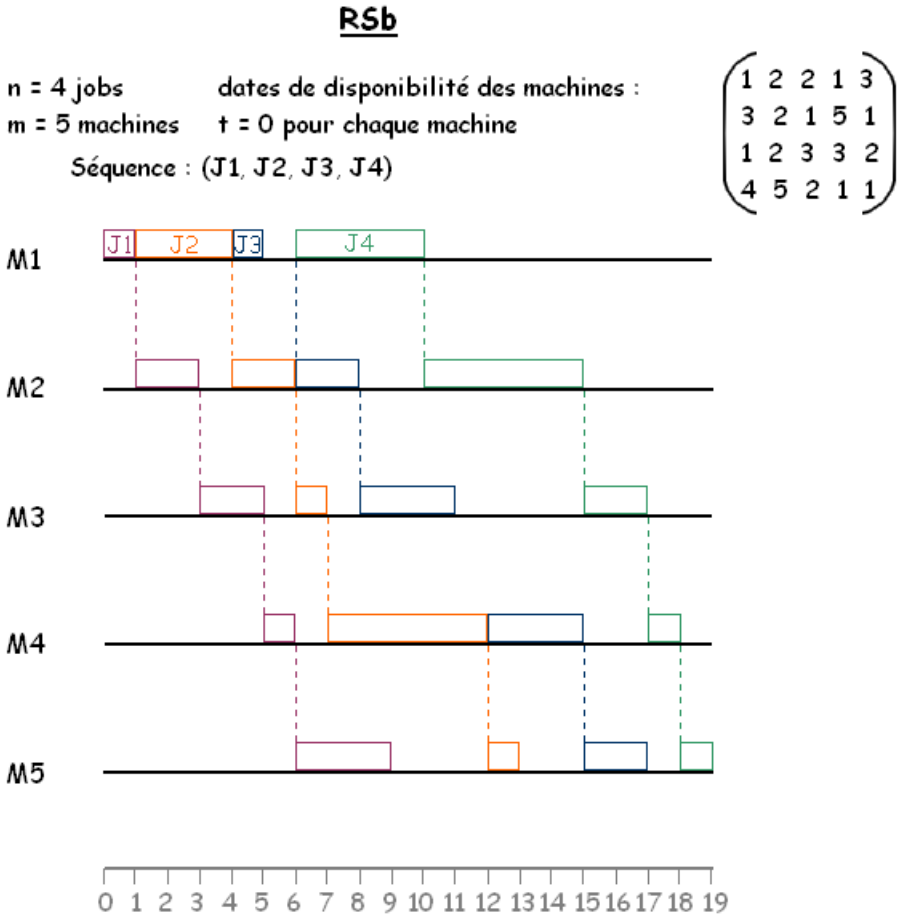


FIGURE A.1 – Exemple d'application de la contrainte RSb

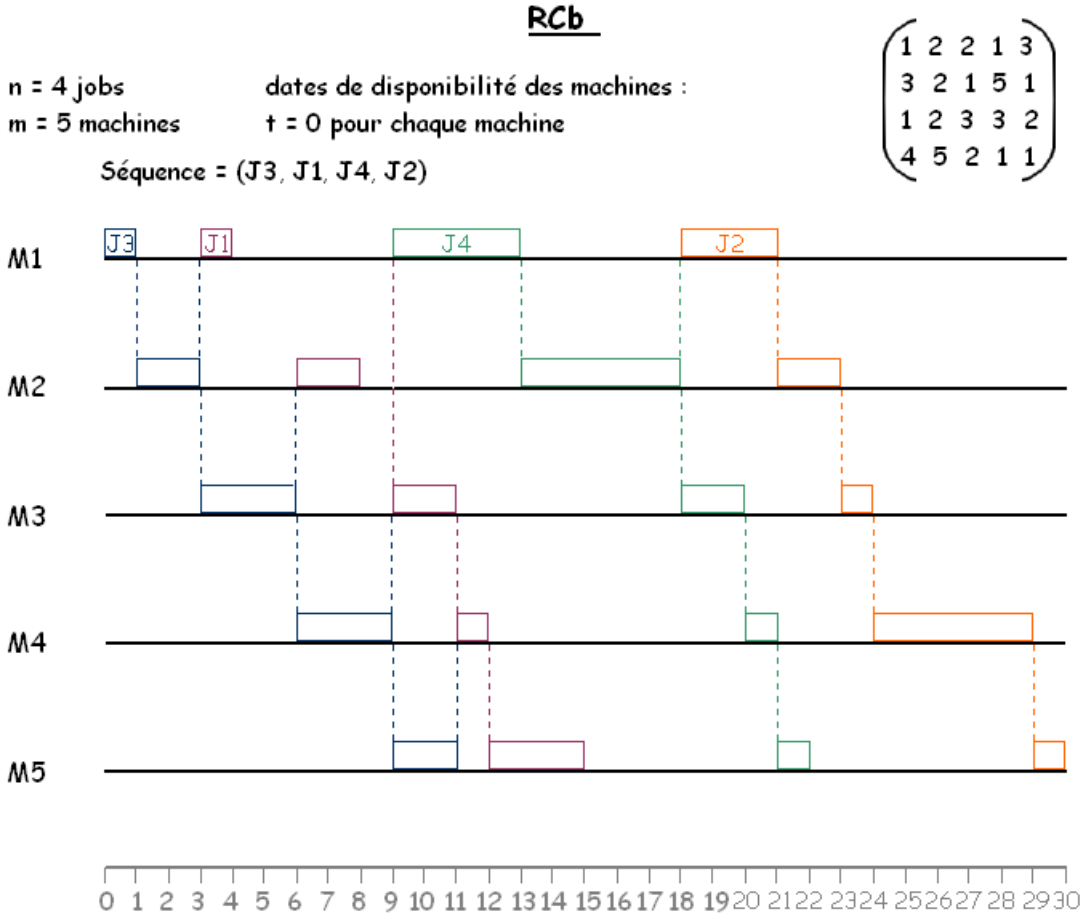


FIGURE A.2 – Exemple d'application de la contrainte RCb

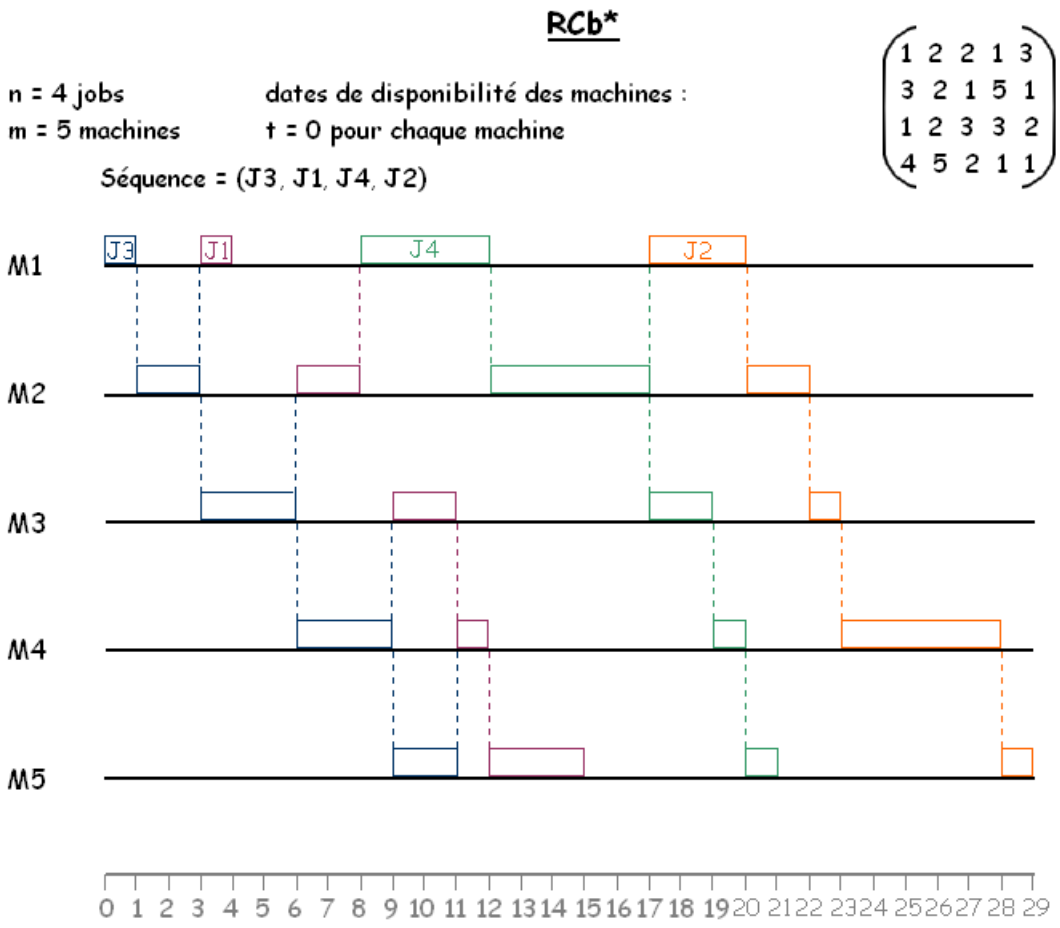


FIGURE A.3 – Exemple d'application de la contrainte RCb*

Fichiers de comparaison des performances de la PSE

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	1	155	155
fd_9	9	9	RSb	1	1	167	167
fd_2	9	9	RSb	1	1	153	153
fd_1	9	9	RSb	1	1	156	156
fd_8	9	9	RSb	1	1	164	164
fd_10	9	9	RSb	1	1	158	158
fd_7	9	9	RSb	1	1	168	168
fd_5	9	9	RSb	1	1	165	165
fd_4	9	9	RSb	1	1	170	170
fd_6	9	9	RSb	1	1	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	2	155	155
fd_9	9	9	RSb	1	2	167	167
fd_2	9	9	RSb	1	2	153	153
fd_1	9	9	RSb	1	2	156	156
fd_8	9	9	RSb	1	2	164	164
fd_10	9	9	RSb	1	2	158	158
fd_7	9	9	RSb	1	2	168	168
fd_5	9	9	RSb	1	2	165	165
fd_4	9	9	RSb	1	2	170	170
fd_6	9	9	RSb	1	2	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	3	155	155
fd_9	9	9	RSb	1	3	167	167
fd_2	9	9	RSb	1	3	153	153
fd_1	9	9	RSb	1	3	156	156
fd_8	9	9	RSb	1	3	164	164
fd_10	9	9	RSb	1	3	158	158
fd_7	9	9	RSb	1	3	168	168
fd_5	9	9	RSb	1	3	165	165
fd_4	9	9	RSb	1	3	170	170
fd_6	9	9	RSb	1	3	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	4	155	155
fd_9	9	9	RSb	1	4	167	167
fd_2	9	9	RSb	1	4	153	153
fd_1	9	9	RSb	1	4	156	156
fd_8	9	9	RSb	1	4	164	164
fd_10	9	9	RSb	1	4	158	158
fd_7	9	9	RSb	1	4	168	168
fd_5	9	9	RSb	1	4	165	165
fd_4	9	9	RSb	1	4	170	170
fd_6	9	9	RSb	1	4	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	5	155	155
fd_9	9	9	RSb	1	5	167	167
fd_2	9	9	RSb	1	5	153	153
fd_1	9	9	RSb	1	5	156	156
fd_8	9	9	RSb	1	5	164	164
fd_10	9	9	RSb	1	5	158	158
fd_7	9	9	RSb	1	5	168	168
fd_5	9	9	RSb	1	5	165	165

fd_4	9	9	RSb	1	5	170	170
fd_6	9	9	RSb	1	5	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	6	155	155
fd_9	9	9	RSb	1	6	167	167
fd_2	9	9	RSb	1	6	153	153
fd_1	9	9	RSb	1	6	156	156
fd_8	9	9	RSb	1	6	164	164
fd_10	9	9	RSb	1	6	158	158
fd_7	9	9	RSb	1	6	168	168
fd_5	9	9	RSb	1	6	165	165
fd_4	9	9	RSb	1	6	170	170
fd_6	9	9	RSb	1	6	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	7	155	155
fd_9	9	9	RSb	1	7	167	167
fd_2	9	9	RSb	1	7	153	153
fd_1	9	9	RSb	1	7	156	156
fd_8	9	9	RSb	1	7	164	164
fd_10	9	9	RSb	1	7	158	158
fd_7	9	9	RSb	1	7	168	168
fd_5	9	9	RSb	1	7	165	165
fd_4	9	9	RSb	1	7	170	170
fd_6	9	9	RSb	1	7	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	8	155	155
fd_9	9	9	RSb	1	8	167	167
fd_2	9	9	RSb	1	8	153	153
fd_1	9	9	RSb	1	8	156	156
fd_8	9	9	RSb	1	8	164	164
fd_10	9	9	RSb	1	8	158	158
fd_7	9	9	RSb	1	8	168	168
fd_5	9	9	RSb	1	8	165	165
fd_4	9	9	RSb	1	8	170	170
fd_6	9	9	RSb	1	8	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	1	9	155	155
fd_9	9	9	RSb	1	9	167	167
fd_2	9	9	RSb	1	9	153	153
fd_1	9	9	RSb	1	9	156	156
fd_8	9	9	RSb	1	9	164	164
fd_10	9	9	RSb	1	9	158	158
fd_7	9	9	RSb	1	9	168	168
fd_5	9	9	RSb	1	9	165	165
fd_4	9	9	RSb	1	9	170	170
fd_6	9	9	RSb	1	9	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	1	155	155
fd_9	9	9	RSb	2	1	167	167
fd_2	9	9	RSb	2	1	153	153
fd_1	9	9	RSb	2	1	156	156
fd_8	9	9	RSb	2	1	164	164
fd_10	9	9	RSb	2	1	158	158

fd_7	9	9	RSb	2	1	168	168
fd_5	9	9	RSb	2	1	165	165
fd_4	9	9	RSb	2	1	170	170
fd_6	9	9	RSb	2	1	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	2	155	155
fd_9	9	9	RSb	2	2	167	167
fd_2	9	9	RSb	2	2	153	153
fd_1	9	9	RSb	2	2	156	156
fd_8	9	9	RSb	2	2	164	164
fd_10	9	9	RSb	2	2	158	158
fd_7	9	9	RSb	2	2	168	168
fd_5	9	9	RSb	2	2	165	165
fd_4	9	9	RSb	2	2	170	170
fd_6	9	9	RSb	2	2	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	3	155	155
fd_9	9	9	RSb	2	3	167	167
fd_2	9	9	RSb	2	3	153	153
fd_1	9	9	RSb	2	3	156	156
fd_8	9	9	RSb	2	3	164	164
fd_10	9	9	RSb	2	3	158	158
fd_7	9	9	RSb	2	3	168	168
fd_5	9	9	RSb	2	3	165	165
fd_4	9	9	RSb	2	3	170	170
fd_6	9	9	RSb	2	3	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	4	155	155
fd_9	9	9	RSb	2	4	167	167
fd_2	9	9	RSb	2	4	153	153
fd_1	9	9	RSb	2	4	156	156
fd_8	9	9	RSb	2	4	164	164
fd_10	9	9	RSb	2	4	158	158
fd_7	9	9	RSb	2	4	168	168
fd_5	9	9	RSb	2	4	165	165
fd_4	9	9	RSb	2	4	170	170
fd_6	9	9	RSb	2	4	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	5	155	155
fd_9	9	9	RSb	2	5	167	167
fd_2	9	9	RSb	2	5	153	153
fd_1	9	9	RSb	2	5	156	156
fd_8	9	9	RSb	2	5	164	164
fd_10	9	9	RSb	2	5	158	158
fd_7	9	9	RSb	2	5	168	168
fd_5	9	9	RSb	2	5	165	165
fd_4	9	9	RSb	2	5	170	170
fd_6	9	9	RSb	2	5	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	6	155	155
fd_9	9	9	RSb	2	6	167	167
fd_2	9	9	RSb	2	6	153	153
fd_1	9	9	RSb	2	6	156	156

fd_8	9	9	RSb	2	6	164	164
fd_10	9	9	RSb	2	6	158	158
fd_7	9	9	RSb	2	6	168	168
fd_5	9	9	RSb	2	6	165	165
fd_4	9	9	RSb	2	6	170	170
fd_6	9	9	RSb	2	6	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	7	155	155
fd_9	9	9	RSb	2	7	167	167
fd_2	9	9	RSb	2	7	153	153
fd_1	9	9	RSb	2	7	156	156
fd_8	9	9	RSb	2	7	164	164
fd_10	9	9	RSb	2	7	158	158
fd_7	9	9	RSb	2	7	168	168
fd_5	9	9	RSb	2	7	165	165
fd_4	9	9	RSb	2	7	170	170
fd_6	9	9	RSb	2	7	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	8	155	155
fd_9	9	9	RSb	2	8	167	167
fd_2	9	9	RSb	2	8	153	153
fd_1	9	9	RSb	2	8	156	156
fd_8	9	9	RSb	2	8	164	164
fd_10	9	9	RSb	2	8	158	158
fd_7	9	9	RSb	2	8	168	168
fd_5	9	9	RSb	2	8	165	165
fd_4	9	9	RSb	2	8	170	170
fd_6	9	9	RSb	2	8	169	169
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RSb	2	9	155	155
fd_9	9	9	RSb	2	9	167	167
fd_2	9	9	RSb	2	9	153	153
fd_1	9	9	RSb	2	9	156	156
fd_8	9	9	RSb	2	9	164	164
fd_10	9	9	RSb	2	9	158	158
fd_7	9	9	RSb	2	9	168	168
fd_5	9	9	RSb	2	9	165	165
fd_4	9	9	RSb	2	9	170	170
fd_6	9	9	RSb	2	9	169	169

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	1	357	357
fd_9	9	9	RCb*	1	1	355	355
fd_2	9	9	RCb*	1	1	369	369
fd_1	9	9	RCb*	1	1	345	345
fd_8	9	9	RCb*	1	1	404	404
fd_10	9	9	RCb*	1	1	372	372
fd_7	9	9	RCb*	1	1	366	366
fd_5	9	9	RCb*	1	1	353	353
fd_4	9	9	RCb*	1	1	367	367
fd_6	9	9	RCb*	1	1	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	2	357	357
fd_9	9	9	RCb*	1	2	355	355
fd_2	9	9	RCb*	1	2	369	369
fd_1	9	9	RCb*	1	2	345	345
fd_8	9	9	RCb*	1	2	404	404
fd_10	9	9	RCb*	1	2	372	372
fd_7	9	9	RCb*	1	2	366	366
fd_5	9	9	RCb*	1	2	353	353
fd_4	9	9	RCb*	1	2	367	367
fd_6	9	9	RCb*	1	2	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	3	357	357
fd_9	9	9	RCb*	1	3	355	355
fd_2	9	9	RCb*	1	3	369	369
fd_1	9	9	RCb*	1	3	345	345
fd_8	9	9	RCb*	1	3	404	404
fd_10	9	9	RCb*	1	3	372	372
fd_7	9	9	RCb*	1	3	366	366
fd_5	9	9	RCb*	1	3	353	353
fd_4	9	9	RCb*	1	3	367	367
fd_6	9	9	RCb*	1	3	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	4	357	357
fd_9	9	9	RCb*	1	4	355	355
fd_2	9	9	RCb*	1	4	369	369
fd_1	9	9	RCb*	1	4	345	345
fd_8	9	9	RCb*	1	4	404	404
fd_10	9	9	RCb*	1	4	372	372
fd_7	9	9	RCb*	1	4	366	366
fd_5	9	9	RCb*	1	4	353	353
fd_4	9	9	RCb*	1	4	367	367
fd_6	9	9	RCb*	1	4	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	5	357	357
fd_9	9	9	RCb*	1	5	355	355
fd_2	9	9	RCb*	1	5	369	369
fd_1	9	9	RCb*	1	5	345	345
fd_8	9	9	RCb*	1	5	404	404
fd_10	9	9	RCb*	1	5	372	372
fd_7	9	9	RCb*	1	5	366	366
fd_5	9	9	RCb*	1	5	353	353

fd_4	9	9	RCb*	1	5	367	367
fd_6	9	9	RCb*	1	5	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	6	357	357
fd_9	9	9	RCb*	1	6	355	355
fd_2	9	9	RCb*	1	6	369	369
fd_1	9	9	RCb*	1	6	345	345
fd_8	9	9	RCb*	1	6	404	404
fd_10	9	9	RCb*	1	6	372	372
fd_7	9	9	RCb*	1	6	366	366
fd_5	9	9	RCb*	1	6	353	353
fd_4	9	9	RCb*	1	6	367	367
fd_6	9	9	RCb*	1	6	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	7	357	357
fd_9	9	9	RCb*	1	7	355	355
fd_2	9	9	RCb*	1	7	369	369
fd_1	9	9	RCb*	1	7	345	345
fd_8	9	9	RCb*	1	7	404	404
fd_10	9	9	RCb*	1	7	372	372
fd_7	9	9	RCb*	1	7	366	366
fd_5	9	9	RCb*	1	7	353	353
fd_4	9	9	RCb*	1	7	367	367
fd_6	9	9	RCb*	1	7	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	8	357	357
fd_9	9	9	RCb*	1	8	355	355
fd_2	9	9	RCb*	1	8	369	369
fd_1	9	9	RCb*	1	8	345	345
fd_8	9	9	RCb*	1	8	404	404
fd_10	9	9	RCb*	1	8	372	372
fd_7	9	9	RCb*	1	8	366	366
fd_5	9	9	RCb*	1	8	353	353
fd_4	9	9	RCb*	1	8	367	367
fd_6	9	9	RCb*	1	8	398	398
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb*	1	9	357	357
fd_9	9	9	RCb*	1	9	355	355
fd_2	9	9	RCb*	1	9	369	369
fd_1	9	9	RCb*	1	9	345	345
fd_8	9	9	RCb*	1	9	404	404
fd_10	9	9	RCb*	1	9	372	372
fd_7	9	9	RCb*	1	9	366	366
fd_5	9	9	RCb*	1	9	353	353
fd_4	9	9	RCb*	1	9	367	367
fd_6	9	9	RCb*	1	9	398	398

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	1	277	277
fd_9	9	9	RCb	1	1	319	319
fd_2	9	9	RCb	1	1	319	319
fd_1	9	9	RCb	1	1	297	297
fd_8	9	9	RCb	1	1	309	309
fd_10	9	9	RCb	1	1	303	303
fd_7	9	9	RCb	1	1	316	316
fd_5	9	9	RCb	1	1	332	332
fd_4	9	9	RCb	1	1	267	267
fd_6	9	9	RCb	1	1	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	2	277	277
fd_9	9	9	RCb	1	2	319	319
fd_2	9	9	RCb	1	2	319	319
fd_1	9	9	RCb	1	2	297	297
fd_8	9	9	RCb	1	2	309	309
fd_10	9	9	RCb	1	2	303	303
fd_7	9	9	RCb	1	2	316	316
fd_5	9	9	RCb	1	2	332	332
fd_4	9	9	RCb	1	2	267	267
fd_6	9	9	RCb	1	2	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	3	277	277
fd_9	9	9	RCb	1	3	319	319
fd_2	9	9	RCb	1	3	319	319
fd_1	9	9	RCb	1	3	297	297
fd_8	9	9	RCb	1	3	309	309
fd_10	9	9	RCb	1	3	303	303
fd_7	9	9	RCb	1	3	316	316
fd_5	9	9	RCb	1	3	332	332
fd_4	9	9	RCb	1	3	267	267
fd_6	9	9	RCb	1	3	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	4	277	277
fd_9	9	9	RCb	1	4	319	319
fd_2	9	9	RCb	1	4	319	319
fd_1	9	9	RCb	1	4	297	297
fd_8	9	9	RCb	1	4	309	309
fd_10	9	9	RCb	1	4	303	303
fd_7	9	9	RCb	1	4	316	316
fd_5	9	9	RCb	1	4	332	332
fd_4	9	9	RCb	1	4	267	267
fd_6	9	9	RCb	1	4	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	5	277	277
fd_9	9	9	RCb	1	5	319	319
fd_2	9	9	RCb	1	5	319	319
fd_1	9	9	RCb	1	5	297	297
fd_8	9	9	RCb	1	5	309	309
fd_10	9	9	RCb	1	5	303	303
fd_7	9	9	RCb	1	5	316	316
fd_5	9	9	RCb	1	5	332	332

fd_4	9	9	RCb	1	5	267	267
fd_6	9	9	RCb	1	5	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	6	277	277
fd_9	9	9	RCb	1	6	319	319
fd_2	9	9	RCb	1	6	319	319
fd_1	9	9	RCb	1	6	297	297
fd_8	9	9	RCb	1	6	309	309
fd_10	9	9	RCb	1	6	303	303
fd_7							
fd_5	9	9	RCb	1	6	332	332
fd_4	9	9	RCb	1	6	267	267
fd_6	9	9	RCb	1	6	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	7	277	277
fd_9	9	9	RCb	1	7	319	319
fd_2	9	9	RCb	1	7	319	319
fd_1	9	9	RCb	1	7	297	297
fd_8	9	9	RCb	1	7	309	309
fd_10	9	9	RCb	1	7	303	303
fd_7	9	9	RCb	1	7	316	316
fd_5	9	9	RCb	1	7	332	332
fd_4	9	9	RCb	1	7	267	267
fd_6	9	9	RCb	1	7	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	8	277	277
fd_9	9	9	RCb	1	8	319	319
fd_2	9	9	RCb	1	8	319	319
fd_1	9	9	RCb	1	8	297	297
fd_8	9	9	RCb	1	8	309	309
fd_10	9	9	RCb	1	8	303	303
fd_7	9	9	RCb	1	8	316	316
fd_5	9	9	RCb	1	8	332	332
fd_4	9	9	RCb	1	8	267	267
fd_6	9	9	RCb	1	8	295	295
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	9	9	RCb	1	9	277	277
fd_9	9	9	RCb	1	9	319	319
fd_2	9	9	RCb	1	9	319	319
fd_1	9	9	RCb	1	9	297	297
fd_8	9	9	RCb	1	9	309	309
fd_10	9	9	RCb	1	9	303	303
fd_7	9	9	RCb	1	9	316	316
fd_5	9	9	RCb	1	9	332	332
fd_4	9	9	RCb	1	9	267	267
fd_6	9	9	RCb	1	9	295	295

Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	10	9	RCb	1	1	246	246
fd_9	10	9	RCb	1	1	241	241
fd_2	10	9	RCb	1	1	255	255
fd_1	10	9	RCb	1	1	200	200
fd_8	10	9	RCb	1	1	236	236
fd_10	10	9	RCb	1	1	249	249
fd_7	10	9	RCb	1	1	245	245
fd_5	10	9	RCb	1	1	234	234
fd_4	10	9	RCb	1	1	245	245
fd_6	10	9	RCb	1	1	238	238
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	10	9	RCb	1	3	246	246
fd_9	10	9	RCb	1	3	241	241
fd_2	10	9	RCb	1	3	255	255
fd_1	10	9	RCb	1	3	200	200
fd_8	10	9	RCb	1	3	236	236
fd_10	10	9	RCb	1	3	249	249
fd_7	10	9	RCb	1	3	245	245
fd_5	10	9	RCb	1	3	234	234
fd_4	10	9	RCb	1	3	245	245
fd_6	10	9	RCb	1	3	238	238
Fichier	nbJobs	nbMachines	contrainte	méthodeBI	méthodeBS	ForceBrute	Cmax*
fd_3	10	9	RCb	1	5	246	246
fd_9	10	9	RCb	1	5	241	241
fd_2	10	9	RCb	1	5	255	255
fd_1	10	9	RCb	1	5	200	200
fd_8	10	9	RCb	1	5	236	236
fd_10	10	9	RCb	1	5	249	249
fd_7	10	9	RCb	1	5	245	245
fd_5	10	9	RCb	1	5	234	234
fd_4	10	9	RCb	1	5	245	245
fd_6	10	9	RCb	1	5	238	238

duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
40.270000	0.000000	187	0	40505	0
37.830000	51.540000	189	185072	2431	59
38.550000	8.820000	200	27035	20546	52
38.050000	78.970000	170	238528	237	55
39.250000	3.090000	192	9109	3185	30
38.260000	4.300000	205	13125	5479	44
38.910000	15.690000	197	46352	1727	44
40.220000	5.790000	191	11128	3161	48
38.410000	24.580000	198	50629	2251	43
38.860000	4.650000	198	11756	33116	47
duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
39.860000	0.000000	187	0	40212	0
38.970000	74.950000	189	184405	2464	59
44.810000	10.010000	200	27054	20502	52
44.910000	197.530000	170	238540	270	55
42.940000	28.020000	192	9129	3045	30
41.320000	9.390000	205	12958	5463	44
46.360000	15.120000	197	46344	1936	44
42.430000	24.400000	191	11589	3507	48
42.130000	20.950000	198	50754	2066	43
42.920000	4.660000	198	11433	32829	47
duréeForceBrute	duréePSE	premièreBI	nbNoeudsEvalués	nbNoeudsAvantOptimum	écart
38.960000	0.000000	187	0	40429	0
38.590000	58.210000	189	185059	2184	59
38.400000	9.920000	200	26682	19871	52
38.450000	78.330000	170	237898	206	55
38.400000	3.380000	192	9083	2904	30
38.770000	4.400000	205	12785	5348	44
39.260000	15.390000	197	46214	2025	44
38.190000	4.700000	191	11827	3506	48
38.070000	18.390000	198	50754	2111	43
39.520000	5.350000	198	11366	33099	47

Cahier de Spécifications

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

Département Informatique

Cahier de spécification système & plan de développement			
Projet :	**Etude d'un flowshop pour minimiser le C_{max} sous diverses contraintes de blocage**		
Emetteur :	Mélanie MAUGEAIS	Coordonnées : EPU-DI	
Date d'émission :	21 décembre 2012		
Validation			
Nom	Date	Valide (O/N)	Commentaires

Christophe LENTE : 20/12/2012 ; O ; Ok pour 2012-2013
Nhat Vinh VO : 20/12/2012 ; O ; Ok pour 2012-2013
Christophe LENTE : 30/11/2012 ; N ; Planning à détailler
Nhat Vinh VO : 30/11/2012 ; N ; Planning à compléter

Historique des modifications		
Version	Date	Description de la modification

00 : 30/11/2012 ; Version initiale
01 : 20/12/2012 ; Version finale, le planning a été étoffé

Table des matières

Cahier de spécification système

1.1 Introduction

Dans le cadre d'un projet de recherche, il est très difficile d'établir des spécifications détaillées, car le déroulement du projet ne peut pas être défini à l'avance. Ce document a pour but de spécifier la manière dont va être conduite et réalisée l'étude d'un flowshop pour la minimisation du C_{max} sous diverses contraintes de blocage.

Le projet de recherche ainsi cité a été donné comme projet de fin d'étude à Mélanie MAUGEAIS, étudiante en dernière année au département informatique de Polytech'Tours, sous la supervision de Christophe LENTÉ et Vinh Nhat VO, enseignant chercheur et doctorant dans le même établissement.

1.2 Contexte de la réalisation

1.2.1 Contexte

Ce projet a été donné dans le cadre des enseignements de la 5e année de formation d'ingénieur en informatique à Polytech'Tours. Il s'étend sur toute l'année scolaire et sera clôturé par une soutenance.

Il s'agit d'un projet de recherche, qui vise à étudier l'ordonnancement d'un problème de flowshop pour la minimisation de la date de fin d'exécution de toutes les tâches (le C_{max}), avec un nouveau type de contrainte de blocage.
Cette étude se base sur un article publié en 2004 :

Complexity of flowshop scheduling problems with a new blocking constraint

par S. MARTINEZ, S. DAUZÈRE-PÉRÈS, C. GUÉRET, Y. MATI et N. SAUER ;
et vise à établir une heuristique de résolution pour ce type de problème d'ordonnancement.

L'étude d'un article complémentaire pourra être extrêmement précieuse dans le déroulement de ce projet, car il s'agit d'un article présentant une heuristique de résolution d'un problème de flowshop avec des contraintes mixtes de blocage (heuristique TSS) :

Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems

par W. TRABELSI, C. SAUVEY et N. SAUER.

1.2.2 Objectifs

Ce projet de recherche est mené autour d'un sujet tout à fait nouveau : l'étude de la contrainte de blocage RCb/RCb* dans un ordonnancement de type flowshop.

Le blocage de type RCb (Release when Completing Blocking) a pour conséquence qu'un job ne peut commencer sur une machine que si le job précédent a libéré la machine suivante.

Le blocage RCb* (Release when Completing Blocking*) consiste à ce qu'un job ne puisse commencer sur une machine que lorsque le job précédent à terminé son exécution sur la machine suivante.

De manière plus poussée, ce projet s'oriente vers l'étude d'un flowshop soumis à des contraintes mixtes de blocage, parmi RSb, RCb et RCb*.

La contrainte de blocage RSb (Release when Starting Blocking) désigne la contrainte de blocage classique dans un flowshop sans stock intermédiaire.

Les objectifs de ce projet de recherche sont les suivants :

- Etablir la modélisation Max-Plus d'un flowshop sous chacune des contraintes (RSb, RCb et RCb*), sans mixité,
- Réaliser un programme simple permettant de calculer le C_{max} d'une séquence de jobs donnée, à partir des modélisations Max-Plus établies, afin de vérifier qu'elles soient exactes
- Créer un programme mettant en oeuvre une PSE (Procédure par Séparation et Evaluation) et permettant de résoudre un problème de flowshop sous les diverses contraintes précédemment citées
- La PSE devra également mettre en place un calcul de bornes inférieure et supérieure afin d'encadrer la solution optimale

1.2.3 Bases méthodologiques

Pour mener à bien ce projet, nous utiliserons une machine virtuelle afin de programmer en langage C sous une distribution Ubuntu. Cette machine virtuelle est fournie par les encadrants du projet, et résulte d'un projet de fin d'étude qui a eu lieu l'année passée.

Pour une meilleure lisibilité du code, et afin de simplifier des reprises éventuelles futures, je m'appliquerais à respecter quelques conventions simples de nommage, dont notamment :

- Donner un nom explicite aux variables et constantes du programme, mettant ainsi en évidence leur rôle
- Le nom des variables sera écrit sans espace, en minuscules, avec une majuscule à chaque début d'un nouveau mot (*maVariable*)
- Le nom des constantes sera écrit en majuscules avec un underscore pour délimiter chaque nouveau mot (*MA_CONSTANTE*)
- Le nom des fonctions ne comportera pas d'espace et devra commencer par une majuscule, tout comme chaque nouveau mot (*MaFonction*)

De plus, il est nécessaire de commenter au maximum le code, de sorte à connaître l'effet d'une action dans le programme. Un commentaire au dessus de chaque fonction sera obligatoire, afin de préciser son rôle et les éléments sur lesquels elle agit, ainsi que des commentaires intermédiaires, situés à des endroits stratégiques du code, notamment au niveau de suites d'instructions complexes ou particulièrement importantes.

Au delà de ces quelques impératifs, il est du bon sens du programmeur d'utiliser des notations adéquates et des commentaires judicieux.

1.3 Description générale

1.3.1 Environnement du projet

Le projet est établi sur un problème nouveau, par conséquent, il n'y a que peu de ressources existantes, aussi bien dans la littérature que dans les projets antérieurs s'en rapprochant.

Toutefois, il sera bon de s'appuyer sur le stage de M2R (master de recherche) de Vincent AUGUSTO, dans lequel l'étudiant a réalisé, en 2005, une PSE de résolution d'un problème de flowshop classique, qu'il me faudra maîtriser pour reprendre les éléments clefs : construction de l'arborescence, mise en place des bornes inférieures et supérieures..

Ce projet a été repris l'année suivante par Julien PAPILLON, puis en 2011-2012, Ze Yu JI et Fang YAN ont mis en place une machine virtuelle permettant la compilation d'un code écrit en langage C, qui utiliserait la méthode de résolution du PVC (problème du voyageur de commerce), codée en Fortran. Comme je serai peut être amenée à utiliser cette procédure, il est fortement conseillé d'utiliser la machine virtuelle fournie pour réaliser la programmation, en langage C, de la PSE de résolution d'un problème de flowshop pour la minimisation du C_{max} soumis à diverses contraintes de blocage.

De plus, afin de réaliser la programmation des algorithmes fournis dans l'article décrivant des heuristiques de résolution d'un flowshop avec contraintes de blocage mixtes, un mini-projet de GPF (Gestion des Flux et de la Production) sera proposé aux étudiants de DI4.

J'encadrerai leur projet avec M. LENTÉ et leur travail pourra ainsi être utilisé dans la PSE à réaliser dans le cadre de mon projet de fin d'étude.

1.3.2 Caractéristiques des utilisateurs

On distingue un seul type d'utilisateurs : les chercheurs en ordonnancement. Il s'agit donc d'utilisateurs aguerris, qui savent manipuler l'outil informatique, et qui n'ont donc pas de problème à utiliser une console de commandes, à comprendre le langage C ou bien encore à utiliser le programme qui va être réalisé.

De ce fait, aucune précaution particulière n'est à prendre en considération pour la réalisation du programme. Notamment, il n'y aura pas besoin d'interface graphique pour rendre intuitive l'utilisation du programme.

Il sera toutefois nécessaire de documenter le code et les divers fichiers joints ou créés par l'application.

1.4 Conditions de fonctionnement

1.4.1 Performances

Dans ce projet, le but est d'obtenir que la PSE fournisse un résultat en un temps minimum. Rappelons que cette procédure par séparation et évaluation vise à trouver une solution optimale pour un problème d'ordonnancement de type flowshop, soumis à diverses contraintes de blocage, pour la minimisation du C_{max}, ou bien, si le problème ne permet pas de trouver l'optimum en temps polynomial, la PSE pourra fournir un encadrement précis de la solution optimale.

L'objectif est donc de produire des résultats pour différentes instances de ce type de problème, en visant à résoudre des problèmes de plus en plus gros.

Une instance est définie par un nombre de machines, un nombre de jobs, des durées d'exécution et des contraintes de blocage. Les performances seront bonnes si on peut trouver une solution exacte pour des problèmes avec un grand nombre de machines et de jobs, et cela dans un temps relativement court, cette durée dépendant directement de la taille du problème.

1.4.2 Capacités

Le problème d'ordonnancement étudié est un problème NP-difficile, on ne peut donc pas prédire les limites de la PSE qui va être réalisée. Ces limites pourront être établies à partir des expérimentations qui seront faites.

Plan de développement

2.1 Découpage du projet en tâches

Afin d'établir un planning de base pour orienter ce projet de recherche, il faut commencer par le découper en tâches. Ce découpage restera toutefois assez grossier puisque dans un projet de recherche tout reste à définir, on ne connait pas à l'avance la tournure que le projet va prendre.

Une estimation de charge sera intéressante à réaliser afin de préciser le poids de chaque tâche dans le déroulement du projet. Ces estimations de charge seront exprimées en pourcentage de la charge totale du projet, sachant que ce dernier s'étend sur 420 heures obligatoires (nous ignorerons les heures supplémentaires passées sur le projet pour les calculs).

2.1.1 Prendre en main le projet

Description de la tâche

Le projet porte sur une nouvelle contrainte de blocage, qu'il faut donc commencer par comprendre avant de se lancer sur une réflexion plus poussée.

Dans un premier temps, il sera nécessaire de maitriser le contenu des articles support, tant au niveau de la définition du problème qu'au niveau des divers outils de modélisation (graphe, modèle mathématique,...) et des méthodes de résolution proposées.

Estimation de charge

Cette phase de démarrage peut être estimée à 8% environ de la charge totale du projet, soit à peu près 35h.

2.1.2 Etablir la modélisation Max-Plus

Description de la tâche

Un problème de flowshop avec des contraintes de blocage met en jeu diverses contraintes de précédence spécifiques à chaque type de blocage (RSb, RCb, RCb*).

A ce stade, il est prévu d'étudier un flowshop avec une unique contrainte de blocage. La mixité des contraintes sera étudiée a posteriori, si les résultats sont concluants pour un unique type de blocage.

Le choix d'une modélisation de type Max-Plus s'explique par le fait qu'une telle modélisation représente d'une manière simple le problème, et qu'elle est facilement manipulable.

De plus, M. Christophe LENTÉ, enseignant chercheur en charge de l'encadrement du projet, travaille habituellement avec l'algèbre Max-Plus, donc bien qu'il y ait d'autres possibilités pour modéliser un problème d'ordonnancement, c'est celle-ci qui sera privilégiée, sauf si vraiment elle ne permet pas d'obtenir des résultats intéressants.

Estimation de charge

La durée nécessaire à la réalisation de cette tâche dépend non seulement de l'expérience de l'étudiant en charge du projet, en ce qui concerne le fait d'établir une modélisation, mais aussi de la compréhension du problème.

Il est difficile d'en faire une estimation précise, mais comme cette modélisation sera la base de tout le code qui sera produit, cette étape n'est pas à négliger.

Cette première étape pourrait être estimée à environ 12% de la charge totale du projet, soit à peu près 50h.

2.1.3 Vérifier l'exactitude de la modélisation établie

Description de la tâche

Cette deuxième tâche consiste à réaliser un petit programme, en langage C, permettant de calculer le C_{max} d'une séquence de jobs donnée, à partir de la modélisation Max-Plus définie pour la contrainte de blocage spécifiée dans l'instance.

Ce programme devra permettre de tester les modélisations réalisées. En effet, en comparant le résultat obtenu par le programme, au C_{max} qu'il faut obtenir pour l'instance donnée (calculé manuellement par exemple), on pourra aisément vérifier les modélisations : si les deux valeurs sont différentes pour l'une des instances testées, alors la modélisation sera remise en question.

Estimation de charge

La réalisation de ce programme ne devrait pas prendre trop de temps, mais une réflexion sur la manière d'organiser le code sera nécessaire.

Cette étape est estimée à environ 10% de la charge totale du projet, soit approximativement 42h.

2.1.4 Prendre en main l'existant

Description de la tâche

Une PSE ayant été réalisée sur ces dernières années m'a été fournie, afin que je puisse m'appuyer sur le code, le réutiliser, et y intégrer les nouveaux éléments spécifiques au problème étudié dans le cadre de mon PFE.

De ce fait, avant de commencer la programmation de ma propre PSE, je dois prendre en main le code existant, de sorte à déterminer les portions de code particulièrement intéressantes et celles à modifier ou à retirer pour adapter le programme à la recherche d'une solution proche de l'optimum pour un problème comme décrit précédemment : problème de flowshop pour la minimisation du C_{max}, soumis à des contraintes de blocage (RSb, RCb et RCb*).

Estimation de charge

Cette prise en main de l'existant ne devrait pas être trop longue, et surtout, elle se fera en partie en parallèle à la réalisation de la PSE. J'estime sa charge à 7% de la charge totale du projet, soit 30 heures environ.

2.1.5 Réaliser la procédure par séparation et évaluation (PSE)

Description de la tâche

Cette étape concerne le coeur du projet qui va être mené. C'est en effet l'objectif principal de l'étude.
Il s'agit de programmer la procédure qui résoudra un problème de flowshop pour minimiser la date de fin d'exécution de la dernière tâche sur la dernière machine, sous des contraintes de blocage d'abord uniques puis mixées.

Une procédure par séparation et évaluation (PSE, ou branch and bound en Anglais) est une méthode d'énumération implicite, c'est à dire que toutes les solutions possibles du problème peuvent être énumérées mais l'analyse des propriétés du problème permet d'éliminer de larges classes de mauvaises solutions.

C'est donc sur ce principe que le programme va être réalisé. Il devra permettre de trouver une solution proche de l'optimum, et de fournir un encadrement le plus restreint possible.
La résolution sera possible sur des problèmes à contrainte unique de blocage dans un premier temps, puis, dans un second temps, sur des problèmes avec contraintes mixtes de blocage, c'est à dire que différents blocages interviendront d'une machine à l'autre.

Estimation de charge

La réalisation de cette PSE peut s'étaler dans le temps, en fonction des difficultés qui seront rencontrées, aussi bien sur la programmation que sur la modélisation des problèmes.
L'estimation de charge de cette étape correspond à 60% de la charge totale du projet, soit environ 250h.

Décomposition en sous-tâches

La réalisation d'une PSE comporte de nombreuses sous-tâches qu'il est important de citer. Ces tâches vont également être brièvement décrites et estimées en charge.

1. ***Mise en place d'une borne inférieure***

   Afin de prévoir un encadrement précis de la solution optimale, il faut prévoir de calculer une ou plusieurs bornes inférieures, avec différentes méthodes. La mise en place d'une borne inférieure passe par les étapes suivantes :

   - Recherche (théorique) : dans un premier temps, il faut déterminer, parmi tous les calculs possibles de bornes inférieures, celui ou ceux qui sont les plus intéressants pour notre problème.
   - Programmation : une fois les méthodes de calcul choisies, il faut réaliser un programme les mettant en oeuvre sur des instances du problème.
   - Tests de validation : il est nécessaire de vérifier que les valeurs obtenues sont bien inférieures ou égales à la solution optimale des instances.
   - Tests à grande échelle (benchmarks) : le but est d'établir un banc d'essai de sorte à pouvoir mesurer les performances du système.
   - Comparaison des bornes inférieures : si on détermine plusieurs bornes inférieures, il est bon de les comparer, de sorte à voir quelle est la plus adaptée à notre problème.

   La mise en place d'une borne inférieure ne doit pas être négligée car son calcul n'est pas trivial et plus elle sera précise, plus on pourra encadrer précisément la valeur optimale de la solution. De plus, si la valeur de retour de la PSE est égale à la borne inférieure calculée, alors la valeur retournée est l'optimum de l'instance ! On pourrait estimer la charge de cette sous-tâche à environ 50 heures.

2. ***Mise en place d'une borne supérieure***

   Pour finaliser l'encadrement de la solution optimale, il faut également calculer une ou plusieurs bornes supérieures, avec différentes méthodes. La mise en place d'une borne supérieure passe par les mêmes étapes que précédemment :

   - Recherche (théorique) : dans un premier temps, il faut déterminer, parmi tous les calculs possibles de bornes supérieures, celui ou ceux qui sont les plus intéressants pour notre problème.
     Un calcul simple de borne supérieure correspond au calcul du C_{max} pour une séquence quelconque de l'instance : cette valeur sera au mieux l'optimum, et sera donc forcément une borne supérieure.
   - Programmation : une fois les méthodes de calcul choisies, il faut réaliser un programme les mettant en oeuvre sur des instances du problème.
   - Tests de validation : il est nécessaire de vérifier que les valeurs obtenues sont bien supérieures ou égales à la solution optimale des instances.
   - Tests à grande échelle (benchmarks) : le but est d'établir un banc d'essai de sorte à pouvoir mesurer les performances du système.
   - Comparaison des bornes supérieures : si on détermine plusieurs bornes supérieures, il est bon de les comparer, de sorte à voir quelle est la plus adaptée à notre problème.

   Cette sous-tâche peut éventuellement être mise entre parenthèses dans un premier temps, en calculant grossièrement une borne supérieure par la méthode triviale citée plus haut : en récupérant le C_{max} d'un ordonnancement quelconque. Toutefois, la précision n'est pas à sous-estimer car l'objectif est de borner au mieux l'optimum. La charge de cette tâche peut donc être estimée approximativement à 20 heures du projet.

3. ***Réalisation de tests supplémentaires***

   Pour chaque instance, on pourrait calculer l'écart entre la borne inférieure et la borne supérieure, de sorte à trouver le meilleur encadrement de la solution optimale (plus la différence sera faible, plus l'encadrement sera restreint autour de l'optimum).

   Cette phase de test est intéressante mais pas nécessaire. Toutefois, nous prévoirons d'y passer à peu près 10 heures.

4. ***Établissement de conditions de dominance***

   Etablir des conditions de dominance consiste à comparer 2 noeuds de l'arborescence, c'est à dire 2 séquences partielles, afin d'être en mesure de dire si l'un d'eux sera forcément meilleur que l'autre, auquel cas il ne sera pas nécessaire de poursuivre le parcours du second noeud.

   Cette tâche est importante d'un point de vue optimisation du programme, c'est pourquoi je l'estime à environ 20 heures.

5. ***Programmation d'une méthode exacte de résolution de petits jeux de données***

De sorte à pouvoir mesurer la précision de la PSE, qui, rappelons-le, doit être une méthode de résolution exacte renvoyant l'optimum, il sera intéressant de trouver la valeur optimale de certaines instances, afin de comparer l'écart existant entre cette valeur et le résultat de la PSE.

Plusieurs méthodes exactes peuvent être programmées :

- Force brute : cette méthode permet de trouver l'optimum, par énumération de toutes les solutions. Les étudiants de DI4 en charge du mini-projet de GPF en rapport avec mon PFE, seront chargés de réaliser le programme mettant en oeuvre cette méthode.
- PPC (Programmation Par Contraintes) : technique avancée de résolution de problèmes d'optimisation complexes, reposant sur un modèle mathématique, dans lequel sont définies les données du problème, et des contraintes désignant ce qu'il faut éviter, ce qu'on ne veut pas obtenir.
- PLNE (Programmation Linéaire en Nombres Entiers) : sa particularité est que toutes les variables prennent des valeurs entières. De nombreux problèmes sont formulables en PLNE et il existe diverses méthodes de résolution des PLNE qui sont en pratique souvent efficaces. C'est pourquoi, cette méthode pourrait être utilisée pour obtenir l'optimum de certaines instances.

L'une de ces méthodes sera déjà réalisée, on ne s'intéressera, a priori, pas aux deux autres méthodes citées. Dans le cadre de mon projet, la réalisation de cette tâche n'aura de charge propre : elle sera comprise dans l'encadrement du mini-projet de GPF.

6. ***Programmation de l'arborescence***

La programmation de l'arborescence est à reprendre en partie sur la PSE fournie avec la machine virtuelle, en réalisant toutefois quelques adaptations :

- Adaptation des bornes inférieures et supérieures : il y aura peut-être besoin d'adapter les codes produits de sorte à pouvoir les introduire dans le code de la PSE existante, afin de réutiliser au maximum le code fourni.
- Intégration des bornes inférieures et supérieures
- Tests de validation : de nouveau, il faudra vérifier que l'optimum et la valeur obtenue par la PSE pour les instances testées appartient bien à l'intervalle [BI,BS].
- Tests de grande échelle (benchmarks)
- Analyse de tests : trouver l'utilité des bornes inférieures, des bornes supérieures, l'influence du parcours, la taille maximale des problèmes, ...
  Par exemple, il faudrait être capable de déterminer si une borne inférieure est meilleure pour un certain type de problème.
- Améliorations : un programme ne pouvant pas être parfait du premier coup, selon les résultats obtenus, il faudra prévoir de l'améliorer et de recommencer autant de fois que nécessaire.

Cette sous-tâche est la plus longue, et avec toutes les améliorations qui peuvent en ressortir au fur et à mesure de l'avancement, j'estime sa charge à 150 heures environ.

2.1.6 Encadrer le mini-projet de GPF

Description de la tâche

Afin d'alléger ma charge de travail, un mini-projet de GPF est confié à des étudiants de quatrième année au département informatique de Polytech'Tours, de sorte à programmer les heuristiques présentées dans le 2ième article cité en introduction.

Il leur est ainsi demandé de :

- Générer des instances de problèmes de flowshop de manière aléatoire, en prenant en compte des contraintes de blocage mixtes
- Coder l'algorithme NEH
- Coder l'algorithme TSS
- Coder l'amélioration locale de NEH
- Coder l'amélioration locale de TSS
- Générer des résultats statistiques sur un grand nombre d'instances, en combinant les 4 algorithmes (par exemple : NEH + amélioration TSS, NEH + amélioration NEH, TSS + amélioration NEH, ...).

Estimation de charge

Le mini-projet de GPF des étudiants de DI4 s'étend de novembre à janvier, donc l'encadrement se fera en parallèle de mon travail, par des réunions pour faire le bilan de leur avancement et contrôler qu'ils s'orientent dans la bonne direction. De ce fait, on peut estimer la charge de cette tâche à 7% de la charge totale de mon PFE, soit 30 heures.

2.1.7 Prévoir les débordements

Description de la tâche

Le fait de prévoir les débordements n'est pas une tâche à proprement parler mais une ligne à suivre tout au long du projet, de sorte à être capable d'absorber toute difficulté survenant au cours de la réalisation.

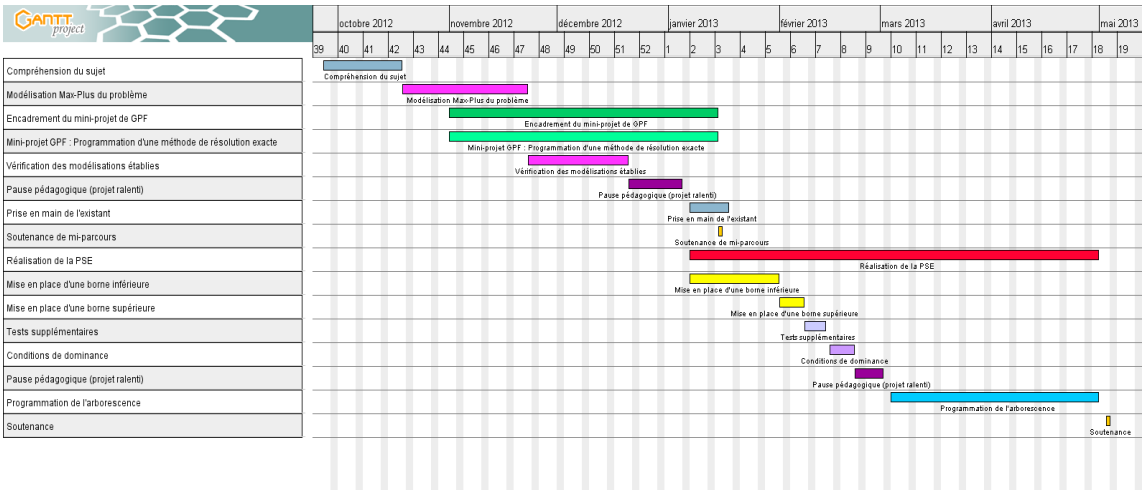
Estimation de charge

Dans le cadre de ce projet, tout débordement pourra être amorti dans les 25 heures de marge laissée (environ 6% de la charge totale du projet).

2.2 Planning

En se basant principalement sur les tâches décrites précédemment, on peut établir un planning grossier du déroulement du projet.

Voici un diagramme de Gantt de ce planning :



Références

x *Complexity of flowshop scheduling problems with a new blocking constraint*
par S. MARTINEZ, S. DAUZÈRE-PÉRÈS, C. GUÉRET, Y. MATI et N. SAUER

x *Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems*
par W. TRABELSI, C. SAUVEY et N. SAUER

Rapport du master de recherche de Vincent Augusto

Université François Rabelais
École Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis
37200 TOURS
http ://www.polytech.univ-tours.fr

Stage de Master de Recherche en Informatique

Une procédure par séparation et évaluation pour le problème de flowshop avec délais minimum et maximum

LI Tours
Polytech'Tours
64, avenue Jean Portalis
37200 TOURS

Encadrant de stage
M. Christophe LENTÉ
Maître de conférences

Auteur du stage
Vincent AUGUSTO
M2R
2004-2005

Remerciements

Ce stage de M2R s'inscrivant dans le prolongement de mon Projet de Fin d'Études de troisième année du Département Informatique, je tiens à remercier une fois de plus l'intégralité de l'équipe *Ordonnancement et Conduite* du Laboratoire d'Informatique de Polytech'Tours pour leur bonne humeur et leurs conseils avisés. Je remercie également MM. Christophe LENTÉ et Jean-Louis BOUQUARD pour leur patience ainsi que pour leur aide qui me furent très précieuses.

Je me dois de remercier également toutes les personnes qui ont appuyé ma candidature pour la thèse de doctorat à l'école des Mines de Saint-Etienne. Leur soutien me fut très précieux et contribua à ma sélection.

Enfin, merci à mes collègues de stage avec qui j'ai pu partager la bonne humeur et la chaleur (dans tous les sens du terme) de la salle Unix-A du DI. Ce fut une expérience... remarquable !

Table des matières

Introduction

Résoudre un problème d'ordonnancement, c'est trouver une adéquation entre un travail à effectuer, décrit sous la forme d'un ensemble de tâches interdépendantes, et les moyens disponibles pour sa réalisation. La variété des domaines d'application est extrême. L'intérêt des entreprises pour une bonne maîtrise de l'ordonnancement de leurs activités est fondamental. Cependant, le savoir permettant de décrire un problème, d'énoncer les objectifs qui président à sa résolution en vue d'adopter une méthodologie de résolution n'est pas toujours à la portée de l'industriel lambda. C'est pour cette raison que le lien entre le monde de l'entreprise et le domaine de la recherche en ordonnancement est aujourd'hui très fort, ce dernier étant en perpetuelle évolution depuis les années 1950, avec les débuts de la Recherche Opérationnelle et plusieurs précurseurs tels Johnson [15], Jackson [14], Conway et al. [8], Baker [1], etc.

Cet ouvrage, bilan de mon stage de Master 2 Informatique, s'inscrit dans la continuité de mon Projet de Fin d'Études effectué au sein du Département Informatique de Polytech'Tours. Après avoir étudié la modélisation de problèmes de flowshop au moyen de l'*algèbre Max-Plus*, et plus précisément la modélisation du problème de flowshop sans attente, je me suis penché sur le problème plus général du flowshop avec délais minimum et maximum. Ce rapport de stage de M2R s'inscrit également dans le prolongement des travaux de MM. Lenté [16] et Bouquard [5].

Nous présenterons dans un premier temps de brefs rappels à propos des algèbres tropicales et de l'ordonnancement en général ; bon nombre d'entre eux figurent également dans le rapport de PFE précédant ce document, bien que plusieurs modifications propres au sujet qui nous interesse ici ont été apportées. Dans un second chapitre nous donnerons une modélisation du problème de flowshop considéré, ainsi qu'une description de la borne inférieure au critère examiné, dans un premier temps dans un cas particulier simple, puis dans le cas général. Il s'agit d'un préliminaire théorique indispensable à la présentation de la Procédure par Séparation et Évaluation proposée dans le troisième chapitre. Plusieurs constatations très brèves seront données dans le quatrième chapitre au sujet des performances du programme réalisé. Enfin, nous conclurons sur l'apport du travail effectué au cours de ce stage et sur son intérêt. Les annexes regroupent les notices utilisateur des programmes développés.

Chapitre 1

Préliminaires théoriques

Ce chapitre a pour but de présenter succintement les notions sur lesquelles s'appuit l'étude du problème d'ordonnancement considéré dans ce rapport. Ne seront rassemblés ici que les éléments indispensables à la compréhension du rapport. Pour de plus amples informations, se référer à [10], [16].

1.1 Algèbres tropicales

1.1.1 Généralités

Les trois définitions qui suivent portent sur un ensemble E muni de deux lois internes notées $\oplus$ et $\otimes$.

Définition 1.1.1 (Demi-anneau)

$(E, \oplus, \otimes)$ *est un demi-anneau ssi* $\oplus$ *est commutative, associative, admet un élt. neutre* $\mathbf{0}$ *;*
$\otimes$ *est associative et admet un élément neutre* $\mathbf{1}$ *;*
$\otimes$ *est distributive sur* $\oplus$ *à droite et à gauche ;*
$\mathbf{0}$ *est absorbant pour* $\otimes$*.*

L'élément $\mathbf{0}$ est appelé l'élément nul et $\mathbf{1}$ est appelé élément unité. L'écriture $a \otimes b$ est souvent simplifiée en $a.b$ ou en ab.

Définition 1.1.2 (Dioïde)

$(E, \oplus, \otimes)$ *est un demi-anneau idempotent ou* ***dioïde*** *ssi* $(E, \oplus, \otimes)$ *est un demi-anneau ;*
$\oplus$ *est idempotente.*

Définition 1.1.3 (Demi-corps)

$(E, \oplus, \otimes)$ *est un demi-corps ssi* $(E, \oplus, \otimes)$ *est un demi-anneau ;*
$(E - \{\mathbf{0}\}, \otimes)$ *est un groupe ;*
soit encore ssi $(E, \oplus, \otimes)$ *est un demi-anneau ;*
tout élément non nul admet un symétrique pour $\otimes$*.*

L'étude menée dans ce rapport repose sur le demi-corps idempotent qui est noté $\mathbb{R}_{max} = (\mathbb{R} \cup \{-\infty\}, max, +)$ également appelé *algèbre Max-Plus* L'opérateur max est noté additivement $\oplus$ et l'opérateur $+$ désignant l'addition usuelle est noté multiplicativement $\otimes$. L'élément nul est

$-\infty$, noté **0** et l'élément unité est 0, noté **1**. Lorsque l'on a besoin de faire référence au nombre 0 on le note sous forme de chiffre, les termes élement nul ou zéro font forcément référence à **0**. Par convention et par commodité, dans ce document le signe $\equiv$ signale la traduction d'une expression de l'algèbre Max-Plus dans l'algèbre usuelle ou inversement.

Définition 1.1.4 (Ordre canonique)
Tout dioïde est muni de la relation d'ordre partielle

$$\forall(x,y) \in E^2, x \preceq y \Leftrightarrow x \oplus y = y \tag{1.1}$$

Dans $\mathbb{R}_{max}$ cet ordre est total et correspond à l'ordre naturel des réels, mais il restera néanmoins noté $\preceq$ dans un soucis de généralité.

1.1.2 Algèbres de matrices

Cette section décrit des considérations particulières liées à la modélisation matricielle des problèmes de types flowshop comme nous le verrons par la suite dans le cas particulier du flowshop sans attente.

Convention de notations

Comme cela a été dit, il est nécessaire de définir de nombreuses matrices, dont le plupart dépendent d'un paramètre. Ce sont donc plus exactement des fonctions matricielles d'un ensemble de travaux ou de séquences sur un dioïde, d'autres sont des transformées de ces matrices. Nous adopterons dans ce rapport la même notation utilisée que dans [16] : une lettre capitale désigne une matrice, la lettre minuscule désigne leurs composantes. Ainsi $a_{i,j}$ est une composante de la matrice A.

Exemple 1.1.1

$$A = \begin{pmatrix} a_1 & a_{1,2} & a_{1,3} \\ a_{2,1} & a_2 & a_{2,3} \\ a_{3,1} & a_{3,2} & a_3 \end{pmatrix}, \quad t(\sigma) = \begin{pmatrix} t_1(\sigma) & t_{1,2}(\sigma) & t_{1,3}(\sigma) \\ t_{2,1}(\sigma) & t_2(\sigma) & t_{2,3}(\sigma) \\ t_{3,1}(\sigma) & t_{3,2}(\sigma) & t_3(\sigma) \end{pmatrix}$$

La matrice $R^{(u,v)}(k)$ est composée des éléments $r^{u,v}_{\ell,c}(k)$.

La notation employée en particulier dans [12] est aussi utilisée, la composante à l'intersection de la ligne ℓ et de la colonne c de la matrice A est notée $[A]_{\ell,c}$, ainsi $r^{u,v}_{\ell,c}(k) = [R^{(u,v)}(k)]_{\ell,c}$.

Généralités

Les structures de demi-anneau et de dioïde peuvent se transmettre à des ensembles de matrices, il suffit pour cela d'étendre correctement les lois internes.

Soient A et B deux matrices $n \times m$, on définit $C = A \oplus B$ par :

$$\forall(i,j) \in \{1,\ldots,n\} \times \{1,\ldots,m\}, \quad [C]_{i,j} = [A]_{i,j} \oplus [B]_{i,j} \tag{1.2}$$

Soient A une matrice $n \times m$ et B une matrice $n \times p$, on définit $C = A \otimes B$ par :

$$\forall (i,j) \in \{1,\dots,n\} \times \{1,\dots,p\},\ \ [C]_{i,j} = \bigoplus_{k=1}^{m} [A]_{i,k} \otimes [B]_{k,j} \tag{1.3}$$

Théorème 1.1.1
Soit $\mathcal{M}_n(E)$ l'ensemble des matrices carrées d'ordre n sur E.
Si $(E, \oplus, \otimes)$ est un demi-anneau alors $(\mathcal{M}_n(E), \oplus, \otimes)$ en est un également.
Si $(E, \oplus, \otimes)$ est un dioïde alors $(\mathcal{M}_n(E), \oplus, \otimes)$ en est un également.

Résiduation

La résiduation est une théorie vaste [4] quis'appliqueà des applications monotones entre ensembles ordonnés et dont l'objectif est de définir un objet mathématique se rapprochant le plus possible d'une fonction réciproque et de "résoudre" par excès ou par défaut des équations de la forme $f(x) = \alpha$.

On ne retiendra dans ce rapport que la possibilité de définir une sorte de division dans les dioïdes complets et ainsi de pallier la non inversibilité des matrices.

On considère les équations $ax = b$ et $xa = b$ dans un dioïde complet.

Définition 1.1.5
On appelle résidué à gauche le nombre $a\backslash b$ et résidué à droite le nombre b/a définis par

$$a\backslash b = \bigoplus \{x | ax \preceq b\} \tag{1.4}$$

$$b/a = \bigoplus \{x | xa \preceq b\} \tag{1.5}$$

Ainsi $a(a\backslash b) \preceq b$ et $(b/a)a \preceq b$. Il faut noter que si a est inversible alors $a\backslash b = a^{-1}b$ et $b/a = ba^{-1}$.

Exemple 1.1.2
Dans le dioïde complet $\bar{\mathbb{R}}_{max} = (\mathbb{R} \cup \{-\infty, +\infty\},\ max,\ +)$,

$$\forall a, b \in \mathbb{R},\ \ a\backslash b = b/a \equiv b - 1 \tag{1.6}$$

On notera ces nombres $\frac{b}{a}$ dans toute la suite.

$$\forall a \in \bar{\mathbb{R}}_{max},\ \ a\backslash\infty = \infty/a = \infty \tag{1.7}$$

$$\forall b \in \mathbb{R}_{max},\ \ \infty\backslash b = b/\infty = \mathbf{0} \tag{1.8}$$

$$\forall b \in \bar{\mathbb{R}}_{max},\ \ \mathbf{0}\backslash b = b/\mathbf{0} = \infty \tag{1.9}$$

$$\forall a \in \mathbb{R} \cup \{\infty\},\ \ a\backslash\mathbf{0} = \mathbf{0}/a = \mathbf{0} \tag{1.10}$$

Le théorème général dû à Blyth [3] permet de calculer la résiduée de deux matrices. On utilise la notation $\wedge$ pour représenter l'opérateur inf.

Théorème 1.1.2
Soient $A \in \mathcal{M}_n(\bar{\mathbb{R}}_{max})$ et $B \in \mathcal{M}_n(\bar{\mathbb{R}}_{max})$.

$$[A\backslash B]_{\ell,c} = \bigwedge_{k=1}^{n} [A]_{k,\ell}\backslash[B]_{k,c} \tag{1.11}$$

$$[B/A]_{\ell,c} = \bigwedge_{k=1}^{n} [B]_{\ell,k}/[A]_{c,k} \tag{1.12}$$

Exemple 1.1.3
Soient les deux matrices

$$A = \begin{pmatrix} 2 & \mathbf{0} \\ \infty & 3 \end{pmatrix}, \quad B = \begin{pmatrix} \mathbf{0} & \infty \\ 1 & 4 \end{pmatrix}$$

On a alors

$$A\backslash B = \begin{pmatrix} 2\backslash\mathbf{0} \wedge \infty\backslash 1 & 2\backslash\infty \wedge \infty\backslash 4 \\ \mathbf{0}\backslash\mathbf{0} \wedge 3\backslash 1 & \mathbf{0}\backslash\infty \wedge 3\backslash 4 \end{pmatrix} = \begin{pmatrix} \mathbf{0}\wedge\mathbf{0} & \infty\wedge\mathbf{0} \\ \infty\wedge -2 & \infty\wedge 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -2 & 1 \end{pmatrix}$$

et cette matrice est la plus grande matrice X telle que $AX \preceq B$.

On a également

$$A/B = \begin{pmatrix} \mathbf{0}/2 \wedge \infty/\mathbf{0} & \mathbf{0}/\infty \wedge \infty/3 \\ 1/2 \wedge 4/\mathbf{0} & \mathbf{1}/\infty \wedge 4/3 \end{pmatrix} = \begin{pmatrix} \mathbf{0}\wedge\infty & \mathbf{0}\wedge\infty \\ -1\wedge\infty & \mathbf{0}\wedge 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ -1 & \mathbf{0} \end{pmatrix}$$

C'est la plus grande matrice X telle que $XA \preceq B$.

1.2 L'ordonnancement de flowshops

Cette section décrit plusieurs notions générales d'ordonnancement et donne une description du problème particulier qui nous intéresse, à savoir le problème de flowshop avec délais maximum et minimum.

1.2.1 Notions d'ordonnancement

L'ordonnancement de travaux est un domaine de recherche très important pour la santé et la compétitivité d'un grand nombre d'entreprises. Le problème revient à trouver un *ordre* selon lequel un nombre donné de travaux doivent être planifiés sur un ensemble précis de machines afin d'optimiser tel ou tel critère. En général, on considère quatre grands ensembles d'organisation de l'atelier :

- le *flowshop*, où tous les travaux suivent la même séquence sur chaque machine et où chaque travail possède exactement une opération ;
- l'*openshop*, similaire au flowshop, excepté le fait que les opérations d'un travail peuvent être exécutées dans n'importe quel ordre ;
- le *jobshop*, où les travaux peuvent suivre des séquences différentes entre elles sur les machines, et où un travail peut utiliser la même machine plusieurs fois ;
- les *machines parallèles*, où les travaux peuvent être exécutés sur n'importe quelle machine d'un groupe prédéfini.

Presqu'un quart des ateliers sont aujourd'hui conçus selon le modèle du flowshop. La résolution d'un problème classique du flowshop doit permettre l'ordonnancementde n travaux passant par m machines, permettant l'optimisation du makespan par exemple, noté C_{max}. La minimisation du C_{max} est connue pour être NP-difficile, excepté lorsque $m = 2$. On pourra également noter l'existance d'autres critères, le retard maximum par exemple.

Plusieurs contraintes peuvent s'ajouter au problème d'ordonnancement étudié. Par soucis de clarté, ne seront détaillées ici que les contraintes qui seront utilisées dans ce rapport :

- les temps de montage et de démontage : notés $s_{i,j}$ (*setup time*) et $r_{i,j}$ (*removal time*), ils dépendent du travail i et de la machine j considérés. Ces derniers symbolisent les temps nécessaire à la préparation de la machine avant et après l'exécution du travail.
- les délais ou décalages temporels : notés $a_{i,j}$, ils permettent de modéliser le délai existant entre le début de la phase opératoire d'un travail et son exécution effective. Ceux-ci peuvent être négatifs.

Rappelons également que Graham et al. [13] et Blazewicz et al. [2] ont proposé une notation a trois champs notée $\alpha/\beta/\gamma$ largement utilisée dans ce rapport. Le premier champ permet de connaître la nature du problème, le deuxième les critères étudiés, et le troisième le critère à optimiser.

1.2.2 Le problème de flowshop sans attente

Le problème de flowshop sans attente est particulier dans la mesure où il peut être modélisé comme un problème de voyageur de commerce. Il est possible de le formuler de la manière suivante : soient n travaux à ordonnancer sur m machines. Chaque travail doit être programmé sur chaque machine, dans l'ordre des machines, de 1 à m : on décompose donc le travail en m tâches. L'exécution de la tâche du travail j sur la machine k nécessite $p_{j,k}$ unités de temps. La préemption n'est pas autorisée, ainsi à partir du moment où une tâche a débuté sur l'une des machines, elle ne peut pas être interrompue. Chaque machine ne peut exécuter qu'une seule tâche à la fois. Pour chaque travail, le délai entre chaque tâche doit être nul. On remarquera qu'il s'agit ici d'un cas particulier du problème suivant.

Il existe une procédure par séparation et évaluation pour résoudre le problème de voyageur de commerce asymétrique, donnée par Carpaneto, Dell'Amico et Toth [7]. Cette méthode donne de très bons résultats, et un algorithme existe, permettant la résolution de problèmes de flowshop sans attente avec un grand nombre de travaux (jusqu'à 1000) en quelques minutes.

1.2.3 Le problème de flowshop avec délais minimum et maximum

De la même manière que précedemment, le problème de flowshop avec délais (minimum et/ou maximum) peut être formulé de la manière suivante : soient n travaux à ordonnancer sur m machines. Chaque travail doit être programmé sur chaque machine, dans l'ordre des machines, de 1 à m. L'exécution de la tâche du travail j sur la machine k nécessite $p_{j,k}$ unités de temps. La préemption n'est pas autorisée, ainsi à partir du moment où une tâche a débuté sur l'une des machines, elle ne peut pas être interrompue. Chaque machine ne peut exécuter qu'une seule tâche à la fois. Pour chaque travail, il existe des délais minimum et maximum entre chaque tâche successive. Ainsi, le temps d'attente entre deux opérations consécutives d'une tâche doit être plus grand que le délai minimum et plus petit que le délai maximum, tous deux connus. Ce temps d'attente est habituellement noté $[\alpha_{i,j}, \beta_{i,j}]$.

Si cet intervalle est égal à $[0, \infty]$ pour chaque tâche de chaque travail, on retrouve le problème conventionnel $Fm||C_{max}$. Si tous les intervalles sont tels que $\alpha_{i,j} = \beta_{i,j} = 0$ pour toutes les tâches de chaque travail, on retrouve le problème de flowshop sans attente décrit dans la section précédente.

1.2.4 Pourquoi étudier l'ordonnancement ?

Comme cela a été dit brievement en introduction, les principales motivations pour l'étude de tels problèmes particuliers proviennent du milieu industriel. Des temps morts maximum apparaîtront lors de la modélisation de situations où les délais entre les opérations ne doivent pas être trop longs afin d'éviter la détérioration de produits. Par exemple, dans l'industrie agro-alimentaire impliquant des denrées perissables, à partir du moment où la nourriture a été préparée et cuisinée, celle-ci doit être emballée et conditionnée avant qu'un certain laps de temps ne s'écoule, sinon elle sera perdue.

Les délais minimum interviennent lorsque des temps d'attente sont imposés. On peut citer l'exemple d'un laboratoire médical entièrement automatisé, où plusieurs analyses sont réalisées.

Les durées des réactions chimiques sont bornées, ainsi les opérations de manipulations sont strictement minutées. Ces dernières réactions sont ainsi modélisées par des délais maximum et minimum.

C'est pour ces dernières raisons qu'il est intéressant d'étudier et d'appronfondir nos connaissance en la matière, et plus précisément au sujet des problèmes de flowshop avec délais minimum et maximum ou sans attente. Nous nous pencherons comme cela a été dit dans l'introduction sur la résolution du problème de flowshiop avec délais au moyen d'une procédure par séparation et évaluation.

Chapitre 2

Le flowshop avec délais minimum et maximum

Ce chapitre a pour but de détailler la modélisation Max-Plus d'un flowshop avec délais minimum et maximum, d'abord à deux machines, puis à m machines. Le problème $Fm|delai\ min\text{-}max|C_{max}$ est, comme nous allons le voir, NP-difficile, même dans le cas trivial à deux machines.

2.1 Le flowshop avec délais minimum et maximum à deux machines

2.1.1 Matrice associée à un travail

Le problème étudié ici comporte n travaux. Chaque travail x se décompose en deux opérations de durées opératoires $p_{x,1}$ et $p_{x,2}$. Le temps d'attente entre ces deux opérations doit être supérieur ou égal à α_x et inférieur ou égal à β_x ($\alpha_x \leq \beta_x$). Le critère à minimiser est le makespan (C_{max}).

Notre étude portera sur des ordonnancements de permutation, le problème traité ici deviendra donc un $F2|delai\ min-max, pmtn|C_{max}$. Si les machines sont disponibles aux dates t_1 et t_2, les tâches du travail à ordonnancer x se terminerons aux dates $C_{x,1}$ et $C_{x,2}$, définies comme suit :

$$\begin{cases} C_{x,1} = t_1 p_{x,1} \oplus \dfrac{t_2}{\beta_x} \\ C_{x,2} = t_1 p_{x,1} \alpha_x p_{x,2} \oplus t_2 p_{x,2} \end{cases} \tag{2.1}$$

Ces deux équations correspondent aux cas illustrés par les figure 2.1, 2.2 et 2.3.

En posant $\vec{C}_x = (C_{x,1}, C_{x,2})$ et $\vec{t} = (t_1, t_2)$, cela devient

$$\vec{C}_x = \vec{t} \begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} \\ \mathbf{1} & p_{x,2} \end{pmatrix} \tag{2.2}$$

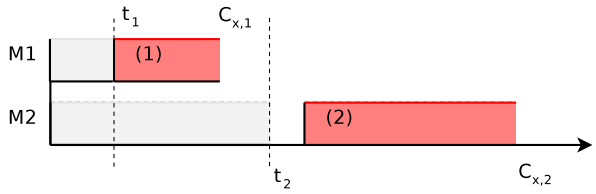


FIG. 2.1 – $F2|delai\ min - max|C_{max}$: premier cas.

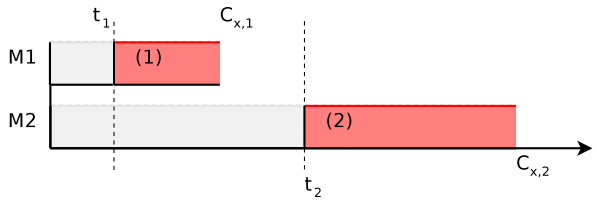


FIG. 2.2 – $F2|delai\ min - max|C_{max}$: deuxième cas.

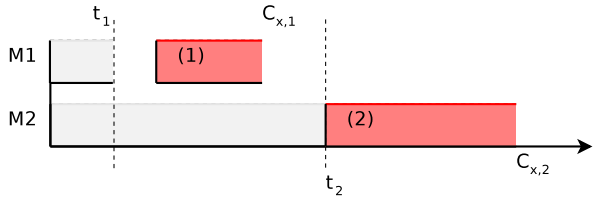


FIG. 2.3 – $F2|delai\ min - max|C_{max}$: troisième cas.

Ce qui permet d'énoncer les définitions et propriétés suivantes.

Définition 2.1.1
On appelle matrice associée à un travail la matrice suivante :

$$T(x) = \begin{pmatrix} p_{x,1} & p_{x,1}\alpha_x p_{x,2} \\ \dfrac{\mathbf{1}}{\beta_x} & p_{x,2} \end{pmatrix} \tag{2.3}$$

Par exemple, les travaux du problème $F2||C_{max}$ ont leurs α_x égaux à $\mathbf{1}$ et les β_x égaux à $+\infty$. Ainsi, la matrice associée à un travail sera de la forme suivante :

$$M_x = \begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} \\ \mathbf{0} & p_{x,2} \end{pmatrix}$$

De la même façon, les travaux du problème $F2|no-wait|C_{max}$ ont leurs α_x et β_x égaux à $\mathbf{1}$. Ainsi, la matrice associée à un travail sera de la forme :

$$M_x = \begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} \\ \mathbf{1} & p_{x,2} \end{pmatrix}$$

Il est possible de simplifier la forme de la matrice associée à un travail. En effet, on remarque que :

$$\begin{pmatrix} p_{x,1} & p_{x,1}\alpha_x p_{x,2} \\ \dfrac{\mathbf{1}}{\beta_x} & p_{x,2} \end{pmatrix} = \frac{\mathbf{1}}{\alpha_x} \begin{pmatrix} \alpha_x p_{x,1} & (\alpha_x p_{x,1})(\alpha_x p_{x,2}) \\ \dfrac{\alpha_x}{\beta_x} & \alpha_x p_{x,2} \end{pmatrix}$$

Si l'on pose $p'_{x,1} = \alpha_x p_{x,1}$, $p'_{x,2} = \alpha_x p_{x,2}$ et $\beta'_x = \dfrac{\beta_x}{\alpha_x}$, il vient :

$$\begin{pmatrix} p_{x,1} & p_{x,1}\alpha_x p_{x,2} \\ \dfrac{\mathbf{1}}{\beta_x} & p_{x,2} \end{pmatrix} = \frac{\mathbf{1}}{\alpha_x} \begin{pmatrix} p'_{x,1} & p'_{x,1}p'_{x,2} \\ \dfrac{\mathbf{1}}{\beta'_x} & p'_{x,2} \end{pmatrix}$$

Finalement, on peut dire que le travail $(p_{x,1}, p_{x,2}, \alpha_x, \beta_x)$ peut être remplacé par le travail $\left(\alpha_x p_{x,1}, \alpha_x p_{x,2}, \mathbf{1}, \dfrac{\beta_x}{\alpha_x}\right)$. On en déduit alors le théorème suivant.

Théorème 2.1.1
Le problème $F2|delai\ min-max, pmtn|C_{max}$, pour lequel les données sont $(p_{x,1}, p_{x,2}, \alpha_x, \beta_x)$, est équivalent au problème $F2|delai\ max, pmtn|C_{max}$, pour lequel les données sont $\left(\alpha_x p_{x,1}, \alpha_x p_{x,2}, \mathbf{1}, \dfrac{\beta_x}{\alpha_x}\right)$.

Ainsi, nous ne considérerons dans la suite que les problèmes où les délais minimum sont nuls.

2.1.2 Matrice associée à une séquence

Nous allons voir dans cette section comment les résultats présentés plus haut s'étendent à des séquences de travaux.

Définition 2.1.2
On appelle matrice associée à une séquence σ la matrice

$$T(\sigma) = \bigotimes_{i=1}^{|\sigma|} T(\sigma(i)) \tag{2.4}$$

De plus, on pose $T_i(\sigma) = T(\sigma(i))$.

Propriété 2.1.1
Si les machines sont disponibles aux dates $\vec{t} = (t_1, t_2)$, le vecteur $\vec{C}_\sigma = (C_{|\sigma|,1}, C_{|\sigma|,2})$ des dates de fin au plus tôt d'une séquence σ est donné par l'équation

$$\vec{C}_\sigma = \vec{t} \otimes T(\sigma) \tag{2.5}$$

2.1.3 Minoration de la matrice associée à une séquence

L'objectif est ici de trouver une minoration de la matrice associée à une séquence. MM. Lenté et Bouquard ont donné trois manières de procéder dans [5] ; nous ne décrirons ici que la méthode relative au flowshop sans attente.

Si β_j n'est pas l'infini, soient $c_{x,1}$ et $c_{x,2}$ deux nombres non négatifs tels que $c_{x,1}c_{x,2} = \beta_x$. On remarque que :

$$c_{x,1} \geq \mathbf{1} \Rightarrow p_{x,1} \geq \frac{p_{x,1}}{c_{x,1}} \tag{2.6}$$

De la même façon :

$$c_{x,2} \geq \mathbf{1} \Rightarrow p_{x,2} \geq \frac{p_{x,2}}{c_{x,2}} \tag{2.7}$$

De plus :

$$p_{x,1}p_{x,2} = \frac{p_{x,1}}{c_{x,1}}\beta_x\frac{p_{x,2}}{c_{x,2}} \tag{2.8}$$

Finalement, il vient :

$$\begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} \\ \dfrac{\mathbf{1}}{\beta_x} & p_{x,2} \end{pmatrix} \geq \begin{pmatrix} \dfrac{p_{x,1}}{c_{x,1}} & \dfrac{p_{x,1}}{c_{x,1}}\beta_x\dfrac{p_{x,2}}{c_{x,2}} \\ \dfrac{\mathbf{1}}{\beta_x} & \dfrac{p_{x,2}}{c_{x,2}} \end{pmatrix}$$

$$\begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} \\ \dfrac{\mathbf{1}}{\beta_x} & p_{x,2} \end{pmatrix} \geq \frac{\mathbf{1}}{\beta_x} \begin{pmatrix} \dfrac{p_{x,1}\beta_x}{c_{x,1}} & \dfrac{p_{x,1}\beta_x}{c_{x,1}}\dfrac{p_{x,2}\beta_x}{c_{x,2}} \\ \mathbf{1} & \dfrac{p_{x,2}\beta_x}{c_{x,2}} \end{pmatrix} \tag{2.9}$$

Cette dernière matrice est une matrice sans attente que nous appellerons T_x^{NW} :

$$\forall x \; travail, \; T_x \geq \frac{\mathbf{1}}{\beta_x} T_x^{NW} \tag{2.10}$$

Ainsi, une borne inférieure peut être obtenue en résolvant un problème sans-attente par la méthode de Gilmore et Gomory [11].

Afin d'obtenir une borne inférieure, il est indispensable de définir les valeurs de $c_{x,1}$ et $c_{x,2}$. Ces valeurs nous permettent de passer des éléments de la diagonale $p_{x,1}$ et $p_{x,2}$ à $\frac{p_{x,1}}{c_{x,1}}$ et $\frac{p_{x,2}}{c_{x,2}}$. Puisque $c_{x,1}$ et $c_{x,2}$ sont positifs, ces derniers peuvent être vus comme des valeurs diminuées de $p_{x,1}$ et $p_{x,2}$.

MM. Bouquard et Lenté proposent cinq stratégies :

- si $p_{x,1} \leq p_{x,2}$ alors $c_{x,1} = \beta_x$ et $c_{x,2} = \mathbf{1}$, sinon $c_{x,1} = \mathbf{1}$ et $c_{x,2} = \beta_x$;
- si $p_{x,1} \geq p_{x,2}$ alors $c_{x,1} = \beta_x$ et $c_{x,2} = \mathbf{1}$, sinon $c_{x,1} = \mathbf{1}$ et $c_{x,2} = \beta_x$;
- si $\sum_{x=1}^{n} p_{x,1} \leq \sum_{x=1}^{n} p_{x,2}$ alors $c_{x,1} = \beta_x$ et $c_{x,2} = \mathbf{1}$, sinon $c_{x,1} = \mathbf{1}$ et $c_{x,2} = \beta_x$;
- si $\sum_{x=1}^{n} p_{x,1} > \sum_{x=1}^{n} p_{x,2}$ alors $c_{x,1} = \beta_x$ et $c_{x,2} = \mathbf{1}$, sinon $c_{x,1} = \mathbf{1}$ et $c_{x,2} = \beta_x$;
- $c_{x,1} = c_{x,2} = \frac{\beta_x}{2}$.

Chacune de ces stratégies donne une borne inférieure.

2.2 Cas général

Nous allons maintenant nous pencher sur la modélisation Max-Plus d'un flowshop de permutation avec délais minimum et maximum ainsi que les propriétés qui en découlent.

2.2.1 Notations

Dans le cas à m machine, nous devons définir les temps d'attente pour chaque paire de tâches de chaque travail. Ainsi, pour un problème à m machines et pour un travail x, on définit $m-1$ bornes inférieures $\alpha_{x,j}$ et $m-1$ bornes supérieures $\beta_{x,j}$, avec $\alpha_{x,1}$ et $\beta_{x,1}$ des délais minimum et maximum entre les deux premières tâche du travail x, et $\alpha_{x,m-1}$ et $\beta_{x,m-1}$ des délais minimum et maximum entre les deux dernières tâche du même travail x.

De la même manière que précedemment, on ne se penchera ici que sur le cas du flowshop de permutation, noté $Fm|delai\ min-max, pmtn|C_{max}$.

2.2.2 Matrice associée à un travail

Il s'agit ici de généraliser les résultats décrits dans la section précédente au cas à m machines. Pour cela, il suffit d'énumérer tous les cas de figures possibles. Dans le cas à trois machines par exemple, il existe six cas de figure à traiter. Si les machines sont disponibles aux dates t_1, t_2 et t_3, les dates de fin des tâches du travail x sont définies de la manière suivante :

$$\begin{array}{ll} C_1 \geq t_1 + p_{x,1} & C_2 \geq t_2 + p_{x,2} \\ C_1 \geq t_2 - \beta_{x,1} & C_2 \geq t_3 - \beta_{x,2} \\ C_1 \geq t_3 - \beta_{x,2} - p_{x,2} - \beta_{x,1} & C_2 \geq t_1 + p_{x,1} + \alpha_{x,1} + p_{x,2} \end{array}$$

$$\begin{array}{l} C_3 \geq t_3 + p_{x,3} \\ C_3 \geq t_2 - p_{x,2} + \alpha_{x,2} + p_{x,3} \\ C_3 \geq t_1 + p_{x,1} + \alpha_{x,1} + p_{x,2} + \alpha_{x,2} + p_{x,3} \end{array}$$

On en déduit aisément l'expression Max-Plus de ces dates de fin :

$$C_1 = t_1 p_{x,1} \oplus \frac{t_2}{\beta_{x,1}} \oplus \frac{t_3}{\beta_{x,1} p_{x,2} \beta_{x,2}}$$

$$C_2 = t_2 p_{x,2} \oplus \frac{t_3}{\beta_{x,2}} \oplus t_1 p_{x,1} \alpha_{x,1} p_{x,2}$$

$$C_3 = t_3 p_{x,3} \oplus t_2 p_{x,2} \alpha_{x,2} p_{x,3} \oplus t_1 p_{x,1} \alpha_{x,1} p_{x,2} \alpha_{x,2} p_{x,3}$$

Il est possible d'exprimer ces résultats sous forme matricielle.

En posant $\vec{C_x} = (C_{x,1}, C_{x,2}, C_{x,3})$ et $\vec{t} = (t_1, t_2, t_3)$, cela devient

$$\vec{C_x} = \vec{t} \begin{pmatrix} p_{x,1} & p_{x,1}\alpha_{x,1}p_{x,2} & p_{x,1}\alpha_{x,1}p_{x,2}\alpha_{x,2}p_{x,3} \\ \mathbf{1}/\beta_{x,1} & p_{x,2} & p_{x,2}\alpha_{x,2}p_{x,3} \\ \mathbf{1}/(\beta_{x,1}p_{x,2}\beta_{x,2}) & \mathbf{1}/\beta_{x,2} & p_{x,3} \end{pmatrix} \tag{2.11}$$

Penchons-nous maintenant sur le cas à m machines.

Définition 2.2.1
A tout travail x est associée une matrice de durées d'exécution P_x, telle que :

$$P_x = P(x) = \begin{pmatrix} p_{x,1} & p_{x,1}\alpha_{x,1}p_{x,2} & \cdots & p_{x,1}\alpha_{x,1}p_{x,2}\cdots p_{x,m} \\ \beta_{x,1}^{-1} & p_{x,2} & \cdots & p_{x,2}\alpha_{x,2}\cdots p_{x,m} \\ \vdots & \vdots & \ddots & \vdots \\ (\beta_{x,1}p_{x,2}\beta_{x,2}\cdots\beta_{x,m-1})^{-1} & (\beta_{x,2}p_{x,3}\cdots\beta_{x,m-1})^{-1} & \cdots & p_{x,m} \end{pmatrix} \tag{2.12}$$

ou plus formellement :

$$\forall(\ell,c)\in\{1,\dots,m\}^2,\ [P_x]_{\ell,c}=\begin{cases}\beta_{x,\ell}^{-1}\bigotimes_{k=c+1}^{\ell-1}(\beta_{x,k-1}p_{x,k})^{-1}\ si\ \ell>c\\ p_{x,\ell}\bigotimes_{k=\ell+1}^{c}(\alpha_{x,k-1}p_{x,k})\ si\ \ell\leq c\end{cases}\tag{2.13}$$

2.2.3 Matrices associées à une séquence

Définition 2.2.2

Soit σ une séquence.

- *$T_i(\sigma)=T(\sigma(i))$ est la matrice associée au i^e travail de σ ;*
- *$\vec{C}_{i,.}(\sigma)=\big(C_{i,1}(\sigma),\dots,C_{i,m}(\sigma)\big)$ est le vecteur des dates de fin des opérations du i^e travail de σ et $\vec{C}_\sigma=\vec{C}_{|\sigma|,.}(\sigma)$ est le vecteur des dates de fin de démontage des opérations du dernier travail de σ.*

Définition 2.2.3 *A toute séquence σ peut être associée la matrice :*

$$T(\sigma)=\bigotimes_{i=1}^{|\sigma|}T(\sigma(i))=\bigotimes_{i=1}^{|\sigma|}T_i(\sigma)$$

Propriété 2.2.1

Soit σ la concaténation de deux séquences σ_1 et σ_2. Il vient alors :

$$T(\sigma)=T(\sigma_1\sigma_2)=T(\sigma_1)T(\sigma_2)\tag{2.14}$$

Preuve 2.2.1

Cette propriété est une conséquence de l'associativité du produit matriciel.

□

Propriété 2.2.2

$\forall\sigma$ *séquence,* $\forall i\in\{1,\dots,|\sigma|\},\ \vec{C}_{i,.}(\sigma)=\vec{C}_{i-1,.}(\sigma)\otimes T_i(\sigma)$

Propriété 2.2.3

Soit $\vec{t}$ le vecteur des dates de disponibilité au plus tôt des machines, les dates de fin d'une séquence σ vérifient :

$$\vec{C}_\sigma=\vec{t}\otimes T(\sigma)\tag{2.15}$$

Preuve 2.2.2

Cette propriété est un corollaire de la propriété précédente.

□

Corollaire 2.2.1

Si toutes les machines sont libres à la date 0, les dates de fin des opérations du dernier travail d'une séquence σ sont :

$$\vec{C}_\sigma=\big(\mathbf{1}\dots\mathbf{1}\big)T(\sigma)\tag{2.16}$$

Corollaire 2.2.2

Pour toute séquence σ, on a :

$$C_{max}(\sigma) = \big(\mathbf{1} \dots \mathbf{1}\big) T(\sigma) \begin{pmatrix} \mathbf{1} \\ \vdots \\ \mathbf{1} \end{pmatrix} \tag{2.17}$$

2.2.4 Minoration de la matrice associée à une séquence

Comme cela a été vu précedemment, l'objectif est ici de trouver une minoration de la matrice associée à une séquence dans le cas à m machines. Nous allons décrire une méthode similaire à celle employée dans le cas à 2 machines.

Penchons-nous tout d'abord sur le cas d'un problème à 3 machines, afin d'appréhender la stratégie de minoration dans le cas général.

Minoration d'une matrice de dimension 3

Soit x un travail, soit T_x sa matrice associée. Si $\beta_{x,1}$ et $\beta_{x,2}$ ne sont pas égaux à l'infini, on définit les nombres a_1, a_2, b_1, b_2 et b_3 de la manière suivante :

$$b_1 \geq \mathbf{1},\ b_2 \geq \mathbf{1},\ b_3 \geq \mathbf{1}$$

$$\frac{\mathbf{1}}{a_1} \leq \frac{\mathbf{1}}{\beta_1},\ \frac{\mathbf{1}}{a_2} \leq \frac{\mathbf{1}}{\beta_2},\ \frac{b_2}{a_1 a_2} \leq \frac{\mathbf{1}}{\beta_1 \beta_2}$$

$$\frac{a_1}{b_1 b_2} \leq \alpha_1,\ \frac{a_2}{b_2 b_3} \leq \alpha_2,\ \frac{a_1 a_2}{b_1 b_2 b_3} \leq \alpha_1 \alpha_2$$

D'après les conditions énoncées ci-dessus, il vient :

$$\forall j \in \{1, \dots, 3\},\ p_{x,j} \geq \frac{p_{x,j}}{b_{x,j}} \tag{2.18}$$

On peut ainsi écrire l'inégalité matricielle suivante :

$$T_x = \begin{pmatrix} p_{x,1} & p_{x,1}\alpha_{x,1}p_{x,2} & p_{x,1}\alpha_{x,1}p_{x,2}\alpha_{x,2}p_{x,3} \\ \dfrac{\mathbf{1}}{\beta_{x,1}} & p_{x,2} & p_{x,2}\alpha_{x,2}p_{x,3} \\ \dfrac{\mathbf{1}}{\beta_{x,1}p_{x,2}\beta_{x,2}} & \dfrac{\mathbf{1}}{\beta_{x,2}} & p_{x,3} \end{pmatrix} \geq$$

$$\begin{pmatrix} \frac{p_{x,1}}{b_{x,1}} & \frac{p_{x,1}}{b_{x,1}}a_{x,1}\frac{p_{x,2}}{b_{x,2}} & \frac{p_{x,1}}{b_{x,1}}a_{x,1}\frac{p_{x,2}}{b_{x,2}}a_{x,2}\frac{p_{x,3}}{b_{x,3}} \\ \frac{\mathbf{1}}{a_{x,1}} & \frac{p_{x,2}}{b_{x,2}} & \frac{p_{x,2}}{b_{x,2}}a_2\frac{p_{x,3}}{b_{x,3}} \\ \frac{b_{x,2}}{a_{x,1}p_{x,2}a_{x,2}} & \frac{\mathbf{1}}{a_{x,2}} & \frac{p_{x,3}}{b_{x,3}} \end{pmatrix}$$

Cette dernière matrice est une matrice sans attente ; on remarquera cependant que les durées opératoires du nouveau problème ainsi créé peuvent devenir négatives en fonction des valeurs des $b_{x,i}$. Il est possible de remédier à ce travers en posant $\Pi_x = b_{x,1} \oplus b_{x,2} \oplus b_{x,3}$. Il vient alors :

$$T_x \geq \frac{\mathbf{1}}{\Pi_x} \begin{pmatrix} \frac{\Pi_x p_{x,1}}{b_{x,1}} & \frac{\Pi_x p_{x,1}}{b_{x,1}}\frac{a_{x,1}}{\Pi_x}\frac{\Pi_x p_{x,2}}{b_{x,2}} & \frac{\Pi_x p_{x,1}}{b_{x,1}}\frac{a_{x,1}}{\Pi_x}\frac{\Pi_x p_{x,2}}{b_{x,2}}\frac{a_{x,2}}{\Pi_x}\frac{\Pi_x p_{x,3}}{b_{x,3}} \\ \frac{\Pi_x}{a_{x,1}} & \frac{\Pi_x p_{x,2}}{b_{x,2}} & \frac{\Pi_x p_{x,2}}{b_{x,2}}\frac{a_{x,2}}{\Pi_x}\frac{\Pi_x p_{x,3}}{b_{x,3}} \\ \frac{\Pi_x^2 b_{x,2}}{a_{x,1}\Pi_x p_{x,2}a_{x,2}} & \frac{\Pi_x}{a_{x,2}} & \frac{\Pi_x p_{x,3}}{b_{x,3}} \end{pmatrix}$$

Cette dernière matrice est une matrice sans attente que nous appellerons T_x^{NW} :

$$\forall x \; travail, \; T_x \geq \frac{\mathbf{1}}{\Pi_x} \; T_x^{NW} \tag{2.19}$$

Il s'agit maintenant d'injecter cette inégalité dans le produit T_τ, où $T_{\sigma\tau} = T_\sigma T_\tau$:

$$\begin{aligned} T_\tau &= \bigotimes_{j=1}^{n-p} T_{\tau(j)} \\ &\geq \bigotimes_{j=1}^{n-p} \left(\frac{\mathbf{1}}{\Pi_x} T_x^{NW} \right) \\ &\geq \left(\bigotimes_{j=1}^{n-p} \frac{\mathbf{1}}{\Pi_x} \right) \bigotimes_{j=1}^{n-p} T_x^{NW} \end{aligned}$$

Minoration dans le cas général

Il est possible de généraliser très aisément au cas à m machines.

Soit x un travail. Si l'ensemble des $\beta_{x,j}$ n'est pas l'infini, soient 2 vecteurs $(b_{x,1}, b_{x,2}, \ldots, b_{x,m})$ et $(a_{x,1}, a_{x,2}, \ldots, a_{x,m-1})$ de nombres non négatifs définis de la manière suivante :

$$\forall i \in \{1, \ldots, m-1\}, \; \frac{\mathbf{1}}{a_i} \leq \frac{\mathbf{1}}{\beta_i}$$

$$\forall (i,j) \in \{1,\dots,m-1\} \times \{2,\dots,m\},\ i<j,\ \frac{\bigotimes_{k=i}^{j-1} a_k}{\bigotimes_{k=i}^{j} b_k} \leq \bigotimes_{k=i}^{j-1} \alpha_k$$

On admet que $\forall j \in \{1,\dots,m\},\ b_{x,j} \geq \mathbf{1}$. Il vient alors :

$$\forall j \in \{1,\dots,m\},\ p_{x,j} \geq \frac{p_{x,j}}{b_{x,j}} \tag{2.20}$$

De plus :

$$\forall (i,j) \in \{1,\dots,m-1\} \times \{2,\dots,m\},\ \bigotimes_{k=i}^{j} p_{x,k} \geq \frac{p_{x,i}}{b_{x,i}} \bigotimes_{k=i}^{j-1} a_{x,k} \frac{p_{x,k+1}}{b_{x,k+1}} \tag{2.21}$$

Finalement, il vient :

$$T_x = \begin{pmatrix} p_{x,1} & p_{x,1}p_{x,2} & \cdots & p_{x,1}p_{x,2}\cdots p_{x,m} \\ \beta_{x,1}^{-1} & p_{x,2} & \cdots & p_{x,2}\cdots p_{x,m} \\ \vdots & \vdots & \ddots & \vdots \\ (\beta_{x,1}p_{x,2}\dots\beta_{x,m-1})^{-1} & (\beta_{x,2}p_{x,3}\dots\beta_{x,m-1})^{-1} & \cdots & p_{x,m} \end{pmatrix} \geq$$

$$\begin{pmatrix} \frac{p_{x,1}}{b_{x,1}} & \frac{p_{x,1}}{b_{x,1}}a_{x,1}\frac{p_{x,2}}{b_{x,2}} & \cdots & \frac{p_{x,1}}{b_{x,1}}a_{x,1}\frac{p_{x,2}}{b_{x,2}}a_{x,2}\cdots\frac{p_{x,m}}{b_{x,m}} \\ \frac{\mathbf{1}}{a_{x,1}} & \frac{p_{x,2}}{b_{x,2}} & \cdots & \frac{p_{x,2}}{b_{x,2}}a_{x,2}\cdots\frac{p_{x,m}}{b_{x,m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{b_{x,2}\cdots b_{x,m-1}}{a_{x,1}p_{x,2}a_{x,2}\cdots a_{x,m-1}} & \frac{b_{x,3}\cdots b_{x,m-1}}{a_{x,2}p_{x,3}a_{x,3}\cdots a_{x,m-1}} & \cdots & \frac{p_{x,m}}{b_{x,m}} \end{pmatrix}$$

$$T_x \geq \frac{\mathbf{1}}{\Pi_x} \begin{pmatrix} \frac{\Pi_x p_{x,1}}{b_{x,1}} & \frac{\Pi_x p_{x,1}}{b_{x,1}} \frac{a_{x,1}}{\Pi_x} \frac{\Pi_x p_{x,2}}{b_{x,2}} & \cdots & \frac{\Pi_x p_{x,1}}{b_{x,1}} \frac{a_{x,1}}{\Pi_x} \frac{\Pi_x p_{x,2}}{b_{x,2}} \frac{a_{x,2}}{\Pi_x} \cdots \frac{\Pi_x p_{x,m}}{b_{x,m}} \\ \frac{\Pi_x}{a_{x,1}} & \frac{\Pi_x p_{x,2}}{b_{x,2}} & \cdots & \frac{\Pi_x p_{x,2}}{b_{x,2}} \frac{a_{x,2}}{\Pi_x} \cdots \frac{\Pi_x p_{x,m}}{b_{x,m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\Pi_x^{m-1} b_{x,2} \dots b_{x,m-1}}{a_{x,1} \Pi_x p_{x,2} a_{x,2} \dots a_{x,m-1}} & \frac{\Pi_x^{m-2} b_{x,3} \dots b_{x,m-1}}{a_{x,2} \Pi_x p_{x,3} a_{x,3} \dots a_{x,m-1}} & \cdots & \frac{\Pi_x p_{x,m}}{b_{x,m}} \end{pmatrix}$$

$$\text{avec } \Pi_x = \bigoplus_{i=1}^{m} b_{x,i}.$$

Cette dernière matrice est une matrice sans attente que nous appellerons T_x^{NW} :

$$\forall x \; travail, \; T_x \geq \frac{\mathbf{1}}{\Pi_x} T_x^{NW} \tag{2.22}$$

La minoration s'effectue de la même manière que dans le cas particulier du problème à 3 machines. On en déduit ainsi une borne inférieure qui peut être obtenue en résolvant le problème sans attente associé décrit ci-dessus. Pour cela, nous utiliserons une procédure par séparation et évaluation existante dédiée à la résolution du problème de voyageur de commerce. Cependant, il faut noter que l'obtention d'une borne inférieure est assujétie à la détermination de plusieurs paramètres, comme nous allons le voir dans le chapitre suivant.

Chapitre 3

Procédure par séparation et évaluation

Une procédure par séparation et évaluation a été développée par MM. Jean-Louis Bouquard et Christophe Lenté afin d'obtenir une résolution du problème de flowshop avec délais minimum et maximum dans le cas particulier à deux machines ; se référer à [5] pour plus de détails. Ce chapitre a pour but de présenter et de détailler l'implémentation de la PSE dans le cas à trois machines ou plus.

Une procédure par séparation et évaluation [12, 6] est une méthode exacte de résolution qui est souvent représentée sous forme arborescente et qui peut s'adapter à de nombreux problèmes combinatoires dont font partie les problèmes d'ordonnancement. Les grands principes d'une PSE ne seront pas décrit ici étant donné qu'ils ont été maintes fois cité dans plusieurs ouvrages. Le lecteur pourra par exemple se reporter à [16] pour plus de détails.

3.1 Stratégie de branchement

Le schéma de branchement est basé sur une séquence partielle. Le noeud racine contient la séquence vide et correspond à l'ensemble de toutes les solutions possibles pour le problème considéré. Pour chacun de ses n fils, un travail est choisi pour être le premier élément de la séquence. Ainsi, à l'étage p, le nœud $(J_{\sigma(1)}, \ldots, J_{\sigma(p)})$ possède $n-p$ fils correspondant au choix du $(p+1)^e$ travail.

On notera σ la séquence partielle correspondant au nœud courant, p sa longueur ($p = |\sigma|$) et $\bar{\sigma}$ l'ensemble des travaux restant. τ désignera toutes les permutations de l'ensemble $\bar{\sigma}$ de manière à ce que $\sigma\tau$ soit une séquence complète.

3.2 Borne inférieure

Le chapitre précédent nous a permis de trouver une méthode de calcul afin de déterminer une borne inférieure pour toute séquence. Il s'agissait de se ramener à un problème de flowshop sans attente à m machines. Bien que ce problème soit également NP-difficile, une procédure par séparation et évaluation très performante existe pour résoudre ce dernier [7]. Il s'agit ainsi

d'appeller ce dernier programme à chaque nœud afin d'obtenir une borne inférieure, ainsi que l'ordonnancement associé.

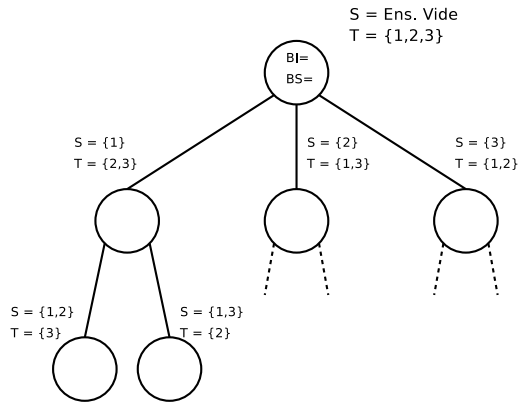


FIG. 3.1 – Schéma représentatif de la PSE.

Au nœud racine, il suffit de résoudre le problème sans attente avec l'intégralité des travaux pour obtenir la borne inférieure ainsi que l'ordonnancement optimal associé. Dans le cas d'un fils où un ordonnancement fixé existe, le calcul de borne inférieure est appliqué uniquement sur le problème avec les travaux restants à ordonnancer ; la borne inférieure pour ce nœud sera obtenue en additionnant simplement la date de disponibilité de la première machine avec la valeur du C_{max} obtenue lors de la résolution du problème sans attente.

Par exemple, soit N un nœud (non racine) et soient S et T respectivement les ensembles des travaux ordonnancés et non encore ordonnancés. Le placement des travaux ordonnancés permet d'affirmer que la première machine sera disponible à la date d_1. D'après la description donnée ci-dessus, on aura $B_{inf} = d_1 + C^{NW}_{max}(T)$.

Il est possible d'affiner ce résultat en tenant compte du premier temps mort juste après la date de disponibilité de la première machine.

Il est clair que le calcul du C_{max} pour le problème sans attente dépend fortement du choix des constantes décrites dans le chapitre précédent. Afin de résoudre ce problème, un solveur (en l'occurrence GLPK) est utilisé afin d'obtenir les valeurs des paramètres à fixer dans le cadre de la résolution du problème sans attente. On se ramène ainsi à la résolution d'un problème linéaire simple en amont de la résolution du problème d'ordonnancement qui se présente à nous.

Prenons le cas particulier du problème à trois machines. Soit x un travail non encore ordonnancé. Ici, il est nécessaire de fixer cinq paramètres, les contraintes ayant été définies naïvement de la manière suivante :

$$\begin{cases} b_{x,1} \geq 0 \\ b_{x,2} \geq 0 \\ b_{x,3} \geq 0 \\ a_{x,1} \geq \beta_{x,1} \\ a_{x,2} \geq \beta_{x,2} \\ a_{x,1} - b_{x,1} - b_{x,2} \leq \alpha_{x,1} \\ a_{x,2} - b_{x,2} - b_{x,3} \leq \alpha_{x,2} \\ a_{x,1} + a_{x,2} - b_{x,1} - b_{x,2} - b_{x,3} \leq \alpha_{x,1} + \alpha_{x,2} \\ b_{x,2} - a_{x,1} - a_{x,2} \leq -\beta_{x,1} - \beta_{x,2} \end{cases}$$

Le problème linéaire peut être reformulé de la manière suivante :

Maximiser

$$Z = -b_{x,1} - b_{x,3}$$

sous contraintes

$$\begin{aligned} a_{x,1} - b_{x,1} - b_{x,2} &\leq \alpha_{x,1} \\ a_{x,2} - b_{x,2} - b_{x,3} &\leq \alpha_{x,2} \\ a_{x,1} + a_{x,2} - b_{x,1} - b_{x,2} - b_{x,3} &\leq \alpha_{x,1} + \alpha_{x,2} \\ b_{x,2} - a_{x,1} - a_{x,2} &\leq -\beta_{x,1} - \beta_{x,2} \end{aligned}$$

les variables étant bornées de la manière suivante

$$a_{x,1} \geq \beta_{x,1},\ a_{x,2} \geq \beta_{x,2},\ b_{x,1} \geq 0,\ b_{x,2} \geq 0,\ b_{x,3} \geq 0$$

L'ajout de variables auxiliaires permet de modifier la forme du problème linéaire afin de le placer en entrée du solveur GLPK. Le problème devient alors :

Maximiser

$$Z = -b_{x,1} - b_{x,3}$$

sous contraintes

$$\begin{aligned} r &= a_{x,1} - b_{x,1} - b_{x,2} \\ s &= a_{x,2} - b_{x,2} - b_{x,3} \\ t &= a_{x,1} + a_{x,2} - b_{x,1} - b_{x,2} - b_{x,3} \\ u &= b_{x,2} - a_{x,1} - a_{x,2} \end{aligned}$$

les variables étant bornées de la manière suivante

$$\begin{array}{ll} -\infty < r \leq \alpha_{x,1} & \beta_{x,1} \leq a_{x,1} < \infty \\ -\infty < s \leq \alpha_{x,2} & \beta_{x,2} \leq a_{x,2} < \infty \\ -\infty < t \leq \alpha_{x,1} + \alpha_{x,2} & 0 \leq b_{x,1} < \infty \\ -\infty < u \leq -\beta_{x,1} - \beta_{x,2} & 0 \leq b_{x,2} < \infty \\ & 0 \leq b_{x,3} < \infty \end{array}$$

La fonction objectif définie ici n'est qu'un exemple. En effet, il est maintenant nécessaire de trouver une fonction objectif à maximiser permettant de maximiser le C_{max} qui sera obtenu lors de la résolution du problème sans attente. On en déduit qu'il est possible de calculer autant de bornes inférieures qu'il existe de manières de calculer nos paramètres d'entrée.

Ramenons-nous maintenant au problème sans attente et examinons l'expression du C_{max} d'un tel problème. Rappelons à cette occasion la définition concernant la modélisation des temps morts dans un problème de flowshop sans attente à trois machines.

Définition 3.2.1 (Modélisation des temps morts dans le $F3|nowait|C_{max}$)
Soit $t_{k,k+1}$ le temps mort entre les travaux k et $k+1$ dans un ordonnancement sans attante. Il s'agit de la durée qui sépare le début des travaux k et $k+1$ sur la première machine. On en déduit une expression du C_{max} :

$$C_{max} = t_{0,1} + t_{1,2} + t_{2,3} + \cdots + t_{n-1,n} + t_{n+1,0}$$

avec T_0 un travail fictif ($x_0 = y_0 = z_0 = 0$). Il vient alors :

$$C_{max} = x_0 \quad \mathbf{1} \oplus \frac{y_0}{x_1} \oplus \frac{y_0 z_0}{x_1 y_1} \quad x_1 \quad \mathbf{1} \oplus \frac{y_1}{x_2} \oplus \frac{y_1 z_1}{x_2 y_2} \quad \ldots x_{n-1} \quad \mathbf{1} \oplus \frac{y_{n-1}}{x_n} \oplus \frac{y_{n-1} z_{n-1}}{x_n y_n} \quad x_n \quad \mathbf{1} \oplus \frac{y_n}{x_0} \oplus \frac{y_n z_n}{x_0 y_0}$$

$$C_{max} = \bigotimes_{i=1}^{n} x_i \quad \mathbf{1} \oplus \frac{y_0}{x_1} \oplus \frac{y_0 z_0}{x_1 y_1} \quad \mathbf{1} \oplus \frac{y_1}{x_2} \oplus \frac{y_1 z_1}{x_2 y_2} \quad \ldots \quad \mathbf{1} \oplus \frac{y_{n-1}}{x_n} \oplus \frac{y_{n-1} z_{n-1}}{x_n y_n} \quad \mathbf{1} \oplus \frac{y_n}{x_0} \oplus \frac{y_n z_n}{x_0 y_0}$$

Finalement :

$$C_{max} = \left(\bigotimes_{i=1}^{n} x_i \right) \left(\mathbf{1} \oplus \frac{y_0}{x_1} \oplus \frac{y_0 z_0}{x_1 y_1} \right) \left(\bigotimes_{i=1}^{n-1} \mathbf{1} \oplus \frac{y_i}{x_{i+1}} \oplus \frac{y_i z_i}{x_{i+1} y_{i+1}} \right) \left(\mathbf{1} \oplus \frac{y_n}{x_0} \oplus \frac{y_n z_n}{x_0 y_0} \right) \tag{3.1}$$

Il est clair que la maximisation des durées opératoires des travaux du problème permettra l'augmentation du C_{max}. Cependant, en observant l'expression du C_{max} de la définition précédente, on se rend compte d'une part que ce sont les travaux figurant sur la troisième machine qui agissent principalement sur la date de fin maximale. D'autre part, si l'on observe l'expression du temps mort entre deux travaux, on constate que les décalages fixés entre les travaux se compensent.

Enfin, rappelons que le problème de flowshop sans attente est très similaire à un problème de flowshop avec décalages fixes. On en déduit qu'il faudra examiner une manière de maximiser également les temps morts entre les travaux.

On en déduit plusieurs fonctions objectifs à tester :

$$Z_1 = -b_{x,1} - b_{x,3}$$
$$Z_2 = -b_{x,1} - b_{x,2} - b_{x,3}$$
$$Z_3 = a_{x,1} + a_{x,2} - b_{x,1} - b_{x,3}$$
$$Z_4 = a_{x,1} + a_{x,2} - b_{x,1} - b_{x,2} - b_{x,3}$$

Ces quatre fonctions définies ainsi donneront quatre bornes inférieures, à savoir LB1(N), LB2(N), LB3(N) et LB4(N), avec N le nœud considéré de notre PSE. Pour des soucis de performances, seule une borne sera choisie pour tout le déroulement de la procédure par séparation et évaluation. Des tests préalables permettront de connaître quelle borne est la plus performante.

3.3 Borne supérieure

Plusieurs heuristiques donnant des bornes supérieures ont été implémentées et testées pour notre problème. La première repose sur un calcul élémentaire de C_{max} suivant la séquence donnée par le calcul de l'optimal pour le problème sans attente ; les deuxième et troisième bornes supérieures sont obtenues grâce à l'heuristique NEH [17]. Ces dernières heuristiques ont bien évidemment été adaptées afin d'être utilisées pour notre problème.

Notre première heuristique sera notée H1, et repose sur un principe très simple. En effet, il se trouve que le calcul d'une borne inférieure dans un nœud donne également une séquence. Il est donc possible de calculer par la même occasion une borne inférieure, simplement en créant l'ordonnancement associé à cette séquence : il s'agit du C_{max}. Contrairement au schéma classique d'une PSE, il est donc possible de trouver la solution du problème sans être dans une feuille de l'arbre de recherche.

Les deux autres heuristiques ont été adaptées à partir de l'heuristique NEH [17] afin qu'elles puissent être utilisées dans le cadre particulier de notre problème. Le principe est le suivant : les travaux sont tout d'abord triés selon un certain critère. Ensuite, l'ordonnancement est construit pas à pas en insérant successivement les travaux à la meilleure place dans la séquence partielle de manière à minimiser le C_{max}. Deux versions de cette heuristique ont été créées, qui diffèrent par la manière de classer les travaux. La première (notée NEH1) utilise l'ordre des travaux donné par la borne inférieure précédemment calculée. La deuxième (notée NEH2) repose sur le classement original de l'algorithme, à savoir selon la somme des durées opératoires sur toutes les machines.

3.4 Implémentation

Il s'agit ainsi d'implémenter une PSE qui appelera à chacun de ses noeuds (du moins, si cela est possible) le programme de résolution du flowshop sans attente. Une borne inférieure sera alors obtenue, garante du bon fonctionnement de notre PSE.

Le programme est implémenté en langage C, et ne comporte par conséquent pas de classes.

3.4.1 Variables globales

Le programme contient un certain nombre de variables globales, définies dans le tableau 3.1.

Nom de la variable	Description
nbmachines	Nombre de machines dans le problème considéré.
nbjobs	Nombre de travaux dans le problème considéré.
nbiterations	Nombre de problèmes à résoudre.
compteur	Nombre de nœuds développés dans la PSE.
job	Travaux du problème considéré.
meilleuresequence	Meilleure séquence trouvée.
sortie	Fichier de logs.

TAB. 3.1 – Variables globales du programme.

3.4.2 Description des fonctions

Le programme de la procédure par séparation et évaluation ainsi implémentée se décompose d'une part en plusieurs fonctions spécifiques au déroulement de cette dernière, d'autre part en plusieurs procédure annexes relatives aux différents appels externes à réaliser.

Programme principal

Voici un court descriptif des différentes procédure du programme principal (`bb-d.c`) :

- `lecture_fichier` : Cette procédure permet la lecture du fichier de données. Les données sont enregistrées dans le tableau job[].
- `cmaxF3d` : Cette procédure permet le calcul d'une borne supérieure (H1).
- `neh` : Cette procédure permet le calcul d'une borne supérieure suivant l'heuristique NEH1 ; utilisation du classement donné par la borne inférieure (pas de tri).
- `neh2` : De la même manière, calcul d'une borne supérieure suivant l'heuristique NEH2 ; les travaux sont cette fois classés selon la somme des durées opératoires sur toutes les machines.
- `miseajourmeilleur` : Cette procédure permet la mise à jour de la borne supérieure ainsi que de la séquence associée à cette dernière.
- `borneinfetsup` : Cette procédure permet de déterminer les bornes inférieures et supérieures du noeud passé en argument. Si plusieurs bornes sont calculées, seules les meilleures sont sauvegardées.
- `developper` : Cette procédure permet de développer tous les nœuds d'un fils, de les évaluer et d'éventuellement les couper ou les explorer. Les nœuds à explorer sont triés selon leur borne inférieure (meilleure d'abord).
- `main` : Procédure principale. Décrit l'exécution de la PSE : gère la lecture des données, l'exécution de la PSE et son arrêt.

Appel externe : résolution d'un simplexe

La résolution du simplexe est réalisée au moyen du solveur GLPK et repose dans le fichier `simplexe.h`. La bibliothèque C est liée au programme et utilisée. La fonction décrivant l'appel au simplexe repose sur l'écriture du problème pour GLPK et sur l'exécution proprement dite du simplexe. Les solutions sont renvoyées au programme principal.

Appel externe : résolution d'un problème de voyageur de commerce

La résolution du problème de voyageur de commerce est réalisée par un programme écrit en FORTRAN. Ce dernier est compilé conjointement au programme principal de la PSE. Une fonction assure l'interface entre ces deux programmes et gère la mise au point des différentes variables utilisées. Les solutions sont de la même façon renvoyées au programme principal.

Chapitre 4

Expérimentations numériques

La procédure par séparation et évaluation décrite dans le chapitre précédent à été testée abondamment grâce à des jeux d'essais divers et variés. Comme cela a été dit, les algorithmes ont été codés en C et les tests ont été réalisés sur une machine de type PC Pentium 4, 3,2 Ghz.

4.1 Schéma de génération des données

Etant donné que le programme a été conçu en vue d'être comparé avec ce qui se faisait dans la littérature sur le même sujet, les données (jeux d'essai) ont été générés d'une manière très spécifique, comme nous allons le voir par la suite. Nous nous baserons essentiellement sur l'article de Fondrevelle et al. [9].

Le programme a été codé de manière à accepter un nombre quelconque de machines. Il a cependant été principalement testé avec des problèmes à 3 et 5 machines, puis marginalement à 10 machines. Le nombre de travaux est fixé dans un premier temps à 10, puis 12 et 20. Pour chaque type de problème, une centaine de jeux d'essai ont été générés aléatoirement selon les contraintes suivantes :

- les durées opératoires sont tirées entre 20 et 50 ;
- les délais minimum et maximum entre les tâches des travaux (i.e. α_x et β_x, x étant un travail) sont tirés entre 0 et 14. Il faut noter que si $\beta_x < \alpha_x$ alors on fixe les deux valeurs à β_x.

On tiendra éventuellement compte dans la suite de la charge des machines ainsi que de l'hétérogénéité des durées opératoires des travaux. En d'autres termes :

- certaines instances de données sont homogènes : toutes les durées opératoires sont générées de la même manière et appartiennent au même intervalle (en général, [20 ;50]) ; d'autres instances sont hétérogènes : certaines durées opératoires seront générés dans l'intervalle initial, les autres dans un intervalle différent ([20 ;50] pour la moitié par exemple, [20 ;100] pour l'autre moitié) ;
- la charge de certaines machines peut-être accrue : ainsi, les machines nommées pourront voir les durées opératoires des travaux concernés augmentés de 75 %.

Nous obtenons finalement plusieurs classes de jeux d'essai définis selon les critères ci-dessus. Se reporter au programme pour plus de détails à propos des jeux de données générés.

4.2 Analyse des performances

D'une manière générale, on peut constater que les performances obtenues sont très encouragentes : en effet, sur plusieurs dizaines de tests, on constate que la procdure par séparation et évaluation implémentée se révèle très performantes, résolvant le problème en moins d'une seconde, même avec un grand nombre de travaux (à partir de 10). Malheureusement, aucun test n'a pu être effectué pour des instances plus importantes en terme de nombre de machines ou pour des instances particulières, comme cela a été décrit précédemment. Cependant, les résultats existent et peuvent être consultés au niveau de l'archive électronique.

En ce qui concerne les performances des bornes inférieures, on se rends compte que la borne LB1 issue de l'optimisation de Z_1 donne les meilleures performances. Pour ce qui est des bornes supérieures, les heuristiques NEH se détachent nettement, avec un leger avantage pour NEH2. Encore une fois, se reporter aux archives pour plus d'informations.

Conclusion

Ce travail s'inscrivait dans le prolongement conjoint de mon rapport de PFE et de l'article de MM. Lenté et Bouquard, et consistait à étudier les applications de l'algèbre Max-Plus au cas du problème de flowshop avec délais minimum et maximum. Ce dernier article servant de base à mon travail de stage, j'ai pu étendre les modélisations données pour le problème particulier à trois machines et les généraliser. Il en est de même concernant le programme, qui fut très largement modifié et qui fonctionne aujourd'hui dans le cas général.

Contrairement à l'étude du problème de flowshop sans attente, les performances mesurées dans le cas du flowshop avec délais minimum et maximum semblent être à la hauteur de nos attentes. En effet, les premiers résultats numériques donnés par l'application semblent très prometteurs et méritent ainsi d'être étendus.

Plusieurs transformations des programmes originaux furent nécessaires afin d'obtenir un résultat convenable, tant au niveau théorique que technique (implémentation). Comme cela a été dit dans le dernier chapitre de ce document, les tests doivent être poursuivi afin d'aboutir à une présentation globale et synthétique de la résolution que nous avons pu donner de ce problème.

Annexes

41

Notice d'utilisation

Cette section a pour but de présenter à l'utilisateur un manuel d'utilisation du logiciel facilement compréhensible. Les programmes originaux ont été modifiés afin de les rendre plus simples à utiliser d'une part, et d'autre part pour obtenir une certaine norme dans les mécanismes de paramètrage, comme nous allons le voir par la suite.

Tous les programmes réalisés fonctionnent sous l'environnement Linux, et on été testé sous Linux Ubuntu. Ils sont cependant entièrement réutilisable dans un environnement MS Windows.

Organisation des fichiers dans l'arborescence

Le répertoire par défaut de l'application se nomme `fdminmax`. Ce dernier contient les éléments suivants (fichiers ou dossiers) :

Fichier ou dossier	Description
`aux`	Dossier contenant plusieurs programmes ou bibliothèques auxiliaires.
`data`	Dossier contenant les jeux de données de la PSE.
`doc`	Dossier contenant le présent document.
`include`	Dossier contenant tous les fichiers .h de l'application.
`src`	Sources de l'application.
`Makefile`	Fichier permettant la compilation et l'installation automatique de l'application.
`readme`	Fichier d'instructions, notes.

Instructions de compilation et d'installation

Afin de compiler l'application, un fichier Makefile a été créé. Ce dernier respecte toutes les spécifications habituelles pour ce type d'outil. Ainsi, un `make` permettra de compiler et de créer les différents exécutables de l'application (à savoir `fdminmax` et `fdminmax-gen`). L'installation de l'application sur le système s'effectue de la manière habituelle grâce à la commande `make install`. La commande `make clean` supprime tous les fichiers créés lors de la compilation.

Le script shell `configure` permet de vérifier la présence de tous les outils indispensables à l'application, à savoir les compilateur C et FORTRAN ainsi que la bibliothèque GLPK.

Le générateur de données

Usage

Le générateur de données permet la génération automatique et entièrement configurable de jeux de données. L'exécutable de l'application `fdminman-gen` est fabriqué lors de la compilation de l'application. Ainsi, le programme fourni permet de générer des données pour un problème avec un nombre quelconque de machines et de travaux. Les intervalles sont également entièrement paramétrables.

Il s'agit d'une application en ligne de commande. Son usage est le suivant :

```
fdminmax-gen <nbmachines> <nbtravaux> <minp> <maxp> <minalpha> <maxalpha>
             <minbeta> <maxbeta> <nbjeux> <destination>
```

Tous les paramètres sont indispensables. L'application va ainsi créer *nbjeux* fichiers de données dans le répertoire *destination* suivant les paramètres spécifiés.

Fichiers générés

Chaque fichier généré contient un unique problème, et sera placé dans le répertoire indiqué en paramètre de l'exécutable. Voici un exemple d'un extrait du fichier de données pour 3 machines et 10 travaux :

```
28 27 33 3 1 10 2
41 32 26 0 0 0 3
40 48 26 2 1 10 12
30 27 21 5 3 5 11
46 28 29 1 5 7 9
28 49 46 0 0 5 2
33 26 33 1 4 2 7
49 48 32 2 3 2 3
37 49 21 3 2 4 8
44 24 30 0 1 1 1
```

Les trois premières colonnes contiennent respectivement les durées opératoires des travaux pour les trois premières machines. Les deux colonnes suivantes contiennent les décalages minimum, les deux dernières les décalages maximum correspondant.

Le programme de la PSE

Une fois compilé, le programme de la PSE est utilisable en ligne de commande. Son exécutable se nomme `fdminmax`. Il prend en argument plusieurs paramètres, eventuellement optionnels. Deux modes d'exécutions sont disponibles :

– le mode PSE sur un seul jeu de données : la PSE sera exécutée sur un unique jeu d'essai, spécifié en paramètre. Il s'agit du mode d'exécution par défaut ;
`fdminmax <cheminfichier>`
– le mode PSE sur plusieurs fichiers : la PSE sera exécutée sur plusieurs jeux d'essai successivement, dont l'emplacament sera spécifié en paramètre. Tout les fichiers recevables à cet emplacement seront alors exploités.
`fdminmax -m <emplacement>`

Il est également possible de logguer toutes les actions réalisées par la PSE. Pour cela il suffit de lancer la commande `fdminmax -l <fichiersortie>`.

Bibliographie

[1] K.R. Baker. *Introduction to sequencing and scheduling.* John Wiley & Sons, 1974.

[2] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints : classification and complexity. *Discrete Appl. Math.*, pages 5 :11–24, 1983.

[3] T.S. Blyth. Matrices over ordered algebraic structures. *J. of London Mathematical Society*, pages 39 :427–432, 1964.

[4] T.S. Blyth and M.F. Janowitz. Residuation theory. *Pergamon Press*, 1972.

[5] J.L. Bouquard and C. Lenté. Two-machine flow shop scheduling problems with minimal and maximal delays. *A paraître*, 2005.

[6] J. Carlier and P. Chrétienne. *Problèmes d'ordonnancement : modélisation/complexité/algorithmes.* Masson, 1988.

[7] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large-scale asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software*, pages 21 :394–409, 1995.

[8] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of scheduling.* Addison-Wesley, 1967.

[9] J. Fondrevelle, A. Oulamra, and M.-C. Portmann. Permutation flowshop scheduling problems with maximal and minimal time lags. *Computers & Operations Research*, 2005.

[10] S. Gaubert. *Théorie des systèmes linéaires dans les dioïdes.* PhD thesis, Ecole des Mines de Paris, 1992.

[11] P.C. Gilmore and R.E. Gomory. Sequencing a one-state variable machine : A solvable case of the travelling salesman problem. *Oper. Res.*, pages 12 :655–679, 1964.

[12] M. Gondran and M. Minoux. *Graphes et algorithmes.* Eyrolles, Paris, 3^e edition, 1995.

[13] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, pages 5 :287–326, 1979.

[14] J.R. Jackson. An extension of johnson's results on job-lot scheduling. *Naval Res Log Quart*, page 3(3), 1956.

[15] S.M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Res Log Quart*, pages 1 :61–68, 1954.

[16] C. Lenté. *Analyse Max-Plus de problèmes d'ordonnancement de type flowshop.* PhD thesis, Université François Rabelais, Tours, 2001.

[17] M. Nawaz, E.E. Jr. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. *Omega*, pages 11 :91–5, 1983.

Résumé

L'algèbre Max-Plus permet de modéliser un grand nombre de problèmes de flowshop, et parmi ceux-ci, le problème de flowshop avec délais minimum et maximum. Ce dernier peut être représenté, comme pour le flowshop de permutation et le flowshop sans attente, par un produit matriciel.

Nous nous sommes alors penché sur l'extension au cas général d'une borne inférieure pour ce type de problème, basée sur la résolution d'un flowshop sans attente. Une Procédure par Séparation et Évaluation existe déjà et a été enrichie de cette dernière méthode de calcul. Plusieurs heuristiques permettant le calcul de bornes supérieures ont également été implémentées. Cette PSE se révèle parfaitement fonctionnelle sur n'importe quel jeu de données, et propose des résultats très satisfaisants en temps très réduit.

Mots Clés : Ordonnancement, Flowshop, Délai, Bornes, Procédure par Séparation et Evaluation.

Abstract

The Max-Plus algebra allows the modeling of many flowshop problems, and among them, the flowshop scheduling problem with maximal ans minimal time lags. This one can be modelized as a matricial product, like the ones studied in the past (no-wait flowshop and permutation flowshop).

We focused then on the extension to the general case of a lower born for this type of problem, based on a no-wait flowshop problem resolution. A Branch and Bounds algorithm already exists, and has been improved with this last method. Several heuristics which allow upper borns calculation have also been implemented. The final program is fully functional with every data set, and gives very satisfying results in very restricted time.

Keywords : Scheduling, Flowshop, Time lag, Borns, Branch and Bound Algorithm.

Stage de Master de Recherche en Informatique

Une procédure par séparation et évaluation
pour le problème de flowshop avec délais minimum et maximum

LI Tours
Polytech'Tours
64, avenue Jean Portalis
37200 TOURS

Encadrant de stage
M. Christophe LENTÉ
Maître de conférences

Auteur du stage
Vincent AUGUSTO
M2R
2004-2005

Rapport du mini-projet de GPF

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

Département Informatique
4e année
2012 - 2013

Rapport
Projet de GPF

Heuristiques pour un problème de flowshop avec contraintes mixtes de blocage

Encadrant
Christophe LENTE
christophe.lente@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Maxime SERRA
maxime.serra@etu.univ-tours.fr
Fabien FARIN
fabien.farin@etu.univ-tours.fr

DI4 2012 - 2013

Version du 24 avril 2013

Table des matières

Table des figures

Introduction

1.1 Présentation du sujet

Notre sujet consiste dans un premier temps à étudier l'article intitulé "*Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems*" publié par Wajdi TRABELSI, Christophe SAUVEY et Nathalie SAUER.

Par la suite, il nous faudra implémenter les méthodes décrites dans cet article dans le but de réaliser des tests pour établir un tableau comparatif final de ces heuristiques.

Ce projet s'inscrit directement dans la continuité du projet de fin d'étude de Mélanie MAUGEAIS et du sujet de la thèse du doctorat de Vinh VO.

1.2 Objectifs

Afin de mener à bien ce projet, il a tout d'abord été nécessaire d'en délimiter précisément les objectifs.

Ainsi, la réalisation du projet a été découpée en **cinq grandes phases** principales :

- Compréhension des contraintes énoncées dans l'article
- Analyse des quatre heuristiques (NEH, NEH improvement, TSS et TSS improvement)
- Implémentation de ces dernières en langage C
- Développement d'un générateur d'instances de problèmes aléatoires
- Génération d'un tableau comparatif final des résultats

Étude préliminaire

2.1 Présentation de l'article

L'article qui a fait l'objet de notre étude, publié début 2012, se focalise sur des problèmes particuliers de flowshops ayant des contraintes mixtes de blocages.

Cela signifie donc concrètement que pour une instance donnée d'un problème mettant en oeuvre m machines, il y aura $m-1$ contraintes qui conditionneront le passage des différents jobs sur les machines suivantes.

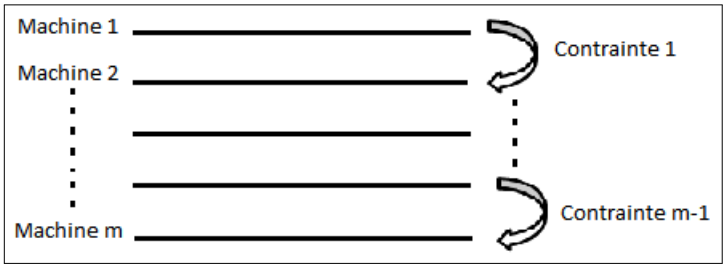


FIGURE 2.1 – Illustration simplifiée d'un problème de flowshop à contraintes mixtes

Dans la suite de l'article, les auteurs nous présentent les différents types de contraintes, au nombre de quatre, Wb, Rsb, Rcb et $Rcb*$ que nous expliciteront dans la section suivante.

Ces flowshops à contraintes mixtes ainsi que les contraintes particulières énoncées plus haut modélisent bien la réalité de certaines industries et c'est la raison pour laquelle on cherche à trouver des méthodes permettant d'obtenir un ordonnancement des travaux le plus optimal possible.

C'est ce qu'ont fait les auteurs de cet article qui nous présentent l'heuristique TSS (Trablsi Sauvey Sauer). Ils comparent les résultats de cette dernière à ceux obtenus par l'heuristique bien connue NEH en y appliquant également des améliorations (TSS Improvement et NEH Improvement). Ceci dans le but d'en apprécier l'efficacité sur diverses instances de paramètres variables (i.e. le nombre de travaux et de machines).

Ces heuristiques feront également l'objet d'explications dans les sections suivantes, sauf pour NEH qui sera supposée comme étant déjà connue.

2.2 Contraintes

2.2.1 Wb

La contrainte Wb signifie en réalité qu'il n'y aura pas de situation de blocage. Un job pourra donc démarrer sur une machine m sans tenir compte de l'avancement de tout autre job sur les machines suivantes ou précédentes. Il ne peut donc y avoir, selon le scénario, seulement des temps d'attente.

2.2.2 Rsb

La contrainte Rsb est la première qui est énoncée dans l'article. Elle signifie en anglais *Release when Starting Blocking* et est qualifiée de contrainte bloquante classique. Avec cette contrainte, on considère qu'il n'y a pas de lieux de stockage intermédiaires entre les machines pour faire "tampon".

En d'autres termes, si on considère deux travaux consécutifs i et j, un travail j ne pourra commencer sur une machine m qu'une fois que le travail i aura été pris en charge sur la machine $m+1$.

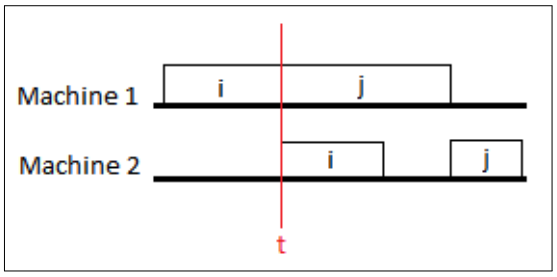


FIGURE 2.2 – Exemple simple de la contrainte Rsb

Sur l'exemple trivial ci-contre, on voit que le travail j ne peut pas commencer sur la machine 1 avant la date t qui correspond à la date de début du travail i sur la machine suivante. Ici, le fait que la date de fin du travail i sur la machine 1 coïncide avec sa date de début sur la machine 2 est un hasard.

Ainsi, on pourrait parfaitement imaginer un hypothétique travail h précédent le travail i, qui nous ferait obtenir le schéma ci-dessous :

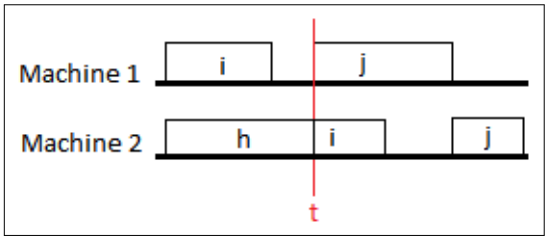


FIGURE 2.3 – Autre exemple de de la contrainte Rsb

Cette fois, la figure ci-dessus illustre bien le temps de blocage qui a été généré sur la machine 1 entre les travaux i et j.

2.2.3 Rcb

La contrainte Rcb est la première contrainte spécifique abordée dans l'article et elle signifie *Release when Completing Blocking*. Dans cette dernière, on considère non plus le début du travail, comme dans la contrainte Rsb, mais la fin de celui-ci.

Cela signifie que si l'on considère de nouveau deux travaux i et j, supposés consécutifs, alors un travail j ne pourra commencer sur une machine m que si le travail i est terminé sur la machine $m+1$ et a été pris en charge sur la machine $m+2$.

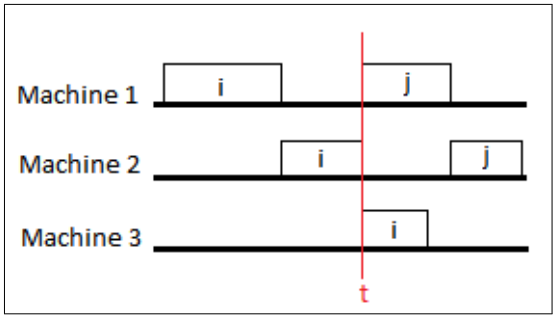


FIGURE 2.4 – Exemple simple de la contrainte Rcb

Sur la figure ci-dessus, on constate donc que le travail j ne peut pas démarrer sur la machine 1 avant la date t qui ici correspond à la date de fin du travail i sur la machine 2 et aussi à sa date de début sur la machine 3.

De même, si un hypothétique travail h se présentait comme présenté sur la figure ci-dessous, alors le travail j serait retardé sur la machine 1 jusqu'à ce que le travail i démarre sur la machine 3 :

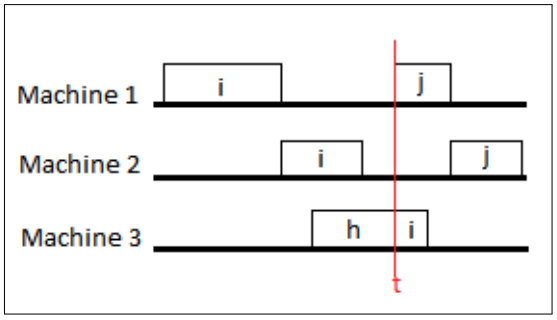


FIGURE 2.5 – Autre exemple de la contrainte Rcb

2.2.4 Rcb*

La contrainte $Rcb*$ quant à elle est une variante de la contrainte Rcb. Comme vu dans la section précédente pour la contrainte Rcb sur les figures 2.4 et 2.5, la date de début du travail j sur la machine 1 correspond au maximum entre la date de fin du travail i sur la machine $m+1$ et la date de début du travail i sur la machine $m+2$.

La subtilité de la contrainte $Rcb*$ réside dans le fait que, pour un travail j donné, son traitement pourra commencer sur la machine m dès que le traitement du travail précédent i sur la machine $m+1$ sera fini, et cela indépendamment de sa prise en charge ou non sur la machine $m+2$.

Concrètement, si on reprend la figure 2.5 de la section précédente, la date t de début du travail j sur la machine 1 correspondra à la date de fin du travail précédent i sur la machine 2 :

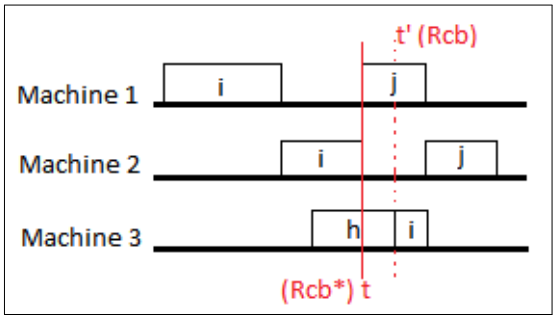


FIGURE 2.6 – Exemple de la contrainte $Rcb*$

2.3 Heuristiques

Toutes les contraintes ainsi que leurs spécificités énoncées dans les sections précédentes représentent des cas de flowshop qui existent dans le milieu industriel.

Maintenant, pour des problèmes de flowshop dans lesquels le nombre de travaux et de machines est important et que le vecteur de contrainte associé met en jeu chacune de ces contraintes de manière aléatoire et répétitive, l'enjeu est de trouver des heuristiques efficaces et rapides dans le but de minimiser un critère qui est le *makespan*, c'est à dire la date finale de fin de traitement de tous les travaux sur toutes les machines.

Dans les sections suivantes, nous expliciterons les heuristiques présentées dans l'article permettant de répondre à cette problématique.

2.3.1 NEH Improvement

NEH Improvement est une heuristique basée sur le principe de l'heuristique NEH. NEH est une heuristique très connue dans le domaine de l'ordonnancement de flowshop puisque celle-ci permet d'obtenir des résultats efficaces et est très simple à implémenter. NEH Improvement utilise la séquence d'ordonnancement crée par NEH pour améliorer son makespan en testant toutes les positions possibles pour un job, sans modifier l'ordre relatif entre les autres jobs.

Cette heuristique est définie par l'algorithme suivant :

DONNÉE : n = Nombre de travaux
1: Soit $J \leftarrow (j1,j2...jn)$ l'ordonnancement des jobs obtenus par l'heuristique NEH
2: C_{pmax} : entier représentant le C_{max} le plus petit connu lors du placement d'un job
3: **TANTQUE** Amélioration du makespan **FAIRE**
4: Mise à jour de l'ordre des jobs à placer qui sont dans J
5: **POUR** i DE 1 À n **FAIRE**
6: On sélectionne le job j_i dans J
7: $C_{pmax} \leftarrow \infty$
8: **POUR** k DE 1 À n **FAIRE**
9: On place le job j_i à la position k dans la séquence $J_{partielle}$ sans changer l'ordre relatif des autres jobs déjà placés
10: Calcul de C_{max} de la séquence $J_{partielle}$
11: **SI** $C_{max} < C_{pmax}$ **ALORS**
12: $Best_k \leftarrow k$
13: $C_{pmax} \leftarrow C_{max}$
14: **FIN SI**
15: **FIN POUR**
16: On place le job j_i à la position $Best_k$ dans J
17: **FIN POUR**
18: **FIN TANTQUE**
19: **RETOURNER** L'ordonnancement de la séquence donnant le plus petit makespan

---

FIGURE 2.7 – NEH Improvement

Afin de mieux illustrer cette heuristique, voici un schéma montrant le déroulement partiel de celle-ci sur un exemple d'instance comprenant trois travaux.

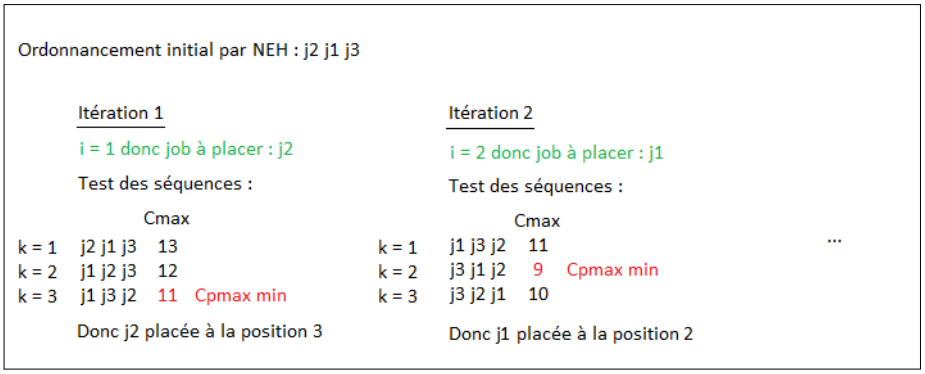


FIGURE 2.8 – Exemple de déroulement de l'heuristique NEH Improvement

2.3.2 TSS

L'heuristique TSS est la principale heuristique présentée dans l'article. Il s'agit d'une méthode incrémentale qui construit, itérations après itérations, une séquence que l'on espère la plus optimale possible en se basant sur un critère discriminant.

Cette méthode est comparable à l'heuristique NEH qui construit également une séquence finale en plaçant les travaux d'une séquence initiale à différents endroits dans le but de satisfaire le critère discriminant.

Dans NEH, ce critère discriminant est le *makespan* final. Ceci correspond au temps total nécessaire à l'exécution de tous les travaux sur toutes les machines. Dans l'heuristique TSS, le critère, bien que différent, fait également intervenir le *makespan* et est présenté comme suit :

$$Cr = min(C_{pmax} + SomTpsIn - SomTpsEx)$$

Avec :
- Cr : Le critère discriminant à minimiser pour une séquence partielle donnée.
- C_{pmax} : Le *makespan* de la séquence partielle en cours.
- $SomTpsIn$: Correspond à la somme des temps d'inactivités sur une machine. Ces temps peuvent soit être des temps d'attente, soit des temps de blocage liés à une contrainte.
- $SomTpsEx$: Correspond à la somme des temps d'exécution des travaux déjà placés dans l'ordonnancement partiel. Soit $\sum_{i=1}^{n}\sum_{j=1}^{m} P_{i,j}$ avec $P_{i,j}$: la durée d'exécution du travail J_i sur la machine M_j

L'algorithme de l'heuristique TSS est décrit ci-après. Il s'agit de la version présentée dans l'article, mais que nous avons remaniée, traduite et commentée pour une meilleure compréhension :

Afin de savoir si cette heuristique donne des résultats satisfaisants, l'efficacité de cet algorithme sera comparée à celle de l'heuristique NEH dans le chapitre suivant.

DONNÉE : n = Nombre de travaux
1: **POUR** i DE 1 À n **FAIRE**
2: On place le travail J_i en premier
3: **TANTQUE** (Il reste des travaux à placer) **FAIRE**
4: **POUR** k DE 1 À (n - nombre de travaux déjà placés) **FAIRE**
5: Pour J_k qui serait placé après J_i :
6: - Calcul de C_{pmax}, $SomTpsIn$, $SomTpsEx$
7: - Mémorisation du Cr
8: **FIN POUR**
9: On place après J_i le travail J_k ayant le plus petit Cr
10: **FIN TANTQUE**
11: Calcul de $C_{max,i}$ /* makespan de la séquence finale commençant par le travail J_i */
12: **FIN POUR**
13: **RETOURNER** L'ordonnancement $C_{max,i}$ donnant le plus petit makespan

FIGURE 2.9 – Heuristique TSS

2.3.3 TSS Improvement

TSS Improvement est une autre heuristique présentée dans l'article et consiste en une amélioration locale de TSS. Cette dernière est basée sur le fait que la contrainte Rcb fait apparaître des temps de blocages supplémentaires qualifiés de "différentiels".

Ces temps différentiels étant le critère de décision pour permuter un travail dans l'algorithme de l'amélioration locale de TSS, il a été nécessaire de bien les comprendre dans un premier temps.

Pour un travail et une machine J_i et M_j donnés, on distingue un temps de blocage différentiel selon si il intervient avant ou bien après le travail J_i.

Temps de blocage différentiel AVANT

Un temps de blocage différentiel avant un travail J_i sur une machine M_j est causé par une insuffisance du temps d'exécution de ce même travail sur la machine précédente, M_{j-1}, face au temps d'exécution du travail précédent J_{i-1} sur la machine suivante M_{j+1}.

Soit, en d'autres termes :

$$BlocageAvant(J_i, M_j) = \max(C_{i-1,j+1} - C_{i,j-1}, 0)$$
avec
$C_{i,j}$: date de fin du travail J_i sur la machine M_j

Temps de blocage différentiel APRÈS

Passons maintenant aux temps de blocages différentiels après. Toujours pour un travail et une machine J_i M_j donnés, ces derniers sont induits par une insuffisance du temps d'exécution du travail précédent,

J_{i-1}, sur la machine M_{j+2} en regard du temps d'exécution du travail J_i sur la machine courante M_j.

Ce qui équivaut, en d'autres termes, à :

$$BlocageAprès(J_i, M_j) = \max(C_{i,j} - C_{i-1,j+2}, 0)$$

toujours avec

$C_{i,j}$: date de fin du travail J_i sur la machine M_j

Si nous reprenons l'exemple qui est donné dans l'article à titre d'illustration, on peut facilement y appliquer les formules précédentes pour le calcul de :

- $BlocageAvant(k, 2) = \max(C_{j,3} - C_{k,1}, 0)$
- $BlocageAvant(j, 2) = \max(C_{j,2} - C_{i,4}, 0)$

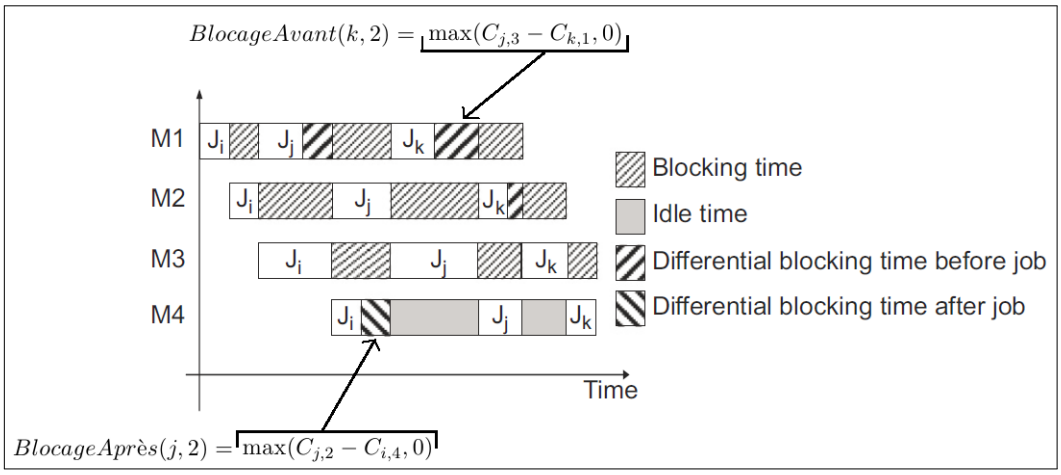


FIGURE 2.10 – Illustration des formules de calcul des temps de blocage différentiels

Algorithme

L'heuristique de l'amélioration locale de TSS repose sur le fait qu'un travail présentant la plus grande somme de temps de blocage différentiel sera changé de position dans la séquence pour conserver le meilleur placement.

Cette opération est ainsi réitérée tant que ces changements apportent une amélioration du makespan de la séquence obtenue par l'heuristique TSS initiale.

Aucun algorithme n'étant précisé dans l'article, voici celui que nous avons utilisé et qui se base sur celui ayant été reconstitué par Mélanie MAUGEAIS (DI5) dans le cadre de son projet de fin d'étude :

DONNÉE : $S = (J_1, ..., J_n)$: Un ordonnancement de travaux obtenu par l'heuristique TSS initiale

1: **TANTQUE** (La séquence S est modifiée) **FAIRE**
2: Sélectionner le travail J_i ayant la plus grande somme de temps différentiels
3: $S' \leftarrow S$
4: **POUR** k DE 1 À n **FAIRE**
5: Mettre J_i en position k dans S'
6: **SI** $(C_{max}(S') < C_{max}(S))$ **ALORS**
7: $S \leftarrow S'$
8: **FIN SI**
9: **FIN POUR**
10: **FIN TANTQUE**
11: **RETOURNER** La séquence S

FIGURE 2.11 – Heuristique TSS Improvement

Implémentation et expérimentation

3.1 Structures de données

Afin d'implémenter au mieux les quatre heuristiques que nous avons présentées plus haut, il a été nécessaire de penser à des structures de données qui permettraient de manipuler aisément les entités concernées (travaux, séquences...etc).

Dans ce but, nous avons, au cours du découpage des fonctionnalités, distingué cinq structures principales. Ces dernières vont être détaillées et sont les suivantes :

- Instance
- Sequence
- Machine
- Job
- Job_Sequence

Instance.h

La structure *Instance* est la structure initiale du programme. C'est dans celle là que seront stockées toutes les données relatives à une instance d'un problème de flowshop. A savoir, le nombre et la liste des travaux, le nombre et la liste des machines et enfin la liste des contraintes associées.

FIGURE 3.1 – Déclaration C de la structure Instance

```
typedef struct {
       int nbr_job;
       int nbr_mach;
       int nbr_cont;
       Job ** jobs;
       Machine ** machines;
       int * contraintes;
} Instance;
```

Les contraintes sont en réalité simplement représentées par un tableau d'entiers dont les valeurs dans le programme sont interprétées comme suit :

- $0 : Wb$
- $1 : Rsb$
- $2 : Rcb$
- $3 : Rcb*$

Sequence.h

La structure *Sequence* est sans doute celle qui est la plus centrale à l'application dans la mesure où elle va intervenir sans cesse dans les calculs des C_{max}.

Les membres de cette structure sont quasiment les mêmes que ceux de la structure *Instance*. On retrouve donc le nombre et la liste des travaux, le nombre et la liste des machines et le vecteur des contraintes.

Ce qui est nouveau, c'est l'ajout des attributs permettant de stocker le C_{max} associé à la séquence ainsi que le critère requis par l'heuristique TSS.

FIGURE 3.2 – Déclaration C de la structure Sequence

```
typedef struct {
        int nbr_job;
        int nbr_mach;
        int nbr_cont;
        Job ** jobs;
        Machine ** machines;
        int * contraintes;
        int Cmax;
        int TSS_Crt;
} Sequence;
```

Les champs C_{max} et TSS_Crt sont initialisés par des fonctions spécialisées propres à la structure *Sequence*. Ces dernières sont, $calcul\_Cmax(Sequence*)$ et $calculer\_TSS\_Crt(Sequence*, int)$.

Machine.h

La structure *Machine* est celle qui se retrouve dans les deux structures vues précédemment. Celle-ci contient toutes les informations relatives à une machine. Celles-ci incluent le numéro de la machine, le nombre total de travaux qu'elle pourra contenir, le nombre de travaux présent dessus à un instant t ainsi que la liste de ces travaux, et enfin, le makespan de la machine. Ce dernier étant mis à jour en temps réel à chaque ajout de travail.

FIGURE 3.3 – Déclaration C de la structure Machine

```
typedef struct {
        int id_mach;
        int total_job;
        int nbr_job;
        Job_Sequence ** jobs;
        int makespan;
} Machine;
```

Une particularité de notre application est que chaque machine n'existe qu'en un seul exemplaire durant toute la durée de vie du programme. Cela signifie donc que chaque occurrence de structure *Sequence* partagera la même liste de machines.

Pratiquement, cela implique un gain dans l'utilisation de la mémoire centrale. Les machines sont ainsi réinitialisées au besoin.

Job.h et Job_Sequence.h

La structure *Job* décrit simplement les informations que doit contenir un travail. On y trouve donc un numéro de travail, le nombre et la valeur des durées d'exécution de ces travaux sur les différentes machines.

FIGURE 3.4 – Déclaration C de la structure Job

```
typedef struct {
       int id_job;
       int nbr_duree;
       int * duree;
} Job;
```

Dans cette représentation, l'indice de la durée dans la liste des durées correspond au numéro de machine qui est concernée. En d'autres termes, pour $travail.id_job = 2$ et $travail.duree[3] = 12$, cela équivaut, avec la notation que nous avons vu dans la chapitre précédent, à $P_{2,3} = 12$. Avec $P_{i,j}$: la durée d'exécution du travail i sur la machine j.

Comme pour la structure *Machine*, les structures *Job* qui sont crées n'existent qu'en un seul exemplaire. Ceci pour la raison que les données caractérisant un travail n'ont pas vocation à être modifiées. Par ailleurs, cela apporte également un gain dans l'utilisation de la mémoire.

Pour stocker toutes les informations concernant un travail à ordonnancer telles que la date de début et la date de fin, nous avons préféré avoir recours à une structure dédiée. Cette dernière est la structure *Job_Sequence* dont la déclaration est la suivante :

FIGURE 3.5 – Déclaration C de la structure Job_Sequence

```
typedef struct {
       int id_job;
       int date_debut;
       int date_fin;
       int duree;
} Job_Sequence;
```

Les *Job_Sequence* sont créés à la volée et les dates de début et de fin sont dynamiquement mises à jour au fur et à mesure de l'ajout sur les machines. Lorsque les machines sont réinitialisées, ces travaux temporaires sont par la même occasion libérés.

3.2 Génération des résultats

Afin de tester les différentes heuristiques, des instances ont été générées aléatoirement dans un premier temps selon les contraintes définies dans la sous-section 3.2.1.

Par la suite, sur chaque séquence, le makespan optimal est déterminé et on applique ensuite les différentes heuristiques et combinaisons de celle-ci pour éprouver leur efficacité.

Lors du recueil des résultats, ces derniers permettent de générer le pourcentage d'erreur par rapport au makespan optimal. Dans un dernier temps, les résultats sont enregistrés dans un fichier xls permettant d'exploiter les résultats sous Excel.

Cette section 3.2 s'attache à détailler les spécifications du déroulement de ces différentes étapes.

3.2.1 Génération des instances

Afin de faire tourner les heuristiques que nous avions implémentées, il nous a fallu générer un jeux de tests complet.

Toutes ces instances de tests ont été générées conformément aux spécifications de l'article qui, pour rappel, sont les suivantes :

- Pour chaque taille d'instance de dimensions n et m fixées, on génère 100 différentes instances
- Pour des dimensions supérieures ou égales à $n = 11$ et $m = 10$, seulement 20 instances seront générées.
- Pour chacune des instances générées, les temps d'exécutions seront choisis aléatoirement dans l'intervalle [0,99]
- Pour les vecteurs contraintes de chaque instance, rien n'était spécifié dans l'article. Il a donc été décidé de générer un vecteur contrainte aléatoire dont les valeurs sont comprises dans l'intervalle [0,3] (i.e $0 => Wb$, $1 => Rsb$, $2 => Rcb$, $3 => Rcb^*$).

Pour chacun des problèmes générés, il était également nécessaire de connaître la solution optimale dans le but de calculer le taux d'erreur avec le résultat des heuristiques.

3.2.2 Format des fichiers d'instances et des répertoires

Notre application étant capable d'importer les données d'une instance depuis un fichier texte, il a été nécessaire de convenir d'un format précis.

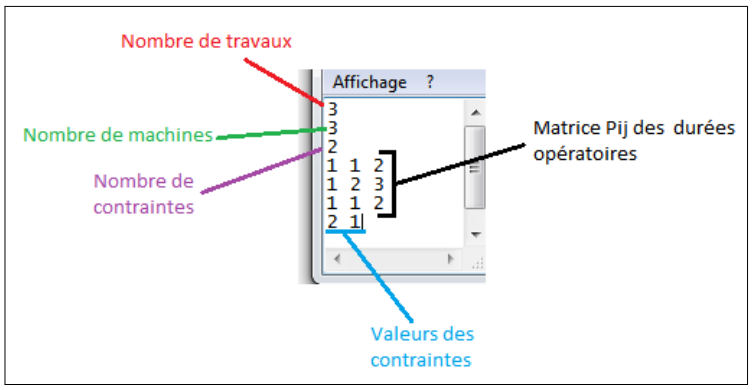


FIGURE 3.6 – Exemple du format des fichiers d'instances

Avec $P_{i,j}$: la durée d'exécution du travail J_i sur la machine M_j

Il a également été décidé qu'un fichier d'instance comme ci-dessus porterait un nom formaté comme suit :

nXXmYYiZZ.txt

Avec :
- XX : le nombre n de travaux
- YY : le nombre m de machines
- ZZ : le numéro du fichier d'instance

Et tous ces fichiers d'instance devaient se trouver dans un répertoire ayant également un nom formaté de la forme /nXXmYY.

3.2.3 Détermination du makespan optimal

Afin d'obtenir le makespan optimal pour une séquence donnée, il faut énumérer toutes les possibilités de positionnement pour chaque travail, ce qui donne pour n travaux $n!$ séquences à calculer.

Le problème est que la plupart des algorithmes permettant d'énumérer toutes les combinaisons sont récursifs, or cela crée un risque de dépassement de la taille de la pile allouée au programme.

Après quelques recherches, nous avons trouvés un algorithme itératif répondant à nos attentes que nous avons adapté au projet : le QuickPerm.

Simple d'implémentation et rapide à l'exécution, il permet de consommer des ressources très limitées lors de son exécution.

Voici l'algorithme du QuickPerm utilisé dans le cadre de notre projet :

DONNÉE : I, l'instance sur laquelle l'énumération va être faite.

tmp : Job déclaration d'un job qui servira pour les permutations
copieListeJobs : tableau de Job $[0..n-1]$ contient les jobs de l'instance I passée en paramètre
sequence_temp : *Sequence* Création d'une séquence qui permettra de calculer le C_{max} pour un ordonnancement
p : tableau de int $[0..n-1]$ Un tableau d'entier qui servira d'indice pour positionner les jobs dans la sequence
POUR i de 0 à $n-1$ par incrément de 1 **FAIRE**
  $p[i] = 0$ Initialisation à 0 des indices du tableau
FIN POUR
$C_{max} = \infty$ On considère le C_max au départ comme étant infini
$c_{maxTemp} \leftarrow calculer\_Cmax(sequence\_temp)$
SI $c_{maxTemp} < c_{max}$ **ALORS**
  $C_{max} \leftarrow c_{maxTemp}$
FIN SI
$reset\_Machines(I)$
$i \leftarrow 1$ Initialisation pour le premier échange
TANTQUE i est inférieur au nombre de jobs de l'instance(n) **FAIRE**
  SI $p[i] < i$ **ALORS**
    $j \leftarrow i\%2 * p[i]$
    $tmp \leftarrow copieListeJobs[j]$
    $copieListeJobs[j] \leftarrow copieListeJobs[i]$
    $copieListeJobs[i] \leftarrow tmp$
    $c_maxTemp \leftarrow calculer_Cmax(sequence_temp)$;
    SI $c_{maxTemp} < C_{max}$ **ALORS**
      $cmax \leftarrow cmaxTemp$
    FIN SI
    $p[i]++$
    $i \leftarrow 1$
  SINON
    $p[i] \leftarrow 0$
    $i++$
  FIN SI
FIN TANTQUE
RETOURNER C_{max}

FIGURE 3.7 – Algorithme du QuickPerm adapté au projet

3.2.4 Application des heuristiques et leurs combinaisons

Pour chaque instance traitée dans le programme, plusieurs appliquations d'heuristiques ont été effectuées. Après détermination du makespan optimal par l'algorithme du QuickPerm, voici les traitements effectués :

- NEH
- TSS
- NEH improvement sur le résultat de TSS
- NEH improvement sur le résultat de NEH
- TSS improvement sur le résultat de TSS
- TSS improvement sur le résultat de NEH
- TSS improvement sur le résultat de NEH improvement sur TSS
- TSS improvement sur le résultat de NEH improvement sur NEH
- NEH improvement sur le résultat de NEH improvement sur TSS
- TSS improvement sur le résultat de NEH improvement sur NEH

Pour chaque application, le C_{max} (makespan) le plus petit trouvé est récupéré. On le compare ensuite au makespan optimal pour calculer le pourcentage d'erreur déterminé par la formule suivante :

$$C_{err} = \frac{C_{max} - C_{maxOptimal}}{C_{maxOptimal}} \times 100$$

L'estimation du pourcentage d'erreur s'effectue en moyennant tout les différents résultats provenant des instances ayant les mêmes paramètres. Le pourcentage d'erreur moyen par type d'application est ensuite calculé puis stocké dans un tableau temporaire pour finalement être écrit dans le fichier de sortie.

3.2.5 Fichier de sortie

Concernant le format du fichier de sortie final généré par notre application, nous avons choisit d'opter pour un fichier Excel (*.xls*) et ceci pour plusieurs raisons.

C'est tout d'abord un format de fichier largement repandu et utilisé qui offre des possibilités de mise en forme assez étendu.

Par ailleurs, une fois les données générées dans une feuille de calcul Excel, les possibilités d'exploitation des résultats sont très riches grâce notamment à l'utilisation potentielle de formules et de génération de graphes.

Voici, ci-dessous, un aperçu de la mise en forme du fichier Excel final généré par notre programme :

	A	B	C	D	E	F
1	**Paramètres d'instance**		**a1**	**a2**	**b1**	**b2**
2	**Jobs**	**Machines**	**TSS(I)**	**NEH(I)**	**NEH_Improvement(TSS(I))**	**NEH_Improvement(NEH(I))**
3	n = 5	m = 05	6,64	0,77	0,79	0,23
4		m = 06	6,24	0,74	0,47	0,16
5		m = 07	6,82	0,88	0,48	0,35
6		m = 10	6,74	0,61	0,66	0,2
7		m = 15	4,62	0,72	0,38	0,19
8		m = 20	4,7	0,77	0,57	0,27
9		m = 50	2,46	0,38	0,32	0,13
10		m = 100	1,54	0,17	0,23	0,11
11	n = 6	m = 05	7,35	0,85	0,91	0,31
12		m = 06	7,31	1,05	0,98	0,29
13		m = 07	8,03	1,11	1,09	0,31
14		m = 10	7,86	0,9	1,05	0,23
15		m = 15	6,22	0,99	0,93	0,35
16		m = 20	5,74	0,79	0,87	0,22
17		m = 50	3,44	0,38	0,48	0,17
18		m = 100	2,23	0,38	0,47	0,16

FIGURE 3.8 – Aperçu du fichier final exporté

3.3 Résultats obtenus

Après obtention des résultats provenant de l'application des heuristiques sur les instances générées, il faut exploiter celles-ci afin de faire ressortir des caractéristiques concernant leur efficacité suivant les différents critères. Pour cela des graphiques on été générés afin de les comparer plus facilement. Cette partie montrera d'abord les résultats obtenus lors du projet, puis ceux donnés dans l'article et enfin un comparatif entre les valeurs du projet et de l'article sera effectué.

3.3.1 Résultats du projet

Les graphiques on entièrement été crées à partir du fichier résultat obtenu lors de l'application des heuristiques sur les instances que nous avons générés. Par manque de temps dû à la quantité de temps nécessaire au calcul du QuickPerm sur des instances supérieures à 10 travaux, le graphique traitera seulement pour un nombre de travaux allant de 5 à 10.

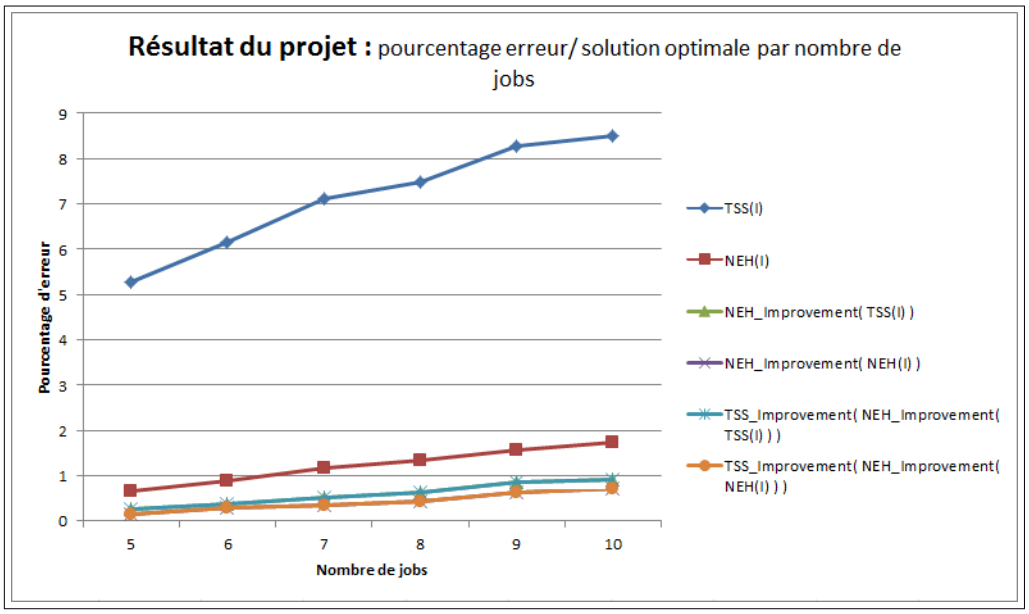


FIGURE 3.9 – Résultat du projet, taux d'erreur pour chaque heuristique en fonction du nombre de jobs d'une instance.

Sur le premier graphique on observe de façon générale un accroissement du pourcentage d'erreur par rapport à la solution optimale lorsque le nombre de travaux augmente.

Plus en détail, on observe que l'heuristique TSS a un pourcentage d'erreur bien plus élevé que NEH, qui est proche des résultats obtenus par les combinaisons d'heuristiques.

Dans les combinaisons d'heuristiques, NEH Improvement appliqué sur les heuristiques initiales donne des résultats meilleurs mais proche de NEH.

En conclusion TSS n'est pas efficace par rapport à NEH. TSS Improvement est moins efficace que NEH Improvement.

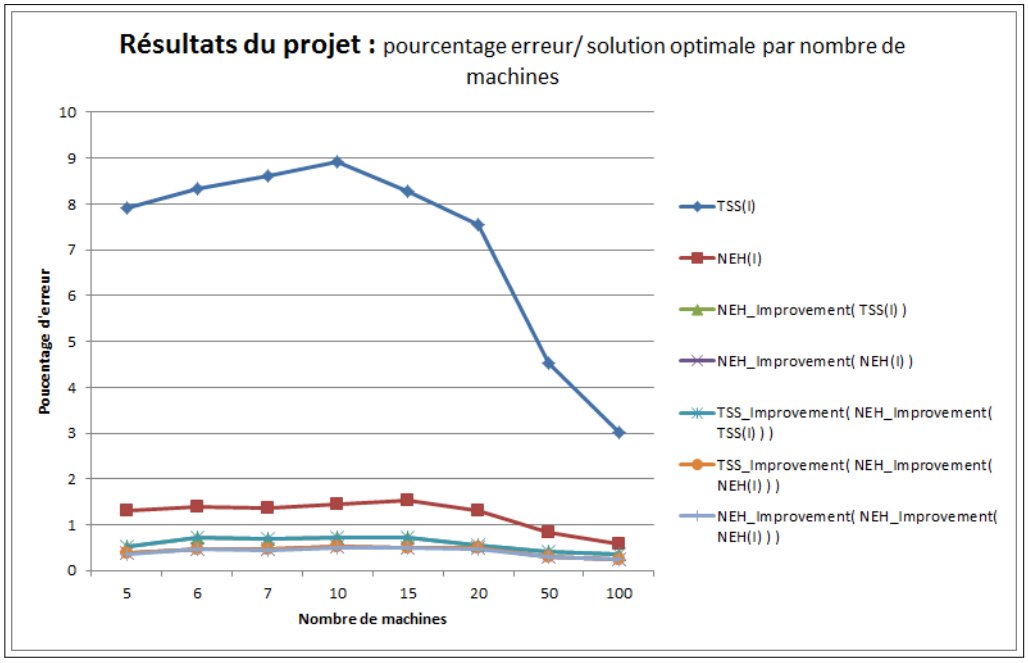


FIGURE 3.10 – Résultat du projet, taux d'erreur pour chaque heuristique en fonction du nombre de machines d'une instance.

Dans ce graphique, TSS tend à s'améliorer lorsque le nombre de machine qui compose une instance augmente, cependant il reste généralement moins efficace que NEH. Dans les combinaisons, c'est NEH Improvement qui obtient les meilleurs résultats, TSS Improvement reste efficace mais n'est pas le plus optimal observé.

3.3.2 Résultats de l'article

L'article donné en temps que base de travail comporte un tableau comprenant des valeurs obtenus par les rédacteurs lors de leurs tests. L'observation des tendances dans les graphiques générés à partir de leurs résultats permettront de comparer nos résultats et valider et/ou invalider leurs expérimentation.

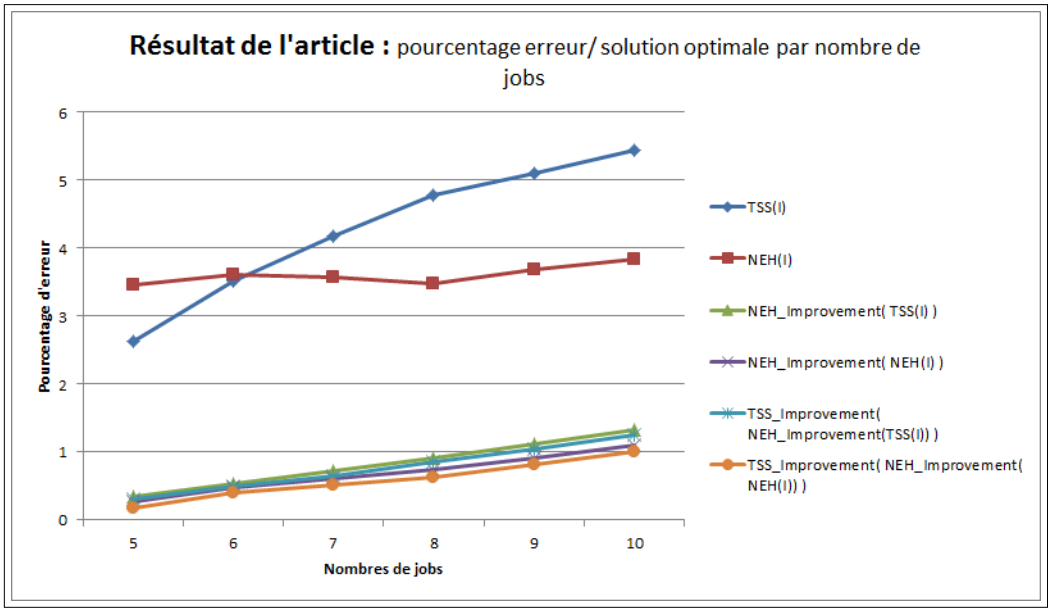


FIGURE 3.11 – Résultat de l'article, taux d'erreur pour chaque heuristique en fonction du nombre de jobs d'une instance.

Dans ce graphique, on observe une augmentation générale du taux d'erreur lorsque le nombre de jobs augmente. Cependant NEH comporte des irrégularités dans la forme de sa courbe, ce qui peut indiquer un manque d'homogénéité dans les résultats obtenus.
Sinon TSS reste globalement moins efficace. En revanche TSS improvement appliqué sur NEH Improvement sur NEH obtient généralement le plus bas taux d'erreur.

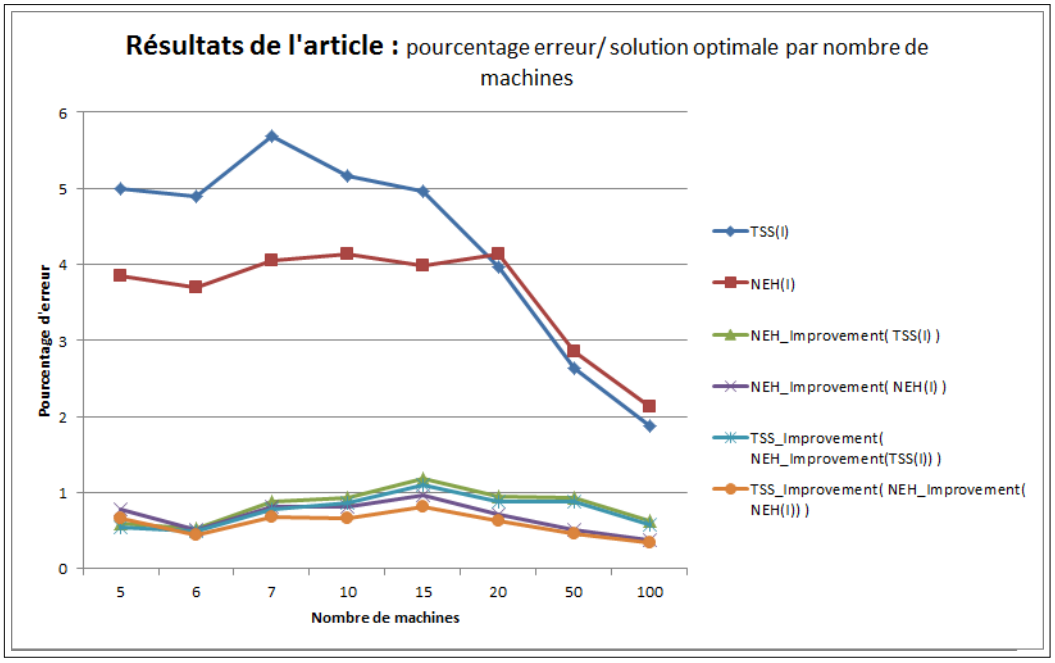


FIGURE 3.12 – Résultat de l'article, taux d'erreur pour chaque heuristique en fonction du nombre de machines d'une instance.

Ce graphique présente des résultats qui ont la forme de résultats non représentatifs du test effectué : des tendances non assumées, un NEH peu efficace et une grande tendance des heuristiques à être plus efficace si le nombre de machine devient grand.

3.3.3 Différences entre le projet et l'article

À partir de la description faite à la section précédente, nous pouvons noter les points suivants :

- TSS reste inefficace face aux autres heuristiques et combinaisons d'heuristiques. Bien que les courbes obtenues dans les différents graphiques aient une forme similaires, les taux d'erreurs obtenus par notre implémentation de TSS reste en moyenne une fois et demie ç deux fois plus grande que les résultats présentés dans l'article ($4,28\%$ contre $7,13\%$).

- NEH reste globalement très efficace dans les résultats de notre projet, ce qui n'est pas représentatif des résultats présentés par l'article avec un taux d'erreur quasiment quatre fois supérieur.

- Le reste des combinaisons d'heuristiques obtient des résultats similaires entre l'article et le projet.

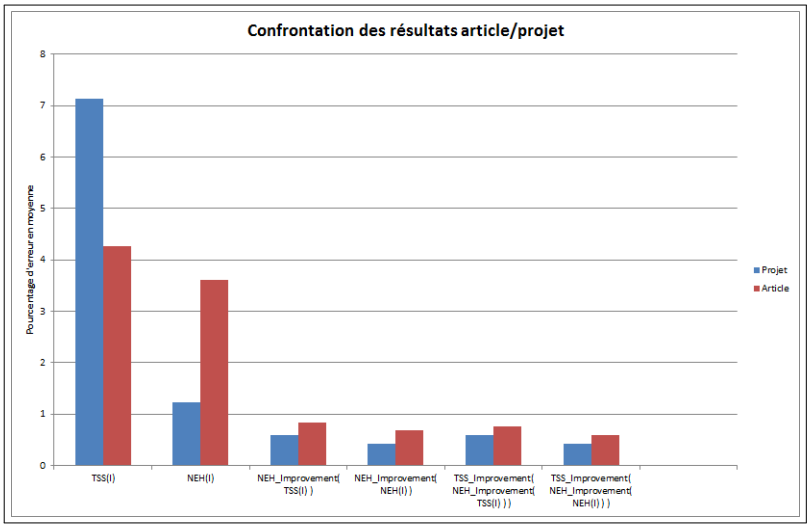


FIGURE 3.13 – Graphique des différences sur le taux d'erreur sur chaque heuristique, indépendamment du nombre de jobs ou de machine.

Conclusion

4.1 Bilan

À l'issue de ce mini projet de GPF, le bilan est majoritairement positif. Effectivement, les cinq grandes phases des objectifs mentionnées dans la partie 1.2 ont toutes été réalisées de manière satisfaisante et dans les temps désirés.

Nous avons ainsi réussi à développer un programme exploitant les quatre heuristiques dont nous avons parlé et générant le tableau de résultats voulu. Ce dernier présente des valeurs cohérentes et satisfaisantes en regard de celui présenté dans l'article.

De plus, au niveau de la gestion du projet, les différentes échéances qui avaient été fixées avec notre encadrant au cours des différentes réunions ont été pour la plupart respectées. Avec une marge d'erreur de plus ou moins deux jours par rapport à la date due.

4.2 Limites

Bien que les objectifs du projet aient été atteints et que les échéances aient été tenues, il est à noter malgré tout un certain nombre de limites et d'améliorations qu'il serait possible d'envisager.

Dans un premier temps, parlons de l'aspect strictement technique du projet, l'implémentation en langage C. En effet, le choix de ce langage de programmation était judicieux pour un certain nombre de raisons. Tout d'abord pour sa rapidité d'exécution, étant un langage proche de la machine. Par ailleurs, le langage C disposant de compilateurs pour une variété de systèmes, il permet de produire un code relativement portable.

Toutefois, à cause d'une mauvaise maîtrise de notre part vis à vis de ce langage ainsi que d'une insuffisance de temps pour parfaire notre code, notre application finale faisant tourner les heuristiques, bien que opérationnelle, souffre de quelques fuites mémoire sur le long terme de son fonctionnement qui pourraient éventuellement se révéler préjudiciables sur une configuration un peu plus modeste.

Dans un second temps, et d'un point de vue fonctionnel, notre programme souffre de lenteur pour le calcul de grandes instances de problèmes. Pour des valeurs de n supérieures ou égales à 10, l'énumération de toutes les solutions d'une instance ainsi que les temps de calcul de toutes les heuristiques prend un temps de plus en plus important.

Une solution à ce problème qui pourrait être envisagée dans le futur serait d'exploiter l'architecture multicoeur d'un ordinateur en rendant notre application multi-thread à des endroits stratégiques dans le but de diminuer au moins par deux le temps total d'exécution.

Heuristiques pour un problème de flowshop avec contraintes mixtes de blocage

Département Informatique
4e année
2012 - 2013

Rapport
Projet de GPF

Résumé : Description en français

Mots clefs : Mots clés français

Abstract: Description en anglais

Keywords: Mots clés en anglais

Encadrant
Christophe LENTE
christophe.lente@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Maxime SERRA
maxime.serra@etu.univ-tours.fr
Fabien FARIN
fabien.farin@etu.univ-tours.fr

DI4 2012 - 2013

Poster

Projet de fin d'études

Minimisation du Cmax d'un flowshop sous diverses contraintes de blocage

Mélanie Maugeais
Encadrants : *C. Lenté, N-Vinh Vo*

Contexte industriel

• Problème de passage d'un poste de travail à un autre, s'il n'existe pas de zone de stockage intermédiaire
➢ Entraîne des blocages
• Etude de certains blocages spécifiques (RSb, RCb, RCb*)

Modélisation

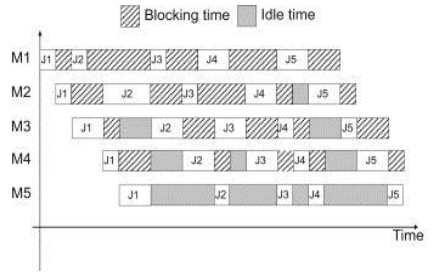
P_1P_2	$P_1P_2P_3$	$P_1P_2P_3P_4$	$P_1P_2P_3P_4$
P_2	P_2P_3	$P_2P_3P_4$	$P_2P_3P_4$
1	P_3	P_3P_4	P_3P_4
0	1	P_4	P_4

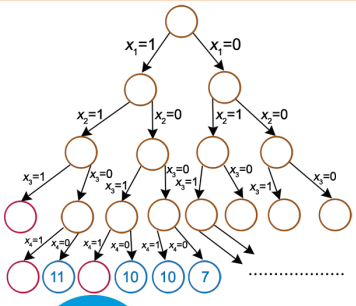
Matrice modélisant un travail sous contrainte de blocage RCb

• Modélisation algébrique
➢ Algèbre Max-Plus
• Permet une analyse du problème

Objectifs

• Comprendre les contraintes de blocage (RSb, RCb, RCb*)
• Etablir la modélisation d'un flowshop soumis à ces contraintes
• Développer une méthode de résolution de type PSE pour optimiser la production sous diverses contraintes de blocage





Département Informatique – Polytech'Tours, 64 av. Portalis 37200 Tours
www.polytech.univ-tours.fr

POLYTECH TOURS *Informatique*

Bibliographie

[1] S. Martinez, S. Dauzère-Pérès, C. Guéret, Y. Mati, N. Sauer. *Complexity of flowshop scheduling problems with a new blocking constraint*. 2005.

[2] W. Trabelsi, C. Sauvez, N. Sauer. *Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems*. 2012.

[3] P. C. Gilmore, R. E. Gomory. *Sequencing a one state-variable machine : a solvable case of the traveling salesman problem*. 1964.

[4] C. Lenté. *Habilitation à diriger des recherches : Mathématiques, ordonnancement et santé*. 2011.

[5] C. Lenté. *Analyse Max-Plus de problèmes d'ordonnancement de type flowshop*. 2001.

[6] Science Direct. *www.sciencedirect.com*.

Etude d'un flowshop pour minimiser le C_{max} sous diverses contraintes de blocage

Département Informatique
5e année
2012 - 2013

Rapport de projet de fin d'étude

Résumé : L'ordonnancement d'atelier est une étape cruciale dans la diminution des coûts de production d'une usine. En effet, selon les problèmes d'atelier étudiés, il est possible d'optimiser le rendement des machines, en diminuant les temps d'attente entre deux ateliers, ou bien encore de réduire les pertes de matières. Dans un contexte industriel, différents blocages peuvent survenir, notamment s'il n'y a pas d'espace de stockage entre deux ateliers. Le projet dont fait état ce rapport est porté sur l'étude de trois différentes contraintes de blocage intervenant dans un flowshop d'atelier, RSb, RCb et RCb*, pour lesquelles peu de travaux ont été réalisés, notamment en ce qui concerne les contraintes RCb et RCb*. Un problème de flowshop pour la minimisation du C_{max} sous les diverses contraintes de blocage précédemment citées est NP-difficile. Le but de cette étude est de créer un modèle en algèbre Max-Plus pour chacun de ces trois problèmes, puis de s'appuyer sur les modélisations établies afin de réaliser une procédure par séparation et évaluation permettant de résoudre de manière optimale des instances de ces problèmes, en garantissant une durée d'exécution bien plus courte qu'une énumération de l'ensemble des solutions.

Mots clefs : RCb, RCb*, RSb, release when starting blocking, release when completing blocking, PSE, procédure par séparation et évaluation, flowshop, Cmax, méthode de résolution optimale, ordonnancement, optimisation, problème NP-difficile

Abstract: Scheduling is a crucial step to reduce manufacturing production costs. Indeed, it is possible to optimize the efficiency on machines, reducing waiting times between two workshops, or to reduce loss of matiere. In an industrial context, different blockings can arise, in particular if there is no storage zone between two workshops. The project this report deals with is about the study of three different kinds of blocking, coming out in flowshops : RSb, RCb and RCb*, for which just a few works has been carried, in particular for RCb anc RCb*. A flowshop problem for the minimization of the C_{max}, under those various blocking constraints is NP-hard. The goal of that study is to create a model in Max-Plus algebra for both problems, then to realize a branch-and-bound procedure to solve optimally those problems by guarenteeing a shortest executing duration than the enumeration of all the solutions.

Keywords: RCb, RCb*, RSb, release when starting blocking, release when completing blocking, branch and bound, flowshop, completion time, scheduling, optimization, NP-hard problem

Encadrants
Christophe LENTÉ
christophe.lente@univ-tours.fr
Nhat Vinh VO
nhat.vo@univ-tours.fr

Étudiante
Mélanie MAUGEAIS
melanie.maugeais@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours