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 PFE

Minimisation de la date moyenne de fin dans un flowshop

Encadrants
Christophe Lenté
christophe.lente@univ-tours.fr
Nhat Vinh Vo
nhat.vo@univ-tours.fr

Étudiant
Pauline Fouillet
pauline.fouillet@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours

Version du 30 avril 2013

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens à remercier Monsieur Lenté pour son calme, sa patience et sa disponibilité.

Je remercie aussi tous les étudiants de $5^{ième}$ année qui ont travaillés à la BU pour leur bonne humeur studieuse.

Introduction

Dans le contexte actuel, il est primordial pour une entreprise industrielle de ne pas perdre de temps, notamment dans un atelier. C'est pourquoi des équipes de recherches présentes dans des laboratoires d'informatiques travaillent sur le sujet de l'optimisation.

C'est le cas du laboratoire d'informatique (LI) de l'université François-Rabelais de Tours. Le LI possède une équipe OC (Ordonnancement et Conduite) qui "prévoit dans le temps l'exécution d'un certain nombre de tâches sur des ressources dans le but d'optimiser un ou plusieurs critères d'évaluation." (http ://li.univ-tours.fr/equipes/equipe-oc)

L'équipe étudie principalement la résolution des problèmes d'ordonnancement et de planification dans le cadre de problématiques issues du monde de l'industrie. Le problème de flowshop $F||\sum C_i$ est étudié dans l'équipe OC, notamment par M. Lenté et Vinh Vo.

Ce projet s'inscrit dans le cadre des projets de fin étude effectués par chaque élève de $5^{\text{ème}}$ année à l'école Polytech'Tours. Ce rapport présente le projet, ses objectifs, le travail réalisé et la démarche entreprie.

Présentation du projet

3.1 Contexte général

La résolution d'un problème de flowshop de type $F||\sum C_i$ a pour but de minimiser les encombrements. En effet, il n'est pas rare de constater ce genre de problèmes dans les ateliers, quand il faut exécuter plusieurs tâches sur plusieurs machines. La minimisation de la somme des dates de fin de chaque tâche sur la dernière machine ($\sum C_i$, notre fonction objectif) est donc essentiel. La somme des dates de fin, $\sum C_i$, ou $\bar{c}$ est illustrée ci-dessous.

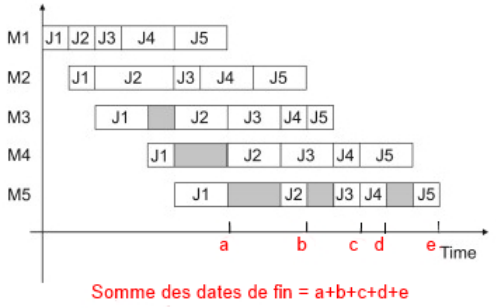


FIGURE 3.1 – La somme des dates de fin ou $\bar{c}$

3.2 Pré-requis

Pour résoudre un problème de flowshop de type $F||\sum C_i$, il existe de nombreuses méthodes exactes. Pour 1 ou 2 machines, c'est un problème qui n'est pas difficile. Cependant, à partir de 3 machines, ce problème devient NP-difficile. Les méthodes exactes peuvent s'avérer trop longues en temps d'exécution.

Une Procédure par Séparation et Évaluation (PSE) est une méthode exacte qui explore les solutions du problème. Pour éviter d'explorer toute l'arborescence, et ainsi gagner du temps, chaque nœud possède des bornes utilisant pour la borne inférieure une modélisation et pour la borne supérieure des heuristiques permettant de s'approcher de la solution optimale. Ces bornes ont pour but de couper des branches de mauvaises solutions. Ces heuristiques conçues spécialement pour répondre au problème souhaité, ici $\sum C_i$, sont calculées à tous les nœuds de l'arborescence.

Lors du déroulement de la PSE, c'est la borne supérieure qui donnera le résultat optimal. La borne supérieure est le résultat de la séquence ordonnancé selon l'heuristique souhaitée, tandis que la borne inférieure permet de couper des branches.

En effet, à chaque nœud, la borne inférieure et la(les) borne(s) supérieure(s) sont calculée(s), comme le montre le schéma ci-dessous. La valeur du meilleur résultat est mise à jour si la borne supérieure est inférieure au meilleur résultat déjà enregistré. La suite d'un nœud n'est pas développé, ie la branche issue du nœud courant est coupée, si la borne inférieure est supérieure au meilleur résultat déjà enregistré.

La déroulement d'une PSE est simple. A chaque nœud courant, ses fils sont développés et on y calcule les bornes. Les fils nœud sont classés par borne inférieure croissante. En effet, c'est le fils qui aura la borne inférieure la plus petite qui sera développé. Si plusieurs fils ont la même borne inférieure, alors le fils ayant la borne supérieure la plus élevée est développé.

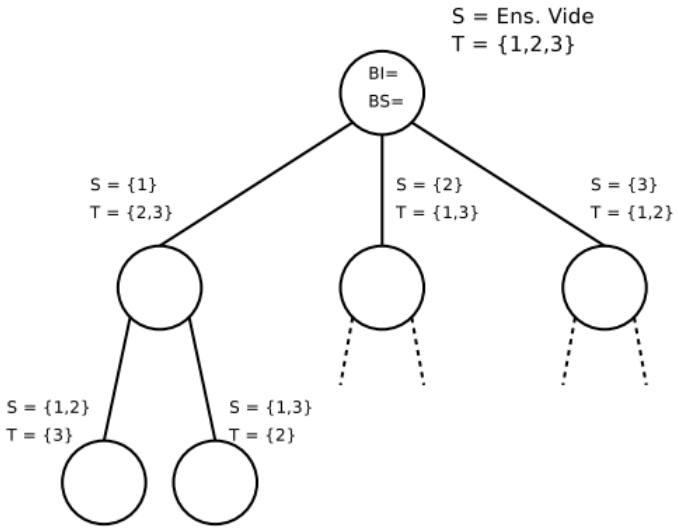


FIGURE 3.2 – Schéma de l'arbre de la PSE (source : Rapport stage de M2R par Vincent Augusto)

C'est donc grâce à de bonnes bornes (supérieures et inférieures) que nous obtiendrons une PSE efficace.

3.3 Existant

3.3.1 Une PSE

Ce projet se base sur une PSE existante réalisée en 2005 par Vincent Augusto dans le cadre de son stage de M2R. Cette PSE a été créée afin de générer des ordonnancements de permutation pour le problème du $F||C_{max}$. L'année dernière, une machine virtuelle Unix a été crée et des améliorations ont été rajoutées à la PSE de 2005. Le fait que la machine virtuelle créée intègre le système d'exploitation UNIX n'est pas anodin.

En effet, la PSE de Vincent Augusto utilise un module en Fortran résolvant le Problème du Voyageur de Commerce (PVC). Il fallait donc un compilateur Fortran et un compilateur C puisque la PSE a été codée en langage C.

3.3.2 Une minoration de la date de fin moyenne d'un flowshop

Une minoration au problème du $F||\sum C_i$ a été modélisée par M. Lenté avec l'algèbre tropicale Max-Plus pour 3 machines et 4 tâches. Cette borne inférieure utilise des graphes qui prennent pour valuation de chaque arc, le temps que met la tache x pour aller de la machine a à la machine b.

Comme le montre le graphique ci dessous, le $Cmax$ pour les 3 premières tâches (ie la séquence sigma : 1 2 3) est le maximum entre les 3 chemins rouges. Nous garderons les tâches sur la machine 1 jusqu'à la dernière tâche (ici, la 3ème tâche).

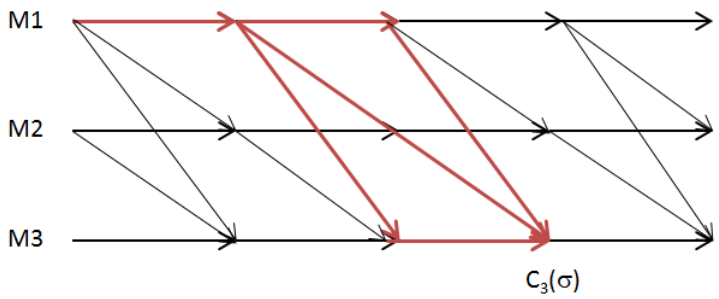


FIGURE 3.3 – Exemple d'arbre pour la borne inférieure

En appliquant cette modélisation à toute les tâches nous pouvons obtenir la somme des C_i pour 4 jobs et 3 machines.

$$\bar{C}_{(\sigma)} \geq T_3 . \left[t_1^{(1)}\right]^3 . \left[t_1^{(2)}\right]^2 . \left[t_1^{(3)}\right]^1 . \left[t_1^{(4)}\right]^0 .\delta[\omega,\sigma(1)]\delta[\sigma(1),\sigma(2)]\delta[\sigma(2),\sigma(3)]\delta[\sigma(3),\sigma(4)]\delta[\sigma(4),\omega]$$

$\sigma(i)$ étant le $i^{ème}$ job de la séquence σ.
$t_{a,b}^{j}$, le temps que met le job j à aller de la machine a à la machine b.

Cette formule nous permet de calculer la borne inférieure puisqu'elle utilise une constante, le résultat de la règle de Smith pour un Flowshop à une machine, et une résolution du problème du voyageur de commerce.

3.4 Objectifs du projet

Le but de ce PFE est de résoudre le problème du $F||\sum C_i$ en réutilisant la PSE créée en 2005. Il s'agit donc ici d'adapter le code au $\bar{c}$ en intégrant les bornes inférieures et supérieures appropriées.

La borne inférieure implémentée est celle modélisée par M. Lenté et les bornes supérieures sont celles présentes sur l'article : "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime" de Quan-Ke Pan et Rube'n Ruiz (Annexe B).

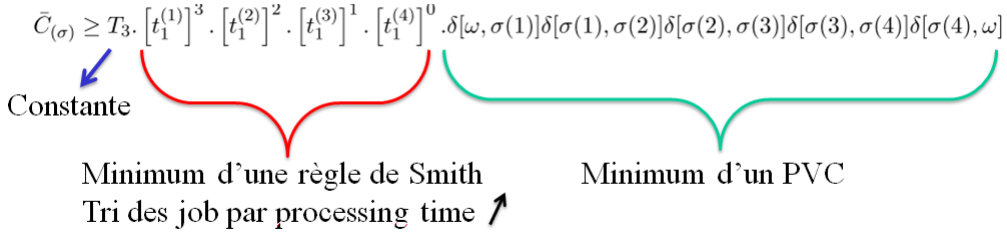
Travail réalisé

4.1 Borne inférieure

4.1.1 Prise en main de la minoration et du module PVC en fortran

Afin de mieux comprendre le fonctionnement de la minoration crée par M. Lenté et le module de résolution d'un PVC, j'ai préféré l'utiliser, dans un premier temps, dans un programme à part et sur un exemple, sur une matrice 4x3.

Rappelons que la formule de la borne inférieure en modélisation Max-Plus est la suivante :



Après initialisation de la matrice Job ($Job[n][m]$ étant le processing time du job n sur la machine m), la constante T_3 fut facile à calculer puisqu'il s'agit de la somme des durées d'exécution sur la dernière machine, ie la $3^{ième}$ machine.

Passons au résultat de la règle de Smith. Pour cela, il suffit de trier les jobs de la machine 1 selon leurs temps d'exécution croissants, et d'y ajouter un coefficient décroissant. Si $tabtri$ est le tableau trié par processing time croissants de la première machine, alors on peut donner la formule du résultat de la règle de Smith comme ci-dessous :
$ResultSmith = (n-1)*tabtri(1)+(n-2)*tabtri(2)+(n-3)*tabtri(3)+(n-4)tabtri(4)$, avec $n = 4$.

Ensuite, pour avoir le résultat d'un PVC (le module en fortran), il faut créer une matrice ligne qui correspond à l'enchainement des colonnes de la matrice des distances. Si nous avons n jobs, nous aurons $n+1$ "villes" dans notre graphe. C'est ce qui explique l'ajout d'un job fictif noté ω.
Pour mieux comprendre, voici la matrice des distances :

	J0	J1	J2	J3	ω
J0	δ(J0,J0)	δ(J0,J1)	δ(J0,J2)	δ(J0,J3)	δ(J0,ω)
J1	δ(J1,J0)	δ(J1,J1)	δ(J1,J2)	δ(J1,J3)	δ(J1,ω)
J2	δ(J2,J0)	δ(J2,J1)	δ(J2,J2)	δ(J2,J3)	δ(J2,ω)
J3	δ(J3,J0)	δ(J3,J1)	δ(J3,J2)	δ(J3,J3)	δ(J3,ω)
ω	δ(ω,J0)	δ(ω,J1)	δ(ω,J2)	δ(ω,J3)	δ(ω,ω)

FIGURE 4.1 – Matrice des distances

La matrice x associée est la matrice ligne suivante :

$$x = [\delta(J0, J0), \delta(J1, J0), \delta(J2, J0), \delta(J3, J0), \delta(\omega, J0), \delta(J0, J1), \delta(J1, J1), \delta(J2, J1), ..., \delta(\omega, \omega)]$$

- $\delta(i, j)$ est la valuation de l'arc i->j et $\delta(i, j) = \frac{t^j_{1,3}}{t^j_3} \oplus \frac{t^i_{1,2}}{t^i_1}\frac{t^j_{2,3}}{t^j_3} \oplus \frac{t^i_{1,3}}{t^i_1}$
  avec $t^j_{a,b}$ le temps que met le job j à aller de la machine a à la machine b.
- $\delta(\omega, i) = \frac{t^i_{1,3}}{t^i_3}$, ω étant un job fictif.
- $\delta(i, \omega) = \mathbb{1}$, $\mathbb{1}$ étant l'élément neutre de $\otimes$ qui équivaut au 0.

Les valeurs des $\delta(i, j)$ sont ici assez facilement calculables, puisqu'il s'agit d'additions et de soustractions de processing times.

Une fois la valeur de la matrice ligne x remplie, il faut lancer la procédure fortran avec la matrice x comme l'un des paramètres.

L'appel à la procédure fortran :

cdt(&n, &ordx, x, &maxnd, &inf, &alpha, &zeur, &binf, fstar, &lb0, &lbc, &nexp, &nprobq, &nass, &active, &lopt, &spars, &avson, &err) ;

Une fois exécutée, il suffit de retourner $binf$ qui est la valeur du chemin optimale. Pour avoir l'enchainement des villes, et donc des jobs, il faut retourner les fstar[i]. La ville fstar[i] est la successeur de la ville (i+1).

C'est en additionnant le résultat de la règle de Smith, la constante T_3 et la valeur $binf$ que nous obtenons une borne inférieure pour le nœud racine. En effet, la minoration est valable pour des machines toutes disponibles à t=0. Ce qui n'est le cas qu'au nœud racine.

4.1.2 Modélisation de la borne inférieure à n jobs et m machines

La minoration de la date de fin moyenne d'un flowshop réalisée par M. Lenté a été établie pour 3 machines. Il a donc fallu généraliser la modélisation pour m machines.

La généralisation a donnée la formule suivante :

$$\bar{C}_{(\sigma)} \geq T_m . \bigotimes_{k=1}^{n-1} \left[t_1^{(k)}\right]^{n-k} . \delta[\omega, \sigma(1)] \delta[\sigma(n), \omega] . \bigotimes_{i=1}^{n-1} \delta[\sigma(i), \sigma(i+1)]$$

avec :

$$T_m = \bigotimes_{b=1}^{n} t_m^{(b)}$$

$$\delta[i,j] = \frac{t_{1,m}^j}{t_m^j} \oplus \frac{t_{1,m}^i}{t_1^i} \bigoplus_{a=2}^{m-1} \frac{t_{1,a}^i}{t_1^i} \frac{t_{a,m}^j}{t_m^j}$$

$$\delta[\omega, i] = \frac{t_{1,m}^i}{t_m^i}$$

$$\delta[i, \omega] = \mathbb{1}$$

Cette modélisation entraîne un codage un peu plus complexe puisque le nombre de quotients (ie de soustractions) n'est pas fixe pour $\delta[i,j]$. Il dépend du nombre de machine m. Lors de l'implémentation, n n'est pas toujours le nombre total de travaux puisque c'est le nombre de travaux qui n'ont pas encore été ordonnancés. Donc n est égal le nombre total de jobs seulement au nœud racine.

4.1.3 Adaptation a un profil de machine, pour une sous séquence donnée

Comme ce qui a été écrit au dessus, il a fallut modifier cette minoration pour qu'elle s'adapte à un profil de machine, où des jobs auraient déjà été placés.

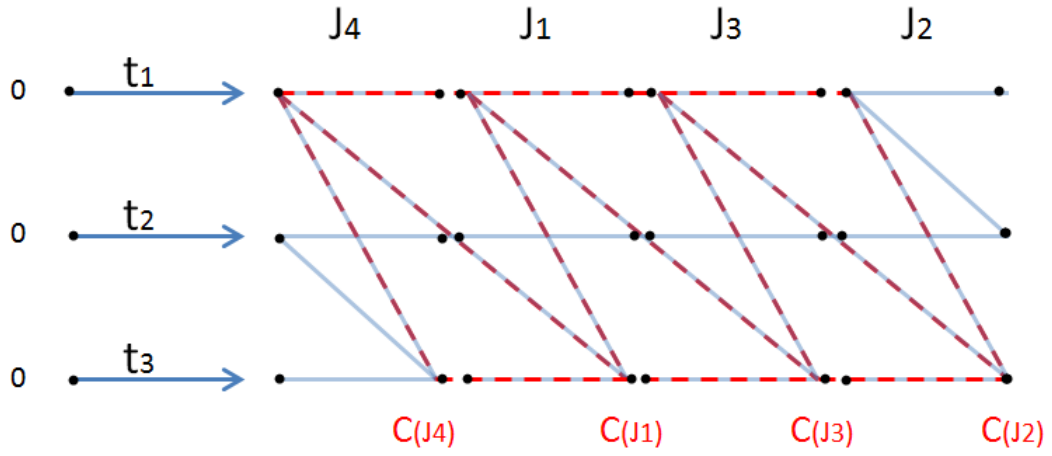


FIGURE 4.2 – Modélisation à un profil de machines

Avant d'expliquer la modélisation, il faut savoir que t_i est la date de disponibilité pour la machine i.
σ est la séquence des jobs qu'il reste à ordonnancer (sur le schéma, $\sigma = J_4, J_1, J_3, J_2$).
τ est la séquence des jobs déjà placés sur les machines.

Comme l'indique le schéma ci-dessus, si l'on reste sur la machine 1, une minoration du $\bar{c}$ est :

$$\bar{c} \geq \bar{c}(\tau) + \sum_{i=1}^{|\sigma|} t_1 + BorneInférieure(\sigma)$$

L'implémentation de la formule est simple. A chaque nœud on applique cette formule sauf au nœud racine. On ajoute $njnp$ (Nombre de Job Non Placés) fois la date de disponibilité sur la machine 1 au résultat de la borne inférieure et on lui ajoute le $\bar{c}$ de la séquence déjà ordonnancée.

4.1.4 Intégration à la PSE

L'implémentation de la borne inférieure à l'intérieur de la PSE se fait dans la fonction $borneinfetsup(pnd, numfils)$ qui prend en paramètre pnd qui est le nœud auquel on veut calculer la borne inférieure et supérieure. En effet, cette fonction calcule les deux bornes mais ne retourne que la valeur de la borne inférieure.

La fonction teste si le nœud est un nœud feuille. En effet, si c'est le cas, il est inutile de calculer une borne inférieure. La valeur du la borne inférieure est mise à jour et correspond à la valeur du $\bar{c}$ du nœud pnd. Ensuite vient le cas d'un nœud non feuille. Dans ce cas la constante et le résultat de la règle de Smith sont calculés. Ensuite le résultat du PVC est retourné grâce à la fonction $appel\_pvc$. Cette fonction est située dans le fichier pvcsimple.h.

Enfin, nous testons si le nœud est au niveau 0, qui correspond au nœud racine. En effet la formule n'est pas la même, comme nous l'avons vu précédemment. La valeur de la borne inférieure calculée est retournée.

4.2 Bornes supérieures

4.2.1 Encadrement du binôme de GPF

Pour la borne supérieure de la PSE, j'ai pu encadrer Yanjing Li et Congqi Wang, un binôme de $4^{ième}$ année dans le cadre de leur mini-projet de Gestion de Flux de Production.

Leur projet avait pour but de coder les heuristiques présentes dans l'article "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime" de Quan-Ke Pan et Rube'n Ruiz. Ces heuristiques simples ou composées sont décrites et leurs algorithmes sont présents dans l'article sus-cité en Annexe B.

Mon travail a consisté à les aider dans leur projet, en leur expliquant les éventuels éléments qu'ils ne comprenaient pas. Pour ce faire, des rendez-vous ont été mis en place de façon hebdomadaire.

Yanjing et Congqi ont réussi à implémenter les 9 heuristiques présentes dans l'article. À savoir LR, NEH, LR_NEH, RZ, VNS, PR1, PR2, PR3 et PR4. Les heurstiques PR sont des heuristiques composées.

À la fin de le projet, ils ont réalisé un programme permettant de comparer ces heuristiques afin de vérifier la cohérence de leurs implémentations avec l'article. En effet, l'article présente un comparatif des fonctions, comme le présente l'image ci-dessous.

En abscisse, il s'agit du RPI (relative percentage increase), qui est le taux d'évolution.
$RPI(c_i) = (c_i - c*)/c* \times 100$
c_i étant la solution obtenue par l'heuristique i et $c*$ la meilleure solution trouvée par n'importe quelle heuristique.

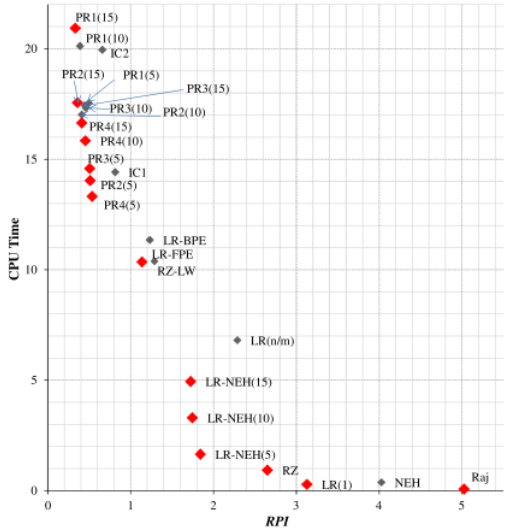


FIGURE 4.3 – Comparatif des différentes heuristiques de l'article

4.2.2 Vérification des codes et choix des heuristiques

Avant de pouvoir utiliser les codes du binôme de GPF, une étape de vérification a été nécessaire pour assurer la cohérence avec l'article. L'article montre que les heuristiques composites telles PR1(15), PR2(15), PR4(15) sont les meilleures. C'est donc par ces heuristiques que j'ai commencé les vérifications.

Hormis quelques erreurs, le code était juste et correspondait aux algorithmes présents sur l'article.

Leur code et le module "comparer" a servit de base pour l'exploitation des résultats.

4.2.3 Macro Excel en VBA

Le programme "comparer" est basé sur les matrices de Taillard. Une instance de Taillard comporte 10 matrices sur un unique fichier txt.

Leur programme retourne un fichier Excel qui pour chaque heuristique et chaque matrice, indique le $\bar{c}$, le temps d'exécution et le RPI. Enfin la dernière colonne du tableau indique le meilleur $\bar{c}$ trouvé.

La macro en langage VBA réalise des calculs sur tous les onglets présents dans le classeur Excel. Ces onglets sont les tableaux Excel du module "comparer" réalisés sur des fichiers de la forme des instances de Taillard.

Lorsque la macro s'exécute, un nouvel onglet "Résultats" se créé, et à la fin de cette feuille, un tableau indique pour chaque heuristique et chaque tableau récapitulatif (onglet), combien de fois l'heuristique est la meilleure et combien de fois elle est strictement la meilleure.

Le tableau ci-dessous est le tableau généré par la macro. (Seule la mise en forme a changée)

Methodes	LR_NEH(5)		LR_NEH(10)		LR_NEH(15)		PR1(5)		PR1(10)	
	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best
1_5x20	0	0	0	0	0	0	0	0	0	0
2_5x15	2	0	2	0	2	0	2	0	2	0
3_5x10	0	0	0	0	0	0	1	0	1	0
4_4x20	0	0	0	0	0	0	0	0	1	0
5_4x15	0	0	0	0	0	0	0	0	1	0
6_4x10	0	0	0	0	0	0	0	0	0	0
7_3x20	0	0	0	0	0	0	0	0	0	0
8_3x15	0	0	0	0	0	0	0	0	0	0
9_3x10	0	0	0	0	0	0	1	0	1	0
somme	2	0	2	0	2	0	4	0	6	0
	PR1(15)		**PR2(5)**		**PR2(10)**		**PR2(15)**		**PR3(5)**	
	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best
1_5x20	0	0	4	0	5	0	10	0	0	0
2_5x15	2	0	2	0	6	0	10	0	2	0
3_5x10	1	0	7	0	9	0	9	0	2	0
4_4x20	1	0	4	0	6	0	9	0	0	0
5_4x15	1	0	3	0	6	0	8	0	1	0
6_4x10	0	0	7	0	10	0	10	0	2	0
7_3x20	0	0	4	0	7	0	10	0	0	0
8_3x15	0	0	5	0	7	0	10	0	0	0
9_3x10	1	0	7	0	10	0	10	0	2	0
somme	6	0	43	0	66	0	86	0	9	0
	PR3(10)		**PR3(15)**		**PR4(5)**		**PR4(10)**		**PR4(15)**	
	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best	Best	Unique Best
1_5x20	0	0	0	0	4	0	5	0	10	0
2_5x15	2	0	2	0	2	0	6	0	10	0
3_5x10	2	0	2	0	8	0	9	0	9	0
4_4x20	1	0	1	0	4	0	6	0	9	0
5_4x15	2	0	2	0	4	0	7	0	9	0
6_4x10	2	0	2	0	7	0	10	0	10	0
7_3x20	0	0	0	0	4	0	7	0	10	0
8_3x15	0	0	0	0	5	0	7	0	10	0
9_3x10	2	0	2	0	7	0	10	0	10	0
somme	11	0	11	0	45	0	67	0	87	0

FIGURE 4.4 – Tableau comparatif des heuristiques

Ici, 90 instances ont été analysées et comparées. Sur aucune instance une heuristique trouve le meilleure $\bar{c}$ toute seule. Cependant, on peut dire que les heuristiques PR2(15) et PR4(15) trouvent le plus souvent le $\bar{c}$ optimal avec un taux de réussite d'environ 95%.

C'est pour cela que les heuristiques PR2(15) et PR4(15) ont été implémentées à la PSE. J'ai aussi implémenté PR1(15) puisque selon l'article de recherche, PR1(15) est la meilleure.

4.2.4 Implémentation des heuristiques

Pour adapter le code créé par Yanjing Li et Congqi Wang, j'ai du modifier le makefile afin que le compilateur compile les fichiers que j'ai directement importer de leur programme.

En effet, PR1, PR2 et PR4 sont des heuristiques composées qui utilisent des heuristiques telles LR, NEH, NEH2, RZ et VNS. Ces heuristiques sont décrites en annexe B.

Cependant, les fonctions PR1, PR2 et PR4 ne pouvaient pas fonctionner dans la PSE puisque les données n'étaient pas les mêmes. En effet, leurs fonctions prennent en paramètre une matrice $M[m][n]$, alors que la PSE utilise une structure $typejobF3d$ job. Les processing times de cette structure sont accessibles par job[numéro_job].p[numéro_machine].

Une fonction d'encapsulation par heuristique a été créée afin de placer les bons processing times dans la matrice M. Cependant, seuls les jobs à ordonnancer doivent être affectés à la matrice M, et non pas ceux déjà placés. Pour cela, $M[i][j] =$ job[$pnd->reste[j]$].p[i], est adapté et fonctionne parfaitement.

L'appel aux fonctions PR1, PR2 et PR3 ont donc été possible. Ces 3 fonctions retournent la nouvelle séquence. Il a donc fallu récupérer la séquence puisque les fonctions retournent le nouvel ordre des indices de la sequence des travaux et non pas le numéro des travaux dans l'ordre.

Prenons un exemple : Nous passons en paramètre la séquence de travaux [2,4,7,1,9]. Si l'ordonnancement retourné par PR est [1,4,0,3,2], cela signifie que les séquence ordonnée est [4,9,2,1,7].
La ligne de code $pnd->reste[resultPR[i]]$ permet de récupérer la bonne séquence.

4.2.5 Heuristique PVC

Une $4^{ième}$ heuristique a été implémentée grâce à la borne inférieure. En effet, quand la borne inférieure résout un PVC, elle donne un ordonnancement des villes. Pour récupérer la séquence des jobs, nous avons le tableau $fstar$. La ville $fstar$[i] est la successeur de la ville (i+1) (i commençant à 0 pour l'indice du tableau, mais les villes commencent à 1 d'où le i+1 : il n'y a pas de ville 0).

Le job fictif, est la ville à l'indice $njnp$, $njnp$ étant le nombre de job non ordonnancé. C'est à partir de ce job fictif que commence l'ordre de notre PVC. L'ordonnancement réel des indices de jobs est :

```
//récupération de l'ordo des villes
seqpvc[0] = fstar[njnp]; //le successeur du job fictif est le premier job de la séquence

for (i = 1 ; i < njnp ; i++)
{
    seqpvc[i] = fstar[seqpvc[i-1]-1];
}
```

FIGURE 4.5 – Récupération de la séquence des jobs suite au PVC

L'ordre des travaux est de la même façon que pour les autres heuristiques :
$pnd->reste[resultPVC$[i]-1]]

4.3 Implémentation au sein de la PSE

La PSE étant établie sur le problème du F$||C_{max}$, diverses fonctions essentielles ont du être modifiées voire refaites.

4.3.1 Modification de la fonction $GetDatesDisponibilite$

La fonction $GetDatesDisponibilite$ permet de calculer les dates de disponibilités des machines d'une sequence d'un nœud. La fonction est initialement située dans le fichier bornes.h.

Cette fonction prend principalement en paramètre le nombre de job à placer pour calculer les dates de disponibilités, la séquence des jobs à placer et les anciennes dates de disponibilités.

Le principe est le suivant, modéliser par les notations Max-Plus.

Avec $D_i =$ nouvelle date de disponibilité sur la machine i

$d_i =$ ancienne date de disponibilité sur la machine i

p_j = processing time du job j

$$
\begin{aligned}
D_0 &= d_0.p_0 \\
D_1 &= D_0 p_1 \oplus d_1 p_1 = (D_0 \oplus d_1) p_1 \\
D_2 &= D_1 p_2 \oplus d_2 p_2 = (D_1 \oplus d_2) p_2 \\
&\vdots \\
D_{m-1} &= (D_{m-2} \oplus d_{m-2}) p_{m-1}
\end{aligned}
$$

Une recopie des dates de disponibilités passées en paramètre sont recopier pour éviter de les modifier.

Algorithm 1 *GetDatesDisponibilite*(NbJobsAPlacer, NbEtages, Sequence, DatesDispo, Pij)

```
for i ← 0 to NbEtages − 1 do
  DatesD[i] ← DatesDispo[i] ;
end for
for i ← 0 to NJobsAPlacer − 1 do
  DatesD[0] ← DatesD[0]+ Pij[Sequence[i]][0] ;
  for j ← 1 to NbEtages − 1 do
    varmax ← max(DatesD[j − 1],DatesD[j]) ;
    DatesD[j] ← varmax + Pij[Sequence[i]][j] ;
  end for
end for
return DatesD ;
```

4.3.2 Ajout de la fonction calculant le $\bar{c}$ d'une séquence

Dans la PSE, il existait une fonction permettant de calculer le C_{max}. Le principe était simple, il suffisait de prendre la date de disponibilité sur la dernière machine.

Le calcule du $\bar{c}$ d'une séquence est plus complexe. La fonction créée prend en paramètre la séquence, les dates de disponibilités des machines, le nombre de jobs non placés et l'ancien $\bar{c}$

L'algorithme de la fonction implémentée est le suivant :

Algorithm 2 *calculcbarre*(Sequence, datesDispo, njnp, cbarre)

```
moncbarre[i] ← cbarre ; {recopie du c̄ passé en paramètre}
for currentJob ← 0 to nbjobs − 1 do
  p[currentJob] ← job[currentJob].p ; {recopie du c̄ passé en paramètre}
end for
for i ← 0 to njnp − 1 do
  sigma[0] ← sequence[i] ;
  datesD ← GetDatesDisponibilite(1,nbmachines, sigma, datesDispo,p) ;
  datesDispo ← datesD ;
  moncbarre ← moncbarre + datesD[nbmachines − 1] ;
end for
return moncbarre ;
```

4.3.3 Choix des bornes supérieures

La PSE de base contient des log (fichier contenant un historique) permettant de suivre l'arborescence. Cette fonctionnalité à été adaptée au problème du $\bar{c}$ et améliorée de manière à analyser les résultats plus facilement.

À chaque nœud calculé, la borne inférieure et les bornes supérieures sont inscrits dans le fichier de log selon les bornes inférieures choisies par l'utilisateur.

En effet, la PSE s'exécute avec une seule, 2, 3 ou les 4 bornes supérieures. L'ajout de booléens permet de choisir les heuristiques que l'on souhaite. Si plusieurs heuristiques sont choisies, la valeur du $\bar{c}$ choisie par la PSE est la plus petite valeur du $\bar{c}$ des séquences retournées par les heuristiques.

4.4 Phase de tests

4.4.1 Jeux de données

La PSE initiale comprend un générateur d'instance nommé fdminmax-gen.

```
./fdminmax-gen m n pmin pmax x x x x nbjeudonnée chemindossier
```

Cette ligne de commande permet de générer $nbjeudonnée$ instances (fichier) dans le dossier spécifié avec m machines n jobs et des processing time compris entre $pmin$ et $pmax$. Les caractères x sont des paramètres pour l'ancienne PSE mais la nouvelle n'en tient pas compte donc ils peuvent être nuls.

Afin de lancer une grande campagne de tests, j'ai généré 5 instances 3x4, 3x5, 3x6, 3x7, 3x8, 4x4, 4x5, 4x6, 4x7, 4x8, 5x4, 5x5, 5x6, 5x7, 5x8 (mxn). La PSE a été lancée sur ces instances et les résultats ont été récupérés.

4.4.2 Programme de force brute

Afin de savoir si la PSE fonctionne correctement, j'ai adapté un programme créé par Maxime Serra et Fabien Farin, étudiants en $4^{ième}$ année à Polytech Tours. Pour leur mini projet de GPF, ils ont travaillé avec Mélanie Maugeais, sur une partie de son PFE. Son PFE porte sur l'étude du problème $F|contraintes|C_{max}$.

Le principe d'un programme de force brute est pour une instance donnée, de générer toutes les séquences possibles, de calculer la solution (dans leur programme, le C_{max}) et de prendre la meilleure. Cette solution est efficace pour des petits jeu de données. En effet, à partir de 8-9 travaux, la force brute met du temps, c'est pourquoi d'autres méthodes comme la PSE existent.

Leur programme étant spécifique au PFE de Mélanie, leurs données étaient spéciales puisqu'elles utilisent diverses contraintes. La modification de la fonction de lecture a donc été nécessaire afin que le programme lise les même instances que la PSE.

La génération de toute la séquence n'a pas été modifiée. Néanmois, j'ai du modifier la fonction qui permettait de calculer le C_{max} et de la transformer en $\bar{c}$. La modification s'est effectuée dans la fonction calculer_Cmax(Sequence * S) issus du fichier Sequence.c. Les structures étant totalement différentes de celle de la PSE, je n'ai pas pu reprendre la fonction calculcbarre que j'ai implémenté au sein de la PSE.

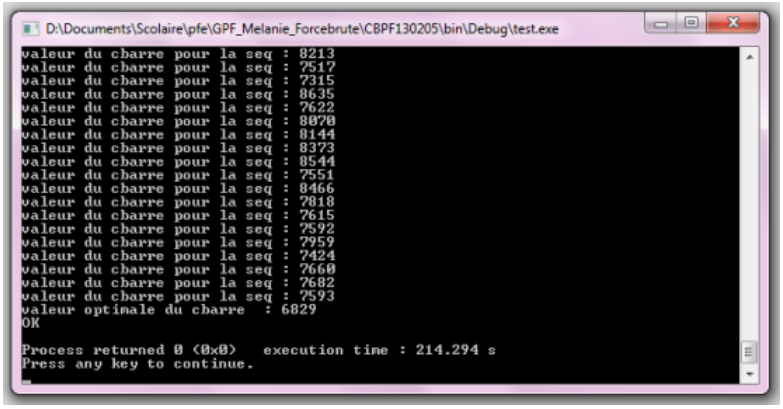


FIGURE 4.6 – Programme de force brute modifié pour le $\bar{c}$

4.4.3 Mise en place de compteurs

Comme il a été dit plus haut, plusieurs heuristiques peuvent faire partie de la PSE simultanément grâce à des booléens.

Le but des compteurs permet de savoir si une heuristique est plus performante qu'une autre. Le principe est simple. Les compteurs sont valables que pour les heuristiques choisies par l'utilisateur parmi les 4. Lors d'un nœud, le compteur de l'heuristique ayant obtenu la plus petite valeur de $\bar{c}$ est incrémenté. C'est le compteur1. Si cette même heuristique est la seule à avoir le plus petit $\bar{c}$ alors son compteur2 est incrémenté.

Puisqu'il y a 4 heuristiques, 8 compteurs ont été mis en place (variables globales) :
`cpt_pr1, cpt_pr2, cpt_pr4, cpt_pvc, cpt_stt_pr1, cpt_stt_pr2, cpt_stt_pr4, cpt_stt_pvc`

Les 4 premiers compteurs sont des compteur1 et les 4 derniers sont des compteur2.

Résultats, limites et améliorations

5.1 Résultats et analyse des résultats

5.1.1 Comparaison Force Brute/Optimal PSE

Pour les 75 instances générées, l'optimal donné par la PSE est toujours le même que celui de la force brute. La PSE est donc fonctionnelle.

5.1.2 Écart borne inférieure - optimal

Une différence Optimal-(Borne inférieure au nœud racine) à été effectuée pour savoir si la borne inférieure est plus efficace sur certains jeux de données que sur d'autres.

Il s'agit de comparer cette différence sur différents jeux de données.

5.1.3 Résultats sur les bornes supérieures

Les heuristiques PR2 et PR4 trouvent quasiment la valeur optimale au nœud racine ; ce qui prouvent que ce sont des heuristiques performantes. D'autre part, cela permet de couper plus de branches.

Les compteurs mis en place montrent aussi la performance de ces deux heuristiques puisque leurs compteur1 sont très élevés. Leur valeur est proche du nombre de nœud évalués par la PSE. Cela signifie que ces heuristiques trouvent l'optimal à quasiment chaque nœud. De plus, leursw< compteur2 sont souvent nuls puisque les deux heuristiques trouvent souvent les mêmes résultats.

Il s'avère que "l'heuristique pvc" (ordonnancement issus du résultat du pvc) est rarement la meilleure. Cependant, quand elle l'est, elle est strictement la meilleure : les autres n'ont pas l'optimal. Son compteur1 et son compteur2 sont peu élevés mais similaires. Cela parait normal, puisqu'elle est totalement différente des heuristiques PR1, PR2 et PR4.

5.1.4 Temps d'exécution

Les temps d'exécution de la PSE s'avèrent très mauvais, la Force brute étant souvent plus rapide que la PSE, si plusieurs heuristiques ont été choisies par l'utilisateur.

Cela est peut-être du à quelques fuites mémoires qui ont pourtant été corrigées pour la plupart d'entre elles.

Cependant, il faut savoir que la PSE tourne sur la machine virtuelle Unix qui est situé sur un périphérique externe tandis que la force brute tourne sous windows. Les conditions ne sont donc pas homogènes.

De plus, le module PVC en fortran est un PSE donc à chaque nœud un problème NP-Difficile est résolu.

5.2 Limites

La PSE est fonctionnelle puisqu'elle trouve le bon optimal. Cependant, à cause de la machine virtuelle ancienne qui fait 20Go située sur un disque dur externe, on ne peut pas en déduire grand chose quant à l'efficacité de la PSE en terme de temps.

De plus, le code a été développé par des personnes différentes. L'ajout de fonctions d'encapsulation de données pour utiliser les différents code, n'accélèrent pas la programme.

5.3 Améliorations

Afin d'améliorer la PSE et de pouvoir juger de son efficacité en terme de temps, il faudrait que la PSE puisse tourner dans les mêmes conditions que la force brute. Il faudrait un système unix capable de compiler du fortran et du C sans utiliser une machine virtuelle.

Il pourrait aussi être utile de transférer tout le projet dans un IDE afin de pouvoir utiliser un debugger et ainsi d'implémenter plus facilement.

La gestion du projet

6.1 Le cahier de spécifications

Pour ce projet, un cahier de spécification a été réalisé en début de parcours. Ce cahier a permis de découper ce projet en tâche et d'avoir un planning à respecter. Il a aussi servit de support en présentant le projet et ses objectifs.

6.2 Le journal de bord

Pendant toute de la durée du projet, un journal de bord a été rédigé. À chaque séance de PFE j'écrivais ce que j'avais fait. Cela permet de suivre son travail, pour ne pas s'éparpiller, même si cela est parfois compliqué.

Il est d'autant plus pratique qu'il est possible de retrouver rapidement quand telle tâche a été faite, combien de temps elle a durée. Cela permet aussi de se rendre compte des difficultés rencontrées et de l'avancement du projet.

6.3 Codage

Le code a été beaucoup commenté afin que d'autres puissent peut-être le réutiliser.

Quant aux sauvegardes, une sauvegarde sur internet de la version courante était faite avant chaque grosse modification.

En effet, comme dans tout projet informatique, le risque de ne plus retrouver ses données est possible, d'autant plus que ce projet est sur une machine virtuelle, sur un disque dur externe.

6.4 Planning

Pour ce projet, j'ai essayé de respecter le planning que je m'étais fixé au départ. Lors de la réalisation du cahier de spécification, un diagramme de Gantt avait été réalisé pour fixer les différentes étapes du projet, le but étant de respecter le planning.

Un nouveau diagramme de Gantt a été fait récemment en montrant le planning des tâches telles qu'elles se sont effectuées réellement.

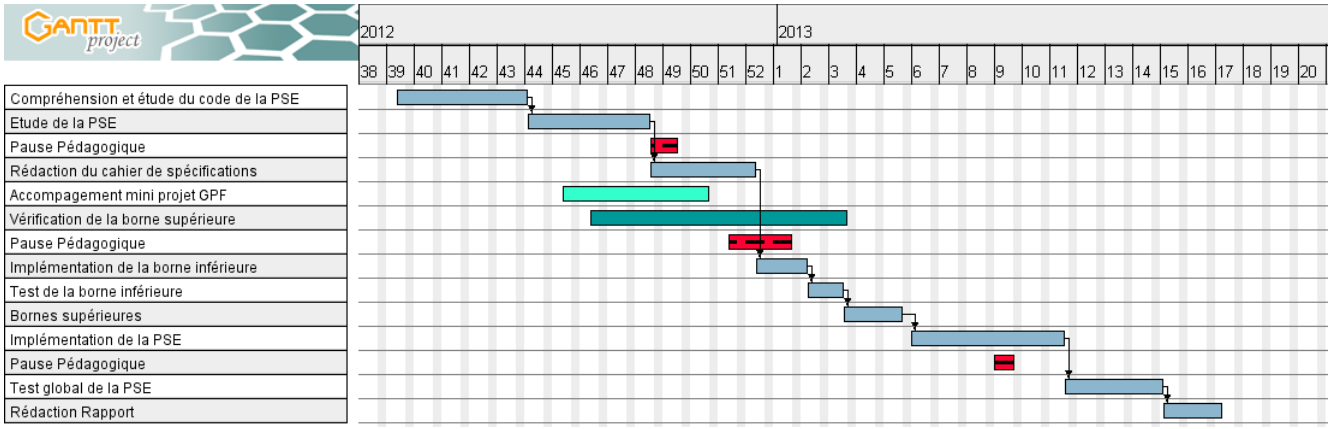


FIGURE 6.1 – Diagramme de Gantt provisoire

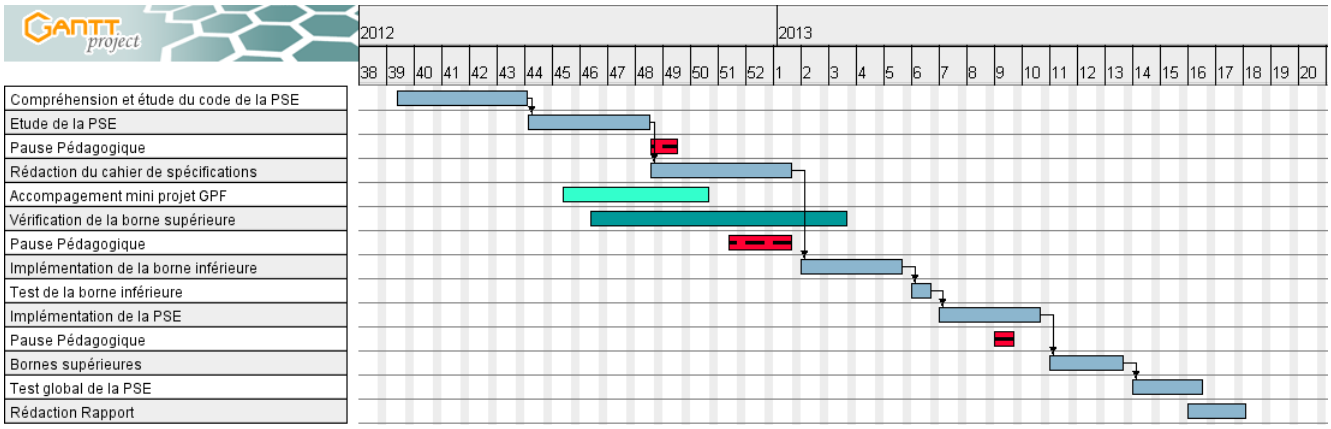


FIGURE 6.2 – Diagramme de Gantt réel

On peut constater que le cahier de spécification m'a pris un peu plus de temps que prévu, ainsi que l'implémentation de la borne inférieure aussi. Cependant les tests sur la borne inférieure ont été moins long puisqu'ils se sont avérés juste. Les bornes supérieures ont aussi pris plus de temps.

Petite remarque concernant l'enchainement des tâches. À la base, je pensais faire l'implémentation des bornes supérieures avant d'implémenter l'ensemble de la PSE (fonction de calcul du $\bar{c}$, dates de disponibilités...). Cela n'était évidement pas possible puisque les bornes supérieures ont besoin de ces fonctions.

Bilan

7.1 Difficultés rencontrées

7.1.1 Reprise de codes

La grosse difficulté de ce PFE, a été de reprendre des codes existants. Même si l'exercice est très répandu, ce n'est pas moins difficile.

La PSE avait déjà été modifiée par plusieurs personnes. Des fonctions ont été rajoutées.

Le nom des fonctions et des variables n'étaient pas toujours explicite, et les commentaires peu nombreux.

Il a aussi fallu que je vérifie et reprenne le code de Yanjing Li et Congqi Wang pour les bornes supérieures. Le code était très peu commenté et peu explicite.

Enfin, le code du binôme de Mélanie était un peu plus commenté, donc j'ai mis moins de temps à le reprendre.

7.1.2 Intégration de nouveaux éléments

L'autre difficulté est de s'adapter au code existant en intégrant nos propres morceaux de code. Les codes ne sont souvent pas compatibles.

7.1.3 Le module du PVC

Le module du PVC en fortran n'a pas été facile à utiliser, notamment pour retourner la séquence et pour lui passer en paramètre la matrice ligne des distances.

7.1.4 Débuguage

Enfin, la dernière difficulté a été de corriger les erreurs. En effet, sans IDE, et donc sans debuguer, débuguer ne fut pas simple. Le mode pas à pas gagne un temps précieux dans le débuguage. Cependant, des printf placés aux endroits stratégiques arrivent aux mêmes fins.

L'installation d'un compilateur Fortran (g77 remplacé par gfortran) sur ma machine unix fut sans succès.

7.2 Compétences acquises

Malgré les difficultés et grâce à ces difficultés, ce PFE m'a beaucoup apporté.

7.2.1 Encadrement d'un mini-projet GPF

J'ai apprécié encadrer Yanjing Li et Congqi Wang pour leur mini-projet GPF. Ces étudiants chinois ont bien travaillé et j'ai apprécié de voir leurs programmes prendre forme.

De plus, les rendez-vous et leur soutenance fut très enrichissant.

7.2.2 Programmation

En matière de programmation j'ai énormément appris. D'une part car j'ai codé moi-même en m'adaptant aux codes déjà implémentés et le langage C est revenu rapidement. D'autre part car j'ai utilisé et vérifié les codes de plusieurs personnes. La diversité des codes m'a fait découvrir de nouvelles manière d'implémenter, et de voir les choses.

Enfin, coder sans IDE demande de la rigueur si l'on souhaite se retrouver dans le code par la suite.

7.2.3 Gestion d'un projet

Ce PFE m'a appris à gérer un projet seule sur une longue durée. L'organisation est essentielle à son bon déroulement.

7.2.4 Connaissance d'heuristiques afin de minimiser le $\bar{c}$

Concernant le projet en lui-même, j'ai découvert le problème du $\bar{c}$ et les heuristiques associées. Ce problème assez général permet d'avoir une vision globale de ce qu'est réellement l'ordonnancement.

7.2.5 Modélisation Max-plus

Enfin, niveau modélisation, j'ai beaucoup apprécié l'utilisation de la notation Max-Plus et la modélisation de la borne inférieure.

Conclusion

Après avoir pris pas mal de temps pour comprendre et intégrer les codes existants, la PSE est finalement fonctionnelle pour le $\bar{c}$. Trois heuristiques composées dont deux très performantes ont été implémentées et comparées. Une borne inférieure utilisant un PVC a aussi été implémentée pour tous les nœuds de la PSE et pour n jobs et m machines.

Cependant, du fait de la machine virtuelle entre autre, la PSE a ses limites mais des améliorations sont possibles.

Finalement, la mise en place de ce projet aura été un excellent moyen de mettre en pratique les compétences qui nous ont été enseignées durant les 3 années à Polytech'Tours. De la gestion de projet à l'implémentation.

Annexe A : Rapport M2R, 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 concluerons 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

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

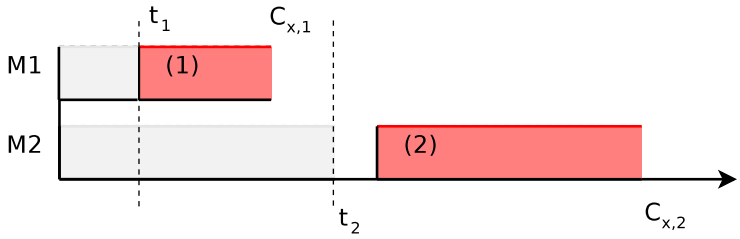


FIG. 2.1 – $F2|delai\ min - max|C_{max}$: premier cas.

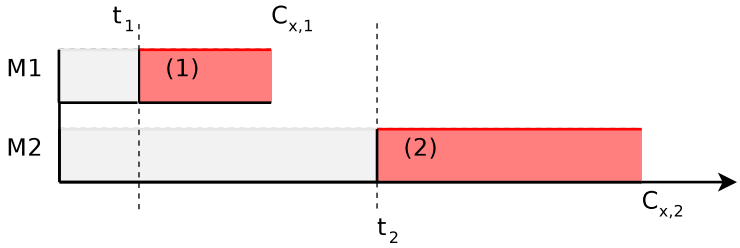


FIG. 2.2 – $F2|delai\ min - max|C_{max}$: deuxième cas.

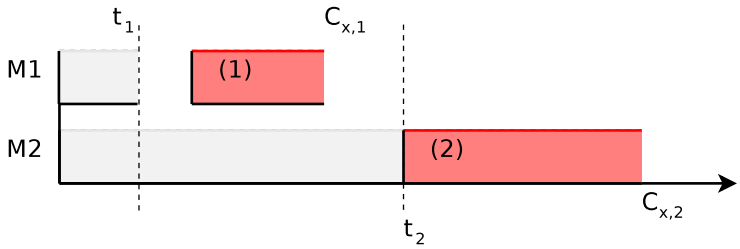


FIG. 2.3 – $F2|delai\ min - max|C_{max}$: troisième cas.

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

$$\vec{C}_x = \vec{t} \begin{pmatrix} p_{x,1} & p_{x,1}\alpha_{x,1}p_{x,2} & p_{x,1}\alpha_{x,1}p_{x,2}\alpha_{x,2}p_{x,3} \\ \mathbf{1} \over \beta_{x,1} & p_{x,2} & p_{x,2}\alpha_{x,2}p_{x,3} \\ \mathbf{1} \over \beta_{x,1}p_{x,2}\beta_{x,2} & \mathbf{1} \over \beta_{x,2} & p_{x,3} \end{pmatrix} \tag{2.11}$$

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 :

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

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

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

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

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.

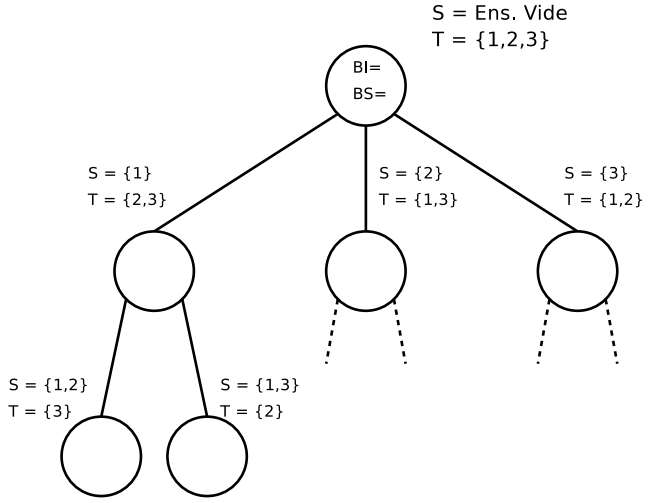


FIG. 3.1 – Schéma représentatif de la PSE.

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 $B_{inf} = d_1 + C_{max}^{NW}(T)$.

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 encourageantes : 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

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

Annexe B : "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime"

Computers & Operations Research 40 (2013) 117–128

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Invited Review

A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime

Quan-Ke Pan [a,b], Rubén Ruiz [c,*]

[a] State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain

ARTICLE INFO

Available online 19 June 2012

Keywords:
Scheduling
Flowshop
Flowtime
Heuristics

ABSTRACT

In recent years, a large number of heuristics have been proposed for the minimization of the total or mean flowtime/completion time of the well-known permutation flowshop scheduling problem. Although some literature reviews and comparisons have been made, they do not include the latest available heuristics and results are hard to compare as no common benchmarks and computing platforms have been employed. Furthermore, existing partial comparisons lack the application of powerful statistical tools. The result is that it is not clear which heuristics, especially among the recent ones, are the best. This paper presents a comprehensive review and computational evaluation as well as a statistical assessment of 22 existing heuristics. From the knowledge obtained after such a detailed comparison, five new heuristics are presented. Careful designs of experiments and analyses of variance (ANOVA) techniques are applied to guarantee sound conclusions. The comparison results identify the best existing methods and show that the five newly presented heuristics are competitive or better than the best performing ones in the literature for the permutation flowshop problem with the total completion time criterion.

Contents

* Corresponding author. Tel.: +34 96 387 70 07x74946; fax: +34 96 387 74 99.
E-mail addresses: panquanke@gmail.com (Q.-K. Pan), rruiz@eio.upv.es (R. Ruiz).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.05.018

1. Introduction

A flowshop is a common layout in production shops where m continuously available machines are disposed in series. Each machine is a production stage and products must visit all machines in order. Scheduling in a flowshop entails the production of n known jobs from a set $J=\{1, 2,..., n\}$. All the n jobs follow the same order of visitation to the machines. This order is, without loss of generality, machine 1, machine 2 and so un until machine m. Each job requires a given known, deterministic and non-negative processing time at each machine, denoted as $p_{i,j}$, $j\in J$, $i=1, 2,\ldots, m$. The flowshop scheduling problem or FSP in short is a theoretical version of reality and several simplifying assumptions apply: all jobs are independent and available for processing at time 0; machines are continuously available; each job is either waiting for processing or being processed by a machine at any given time; machines can only process one job at a time, etc. A complete list of these assumptions is detailed, for example, in Baker [3]. A solution for the FSP is a production sequence or schedule for all jobs which aims at optimizing a given criterion. Most optimization criteria in scheduling are based on the completion times of the jobs or C_j. The time at which a given job finishes processing at a given machine is denoted as $C_{i,j}$ and therefore, $C_{m,j}=C_j$. The most common and widely studied optimization criterion in the flowshop problem is the makespan or $C_{\max}$ minimization. Minimizing makespan is important in situations where a batch of jobs is received and it is required to be completed as soon as possible. For example, a multi-item order submitted by a single customer which needs to be delivered at the earliest possible time. The makespan criterion also increases the utilization of machines. The paper of Johnson [20] is recognized as the pioneering work for the FSP where the specific cases of two and three machines were studied with the objective of makespan minimization. Since then, the FSP has attracted considerable attention from researchers and hundreds of papers have been published in scheduling and related journals. The vast majority of research on flowshop scheduling deals with makespan minimization and several survey papers have been published like those of Framinan et al. [7], Ruiz and Maroto [39], Hejazi and Saghafian [16] and Gupta and Stafford [15].

As of late, there has been an increasing interest in other objective functions. Sometimes each job is needed as soon as it is completed. Similarly, the need to reduce Work In Process (WIP) or in-process inventory has fostered the study of the total flowtime, also referred to as total completion time. When all jobs are available for processing at time 0 (i.e., no release times) the flowtime of a job is equal to its completion time and hence, the total flowtime is equal to $\sum_{j=1}^{n} C_j$. Flowtime minimization leads to a more stable utilization of machines. The FSP with a total flowtime minimization objective was initially classified as $n/m/F/\sum C_j$ following the four parameter notation A/B/C/D of Conway et al. [5]. Later, it has been denoted as $F//\sum C_j$ using the three field notation $\alpha/\beta/\gamma$ of Graham et al. [13]. In the most general setting, the FSP has a search space of $(n!)^m$ sequences. However, the majority of the published research deals with a more restricted version, the so called permutation flowshop scheduling problem of PFSP in which job passing is not allowed and all machines follow the same sequence of jobs. In this case, the search space reduces to $n!$ sequences. The PFSP is classified as $n/m/P/\sum C_j$ or $F/prmu/\sum C_j$ according to Pinedo [34]. We will refer to this last problem with flowtime objective as PFSP-TFT in short. The PFSP-TFT was demonstrated to be NP-Hard in the strong sense for two or more machines by Gonzalez and Sahni [12].

Initial efforts focused on the development of exact implicit enumeration techniques and on approximate approaches to obtain good (but not necessarily optimal) solutions. These solution techniques can be broadly classified into two groups referred to as heuristics and metaheuristics, respectively. Some initial heuristics for the PFSP were introduced by Campbell et al. [4], Gupta [14] and Miyazaki et al. [29], to name just a few. Metaheuristics include many different approaches, like genetic algorithms [42], simulated annealing [44], differential evolution [33] and many others. A metaheuristic method usually obtains better solutions than heuristic algorithms but normally at the cost of significantly added CPU time. Heuristics typically need no more than a few seconds whereas metaheuristics might take several minutes. This is problematic, especially if there are real time requirements or large scale problems [25]. Furthermore, effective and efficient heuristics are still needed in metaheuristic methods for the initial seed sequence. As a result, heuristics are still essential in the scheduling community.

This paper focuses on heuristics for the PFSP-TFT. The flowshop literature already contains some reviews such as Framinan et al. [11]. However, there is room for improvement: comparisons have been performed among no more than a few heuristics; the latest heuristics have not been compared; no common data sets have been used and available results cannot be easily generalized or are not even reproducible; existing comparisons have not carried out comprehensive statistical testing. For all these reasons, we provide an up to date comprehensive review and evaluation of the existing heuristics. From the knowledge obtained after such evaluation we also present five heuristics for the problem under consideration. In total we compare 27 heuristics, which are put through comprehensive computational and statistical testing. The benchmark of choice is given by Taillard [41]. Our results attest to the fact that the five presented heuristics outperform all heuristics proposed up to date.

The rest of the paper is organized as follows: in Section 2, the most well-known heuristics for the PFSP-TFT are reviewed. Section 3 presents the five new heuristics in detail. A comprehensive comparison of the various heuristics is given in Section 4. Finally, we conclude the paper in Section 5.

2. Heuristics for the flowshop scheduling problem

Framinan et al. [11] divided the existing heuristics into two groups: simple and composite methods. A heuristic commonly consists of one or more of three typical phases, namely index development, solution construction, and solution improvement. According to Framinan et al. [11], the method is regarded as composite if it employs a simple heuristic for one or more of the three above-mentioned phases [11]. Conversely, it is regarded as a simple method if no phase contains a heuristic. This distinction is sometimes not easy to apply for some methods but it represents a simple framework. Our literature review is therefore divided between simple and composite heuristics.

2.1. Simple heuristics

The CDS heuristic introduced by Campbell et al. [4] is a simple heuristic for the PFSP. It is basically an extension of the algorithm of Johnson [20]. The CDS creates m^{-1} problems with of two "virtual"

machines, each of them containing some of the original m machines. Johnson's algorithm is then applied to the m^{-1} problems with two virtual machines and m^{-1} sequences are obtained. The schedule with the minimum flowtime is selected. The CDS heuristic has a computational complexity of $O(m^2n+mn\log n)$ and researchers have typically used CDS as benchmark for comparisons. Gupta [14] introduced three simple heuristics, named minimum idle time (MINIT), minimum completion time (MICOT) and MINIMAX algorithms, and compared the results against the CDS heuristic providing better results with less computational time. However, it has to be noted that the maximum instance size tested at that time was really small with just 7 jobs and 20 machines maximum (7×20). Krone and Steiglitz [21] presented an early heuristic in which in the first phase, permutation sequences were improved by insertion movements. In the second phase, job passing was allowed. Miyazaki et al. [29] also presented a heuristic but in this case based on the improvement of the sequence by the interchange of adjacent jobs. Later, Miyazaki and Nishiyama [28] provided a similar extension but for the additional consideration of job weights. Ho and Chang [18] proposed a heuristic that works by minimizing the idle times between jobs in the m machine case. The heuristic was evaluated against other existing methods but mainly those proposed for makespan minimization. Rajendran and Chaudhuri [37] introduced three simple heuristics and compared them with those of Gupta [14], Miyazaki et al. [29] and the aforementioned heuristic of Ho and Chang [18]. The results favored the introduced methods for the studied instances. In a related work, Rajendran and Chaudhuri [36], the same authors presented another heuristic that uses a lower bound in the construction phase of the sequence. The proposed heuristic is applied also to the no-wait problem. No comparisons against the three heuristics of Rajendran and Chaudhuri [37] are shown.

The NEH heuristic of Nawaz et al. [32] is regarded as the best heuristic for the PFSP with makespan criterion [40,39]. It is based on the idea that jobs with larger total processing times should be scheduled as early as possible. Consequently, the heuristic first generates an initial order of jobs with respect to descending sums of their total processing times. Then a job sequence is constructed by evaluating all the sequences obtained by inserting a job from this initial order into all the possible positions of the current partial sequence. The NEH heuristic evaluates $[n(n+1)/2-1]$ sequences and has a complexity of $O(n^3m)$ for the TFT criterion. Due to its effectiveness, the NEH heuristic has been inspiring research on the total completion time criterion since its publication. Rajendran [35] proposed an insertion heuristic, denoted as Raj, having many similarities with the NEH heuristic. The heuristic arranges the jobs according to the weighted total processing times and inserts a job into a restricted subset of all possible positions of the current partial sequence. According to the author's results, the proposed heuristic is more efficient than the methods of Gupta [14], Miyazaki et al. [29] and Ho and Chang [18]. Another heuristic was proposed by Woo and Yim [46] (denoted as WY in short). Unlike the Raj heuristic, WY does not require an initial starting job sequence. However, it also has an insertion phase where a schedule is constructed by inserting all non-scheduled jobs in all possible positions of the partial sequence. This heuristic is also based on the aforementioned NEH heuristic but has a higher complexity of $O(n^4m)$. The authors concluded that their algorithm outperforms the adaptation for flowtime minimization of the NEH, CDS and Raj heuristics.

Framinan et al. [10] investigated the phases of the NEH heuristic and their contribution to its excellent performance regarding makespan minimization. They proposed to modify the NEH heuristic in order to accomplish total flowtime criterion, and proved that the NEH heuristic starting with an initial sequence of jobs sorted by an increasing (instead of decreasing) sum of processing times performs better than the adaptation of the original NEH heuristic. It almost equals the WY heuristic in terms of the quality of the solutions but with smaller computational times. Later, Framinan et al. [9] further delved into the NEH initialization and studied 177 different initial orders for the NEH, including some specially geared towards TFT minimization. Among the proposed methods, a heuristic called B5FT, consisting of the best of five-tuples among the 177 approaches, is shown to outperform the RZ heuristic of Rajendran and Ziegler [38] (to be discussed later) and the WY method which were regarded as the best constructive heuristics for the problem prior to the year 2000 according to Framinan et al. [11]. Framinan and Leisten [8] presented another NEH-based heuristic, referred to as FL, with the same complexity as the WY method. After the insertion process in the basic NEH heuristic, the obtained partial sequence is improved by performing a pairwise interchange improvement procedure. If a better result is obtained, the new partial solution is retained as the current partial sequence. Computational results indicated that this approach outperformed RZ and WY heuristics. More recently, Laha and Sarin [22] have presented a modification of the FL heuristic, denoted as FL-LS. It implements the iteration of the insertion step of the NEH heuristic by performing job insertions rather than the pairwise interchanges. The authors proved by numerical experiments that the modification significantly improves the performance of the FL heuristic while not affecting its computational complexity.

Ho [17] presented a sorting-based heuristic that includes an iterated improvement scheme based on job insertions and pairwise interchanges. The author compared the method with the heuristics of Rajendran and Chaudhuri [37] and Raj of Rajendran [35]. In this case, larger instances of up to 50×20 were tested and the proposed heuristic was shown to be superior. However, this heuristic seems closer to local search techniques such as simulated annealing or tabu search rather than to constructive heuristics as its computational effort does not make it suitable for large problem sizes and/or in those environments where sequencing decisions are required in a short time [11].

Other heuristics assign a weight or index to every job and then arrange the sequence by sorting the jobs according to the assigned index. This idea was exploited by Wang et al. [45]. The authors presented two heuristic approaches by choosing jobs according to a given weight or index function and appending them to a current partial sequence. The first one, named less idle time rule (LIT), focuses on reducing machine idle times, while the second one, named smallest process distance rule (SPD), focuses on reducing both machine idle times and job waiting times. The second approach also consists of two heuristics; one is based on the Euclidean distance measure, while the other is based on the linear distance. The authors did not compare their heuristics with previous ones. Instead, they compared them against the lower bound provided by Ahmadi and Bagchi [1]. The heuristics proposed by Wang et al. [45] have a computational complexity of $O(n^2m)$. The already mentioned RZ heuristic of Rajendran and Ziegler [38] consists of two phases. The first phase involves the generation of a seed sequence according to a priority rule similar to the shortest weighted processing time, whereas the second phase improves the solution by carrying out a local search based on the sequential insertion of each job in the seed sequence at each possible different position of the incumbent partial sequence. The RZ heuristic has a complexity of $O(n^3m)$. Comparisons between the RZ and WY heuristics have been performed by several researchers [9,26]. It was found that the RZ heuristic performs better than the WY heuristic for small-sized problem instances but the relative performance of the WY heuristic improves with increasing number of jobs and finally it surpasses

the RZ heuristic. In addition, the effectiveness of the improvement scheme of the RZ heuristic was also demonstrated by Rajendran and Ziegler [38], and it has been used as an improvement procedure in several composite heuristics [11,25] and Allahverdi and Aldowaisan [2]). Li and Wu [26] have developed an improved RZ heuristic, denoted RZ-LW, where the authors generate an initial sequence by sorting the jobs in ascending order of the sum of processing times, and then perform the RZ local search to the solution until no improvement is found. The performance of RZ-LW is shown to be comparable to that of the Framinan and Leisten [8] but needs far less computational time.

Liu and Reeves [27] proposed a constructive heuristic, referred to as LR that initially sorts jobs according to some indexes that consider both the machine idle times and the effects on the completion times of later jobs. The LR heuristic does not fix the number of sequences to be generated and it is therefore flexible in its computational effort. It can be adjusted according to the requirements of the problem. The benchmark of Taillard [41] has been used to compare the proposed heuristic against the previous ones including Wang et al. [45], Ho [17], Rajendran and Ziegler [38] and Woo and Yim [46]. The computational results demonstrated that the LR heuristic is the best performer, especially in large sized problems.

2.2. Composite heuristics

Liu and Reeves [27] proposed an improvement scheme based on job pairwise exchanges. Starting from an initial sequence, the procedure tries to exchange every job with a certain number of jobs following it in the sequence. If the best sequence obtained by these exchanges is better than the current sequence, it is replaced. After all the jobs are tested, the procedure starts over again from the first job in the sequence. The above procedure is repeated until no improvement can be found for a round of trials (i.e., a form of local search up to local optimality). This procedure is known as the forward pairwise exchange (FPE). The reversed version which checks the exchanges of jobs from right to left in the sequence is called backward pairwise exchange (BPE). The authors studied the effectiveness of different combinations of their heuristics with local search, referred to as LR(x)-FPE and LR(x)-BPE, respectively. The result is that composite methods are more effective than the simple ones at the expense of additional computation time.

Allahverdi and Aldowaisan [2] proposed a total of seven composite heuristics by combining the NEH, WY and RZ methods with local search procedures including FPE with restart (FPE-R in short) and the local search of the RZ heuristic. The authors compared their methods (named IH1∼IH7) against many of the earlier heuristics like those of Ho [17], Wang et al. [45], Rajendran and Ziegler [38] and Woo and Yim [46]. The experimental results indicated that the performance of the heuristic by Ho [17] is good but computationally demanding. They also reported that the heuristics by Wang et al. [45] do not perform well when compared with the others except the CDS method. The proposed heuristics outperform all others in terms of solution quality, and IH7 is the best performer. Framinan and Leisten [8] proposed an improvement to the IH7 heuristic, called IH7-FL, by employing the FL heuristic as an initial solution instead of the WY heuristic as in the original IH7. Later, Framinan et al. [11] presented a comprehensive comparison of recent heuristics for the problem. A total of eight heuristics were compared and a number of composite methods were also presented. One of these new composite heuristics, named C2-FL, is observed to produce better solutions than those of the best method from the earlier study [8].

More recently, Li et al. [25] presented three composite heuristics, denoted as IC1, IC2, and IC3, respectively, by integrating FPE, FPE-R and RZ local search with an effective iterative method where the procedure is repeated until no better solution is found or a given stopping criterion is reached. Computational results show that the three proposed algorithms outperform the existing best composite ones including the C1-FL and C2-FL of Framinan et al. [11] and IH7-FL of Framinan and Leisten [8]. Among the presented heuristics, IC3 performs best in terms of solution quality but needs much more CPU time than both IC1 and IC2. In a related work [24], the authors presented two composite heuristics, named ECH1 and ECH2, which were similar to the heuristics IC1, IC2 and IC3.

A summary of the different heuristics reviewed in chronological order is reported in Table 1.

3. Proposed heuristics

The previous evaluation has prompted us to test some new composite heuristics. We present five new high performing methods. The first one is a simple procedure which combines the LR heuristic of Liu and Reeves [27] and the NEH algorithm. The others are composite heuristics based on this first one and local search methods. More specifically, the RZ local search of Rajendran and Ziegler [38] and a Variable Neighborhood Search scheme (VNS) based on the work of Mladenovic and Hansen [30].

3.1. The presented LR-NEH(x) heuristic

3.1.1. The LR(x) heuristic in detail

The LR(x) heuristic developed by Liu and Reeves [27] constructs x different sequences by appending jobs one by one using an index function. The sequence with the minimum flowtime is selected as the final solution. The index function consists of two terms: the weighted total machine idle time and the artificial total flowtime. Let π be a partial sequence formed by k already scheduled jobs, and U be the set of unscheduled jobs, i.e., those not yet in π. A job $j \in U$ is selected and appended to π according to an index function $\xi_{j,k}$. The weighted total machine idle time between the processing of the job occupying the kth position of the sequence and job j is computed as follows:

$$IT_{j,k} = \sum_{i=2}^{m} \frac{m\max\{C_{i-1,j} - C_{i,[k]}\}}{i + k(m-i)/(n-2)} \tag{1}$$

where $C_{i,[k]}$ is the completion time of the job in the kth position of π at machine i.

The other jobs in U are considered as a single artificial job λ. Its processing time is the average of the processing times of these jobs. Job λ is appended to job j and its completion time $C_{i,\lambda}$ is calculated. Then the total flowtime of jobs j and λ, $AT_{j,k}$, is given below:

$$AT_{j,k} = C_{m,j} + C_{m,\lambda} \tag{2}$$

And the index function $\xi_{j,k}$ is finally defined as follows:

$$\xi_{j,k} = (n-k-2)IT_{j,k} + AT_{j,k} \tag{3}$$

The index function $\xi_{j,k}$ is calculated for all jobs in U. The job with the minimum value of this index function is selected, and ties are broken by selecting the one with the minimum weighted total machine idle time $IT_{j,k}$.

Finally, the procedure of LR(x) is outlined in Fig. 1.

LR(x) does not fix the number of sequences to be generated, and it can be adjusted to the requirements of the problem.

3.1.2. The NEH heuristic

The NEH heuristic of Nawaz et al. [32] was originally designed for the FPSP with the objective of minimizing the makespan. The first step consists of ordering jobs according to descending total

Table 1
Summary of heuristics for flowshop scheduling with total flowtime criterion.

Year	Authors	Acronym	Heuristic type	Comments
1970	Campbell et al.	CDS	Simple	Based on Johnson's rule
1972	Gupta	MINIT	Simple	Based on job pair exchange
		MICOT		Based on job pair exchange
		MINIMAX		Based on Johnson's rule
1974	Krone and Steiglitz		Simple	Based on insertion improvement and job passing
1978	Miyazaki et al.		Simple	Based on interchange of adjacent jobs
1980	Miyazaki and Nishiyama		Simple	Based on interchange of adjacent jobs and job weights
1991	Ho and Chang		Simple	Minimizing the idle time between jobs
1991	Rajendran and Chaudhuri		Simple	Based on lower bound
1992	Rajendran and Chaudhuri		Simple	Considering a job's impact to its immediate successor
1993	Rajendran	Raj	Simple	Based on NEH
1995	Ho		Simple	Based on sorting
1997	Wang et al.	LIT	Simple	Assigning a weight to every job
		SPD1	Simple	Assigning a weight to every job
		SPD2	Simple	Assigning a weight to every job
1997	Rajendran and Ziegler	RZ	Simple	Assigning a weight to every job and performing RZ local search
1998	Woo and Yim	WY	Simple	Based on NEH
2001	Liu and Reeves	LR(x)	Simple	Assigning a weight to every job
		LR(x)-FBE	Composite	Based on LR(x) and FPE
		LR(x)-BPE	composite	Based on LR(x) and BPE
2002	Framinan et al.	NEH-flowtime	Simple	Based on NEH
2002	Allahverdi and Aldowaisan	IH1	Composite	Base on NEH and FPE-R
		IH2	Composite	Based on NEH
		IH3	Composite	Consisting of IH2 and FPE-R
		IH4	Composite	Consisting of WY and FPE-R
		IH5	Composite	Consisting of RZ and FPE-R
		IH6	Composite	Consisting of WY and RZ local search
		IH7	Composite	Consisting of IH6 and FPE-R
2003	Framinan et al.	B5FT	Simple	Based on NEH
2003	Framinan and Leisten	FL	Simple	Based on NEH and interchange
		IH7-FL	Composite	consisting of FL, RZ and FPE-R
2005	Framinan et al.	C1-FL	Composite	Based on LR and FL
		C2-FL	Composite	Based on C1, RZ and FIE-R
2005	Li and Wu	RZ-LW	Simple	Based on iterated RZ local search
2006	Li and Wang	ECH1	Composite	Similar to IC3
		ECH2	Composite	Similar to IC2
2009	Li et al.	IC1	Composite	Consisting of LR and iterated RZ local search.
		IC2	Composite	Consisting of LR, iterated RZ and FPE
		IC3	Composite	Consisting of LR, iterated RZ and FPE-R
2009	Laha and Sarin	FL-LS	Simple	Based on NEH and Insertion

Procedure LR(x)

Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,...,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).

for $l:=1$ **to** x **do** %(generate x sequences)

$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.

for $k:=2$ **to** n **do** %(construct a complete sequence)

Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.

endfor

endfor

return the sequence $\pi\in\{\pi^1,\pi^2,...,\pi^x\}$ with the minimum total flowtime.

Fig. 1. LR(x) heuristic.

processing times. The job with the maximum total processing time is placed first. All other jobs are inserted in all possible positions of the incumbent sequence and finally placed in the position with the lowest partial objective value. The procedure of NEH is described in Fig. 2.

Framinan et al. [10] adapted the NEH heuristic for total flowtime criterion, and found that ranking jobs according to their ascending total processing times performs much better than descending total processing times. As a result, we also employ this improved version. As we can see, the main loop of the NEH can be regarded as an insertion local search around the seed sequence β. We denote this local search as NEH(β) for our other composite heuristics.

3.1.3. The proposed LR-NEH(x) heuristic

The first presented heuristic is denoted as LR-NEH(x). It uses LR(x) and NEH to generate sequences. More specifically, we first generate a partial sequence with d jobs using the LR(x), and then the remaining $n-d$ jobs are inserted into the partial sequence using the NEH heuristic. The relative positions of jobs generated

by the LR(x) are not changed as the algorithm progresses. The procedure of the proposed LR-NEH(x) is outlined in Fig. 3.

LR-NEH(x) has a single parameter d, which is basically the number of jobs after which NEH kicks in. Initial experiments showed that the best value was $d=3n/4$. However, we found that for reasonable values, LR-NEH(x) is robust as regards this parameter. We leave for a further study a deeper examination of the effect of this parameter.

3.2. Composite heuristic PR1(x)

3.2.1. Iterated RZ local search

The improvement procedure presented by Rajendran and Ziegler [38] is a typical local search based on an insertion neighborhood, which sequentially inserts each job in the seed sequence in all possible positions in the incumbent sequence and is, as mentioned, similar to the NEH heuristic. Let $\pi^s=\pi_1^s$, $\pi_2^s,\ldots,\pi_n^s$ be a seed sequence, and π be the incumbent sequence. The procedure of the RZ local search is given in Fig. 4.

The above RZ procedure is a one-pass local search process. The process can be iterated while improvements are found and local optimality is reached. This iterated process can find better results but at the expense of more computational effort. Therefore, a trade-off between effectiveness and efficiency arises. We denote the iterated RZ procedure as iRZ in short.

3.2.2. The proposed composite heuristic PR1(x)

Based on the LR-NEH(x) heuristic and the iRZ local search, we propose a composite heuristic PR1(x). PR1(x) improves each of the solutions generated by LR-NEH(x) using iRZ. To save computational effort, we terminate the iteration if the CPU time is longer than 0.01 mn seconds. The procedure of PR1(x) is outlined in Fig. 5.

Procedure NEH
Generate a job sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_n\}$ by descending order of total processing times.
$\pi:=\{\beta_1\}$
for $k:=2$ **to** n **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the k possible positions of π.
Place job β_k in π at the tested position resulting in the lowest objective value.
endfor
return π

Fig. 2. NEH heuristic.

Procedure $RZ(\pi)$
$\pi^s:=\pi$
for $i:=1$ **to** n **do**
$\pi':=\pi$
Remove job π_i^s from π'.
Take job π_i^s and insert it in all possible positions of π' except for its original position.
Place job π_i^s in π' at the position resulting in the lowest total flowtime.
if $TFT(\pi')<TFT(\pi)$ **then** $\pi:=\pi'$ %($TFT(\pi)$ denotes the total flowtime of π)
endfor
return π

Fig. 4. RZ local search.

Procedure PR1(x)
Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).
$l:=1$
repeat
$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.
for $k:=2$ **to** d **do** %(construct a partial sequence with d jobs)
Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.
endfor
Generate a partial sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_{n-d}\}$ ($\beta_j\in U$, $j=1,2,\ldots,n-d$) by ascending order of total processing times.
for $k:=1$ **to** $n-d$ **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the $k+d$ possible positions of π^l.
Place job β_k in π^l at the tested position resulting in the lowest total flowtime.
endfor
$\pi^l:=iRZ(\pi^l)$ %(perform iRZ local search to π^l)
$l:=l+1$
until $l>x$ **or** $CPUTime>0.01mn$ seconds
return the sequence $\pi\in\{\pi^1,\pi^2,\ldots,\pi^l\}$ with the minimum total flowtime.

Fig. 5. PR1(x) heuristic.

Procedure LR-NEH(x)
Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).
for $l:=1$ **to** x **do** %(generate x sequences)
$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.
for $k:=2$ **to** d **do** %(construct a partial sequence with d jobs)
Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.
endfor
% (NEH heuristic)
Generate a partial sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_{n-d}\}$ ($\beta_j\in U$, $j=1,2,\ldots,n-d$) by ascending order of total processing times.
for $k:=1$ **to** $n-d$ **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the $k+d$ possible positions of π^l.
Place job β_k in π^l at the tested position resulting in the lowest total flowtime.
endfor
endfor
return the sequence $\pi\in\{\pi^1,\pi^2,\ldots,\pi^l\}$ with the minimum total flowtime.

Fig. 3. Proposed LR-NEH(x) heuristic.

3.3. Composite heuristic PR2(x)

The variable neighborhood search (VNS) is an effective metaheuristic presented by Mladenovic and Hansen [30]. Tasgetiren et al. [43] proposed a local search based on the *insertion+interchange* variant of the VNS method and embedded it in a particle swarm optimization algorithm to solve the permutation flowshop with both makespan and total flowtime criterion. As a result, it seems promising to employ VNS in the heuristics. Let π be an incumbent job permutation to improve, and $l_{\max}$ be the maximum number of iterations. The VNS is detailed in Fig. 6.

In the above VNS procedure, the pairwise interchange movement randomly selects two jobs in the sequence ϕ and exchanges their positions. The insertion movement removes a random job from its original position and inserts it in another randomly selected position. In order to have a sufficient exploration of both interchange and insert neighborhoods, we set $l_{\max}$ to $2n^2$. We simply change the iRZ of PR1(x) by VNS, resulting in the PR2(x) heuristic.

3.4. Composite heuristics PR3(x) and PR4(x)

The composite heuristic PR3(x) first generates an initial solution using LR-NEH(x), and then improves the solution using a different improvement procedure. The procedure of PR3(x) is given in Fig. 7.

In the above procedure, the parameter y for the LR-NEH(y) initialization is fixed at 10. The final proposed heuristic PR4(x) uses the VNS local search as an improvement procedure instead of the iRZ local search of PR3(x).

Procedure $VNS(\pi)$
  for $l := 1$ **to** $l_{\max}$ **do**
    improved$:= true$
    repeat
      $\phi := \pi$
      if improved$= true$ **then** Perform a pairwise interchange movement in ϕ.
      else Perform an insertion movement in ϕ.
      if $TFT(\phi) \le TFT(\pi)$ **then** $\pi := \phi$, improved$:= true$
      else improved$:= false$
    until improved$= false$
  endfor
return π

Fig. 6. VNS local search.

Procedure PR3(x)
  $\pi := \mathrm{LR-NEH}(y)$ %(generate an initial solution)
  $\pi^b := \pi$, $l := 1$
  repeat %(improvement procedure)
    $\pi' := iRZ(\pi)$ % (iRZ local search)
    if $TFT(\pi') < TFT(\pi^b)$ **then** $\pi^b := \pi'$.
    $\pi'' := NEH(\pi')$ % (NEH local search)
    if $TFT(\pi'') < TFT(\pi^b)$ **then** $\pi^b := \pi''$
    $\pi := NEH(\pi'')$ % (NEH local search)
    if $TFT(\pi) < TFT(\pi^b)$ **then** $\pi^b := \pi$
    $l := l+1$
  until $l > x$ **or** $CPUTime > 0.01mn$ seconds
return π^b

Fig. 7. PR3(x) heuristic.

4. Computational and statistical experiments

In this section we conduct a comprehensive computational and statistical evaluation of most existing high-performing heuristics as well as of the presented methods. The tested heuristics comprise 14 simple and 13 composite heuristics as follows:

4.1. Simple heuristics

1. Raj heuristic of Rajendran [35],
2–4. LIT, SPD1 and SPD2 heuristics by Wang et al. [45],
5. RZ heuristic of Rajendran and Ziegler [38],
6. WY heuristic by Woo and Yim [46],
7–9. LR(1), LR(n/m) and LR(n) of Liu and Reeves [27],
10. NEH heuristic modified by Framinan et al. [10],
11. FL heuristic of Framinan and Leisten [8],
12. RZ-LW heuristic of Li and Wu [26],
13. FL-LS heuristic by Laha and Sarin [22],
14. Proposed LR-NEH(x) heuristic.

4.2. Composite heuristics

15–16. LR-FPE and LR-BPE of Liu and Reeves [27],
17. IH7 heuristic of Allahverdi and Aldowaisan [2],
18. IH7-FL heuristic of Framinan and Leisten [8],
19–20. Composite heuristics C1-FL and C2-FL of Framinan et al. [11],
21–23. IC1, IC2, IC3 heuristics of Li et al. [25],
24–27. The presented composite heuristics PR1(x), PR2(x), PR3(x) and PR4(x).

All other reviewed heuristics from Section 2 were clearly outperformed by the above heuristics in previous research and are not tested in this work. In addition, the general flowtime computing method presented by Li et al. [19] is employed to save computation time in all heuristics that allow it.

The test bed presented by Taillard [41] is a well-known set for the PFSP with makespan criterion, which consists of a total of 120 instances of various sizes, having 20, 50, 100, 200, and 500 jobs and 5, 10, or 20 machines. These instances are divided into 12 subsets, each of which consists of 10 instances with the same size. These subsets are denoted according to their sizes: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20 and 500×20. Recently, an increasing number of researchers have used this test bed to evaluate their algorithms dealing with total or mean flowtime criterion [27,23,24,43,19,6,25] and Zhang et al. [47], possibly among many others). Thus, we evaluate the above mentioned heuristics based on this test bed, and the performance measure is the relative percentage increase (*RPI*) as follows:

$$RPI(c_i) = (c_i - c^*)/c^* \times 100 \quad (4)$$

where C_i is the solution obtained by the i^{th} heuristic, and c^* is the best solution found by any of the heuristics.

All methods have been coded in Visual C++ 6.0 and run on a cluster of 30 blade severs each one with two Intel XEON 5254 processors running at 2.5 GHz with 16 GB of RAM memory. Each processor has four cores and the experiments are carried out in virtualized Windows XP machines, each one with one virtualized processor and 2 GB of RAM memory. In order to better estimate the performance and elapsed CPU time of the compared algorithms, a total of 5 replications for each instance are carried out. Results are then averaged across the 5 replications for each instance. In our five proposed heuristics LR-NEH(x), PR1(x), PR2(x), PR3(x) and PR4(x), x is tested at three values: 5, 10 and

15. This gives a total of 37 heuristics which are run 5 times each for the 120 instances of Taillard for a grand total of 22,200 results. The average *RPI* values, grouped for each subset (600 results averaged at each cell) are given in Tables 2 and 3 for simple and composite heuristics, respectively. Both types of heuristics are summarized in Table 4, ordered by *RPI*. All CPU times are given in seconds.

It can be seen that simple heuristics clearly perform worse than the composite ones except FL-LS and RZ-LW, which produce slightly smaller *RPI* values than IH7, IH7-FL and C1-FL. Among the simple heuristics, the worst performing algorithms are the three heuristics presented by Wang et al. [45], with SPD1 and SPD2 being more than 15% over the best solution found by any of the compared methods. The best simple heuristic is FL-LS, which produces the smallest mean *RPI* value of 1.22%. However, this is a very costly method, which needs, on average, 120.24 s. As a matter of fact, the column "PARETO" in Table 4 indicates the number of heuristics that Pareto-dominate a given one as regards average *RPI* and average CPU time. For FL-LS we see that there are 16 methods that dominate it: from the 17 heuristics with lower *RPI* values than FL-LS, all of them, except C2-FL, need less CPU time and therefore, dominate, in a Pareto sense, FL-LS.

The presented LR-NEH(x) heuristic yields a much smaller overall *RPI* value than both the NEH and LR heuristics for $x=5$, 10 or 15. Additionally, the proposed LR-NEH(x) is not dominated, i.e., it represents the best trade-off between CPU time and *RPI* among the simple heuristics. Raj is the fastest method, needing barely a tenth of a second, on average.

Among existing composite heuristics from the literature, IC3 is obviously the best performer producing 0.62% *RPI* value within 77.25 s. However, all presented composite heuristics at all tested x values result in lower *RPI* values at a lower computational cost.

Fig. 8 shows a scatter plot of average *RPI* versus average CPU time for the best performing methods. Pareto dominating heuristics are depicted in red (boldface in Table 4).

Previous tables and plots contain average results. We conduct comprehensive statistical analyses to ascertain if the observed differences in *RPI* values are indeed statistically significant. Design of experiments (DOE) and analyses of variance (ANOVA) [31] are conducted for all results.

We consider all 27 tested heuristics. Recall that the proposed methods are tested with three values of x, namely 5, 10 and 15. As a result, we have 37 methods, all of them present in Table 4. Five replicates and 120 instances are tested which recall results in 22,200 treatments. In Taillard's benchmark, not all combinations of n and m are present and therefore, n and m are not orthogonal. In order to study these two factors, we define a factor called type of instance "Type" which has 12 levels, 1 for 20×5, 2 for 20×10 and so on until 12 for 500×20. The replicate (note that all methods are run five independent times) is a witness factor that is shown to be statistically not significant with a p-value close to 1.0. This factor is then removed after validating the experiment. As a result, the initial ANOVA has two factors, Algorithm, with 37 levels, and type of instance, at 12 levels. The response variable is the *RPI*. ANOVA is a parametric statistical tool and there are three main assumptions: normality, homogeneity of variance (homoscedasticity) and independence of the residuals. We carefully checked all three assumptions and the results showed that no major departures were found. The only minor problem is a slight small departure from normality. However, as is well known, ANOVA is robust with respect to the normality assumption. The result of the ANOVA is that the two factors as well as the interaction between the two are statistically significant with p-values very close to 0. The most significant factor is the

Table 2
Results of the simple heuristics.

Instance	Raj		LIT		SPD1		SPD2		RZ		WY		LR(1)		LR(n/m)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	5.80	0.00	9.40	0.00	18.85	0.00	19.35	0.00	2.90	0.00	3.03	0.00	2.74	0.00	2.55	0.00
20×10	4.61	0.00	11.23	0.00	15.56	0.00	14.37	0.00	1.90	0.00	2.80	0.00	3.77	0.00	3.42	0.00
20×20	4.55	0.00	6.95	0.00	10.23	0.00	10.07	0.00	1.97	0.00	2.96	0.00	3.21	0.00	3.21	0.00
50×5	4.51	0.00	7.69	0.00	21.01	0.00	21.56	0.00	2.70	0.00	3.70	0.01	2.20	0.00	1.56	0.01
50×10	6.28	0.00	9.45	0.01	15.96	0.01	14.24	0.00	3.22	0.01	3.39	0.02	5.26	0.00	2.97	0.01
50×20	5.88	0.00	10.19	0.02	11.57	0.01	10.74	0.00	2.94	0.01	3.22	0.04	3.85	0.01	3.40	0.01
100×5	4.09	0.00	5.42	0.03	23.05	0.03	23.31	0.00	2.57	0.02	2.32	0.17	1.25	0.01	0.63	0.17
100×10	4.73	0.00	8.04	0.06	20.98	0.05	19.55	0.00	3.26	0.05	2.78	0.35	2.89	0.01	2.03	0.14
100×20	6.03	0.01	8.19	0.14	12.41	0.11	10.22	0.00	2.72	0.09	3.03	0.68	4.28	0.03	3.36	0.14
200×10	4.64	0.03	6.39	0.54	22.25	0.46	21.50	0.10	2.77	0.33	2.56	6.03	2.47	0.10	1.39	1.99
200×20	4.98	0.06	9.14	1.01	17.80	0.90	14.95	5.82	2.51	0.70	2.35	11.72	3.81	0.20	2.00	2.00
500×20	4.21	0.81	7.03	15.28	18.76	12.61	18.87	11.29	2.34	10.01	1.79	483.79	1.79	3.10	0.94	77.23
Average	5.02	0.08	8.26	1.42	17.37	1.18	16.56	1.43	2.65	0.94	2.83	41.90	3.13	0.29	2.29	6.81
Instance	LR(n)		NEH		FL		RZ-LW		FL-LS		LR-NEH(5)		LR-NEH(10)		LR-NEH(15)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	2.44	0.00	5.05	0.00	2.49	0.00	1.54	0.00	1.74	0.00	2.26	0.00	2.26	0.00	2.26	0.00
20×10	3.35	0.00	4.14	0.00	2.40	0.00	1.43	0.00	1.25	0.00	2.51	0.00	2.51	0.00	2.51	0.00
20×20	2.42	0.01	3.88	0.00	2.44	0.00	1.07	0.00	1.04	0.01	2.01	0.00	1.89	0.00	1.89	0.01
50×5	1.56	0.05	4.25	0.00	2.02	0.03	1.63	0.01	1.37	0.04	1.33	0.01	1.33	0.01	1.33	0.02
50×10	2.81	0.10	5.08	0.00	2.54	0.05	1.44	0.04	1.61	0.08	2.48	0.01	2.43	0.02	2.43	0.04
50×20	3.07	0.23	4.39	0.01	2.01	0.10	1.63	0.05	1.28	0.17	2.67	0.02	2.43	0.05	2.43	0.07
100×5	0.63	0.83	3.18	0.01	1.20	0.31	1.69	0.10	1.12	0.52	1.35	0.04	1.25	0.07	1.25	0.11
100×10	2.00	1.37	4.54	0.02	2.41	0.73	1.09	0.28	1.12	1.24	1.67	0.08	1.54	0.16	1.54	0.24
100×20	2.67	2.76	4.48	0.04	2.04	1.47	0.87	0.57	0.91	2.59	2.44	0.17	2.16	0.34	1.98	0.52
200×10	1.39	19.85	3.16	0.14	1.39	10.16	1.40	2.42	1.11	18.38	1.03	0.59	0.96	1.16	0.95	1.74
200×20	1.94	39.98	3.88	0.29	1.62	20.91	0.80	6.26	1.32	38.61	1.57	1.27	1.52	2.57	1.48	3.74
500×20	0.82	1544.32	2.32	4.00	1.29	701.12	0.88	114.84	0.73	1381.23	0.80	17.66	0.68	35.24	0.65	52.85
Average	2.09	134.12	4.03	0.37	1.99	61.24	1.29	10.38	1.22	120.24	1.84	1.65	1.75	3.30	1.72	4.94

Table 3
Results of the composite heuristics.

Instance	LR-FPE		LR-BPE		IH7		IH7-FL		C1-FL		C2-FL		IC1		IC2		IC3		PR1(5)		PR1(10)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	1.30	0.00	1.40	0.00	1.31	0.00	1.66	0.00	1.68	0.00	1.22	0.00	1.00	0.00	0.66	0.00	0.66	0.00	0.57	0.00	0.53	0.01
20×10	1.91	0.00	1.91	0.00	1.45	0.00	1.45	0.00	1.74	0.00	0.90	0.00	1.30	0.00	1.07	0.00	1.06	0.00	0.84	0.01	0.63	0.01
20×20	1.60	0.00	2.07	0.00	1.24	0.00	1.34	0.00	2.32	0.00	1.12	0.01	1.29	0.00	1.35	0.00	1.32	0.00	0.56	0.01	0.40	0.03
50×5	0.72	0.02	0.83	0.02	1.61	0.03	1.35	0.03	1.67	0.03	1.04	0.05	0.84	0.02	0.61	0.03	0.62	0.03	0.51	0.05	0.46	0.11
50×10	1.37	0.03	1.51	0.02	1.84	0.04	1.63	0.07	2.27	0.06	1.46	0.09	1.10	0.04	0.87	0.06	0.82	0.07	0.50	0.14	0.30	0.27
50×20	1.70	0.04	2.12	0.05	1.78	0.08	1.33	0.13	2.63	0.11	1.36	0.17	1.05	0.07	0.77	0.12	0.68	0.15	0.77	0.27	0.44	0.54
100×5	0.34	0.22	0.33	0.21	1.01	0.55	0.71	0.47	0.94	0.43	0.67	0.54	0.30	0.24	0.14	0.30	0.18	0.38	0.56	0.43	0.47	0.90
100×10	0.87	0.29	0.87	0.29	1.52	0.81	1.71	0.91	1.80	0.85	0.74	1.33	0.69	0.36	0.52	0.57	0.48	1.02	0.53	1.06	0.46	2.24
100×20	1.86	0.40	2.04	0.51	1.49	1.40	1.44	1.77	1.98	1.57	1.01	2.90	1.01	0.61	0.92	1.13	0.77	1.91	0.37	2.81	0.26	5.53
200×10	0.59	3.08	0.53	3.10	1.41	15.77	0.91	14.28	1.19	11.48	0.60	20.66	0.52	3.80	0.33	6.29	0.29	14.29	0.22	10.10	0.21	19.35
200×20	0.97	4.63	0.80	5.07	1.32	24.18	1.19	25.56	1.60	22.22	0.82	49.46	0.48	6.50	0.47	11.08	0.43	21.86	0.18	26.17	0.18	43.16
500×20	0.39	115.48	0.38	126.84	1.13	1006.51	0.89	989.50	0.82	730.96	0.40	1583.46	0.20	161.33	0.17	219.79	0.14	887.24	0.34	169.30	0.34	169.30
Average	1.14	10.35	1.23	11.34	1.43	87.45	1.30	86.06	1.72	63.98	0.95	138.22	0.81	14.41	0.66	19.95	0.62	77.25	0.50	17.53	0.39	20.12

Instance	PR1(15)		PR2(5)		PR2(10)		PR2(15)		PR3(5)		PR3(10)		PR3(15)		PR4(5)		PR4(10)		PR4(15)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	0.37	0.01	0.56	0.01	0.45	0.02	0.37	0.02	0.88	0.00	0.85	0.01	0.84	0.01	0.64	0.01	0.46	0.02	0.40	0.02
20×10	0.48	0.02	0.86	0.02	0.62	0.03	0.52	0.05	0.68	0.01	0.68	0.01	0.68	0.02	0.75	0.02	0.57	0.03	0.49	0.04
20×20	0.31	0.04	0.56	0.03	0.45	0.06	0.31	0.08	0.65	0.02	0.57	0.02	0.57	0.04	0.57	0.03	0.45	0.05	0.40	0.07
50×5	0.45	0.16	0.33	0.10	0.30	0.20	0.30	0.30	0.61	0.07	0.57	0.14	0.57	0.20	0.45	0.12	0.39	0.22	0.34	0.33
50×10	0.24	0.39	0.68	0.23	0.55	0.45	0.45	0.67	0.59	0.17	0.43	0.30	0.42	0.44	0.87	0.27	0.69	0.51	0.58	0.75
50×20	0.38	0.83	0.86	0.44	0.63	0.90	0.54	1.34	0.50	0.32	0.39	0.62	0.32	0.92	0.77	0.51	0.64	0.98	0.56	1.44
100×5	0.43	1.37	0.27	0.64	0.22	1.28	0.21	1.92	0.56	0.58	0.50	1.14	0.49	1.66	0.33	0.78	0.30	1.49	0.27	2.20
100×10	0.41	3.41	0.39	1.58	0.36	3.16	0.33	4.73	0.43	1.49	0.40	2.86	0.36	4.24	0.48	1.88	0.46	3.61	0.44	5.35
100×20	0.20	8.22	0.65	3.38	0.48	6.76	0.40	10.14	0.26	3.18	0.25	6.11	0.22	9.01	0.72	3.93	0.66	7.56	0.60	11.17
200×10	0.19	21.18	0.16	11.11	0.14	21.18	0.14	21.25	0.28	13.11	0.26	21.22	0.26	21.18	0.21	13.34	0.21	20.96	0.21	20.96
200×20	0.18	43.14	0.49	24.19	0.39	43.44	0.39	43.44	0.42	25.78	0.42	43.12	0.42	43.14	0.44	28.37	0.43	43.98	0.43	44.00
500×20	0.34	172.36	0.33	126.74	0.33	126.74	0.33	126.78	0.21	130.25	0.21	132.19	0.21	128.32	0.19	110.61	0.19	110.61	0.19	113.38
Average	0.33	20.93	0.51	14.04	0.41	17.02	0.36	17.56	0.51	14.58	0.46	17.31	0.45	17.43	0.54	13.32	0.45	15.84	0.41	16.64

Table 4
Summary of all heuristics, ordered by *RPI*.

#	Algorithm	*RPI*	Time	Type	PARETO	#	Algorithm	*RPI*	Time	Type	PARETO
1	**PR1(15)**	0.33	20.93	Composite	0	20	RZ-LW	1.29	10.38	Simple	1
2	**PR2(15)**	0.36	17.56	Composite	0	21	IH7-FL	1.30	86.06	Composite	18
3	PR1(10)	0.39	20.12	Composite	1	22	IH7	1.43	87.45	Composite	19
4	PR2(10)	0.41	17.02	Composite	1	23	C1-FL	1.72	63.98	Composite	17
5	**PR4(15)**	0.41	16.64	Composite	0	24	**LR-NEH(15)**	1.72	4.94	Simple	0
6	PR3(15)	0.45	17.43	Composite	2	25	**LR-NEH(10)**	1.75	3.30	Simple	0
7	**PR4(10)**	0.45	15.84	Composite	0	26	**LR-NEH(5)**	1.84	1.65	Simple	0
8	PR3(10)	0.46	17.31	Composite	3	27	FL	1.99	61.24	Simple	20
9	PR1(5)	0.50	17.53	Composite	5	28	LR(n)	2.09	134.12	Simple	26
10	**PR3(5)**	0.51	14.58	Composite	0	29	LR(n/m)	2.29	6.81	Simple	3
11	**PR2(5)**	0.51	14.04	Composite	0	30	**RZ**	2.65	0.94	Simple	0
12	**PR4(5)**	0.53	13.32	Composite	0	31	WY	2.83	41.90	Simple	22
13	IC3	0.62	77.25	Composite	12	32	**LR(1)**	3.13	0.29	Simple	0
14	IC2	0.66	19.95	Composite	10	33	NEH	4.03	0.37	Simple	1
15	IC1	0.81	14.41	Composite	2	34	**Raj**	5.02	0.08	Simple	0
16	C2-FL	0.95	138.22	Composite	15	35	LIT	8.26	1.42	Simple	4
17	**LR-FPE**	1.14	10.35	Composite	0	36	SPD2	16.56	1.43	Simple	5
18	FL-LS	1.22	120.24	Simple	16	37	SPD1	17.37	1.18	Simple	4
19	LR-BPE	1.23	11.34	Composite	1						

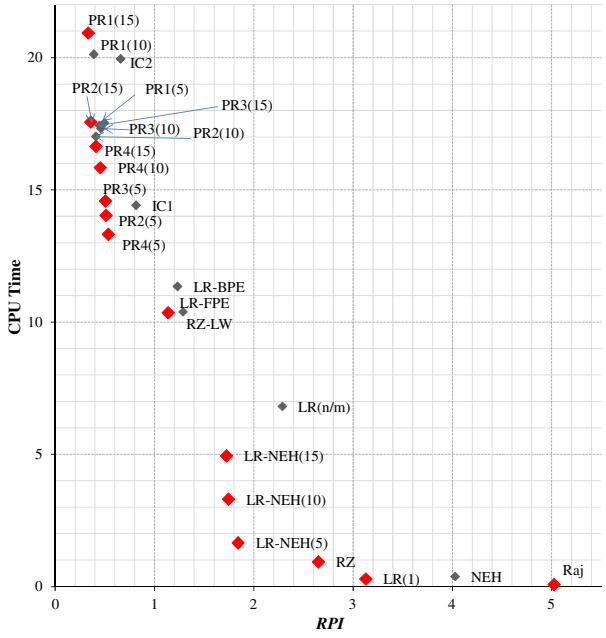


Fig. 8. Average *RPI* versus average CPU time for the heuristics. Pareto dominant methods shown in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

Algorithm with an *F*-Ratio close to 5000. An initial means plot with 99% confidence level intervals (not shown due to space limitations) clearly shows that the following heuristics are statistically different (from worst to best): SPD1, SPD2, LIT, Raj, NEH and LR(1). Note that all these methods have average RPI over 3%. They are statistically worse than all other heuristics and are therefore disregarded from the ANOVA and from following plots (this in turn also helps with the normality assumption). We employ the most restrictive technique for calculating the confidence intervals around the means: the Tukey's Honest Significant Difference (HSD). If the intervals around two plotted means overlap, it means that there are no statistically significant differences between the means at the given confidence level. The means plot with the remaining 31 heuristics is given in Fig. 9.

It can be observed from Fig. 9 that heuristics can be divided into 14 homogenous groups where no significant differences can be found within each group. The last three heuristics are each in a group, i.e., from worst to best, WY, RZ and LR(n/m). Group 11 is formed by methods FL and LR(n). In Fig. 9 we see how the confidence intervals for the average *RPI* of these two methods overlap. All groups are formed in a similar way. Fig. 9 also shows, in red, the intervals from those Pareto non-dominated heuristics.

It is shown that our proposed methods, from PR1 to PR4, in all three values of x, are better than all other heuristics. Statistically speaking, from PR1(15) to PR3(10) we have significant differences with IC3, the best competing method from the literature. It has to be pointed out that IC3 needs, on average, more than 77 s of CPU time while PR1(15) needs less than 21 s on average.

Of course, all previous statistical analyses depict the overall picture of the heuristics across all instances. The results vary slightly from one instance size to another. The interaction between the type of instance and the heuristics is relatively weak (still statistically significant but with a rather small *F*-Ratio). An example of this type of interaction is given in Fig. 10. Only four heuristics are shown for clarity.

We see that the confidence intervals are very wide (there are only 10 instances at each group) and there is not enough data to draw strong conclusions. We see, for example, how PR1(15) is statistically equivalent to IC3 for several instance groups, while being better for the others. Only for 100×5, IC3 results in a lower *RPI*, albeit this difference is not statistically significant.

There are other cases where some differences appear. However, we believe that the main interest lies with the average performance depicted in the previous Fig. 9.

In a nutshell, we put forward the following statements based on the above comparison and analysis. (1) All the simple heuristics are surpassed by the composite ones except FL-LS and RZ-LW regarding to the quality of solutions. On the other hand, most simple heuristics run much faster than their composite counterparts. (2) The best four simple heuristics are FL-LS, RZ-LW, the proposed LR-NEH(x) and FL in terms of effectiveness with the computational time of the proposed LR-NEH(x) being much less than that of the other three heuristics. (3) Both IC2 and IC3 are significantly better than the other existing composite heuristics in terms of overall *RPI* value. However, both are outperformed by the presented heuristics PR1(x)–PR4(x) both in terms of *RPI* as well as CPU time.

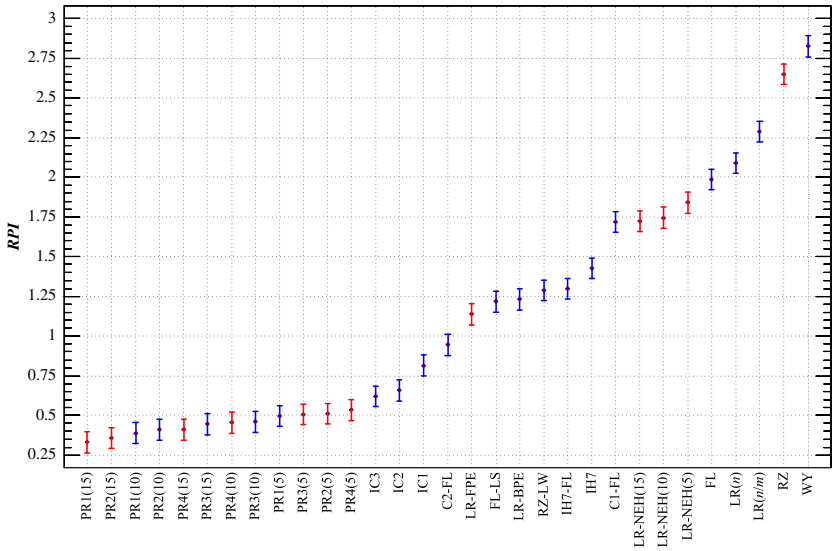


Fig. 9. Means plot for the *RPI* with Tukey's Honest Significant Difference (HSD) 99% confidence intervals for the 31 best performing heuristics.

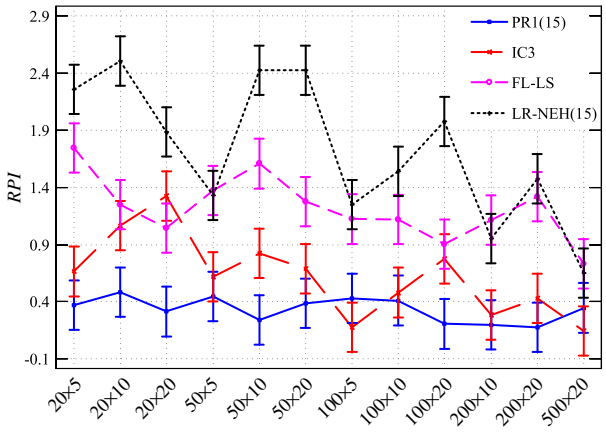


Fig. 10. Means plot for the *RPI* with Tukey's Honest Significant Difference (HSD) 99% confidence intervals for the interaction between the instance type and some chosen heuristics.

5. Conclusions

In this paper we have carried out an extensive review and comparison of the heuristics for the permutation flowshop scheduling problem with total or mean flowtime minimization criterion. A total of 22 existing heuristics have been coded and tested. With the knowledge obtained, five new methods have been presented. The well known Taillard [41] benchmark has been employed. All heuristics have been coded in the same language and have been tested on the same computing platform therefore the results are fully comparable. Furthermore, extensive use of the design of experiments (DOE) approach and analysis of variance (ANOVA) statistical technique results in sound conclusions.

Among simple heuristics, the FL-LS of Laha and Sarin [22] is the best performer. However, it is computationally costly; needing about one order of magnitude more CPU time than other composite heuristics that are, in turn, better performers. Our simple proposed LR-NEH(x) method represents a good trade-off between CPU time and quality, dominating most other existing simple heuristics from a Pareto perspective.

It is demonstrated that the heuristics IC2 and IC3 presented by Li et al. [25] were the best two performers from the literature as regards the quality of solutions and composite heuristics. However, our four presented composite heuristics PR1(x)–PR4(x) result both in lower average relative percentage deviations and lower average CPU time. For example, PR1(x) results in an average deviation of just 0.33% and average CPU time of 20.93 s whereas IC3 has almost double the deviation (0.62%) and more than three times more CPU time (77.25 s). In conclusion, our presented methods can now be considered state-of-the-art heuristics for the permutation flowshop scheduling problem with total flowtime minimization criterion.

Acknowledgments

This research is partially supported by National Science Foundation of China (60874075, 61174187), and Science Foundation of Shandong Province, China (BS2010DX005), and Postdoctoral Science Foundation of China (20100480897). Rubén Ruiz is partially funded by the Spanish Ministry of Science and Innovation, under the project "SMPA—Advanced Parallel Multiobjective Sequencing: Practical and Theorerical Advances" with reference DPI2008-03511/DPI and by the Small and Medium Industry of the Generalitat Valenciana (IMPIVA) and by the European Union through the European Regional Development Fund (FEDER) inside the R+D program "Ayudas dirigidas a Institutos Tecnológicos de la Red IMPIVA" during the year 2011, with project number IMDEEA/2011/142.

References

[1] Ahmadi RH, Bagchi U. Improved lower bounds for minimizing the sum of completion times of n jobs over m machines in a flow shop. European Journal of Operational Research 1990;44(3):331–6.
[2] Allahverdi A, Aldowaisan T. New heuristics to minimize total completion time in m-machine flowshops. International Journal of Production Economics 2002;77(1):71–83.
[3] Baker KR. Introduction to sequencing and scheduling. New York: Wiley; 1974.

[4] Campbell HG, Dudek RA, Smith ML. Heuristic algorithm for n job, m machine sequencing problem. Management Science Series B-Application 1970;16(10): B630–7.
[5] Conway RW, Maxwell WI, Miller LW. Theory of scheduling. Reading, Mass: Addison-Wesley; 1967.
[6] Dong XY, Huang HK, Chen P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Computers & Operations Research 2009;36(5):1664–9.
[7] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society 2004;55(12):1243–55.
[8] Framinan JM, Leisten R. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega—International Journal of Management Science 2003;31(4):311–7.
[9] Framinan JM, Leisten R, Rajendran C. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. International Journal of Production Research 2003;41(1):121–48.
[10] Framinan JM, Leisten R, Ruiz-Usano R. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. European Journal of Operational Research 2002;141(3):559–69.
[11] Framinan JM, Leisten R, Ruiz-Usano R. Comparison of heuristics for flowtime minimisation in permutation flowshops. Computers & Operations Research 2005;32(5):1237–54.
[12] Gonzalez T, Sahni S. Flowshop and jobshop schedules: complexity and approximation. Operations Research 1978;26(1):36–52.
[13] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5(2):287–326.
[14] Gupta JND. Heuristic algorithms for multistage flowshop scheduling problem. AIIE Transactions 1972;4(1):11–8.
[15] Gupta JND, Stafford EF. Flowshop scheduling research after five decades. European Journal of Operational Research 2006;169(3):699–711.
[16] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14):2895–929.
[17] Ho JC. Flowshop sequencing with mean flowtime objective. European Journal of Operational Research 1995;81(3):571–8.
[18] Ho JC, Chang Y-L. A new heuristic for the n-job, M-machine flow-shop problem. European Journal of Operational Research 1991;52(2):194–202.
[19] Jarboui B, Eddaly M, Siarry P. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Computers & Operations Research 2009;36(9):2638–46.
[20] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1(1):61–8.
[21] Krone MJ, Steiglitz K. Heuristic-programming solution of a flowshop-scheduling problem. Operations Research 1974;22(3):629–38.
[22] Laha D, Sarin SC. A heuristic to minimize total flow time in permutation flow shop. Omega—International Journal of Management Science 2009;37(3): 734–9.
[23] Li X, Liu L, Wu C. A fast method for heuristics in large-scale flow shop scheduling. Tsinghua Science & Technology 2006;11(1):12–8.
[24] Li X, Wang Q. Iterative heuristics for permutation flows hops with total flowtime minimization. In: Shen W, editor. Information technology for balanced manufacturing systems. IFIP TC5, Proceedings of the WG 5.5 seventh international conference on information technology for balanced automation systems in manufacturing and services. New York: Springer; 2006. p. 349–56.
[25] Li XP, Wang Q, Wu C. Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega—International Journal of Management Science 2009;37(1):155–64.
[26] Li XP, Wu C. An efficient constructive heuristic for permutation flow shops to minimize total flowtime. Chinese Journal of Electronics 2005;14(2):203–8.
[27] Liu JY, Reeves CR. Constructive and composite heuristic solutions to the $P//\Sigma C_i$ scheduling problem. European Journal of Operational Research 2001;132(2):439–52.
[28] Miyazaki S, Nishiyama N. Analysis for minimizing weighted mean flow-time in flow-shop scheduling. Journal of the Operations Research Society of Japan 1980;23(2):118–32.
[29] Miyazaki S, Nishiyama N, Hashimoto F. An adjacent pairwise approach to the mean flow-time scheduling problem. Journal of the Operations Research Society of Japan 1978;21(2):287–99.
[30] Mladenovic N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[31] Montgomery DC. Design and analysis of experiments. Hoboken, NJ: Wiley; 2008.
[32] Nawaz M, Enscore Jr EE, Ham I. A heuristic algorithm for the m machine, n job flowshop sequencing problem. Omega—International Journal of Management Science 1983;11(1):91–5.
[33] Pan QK, Tasgetiren MF, Liang YC. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 2008;55(4):795–816.
[34] Pinedo M. Scheduling: theory, algorithms, and systems. New York: Springer; 2008.
[35] Rajendran C. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. International Journal of Production Economics 1993;29(1): 65–73.
[36] Rajendran C, Chaudhuri D. A flowshop scheduling algorithm to minimize total flowtime. Journal of the Operations Research Society of Japan 1991;34(1): 28–46.
[37] Rajendran C, Chaudhuri D. An efficient heuristic approach to the scheduling of jobs in a flowshop. European Journal of Operational Research 1992;61(3): 318–25.
[38] Rajendran C, Ziegler H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research 1997;103(1):129–38.
[39] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2): 479–94.
[40] Taillard E. Some efficient heuristic methods for the flow-shop sequencing problem. European Journal of Operational Research 1990;47(1):65–74.
[41] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[42] Tang LX, Liu JY. A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. Journal of Intelligent Manufacturing 2002;13(1):61–7.
[43] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research 2007;177(3):1930–47.
[44] Varadharajan TK, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research 2005;167(3): 772–95.
[45] Wang CG, Chu CB, Proth JM. Heuristic approaches for $n/m/F/\Sigma C_i$, scheduling problems. European Journal of Operational Research 1997;96(3):636–44.
[46] Woo HS, Yim DS. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers & Operations Research 1998;25(3):175–82.
[47] Zhang Y, Li XP, Wang Q. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. European Journal of Operational Research 2009;196(3):869–76.

Minimisation de la date moyenne de fin dans un flowshop

Département Informatique
5e année
2012 - 2013

Rapport de PFE

Résumé : Ce document présente le travail du projet de fin d'étude "Minimisation de la date moyenne de fin dans un flowshop". Minimiser la date moyenne de fin permet d'éviter les problèmes d'encombrement. Le problème a été modélisé avec l'algèbre Max-Plus et cette modélisation utilise une résolution d'un problème de voyageur de commerce. Le projet utilise une procédure par séparation et évaluation.

Mots clefs : PSE, date de fin moyenne, ordonnancement, flowshop, PVC

Abstract: This paper presents the work of the finalstudy projet "Minimizing flowtime in a flowshop." Minimize flowtime avoids congestion problems. The problem was modeled with the Max-plus algebra and this model uses a traveling salesman problem solver. The project uses a branch & bound procedure.

Keywords: Branch & Bound, flowtime, scheduling, flowshop, TSP

Encadrants
Christophe Lenté
christophe.lente@univ-tours.fr
Nhat Vinh Vo
nhat.vo@univ-tours.fr

Étudiant
Pauline Fouillet
pauline.fouillet@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours