Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible

Abir Ben Hmida

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Institut National des Sciences Appliquées de Toulouse*
Discipline ou spécialité : *Systèmes Informatiques*

Présentée et soutenue par *Abir BEN HMIDA SAKLY*
Le *12/12/2009*

Titre : *Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible*

JURY

Mohamed MOALLA	*Professeur*
Taïcir LOUKIL	*Professeur*
Emmanuel NERON	*Professeur*
Mohamed HAOUARI	*Professeur*
Pierre LOPEZ	*Chargé de recherche*
Marie José HUGUET	*Maître de conférences*

Ecole doctorale : *Ecole Doctorale Systèmes (EDSys)*
Unité de recherche : *LAAS de Toulouse*
Directeur(s) de Thèse : *Pierre LOPEZ/Marie-José HUGUET/Mohamed HAOUARI*
Rapporteurs : *Taïcir LOUKIL /Emmanuel NERON*

Avant-propos

Le présent travail a été réalisé à l'unité de recherche ROI (Recherche Opérationnelle pour l'Industrie) à l'Ecole Polytechnique de Tunisie en collaboration avec le groupe Modélisation, Optimisation et Gestion Intégrée de Systèmes d'Activités (MOGISA) du Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du C.N.R.S. à Toulouse.

Ce travail s'inscrit dans le cadre d'une thèse en cotutelle entre la Faculté des Sciences de Tunis (F.S.T.) spécialité Informatique : Productique et Temps Réel, et l'Institut National des Sciences Appliquées de Toulouse (I.N.S.A.T.) spécialité Systèmes Informatiques.

L'objectif que nous avons visé est la conception et le développement des méthodes arborescentes à base de divergences pour la résolution de problèmes d'ordonnancement avec flexibilité des ressources, tout en expliquant les différentes démarches et orientations suivies ainsi que la méthodologie adoptée.

De plus, une étude statistique ainsi qu'une comparaison aux méthodes existantes a largement contribué à la discussion des différents résultats obtenus.

Remerciements

Je tiens à remercier vivement Monsieur **Mohamed HAOUARI**, Professeur à l'Institut National des Sciences Appliquées de Tunis (I.N.S.A.T.) pour m'avoir accueillie au sein de l'unité de recherche ROI (Recherche Opérationnelle pour l'Industrie) et pour toute l'aide et la disponibilité qu'il m'a offert afin de mener ce travail en terme.

Je voudrais remercier, également, Monsieur **Pierre LOPEZ**, Chargé de Recherche C.N.R.S au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS). Ses conseils, ses critiques, sa compétence et ses encouragements m'ont permis de mener à bien cette thèse.

J'exprime, également, toute ma gratitude à Madame **Marie-José HUGUET**, Maître de Conférences à l'Institut National des Sciences Appliquées de Toulouse (I.N.S.A.T.). Ses idées fécondes et brillantes ont été un élément essentiel à l'aboutissement de cette étude.

Je remercie très sincèrement Monsieur **Mohamed MOALLA**, Professeur à la Faculté des Sciences de Tunis, pour avoir accepté être le président de jury de ma soutenance.

J'adresse tous mes remerciements à Madame **Taïcir LOUKIL**, Professeur à l'Institut Supérieur de Gestion Industrielle de Sfax (I.S.G.I.S), et Monsieur **Emmanuel NERON**, Professeur à l'Ecole d'ingénieurs Polytechnique de l'Université de Tours, pour m'avoir fait l'honneur d'accepter d'être rapporteurs de ma thèse.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Le problème d'ordonnancement est l'un des problèmes fréquemment rencontrés dans la gestion des systèmes de production. Il apparaît dans des domaines aussi variés que nombreux ; citons le cas de la planification de grands projets, l'organisation d'activités de services, la conception des emplois de temps, l'organisation des différentes tâches et l'allocation des ressources dans les systèmes informatiques, etc.

Les problèmes d'ordonnancement étant très variés, ils sont classés en plusieurs familles. Si l'on dispose d'une seule machine pour exécuter tous les jobs, on parle de problème à une machine. À l'inverse, on peut disposer de plusieurs machines. Ce sont les problèmes à machines parallèles. Lorsqu'une tâche est composée de plusieurs opérations nécessitant des ressources différentes, on parle de problèmes d'atelier. On peut citer trois grands problèmes d'atelier. Lorsque l'ordre des ressources à visiter est fixé et est le même pour tous les jobs, on est face à un problème de type flow shop. Si l'ordre est fixé mais varie d'un job à l'autre, c'est un problème de job shop. Si cet ordre peut être quelconque, on parle d'un open shop.

Par ailleurs, plusieurs types de contraintes portent sur la nature des jobs à exécuter. S'il est possible d'interrompre l'exécution d'un job et de la reprendre à une date ultérieure, on dit que le problème est préemptif. Des contraintes de précédence peuvent également régir l'ordre d'exécution des jobs. Les jobs peuvent avoir une date de disponibilité avant laquelle leur exécution ne peut commencer. On peut également citer les problèmes particuliers où les jobs sont de durées unitaires, ceux où ils sont de durées égales et le cas général où les durées sont différentes. La définition des ressources et des jobs permet de fixer les contraintes. Ces dernières ne suffisent pas à définir les problèmes d'optimisation afférents : il faut se donner une fonction objectif définie par un critère que l'on va chercher à optimiser. Le problème consiste alors à trouver une solution admissible qui optimise le critère. Le critère le plus classique en ordonnancement est sans doute la minimisation de la date de fin de projet. Il existe également de nombreux critères qui s'expriment en fonction d'une somme : la somme des dates de fin, la somme des retards, le nombre de jobs en retard. Ces critères existent aussi en version pondérée, on affecte alors un poids à chaque job.

Au cours des dernières années, les systèmes de production présentant une flexibilité sur les ressources ont largement attiré l'attention des chercheurs dans le domaine de la recherche opérationnelle. L'intérêt porté à ces systèmes est largement motivé par le caractère de la flexibilité. Ainsi, les caractéristiques des systèmes peuvent être rapidement et facilement modifiées de manière à optimiser les rendements en termes de coûts et de délais de la production.

Ce travail concerne le domaine académique des problèmes d'ordonnancement et plus particulièrement les problèmes de nature flexible. La caractéristique essentielle de ces systèmes est leur complexité qui rend difficile la modélisation et l'optimisation par des approches exactes. Il s'agit de développer des algorithmes basés sur la notion de divergence pour la résolution efficace de ces problèmes. Il s'agit également du couplage de différentes approches existantes pour l'amélioration de performances du système. Cette combinaison de méthodes de résolution se révèle un moyen efficace de pallier la difficulté du problème énoncé et d'apporter des solutions efficaces.

Contributions de la thèse

Les travaux de recherche présentés dans ce mémoire portent sur la résolution de problèmes d'ordonnancement d'atelier avec flexibilité des ressources en utilisant des méthodes arborescentes basées sur la recherche à divergences limitées.

Notre contribution porte sur plusieurs points :

- le développement d'une nouvelle méthode à base de divergences. Cette nouvelle méthode combine à la fois des techniques existant dans d'autres méthodes à divergences et des nouvelles propositions visant à améliorer l'exploration de l'arborescence de recherche de solutions ;
- l'adaptation de cette méthode aux problèmes considérés (flow shop hybride et job shop flexible). Les adaptations proposées tirent parti des travaux de la littérature sur ces problèmes en exploitant des résultats comme des bornes ou des voisinages ;
- l'évaluation de nos méthodes à travers des jeux-tests de la littérature. Ces évaluations nous permettent de nous comparer à différentes approches existantes pour la résolution des problèmes qui nous concernent, et de mettre en évidence les performances de nos méthodes.

Organisation du rapport

Ce mémoire est organisé en cinq chapitres. Les trois premiers concernent un état de l'art des domaines que nous abordons et les deux derniers traitent de nos travaux pour la résolution de problèmes d'ordonnancement avec flexibilité de ressources.

Dans le premier chapitre, nous donnons un bref aperçu du domaine vaste de l'ordonnancement.

Dans le deuxième chapitre, nous effectuons un état de l'art sur les problèmes d'ordonnancement à flexibilité de ressources considérés, à savoir le flow shop hybride et le job shop flexible. Plusieurs méthodes de résolution de ces problèmes sont exposées.

Dans le troisième chapitre, nous présentons différentes méthodes arborescentes à base de divergences en nous focalisant sur certaines d'entre elles qui serviront de base au développement d'une nouvelle méthode à base de divergences.

Les deux chapitres suivants abordent la résolution de problèmes d'ordonnancement avec flexibilité de ressources : le flow shop hybride à plusieurs étages, à deux étages et le job shop flexible.

Dans le quatrième chapitre, nous présentons tout d'abord une méthode arborescente générale basée sur la notion de divergence. Cette méthode est ensuite adaptée aux problèmes de type flow shop hybride à plusieurs étages ainsi qu'au cas particulier du flow shop hybride à deux étages. Puis, nous évaluons les performances de cette méthode sur des jeux-tests issus de la littérature.

Le cinquième chapitre aborde la résolution de problèmes d'ordonnancement de type job shop flexible. Cette partie se décompose plus précisément en deux parties. Nous présentons tout d'abord différentes adaptations de la méthode à base de divergences proposée dans le cadre du Flow Shop Hybride. Puis, nous en évaluons ses performances à travers différents jeux-tests de la littérature.

Finalement, nous présentons la conclusion en synthétisant les apports de nos travaux et en formulant quelques propositions pour les évolutions futures des méthodes que nous avons développées.

Chapitre 1. Présentation des problèmes d'ordonnancement

Dans ce chapitre, nous présentons, dans un premier temps, les problèmes classiques d'ordonnancement des systèmes de production, ainsi que les problèmes particuliers présentant la caractéristique de flexibilité sur lesquels porte notre étude. Dans un second temps, nous présentons des notions de base sur la théorie de la complexité, ainsi que la complexité de certains problèmes d'ordonnancement.

1.1. Définition

Un problème d'ordonnancement se pose lorsqu'il s'agit d'organiser dans le temps, l'exécution de diverses tâches soumises à des contraintes et auxquelles sont attribuées des ressources, de manière à satisfaire un ou plusieurs objectifs donnés [Carlier et Chrétienne, 1988 ; Esquirol et Lopez, 1999].

Une solution au problème d'ordonnancement décrit l'exécution des tâches et l'allocation des ressources au cours du temps : il faut décider, à la fois, quelles ressources réalisent les tâches et quand.

Très fréquemment en ordonnancement de production, les problèmes à résoudre consistent à ne déterminer que le positionnement des tâches dans le temps, le problème d'affectation de ces tâches aux ressources étant résolu en amont. On parle parfois de problème d'ordonnancement « pur ». Ce positionnement peut être relatif (on détermine le séquencement des tâches sur les ressources) ou absolu (on fixe les dates de réalisation des tâches). Bien qu'en pratique, il semble difficile de faire abstraction de l'allocation de ressources, l'étude de ce cas a toutefois permis le développement de plusieurs méthodes génériques qui ont été utilisées dans l'étude de problèmes d'ordonnancement avec flexibilité des ressources.

Dans ce mémoire, nous parlons de problème d'ordonnancement *flexible* lorsqu'il s'agit de déterminer les dates de début et/ou de fin des tâches ainsi que les ressources permettant leur exécution.

Les problèmes d'ordonnancement se rencontrent dans divers domaines. Citons par exemple : les systèmes informatiques, où les tâches représentent les programmes et les ressources sont les processeurs ou la mémoire, la gestion de production, la conception des emplois des temps, etc.

Les caractéristiques des problèmes d'ordonnancement peuvent dépendre du contexte ; toutefois, on peut dégager des éléments de base qui sont les tâches, les ressources, les contraintes et les objectifs à réaliser ainsi que les critères d'évaluation, définis tour à tour ci-après.

1.2. Concepts de base

La présentation de ces concepts est issue de [Pinedo, 1995 ; Esquirol et Lopez, 1999].

1.2.1. Les tâches

Une tâche *Tâche(i)* est, par définition, une entité élémentaire localisée dans le temps par une date de début (S_i) et/ou de fin (C_i), et par une durée d'exécution (p_i) ; par ailleurs, les tâches nécessitent l'utilisation de ressources (voir Fig.1).

Les tâches sont, le plus souvent, liées entre elles par des conditions d'origines diverses ; on dit alors que les tâches sont *interdépendantes*. Dans le cas contraire, elles sont dites *indépendantes*.

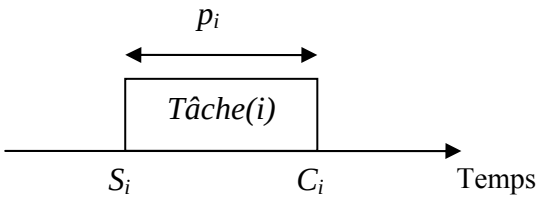


Fig.1. Une tâche

Dans certains problèmes, les tâches peuvent être *interruptibles* ce qui signifie, en pratique, qu'elles peuvent être exécutées par morceaux par une ou plusieurs ressources ; dans d'autres cas contraires, on ne peut interrompre une tâche une fois celle-ci commencée. On parle respectivement de problèmes *préemptif* et *non préemptif*.

1.2.2. Les ressources

Une ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'au moins une tâche et disponible en quantité limitée appelée capacité de la ressource et notée A_k.

On distingue différents types de ressources. Elles peuvent être classées selon leurs disponibilités au cours du temps et on parle dans ce cas de ressources renouvelables et consommables, ou selon leurs capacités et on parle dans ce cas de ressources disjonctives (non partageables) et cumulatives (partageables).

Ressource renouvelable : il s'agit d'une ressource pouvant redevenir disponible en même quantité après avoir été allouée à une tâche (les machines, les hommes, l'équipement,…).

Ressource consommable : il s'agit d'une ressource dont la disponibilité décroît après avoir été allouée à une tâche, par exemple la matière première.

Ressource disjonctive : il s'agit d'une ressource qui ne peut exécuter qu'une seule tâche à la fois.

Ressource cumulative : il s'agit d'une ressource qui peut être utilisée simultanément par plusieurs tâches.

Lorsque plusieurs ressources sont nécessaires simultanément pour assurer la réalisation d'une tâche, on parle de problèmes d'ordonnancement *multi-ressources*. En revanche, un problème d'ordonnancement est dit *mono-ressource* lorsqu'une seule ressource est nécessaire et suffisante pour la réalisation de chaque tâche.

1.2.3. Variables de décision et contraintes

Les variables diffèrent selon qu'elles portent sur des décisions liées au temps ou aux ressources. On parle respectivement de *variables temporelles* et de *variables d'affectation*.

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision. On distingue deux types de contraintes en ordonnancement, à savoir les contraintes temporelles et les contraintes de ressources. Le tableau 1 présente différentes contraintes de temps et de ressources habituellement rencontrées en ordonnancement.

Tableau 1. *Variables de décision et contraintes*

Contraintes	**type**	**Illustration**
Contraintes temporelles	**Contrainte de temps absolu**	Cette contrainte permet de représenter la limitation des valeurs possibles pour les dates des tâches. Par exemple : date de début au plus tôt (notée r_i), date de fin au plus tard (notée d_i).
	Contrainte de temps relatif	Cette contrainte est relative aux contraintes de cohérence technologique telle que les contraintes de gammes dans lesquelles il faut respecter le positionnement relatif entre tâches. Exemple : $Tâche(i) \prec Tâche(j)$, relation de précédence entre tâches qui impose que *Tâche(i)* soit avant *Tâche(j)*.
Contraintes de ressources	**Contraintes de capacité**	**Contrainte (de ressource) disjonctive :** Cette contrainte impose la réalisation disjointe de deux tâches *Tâche(i)* et *Tâche(j)*, càd $(Tâche(i) \prec Tâche(j)) \vee (Tâche(j) \prec Tâche(i))$.

		Contrainte (de ressource) cumulative : Cette contrainte interdit, pour une ressource donnée, la réalisation simultanée d'un certain nombre de tâches utilisant une quantité de la ressource excédant sa capacité.
	Contraintes d'affectation (cas des problèmes d'ordonnancement flexible)	**Contrainte de domaine :** Cette contrainte représente l'ensemble des ressources candidates pour l'exécution d'une tâche.
		Contrainte de différence : Cette contrainte impose l'utilisation de ressources différentes pour la réalisation d'un certain nombre de tâches.

1.2.4. Les objectifs de l'ordonnancement

L'ordonnancement de tâches tenant compte des ressources disponibles et respectant les divers types de contraintes définis ci-avant n'est en général pas unique. L'objectif de l'ordonnancement vise donc à classer les solutions possibles, voire à guider la méthode d'élaboration de l'ordonnancement utilisée. Il est alors possible de définir un ordonnancement « optimal », c'est-à-dire satisfaisant au mieux l'objectif considéré.

Des objectifs élémentaires variés, qu'il serait fastidieux d'essayer de tous lister ici, peuvent être assignés à un ordonnancement. Il est néanmoins important de préciser quelques points.

Dans certains cas, on cherche à optimiser plusieurs fonctions objectifs à la fois. Dans ce cas, le problème d'ordonnancement n'est plus un problème d'optimisation simple mais devient un problème multicritère. Autrefois souvent abordé de manière empirique par des pondérations de critères, ce problème est maintenant souvent abordé au moyen de la notion de dominance de Pareto (une solution est dominante au sens de Pareto si aucune autre solution n'améliore la satisfaction d'un critère sans dégrader la satisfaction d'un autre). Il est ainsi possible de définir des ensembles de solutions candidates (front de Pareto) parmi lesquelles le décideur pourra choisir.

Une difficulté supplémentaire est que des objectifs apparemment différents peuvent être liés (le lien entre diminution des encours et diminution des temps de cycle en est un exemple flagrant), tandis que d'autres peuvent être antagonistes (diminuer les encours et augmenter le taux d'occupation des ressources est en général difficile).

Plusieurs fonctions objectifs sont considérées dans la littérature (voir Fig.2). On note par exemple :

- Le temps total d'exécution (ou *makespan*) ou le temps moyen d'achèvement d'un ensemble de tâches ; il est noté $C_{\max} = \max_{i=1..n} C_i$.

- La somme des dates de fin des travaux, notée $\sum C = \sum_{i=1...n} C_i$; la date de fin moyenne des travaux, notée $\overline{C} = \frac{1}{n} \sum_{i=1,...,n} C_i$ ou la date de fin pondérée des travaux, notée $\overline{C^w}$.

- Le retard absolu par rapport aux dates limites fixées noté T_i=max(0, C_i-d_i), le plus grand retard noté $T_{\max}$, la moyenne des retards notée $\overline{T}$ ou la moyenne pondérée des retards notée $\overline{T^w}$.

- Le retard algébrique L_i=(C_i-d_j), le plus grand retard algébrique noté $L_{\max}$, la moyenne des retards algébriques notée $\overline{L}$, ou la moyenne pondérée $\overline{L^w}$.

- La pénalité unitaire de retard, notée U_i (U_i=0 si $C_i \leq d_i$, U_i=1 sinon), la moyenne notée $\overline{U}$, ou la moyenne pondérée notée $\overline{U^w}$.

- Les encours de fabrication ; le plus grand encours noté $F_{\max}$, la moyenne notée $\overline{F}$, ou la moyenne pondérée notée $\overline{F^w}$.

- L'avance par rapport aux dates limites fixées ; la plus grande avance noté $E_{\max}$, la moyenne notée $\overline{E}$, ou la moyenne pondérée notée $\overline{E^w}$.

Il est parfois difficile de traduire l'ensemble des objectifs de résolution par un ou plusieurs critères numériques. On peut, dans ce cas, avoir recours à une approche par satisfaction de contraintes. L'ensemble des contraintes regroupe alors, à la fois des contraintes intrinsèques du problème (cohérence technologique par exemple) et des objectifs de type seuil à atteindre ou à ne pas dépasser. On pourra dans ce cas se contenter d'une solution quelconque pourvue qu'elle soit admissible. La stratégie d'exploration de l'espace des solutions admissibles doit exploiter intelligemment les contraintes pour élaguer cet espace de recherche [Dechter, 2003]. C'est le principe de *la propagation de contraintes*, qui désigne un ensemble de techniques de filtrage (retrait de valeurs inconsistantes).

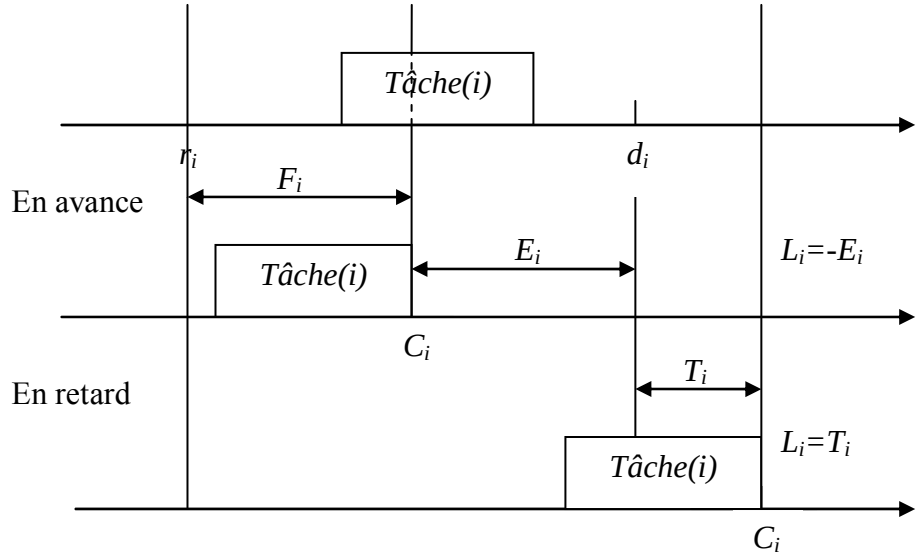


Fig.2. Quelques critères d'évaluation

1.3. Ordonnancement d'atelier

Dans les problèmes d'ordonnancement d'atelier, les ressources considérées sont renouvelables et ne peuvent réaliser qu'une seule tâche à la fois. Les tâches, quant à elles, sont le plus souvent non interruptibles. Ces problèmes peuvent être mono-ressource, une seule ressource est nécessaire à la réalisation de chaque tâche, ou multi-ressources lorsque plusieurs ressources sont nécessaires simultanément pour effectuer une tâche.

Dans le cas particulier de problèmes d'ordonnancement d'atelier, on parle indifféremment de tâche ou d'opération, de ressources ou de machines. Une opération, nécessite une machine pour sa réalisation. Les opérations sont relatives à la fabrication d'un produit et doivent être exécutées en séquence : cela constitue un travail classiquement appelé "**job**".

Une manière usuelle de classer les problèmes d'ordonnancement d'atelier est de positionner ces derniers par rapport aux caractéristiques des ressources engagées. On distingue, alors, des problèmes d'atelier sans ou avec flexibilité de ressources.

1.3.1. Problèmes d'atelier sans flexibilité des ressources

Les problèmes d'ordonnancement sans flexibilité des ressources s'intéressent à ordonnancer des tâches interdépendantes. Dans ce genre de problème, seules les contraintes temporelles et les contraintes de capacité des ressources sont considérées : cela revient à supposer que le problème d'affectation est résolu.

Une solution au problème d'ordonnancement sans flexibilité des ressources consiste à associer à chaque tâche une date de début et/ou de fin d'exécution qui respecte les contraintes temporelles et les contraintes de limitation de capacité des ressources du problème.

Ce problème peut être divisé suivant les caractéristiques décrites ci-après en : flow shop, job shop et open shop.

Ordonnancement d'ateliers en flow shop

En flow shop, tous les jobs visitent les machines dans le même ordre, avec des durées opératoires pouvant être différentes. Les machines sont disponibles sur tout l'horizon de planification et elles ne peuvent exécuter qu'une seule opération à la fois. Par ailleurs, une opération ne peut s'exécuter que sur une seule machine à la fois.

En pratique, ce cas est fréquemment rencontré ; citons l'exemple d'une chaîne de fabrication ou de montage.

Ordonnancement d'ateliers en job shop

Dans un job shop, chaque travail passe sur les machines dans un ordre fixé, mais, à la différence du flow shop, cet ordre peut être différent pour chaque travail. Chaque job possède donc une gamme spécifique.

Il s'agit ici de déterminer les dates de passage sur différentes ressources des jobs ayant des gammes différentes dans l'atelier. Ces jobs partageant des ressources communes, des conflits sont susceptibles de survenir, résultant des croisements des flux. Dans son expression la plus simple, le problème est donc de gérer ces conflits tout en respectant les contraintes données, et en optimisant les objectifs poursuivis.

Ordonnancement d'ateliers en open shop

C'est un problème sans contrainte d'enchaînement prédéfinie, où chaque produit à fabriquer doit subir diverses opérations sur des machines, mais dans un ordre totalement libre. Cela est rarement généralisé dans un atelier manufacturier, mais peut survenir ponctuellement.

La résolution de ce sous-problème supplémentaire (déterminer la séquence d'opérations de chaque job) complique la production d'un ordonnancement, mais offre des degrés de liberté intéressants.

Il est à noter que la notion de « gamme linéaire », très ancrée au niveau des bureaux des méthodes des entreprises, mais aussi la rigidité des outils de gestion des données techniques, a souvent fait que, lorsque des permutations entre opérations étaient possibles, un choix plus ou moins arbitraire était réalisé lors de l'élaboration de la gamme.

1.3.2. Problèmes d'atelier avec flexibilité des ressources

La nécessité d'être toujours plus réactives a conduit les entreprises à rechercher des nouveaux degrés de liberté. Une partie de ces degrés de

liberté réside dans l'élaboration d'alternatives, comme la duplication des exemplaires des machines, pour effectuer une ou plusieurs opérations.

Ces problèmes présentent alors une difficulté supplémentaire par rapport aux problèmes sans flexibilité des ressources dans la mesure où les ressources sont en exemplaires multiples. Ainsi, la machine nécessaire pour exécuter une opération n'est pas connue d'avance, mais doit être sélectionnée parmi un ensemble donné. Ces machines peuvent être de trois types à savoir, des machines identiques notées *P*, des machines uniformes, notées *Q*, où les durées opératoires dépendent de leurs vitesses, et des machines non-reliées (ou indépendantes) notées *R* où leurs vitesses varient d'une tâche à une autre.

Dans ce genre de problème, il faut considérer simultanément les contraintes temporelles, les contraintes de limitation de capacité des ressources et les contraintes d'affectation.

Les problèmes avec flexibilité de ressources les plus étudiés dans la littérature sont : les problèmes à machines parallèles, le flow shop hybride et le job shop flexible présentés ci-dessous.

Ordonnancement d'ateliers en machines parallèles

Dans ce cas, on dispose d'un ensemble de machines pour réaliser les travaux. Les travaux se composent d'une seule opération et un travail exige une seule machine. L'ordonnancement s'effectue en deux phases : la première phase consiste à affecter les travaux aux machines et la deuxième phase consiste à établir la séquence de réalisation sur chaque machine.

Ordonnancement d'ateliers en flow shop hybride

Le "flow shop hybride" à *k* étages est un flow shop pour lequel chaque travail doit subir *k* opérations différentes telles que chaque opération doit se faire sur une unique machine choisie parmi un ensemble, appelé *étage*, de machines parallèles dédiées à cette opération. Une machine ne peut appartenir qu'à un seul étage.

Ordonnancement d'ateliers en job shop flexible

Un job shop flexible est un problème à cheminements multiples, mais qui présente une difficulté supplémentaire dans la mesure où chaque opération nécessite une machine pour être réalisée et cette machine doit être choisie dans un ensemble défini a priori.

1.4. Complexité des problèmes d'ordonnancement

La théorie de la complexité s'intéresse à l'étude formelle de la difficulté théorique intrinsèque des problèmes en informatique ou d'un problème par rapport à un autre et de l'analyse de la complexité des programmes et des algorithmes. Concrètement, on cherche à savoir si le problème étudié est plutôt « facile » ou plutôt « difficile » à résoudre en se basant sur une

estimation (théorique) des temps de calcul et des besoins en mémoire informatique.

Les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire. Pour résoudre un problème d'ordonnancement, nous devons toujours chercher à établir sa complexité, car cela détermine la nature de l'algorithme à mettre en œuvre. Si le problème étudié appartient à la classe P, nous savons d'avance qu'un algorithme polynomial existe pour le résoudre. Dans le cas contraire, si le problème est NP-difficile, deux approches sont possibles. La première est de proposer un algorithme approché, donc une heuristique, qui calcule en temps polynomial une solution s'approchant au mieux de la solution optimale. Une alternative est de proposer un algorithme qui calcule la solution optimale du problème, mais pour lequel la complexité dans le pire des cas est exponentielle. Dans ce cas, le défi est de concevoir un algorithme qui peut résoudre en temps acceptable les problèmes de la plus grande taille possible.

Les problèmes d'ordonnancement sont spécifiés selon une notation à trois champs α | β | γ [Graham *et al.*, 1979] où α précise l'environnement machines ou le système à ordonnancer, β précise les caractéristiques des opérations ou les contraintes et γ précise le(s) critère(s) à optimiser.

Pour calculer la complexité des problèmes d'ordonnancement, plusieurs résultats traditionnels existent dans la littérature. Ainsi, on rappelle la complexité des problèmes de base de l'ordonnancement de type $\alpha \| Z$ avec Z un critère donné.

En considérant les problèmes à une seule machine, la minimisation de la fonction $f_{\max} = \max_{i=1,\ldots,n} (f_i(C_i))$ (avec f_i une fonction croissante de la date de fin d'exécution) est un problème qui se résout polynomialement. En conséquence, les problèmes de type $1 \| Z$ avec $Z \in \{C_{\max}, F_{\max}, T_{\max}, L_{\max}\}$ le sont également. Notons que toute séquence est optimale pour le problème $1 \| C_{\max}$. Les problèmes de type $1 \| \overline{C^w}$ et $1 \| \overline{C}$ peuvent être résolus polynomialement [Lawler *et al.*, 1989] et de même pour le problème $1 | d_i | \overline{U}$ [Pinedo, 1995]. Seuls les problèmes $1 | d_i | \overline{T}$ et $1 | d_i | \overline{U^w}$ sont NP-difficiles au sens faible et $1 | d_i | \overline{T^w}$ au sens fort [Lawler *et al.*, 1989]. En conséquence, tous les problèmes de type $\alpha | d_i | \overline{T^w}$ sont NP-difficiles au sens fort, quelle que soit la valeur de α.

Pour les problèmes à machines parallèles, le problème $P\|f_{\max}$ est NP-difficile au sens fort [Lawler *et al.*, 1989]. De même pour les problèmes $P\|Z$ avec $Z \in \{C_{\max}, F_{\max}, T_{\max}, L_{\max}\}$ [Brucker, 2004].

On note aussi que, pour résoudre le problème $Rm\|C_{\max}$ (le nombre de machines *m* est fixé), il existe un algorithme de programmation dynamique s'exécutant en temps pseudo-polynomial [Lawler *et al.*, 1989].

La minimisation du critère $\overline{C}$ est un problème polynomial quel que soit le type de machines considérées [Pinedo, 1995]. D'autre part, le problème $P\|\overline{C^w}$ est NP-difficile au sens fort [Brucker, 2004].

Sachant que les problèmes $\alpha|d_i|L_{\max}$ avec $\alpha \in \{P, Q, R\}$ sont NP-difficiles au sens fort, on déduit que les mêmes problèmes avec les critères $\overline{U}, \overline{U^w}$ et $\overline{T}$ sont aussi NP-difficiles. On note aussi qu'il existe un algorithme pseudo-polynomial pour la résolution du problème $Rm|d_i|\overline{U^w}$ [Lawler *et al.*, 1989 ; Pinedo, 1995].

D'un point de vue théorique, les problèmes de type flow shop, à quelques cas particulier près, sont NP-difficiles. Pour le cas particulier à deux machines $F2|prmu|C_{\max}$, une solution optimale peut être donnée en un temps polynomial en utilisant la règle de Johnson [Johnson, 1954].

Un autre cas particulier noté $F2|nwt|C_{\max}$ se présente lorsque toutes les opérations du même job doivent passer sans attente (*nwt* = « no-wait »). Ce problème est résolu polynomialement par l'algorithme proposé dans [Gilmore et Gomory, 1964], qui transforme le problème en un problème de voyageur de commerce avec des distances particulières. Ce dernier problème est équivalent à $Fm|block|C_{\max}$ pour lequel la capacité de stock entre les machines est nulle.

Les problèmes de type job shop sont aussi, à quelques cas particulier près, NP-difficiles. Dans [Jackson, 1956], l'auteur a proposé une généralisation des règles de Johnson pour la résolution du cas particulier avec deux machines $J2\|C_{\max}$ par un algorithme polynomial.

Le problème à deux jobs $J|n=2|f$ peut lui aussi être résolu optimalement en temps polynomial. Les résultats de complexité sur les problèmes à deux jobs sont dus aux travaux de [Satskov, 1985 ; Brucker, 1988]. Ces derniers ont proposé une approche géométrique qui permet de trouver la solution optimale pour le problème du job shop à deux jobs avec n'importe quel critère régulier.

Les premiers travaux s'intéressant aux problèmes d'ordonnancement de type flow shop hybride (ou général) réalisés par [Gupta, 1988] s'intéressant à des ateliers composés de deux étages ont montré que la minimisation de la plus grande date de fin est NP-difficile au sens fort même dans le cas de deux étages, dès lors qu'un étage comporte plus d'une machine.

Les problèmes de type job shop flexible (ou général), sont aussi des problèmes NP-difficiles au sens fort, sauf cas particulier. Par exemple, le problème $JMPM\ |n=2|f$ avec n'importe quel critère objectif régulier a été résolu optimalement avec une complexité polynomiale [Brucker *et al.*, 1990]. Avec *MPM* pour « multi-purpose machine » ; Ce sont des machines parallèles partiellement multi-fonctions. Ces machines sont capables de traiter divers types de produit qui sont au nombre de *n*, cependant le traitement de chaque produit demande une mise au point spécifique de la machine et une machine ne peut traiter qu'un ensemble particulier de produits.

L'algorithme de résolution du problème $JMPM\,|n=2, prec; r_i|L_{\max}$ présente aussi une complexité polynomiale [Brucker *et al.*, 1993]. Le problème de job shop flexible devient NP-difficile à partir de deux machines et trois jobs : $JMPM|n=3|C_{\max}$.

En se basant sur les résultats présentés dans [Tanaev *et al.*, 1994], on note que les problèmes de type flow shop, job shop et open shop sont NP-difficiles au sens fort, sauf cas particulier.

En conséquence, tous les problèmes de type flow shop hybride et job shop flexible sont aussi NP-difficiles au sens fort (voir tableau 2).

Tableau 2. *Classe de complexité des problèmes d'ordonnancement (mono-critère)*

Critères / problèmes		C_{max}	F_{max}	T_{max}	L_{max}	f_{max}	$\overline{C}$	$\overline{C^w}$	$\overline{T}$	$\overline{T^w}$	$\overline{U}$	$\overline{U^w}$
Une seule machine		*P*	*P*	*P*	*P*	*P*	*P*	*P*	*NP**	*NP*	*P*	*NP**
Machines parallèles	**(P)**	*NP*	*NP*	*NP*	*NP*	*NP*	*P*	*NP*	*NP*	*NP*	*NP*	*NP*

	(Q)	*NP*	*NP*	*NP*	*NP*	*NP*	*P*	*NP*	*NP*	*NP*	*NP*	*NP*
	(R)	*NP*	*NP*	*NP*	*NP*	*NP*	*P*	*NP*	*NP*	*NP*	*NP*	*NP*
Flow shop		*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*
Flow shop hybride		*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*
Job shop		*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*
Job shop flexible		*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*
Open shop		*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*	*NP*

Notations : P : *Classe P*

NP^* : *Classe* $NP-difficile$ *au sens faible*

NP : *Classe* $NP-difficile$ *au sens fort.*

Les problèmes abordés dans cette thèse, à savoir le job shop flexible et le flow shop hybride sont NP-difficiles au sens fort.

1.5. Conclusion

Dans ce chapitre, nous avons tout d'abord présenté les caractéristiques générales des problèmes d'ordonnancement. Puis, nous nous sommes intéressé plus spécifiquement à des problèmes d'ordonnancement ayant de la flexibilité dans le choix des ressources pour la réalisation des tâches. Ce sont ces problèmes avec flexibilité de ressources qui sont concernés par notre étude.

Dans un second temps, nous avons fait un survol de la complexité de différents problèmes d'ordonnancement. En particulier, les problèmes que nous étudions par la suite sont connus pour être NP-difficiles.

Le chapitre suivant dresse un état de l'art de différentes méthodes utilisées pour la résolution des problèmes d'ordonnancement flexible abordés dans cette thèse.

Chapitre 2. Etat de l'art des problèmes d'ordonnancement d'atelier avec flexibilité de ressources

Dans ce chapitre, nous dressons un état de l'art non exhaustif des problèmes d'ordonnancement de type flow shop hybride et job shop flexible mono-/multi-critères. Seuls les problèmes d'ordonnancement flexible *statiques* sont pris en compte. On ne considère ni pannes, ni opérations de maintenance sur les ressources. Notre étude se focalise sur les problèmes d'ordonnancement de production mono-ressource où chaque opération nécessite une seule ressource pour sa réalisation et pour lesquels les ressources sont renouvelables.

2.1. Définition

Au cours des dernières années, les systèmes de production présentant une flexibilité sur les ressources ont largement attiré l'attention des chercheurs dans le domaine de la recherche opérationnelle. L'intérêt porté à ces systèmes est largement motivé par ce caractère de flexibilité donnant plus de degrés de liberté aux systèmes de production et s'approchant de plus en plus des problèmes d'ateliers réels.

2.2. Problèmes de flow shop hybride

2.2.1. Cas du flow shop hybride à deux étages

Le problème de flow shop hybride à deux étages est un cas particulier du problème de flow shop hybride. Il peut être décrit comme suit. Soient un ensemble $J = \{1,2,...,n\}$ de n jobs, un ensemble $M = \{M_1, M_2, ..., M_m\}$ de m machines, deux étages E_1 et E_2 dans lesquels chaque étage *i* contient m_i $(i = 1,2)$ machines parallèles $\{M_{i,1}, M_{i,2}, ..., M_{i,m_i}\}$ (voir Fig.3). Chaque job $j \in J$ doit passer sur une machine du premier étage, sans interruption, durant une durée $p_{1,j}$. Ensuite, le job j passe sur une machine du deuxième étage, sans interruption, pendant une durée $p_{2,j}$. Le but est de trouver un ordonnancement admissible minimisant la date de fin d'exécution totale (makespan) de l'ensemble des opérations.

En se basant sur la notation à trois champs proposée dans [Hoogeveen *et al.*, 1996], le problème est noté *F2(P)* | | C_{max}. Comme cela a été vu au chapitre précédent, ce problème est NP-difficile au sens fort dans le cas où max(m_1, m_2)>1 [Gupta, 1988]. Notons que dans les travaux dont nous avons connaissance, aucun auteur ne traite du problème de Flow Shop Hybride à étages avec machines quelconques *F2(R)* | | C_{max}.

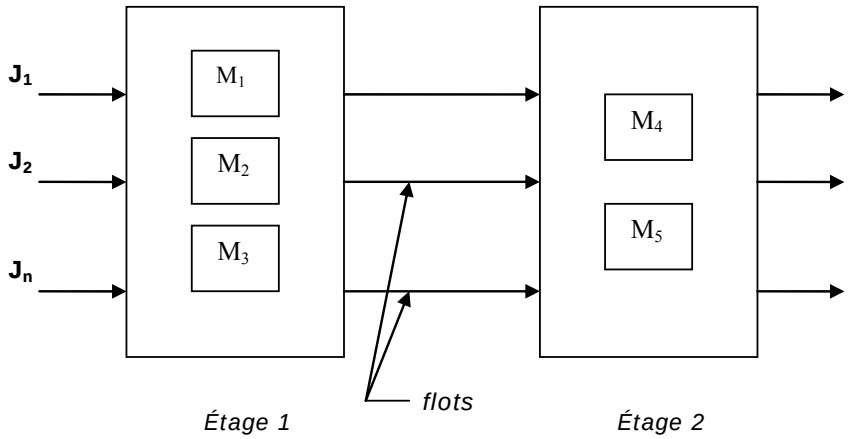


Fig.3. Exemple de flow shop hybride à deux étages et n jobs avec m_1=3 et m_2=2

Lorsque $m_1=m_2=1$, le problème correspond à un flow shop classique à deux machines qui peut être résolu optimalement à l'aide de l'algorithme de Johnson dont la complexité est polynomiale. Cette règle garantit l'optimalité d'une séquence dans laquelle le job *i* est exécuté avant le job *j* lorsque :

$$\min\{p_{1,i}, p_{2,j}\} \leq \min\{p_{1,j}, p_{2,i}\}$$

Dans la majorité des publications traitant du flow shop hybride à deux étages, les auteurs supposent qu'il existe une seule machine dans l'un des deux étages.

Pour résoudre ce problème de flow shop hybride à 2 étages, différents éléments ont été proposés : bornes inférieures, méthodes approchées et méthodes exactes. La suite de ce paragraphe en présente un certain nombre sans prétendre à l'exhaustivité.

Dans la suite, on note $\underline{p}_{i,k}$ la $k^{ème}$ plus petite valeur de $p_{i,j}$ pour $(i=1,2)$ et $j \in J$.

a. Flow shop hybride à 2 étages avec 1 seule machine au 2ème étage

Dans [Gupta, 1988], l'auteur s'est intéressé aux problèmes de flow shop hybride ayant une seule machine au deuxième étage ($m_2=1$). Il a de plus prouvé que le problème *F2(P)* | | C_{max} est NP-difficile au sens fort dans le cas où $\max(m_1, m_2)>1$.

En se basant sur l'idée qu'aucun job $j \in J$ ne peut être ordonnancé sur E_2 avant de passer $\underline{p}_{1,1} = \min_{j \in J} p_{1,j}$ unités de temps dans E_1, l'auteur a proposé la borne inférieure du makespan suivante :

$$LB_1 = \underline{p}_{1,1} + \sum_{j \in J} p_{2,j}$$

Ensuite, l'auteur a remarqué que, puisque le deuxième étage contient une seule machine, le temps d'attente au deuxième étage est nécessairement supérieur à $\max\{\underline{p}_{1,1}; \underline{p}_{1,2} - \underline{p}_{2,1}\}$. D'où l'expression de la deuxième borne inférieure :

$$LB_2 = \max\left\{\underline{p}_{1,1}; \underline{p}_{1,2} - \underline{p}_{2,1}\right\} + \sum_{j \in J} p_{2,j}$$

On note que $LB_2 > LB_1$. De plus, le dernier job exécuté au premier étage nécessite au moins $\underline{p}_{2,1}$ unités de temps pour la fin de son exécution, d'où la proposition de la troisième borne inférieure :

$$LB_3 = \left\lceil \frac{1}{m_1} \sum_{j \in J} p_{1,j} \right\rceil + \underline{p}_{2,1}$$

Ainsi,

$$LB = \max\{LB_2, LB_3\}$$

Dans ce même article, l'auteur a proposé une méthode heuristique qui procède par décomposition du problème initial en deux phases, à savoir la résolution du problème de séquencement, puis celui du problème d'affectation. Afin de résoudre le problème de séquencement, l'auteur suppose que chaque étage contient une seule machine (flow shop classique) et il applique la règle de Johnson afin de définir une liste des jobs ordonnés selon un ordre croissant de priorité. Pour résoudre le problème d'affectation, une règle de priorité a été définie permettant de faire passer le job prioritaire sur la dernière machine disponible au premier étage, et ce pour minimiser le temps d'inactivité de la machine au deuxième étage.

Dans [Gupta et Tunc, 1991], les auteurs se sont intéressés au cas où le deuxième étage domine le premier étage c.-à-d. où $\left(\sum_{j \in J} p_{1,j} \Big/ m_1 < \sum_{j \in J} p_{2,j} \right)$. Ils ont remarqué que dans ce cas, l'algorithme développé dans [Gupta, 1988] n'était pas efficace en termes de qualité du makespan. Pour remédier à ce problème, les auteurs proposent d'utiliser la règle *LPT* (Longest Processing Time) au deuxième étage au lieu d'utiliser la règle de Johnson pour le séquencement des jobs.

Par la suite, une amélioration de LB_3 a été proposée dans [Guinet *et al.*, 1996]. Les auteurs ont remarqué qu'on peut avoir $\left\lceil \frac{1}{m_1} \sum_{j \in J} p_{1,j} \right\rceil < \max_{j \in J} p_{1,j}$, ce qui donne une autre borne dont l'expression est :

$$LB_4 = \max\left(\max_{j \in J} p_{1,j}, \left\lceil \frac{1}{m_1} \sum_{j \in J} p_{1,j} \right\rceil \right) + \underline{p}_{2,1}$$

Ensuite, une autre borne inférieure se reposant sur la règle de Johnson a été proposée dans [Gupta *et al.*, 1997] :

$$LB_5 = \max_{j \in J} \left(\sum_{i=1}^{j} \left(p_{1,\sigma(i)} \right) \Big/ m_1 + \sum_{i=j}^{n} p_{2,i} \right)$$

avec $\sigma(i)$ la séquence obtenue en appliquant la règle de Johnson. En effet, pour chaque ordonnancement *S* et son correspondant $\sigma(S)$ au deuxième étage E_2, il existe au moins un job j_0 tel que :

$$c_{1,j_0} + \sum_{i=j_0}^{n} p_{2,\sigma(i)} = C_{\max}$$

Ce job est appelé job critique. En particulier, si *S** et $\sigma^*(i)$ sont respectivement un ordonnancement optimal et sa permutation correspondante au second étage, alors

$$c_{1,j_0} + \sum_{i=j_0}^{n} p_{2,\sigma^*(i)} = C^*_{\max}$$

Ainsi,

$$c_{1,j} + \sum_{i=j}^{n} p_{2,\sigma^*(i)} \le C^*_{\max} \quad \text{pour chaque job } j \in J.$$

Etant donné que le job $\sigma^*(j)$ ne peut passer sur E_2 qu'après l'ordonnancement de tous les jobs $\{\sigma^*(i), i = 1..j\}$ dans E_1, alors $\sum_{i=1}^{j} p_{1,\sigma^*(i)} \le m_1 c_{1,j}$. Par conséquent,

$$\left(\sum_{i=1}^{j} \left(p_{1,\sigma^*(i)} \right) \Big/ m_1 + \sum_{i=j}^{n} p_{2,i} \right) \le C^*_{\max} \text{ pour chaque job } j \in J.$$

De plus, $\max_{j \in J} \left(\sum_{i=1}^{j} \left(p_{1,\sigma^*(i)} \right) \Big/ m_1 + \sum_{i=j}^{n} p_{2,\sigma^*(i)} \right)$ est le makespan de $\{\sigma^*(i), i \in J\}$ du problème du flow shop hybride à deux machines ayant les durées opératoires $\left\{ \frac{p_{1,\sigma^*(i)}}{m_1}, p_{2,\sigma^*(i)}, i \in J \right\}$. Le makespan optimal pour cette dernière instance est obtenu par application de la règle de Johnson qui déduit la permutation σ. Par conséquent,

$$\max_{j \in J} \left(\sum_{i=1}^{j} \left(p_{1,\sigma(i)} \right) \Big/ m_1 + \sum_{i=j}^{n} p_{2,\sigma(i)} \right) \le \max_{j \in J} \left(\sum_{i=1}^{j} \left(p_{1,\sigma^*(i)} \right) \Big/ m_1 + \sum_{i=j}^{n} p_{2,\sigma^*(i)} \right) \le C^*_{\max}$$

Les auteurs ([Gupta *et al.*, 1997]) ont développé une autre borne inférieure par relaxation de la contrainte de capacité de machine au premier étage. Le problème est alors relaxé au problème d'ordonnancement à une seule machine $1 \mid r_j \mid C_{\max}$ avec r_j la date de début au plus tôt du job j et qui est égale à $p_{1,j}$ après cette relaxation. La solution optimale pour ce dernier problème est obtenue en ordonnançant les jobs dans l'ordre croissant de leur date de début au plus tôt. Se basant sur ce principe, on a la borne donnée par l'expression :

$$LB_6 = \max_{i \in J} \left(p_{1,\sigma(i)} + \sum_{i=j}^{n} p_{2,\sigma(i)} \right)$$

Dans ce travail, les auteurs ont développé un algorithme de type branch-and-bound (B&B), caractérisé par une règle de dominance permettant de réduire la taille de l'arbre de recherche. L'évaluation de cette procédure a été faite sur des instances de petite taille. Ils ont également introduit deux résultats importants. Le premier est la symétrie du problème de flow shop hybride et le deuxième résultat concerne le cas où m_2=1. Etant donné un ordonnancement S au premier étage, un ordonnancement optimal au second étage peut être obtenu en ordonnançant les jobs dans l'ordre croissant de leur date de fin d'exécution au premier étage $C_{1,j}$ ($j \in J$).

b. Flow shop hybride à 2 étages avec 1 seule machine au 1er étage

Dans [Uetake *et al.*, 1995], les auteurs se sont intéressés au cas où le premier étage est constitué d'une seule machine (m_1=1). Ils ont défini une liste de priorité entre les jobs en se basant sur trois heuristiques : la règle *SPT* (Shortest Processing Time), la règle *LPT* (Longest Processing Time) et le classement par ordre croissant des rapports ($p_{1,j}/p_{2,j}$). Cette dernière règle a donné les meilleures solutions.

c. Flow shop hybride à 2 étages avec au moins 2 machines par étage

Dans [Lee et Vairaktarakis, 1994], les auteurs se sont intéressés au cas où $m_1 \geq 2$ et $m_2 \geq 2$. Ils ont montré que la date de fin d'exécution C_{LB} , obtenue par application de la règle de Johnson sur le problème du flow shop hybride ayant les durées opératoires $\left\{\frac{p_{1,j}}{m_1}, \frac{p_{2,j}}{m_2}, j \in J\right\}$, est une borne inférieure pour le problème du flow shop hybride à deux étages. De plus, si au deuxième étage il existe une machine non active au moins pendant une durée allant de 0 à $\underline{p}_{1,1}$ et ainsi de suite jusqu'à $\underline{p}_{1,m_2}$, alors

$$LB_7 = \left\lceil \frac{\sum_{1 \leq j \leq m_2} \underline{p}_{1,j} + \sum_{j \in J} p_{2,j}}{m_2} \right\rceil$$

est une borne inférieure dans le cas où $m_1 \geq m_2$. Dans le cas contraire, il existe au moins une machine dans E_2 ne pouvant commencer avant $\underline{p}_{1,1} + \underline{p}_{1,m1+1}, \ldots,$ et une machine non active au moins pour une durée de $\underline{p}_{1,1} + \underline{p}_{1,m2}$. Par conséquent, on obtient une borne inférieure exprimée par :

$$LB_8 = \left\lceil \frac{\sum_{1 \leq j \leq m_2} \underline{p}_{1,j} + (m_1 - m_2)\underline{p}_{1,1} + \sum_{j \in J} p_{2,j}}{m_2} \right\rceil$$

En se basant sur l'idée qu'un flow shop hybride à deux étages peut être considéré comme la combinaison du problème de flow shop classique à deux machines et du problème à machines parallèles, les auteurs ont proposé plusieurs heuristiques pour les deux types de problèmes. En effet, ils ont suggéré l'utilisation de la règle de Johnson afin d'obtenir une liste de priorité entre les jobs, puis une heuristique d'affectation des jobs à la première machine disponible (plus connue sous l'acronyme *FAM* : First Available Machine).

Dans [Haouari et M'Hallah, 1997], les auteurs proposent une minimisation de la quantité du temps d'inactivité dans E_2. Ils utilisent pour cela la règle *SPT* qui entraîne la borne inférieure suivante :

$$LB_9 = \left\lceil \frac{SPT(m_2) + \sum_{j \in J} p_{2,j}}{m_2} \right\rceil$$

avec $SPT(m_2)$ correspondant à la somme minimale des dates de fin d'exécution des m_2 plus petites $p_{1,j}$. En procédant par symétrie, on obtient :

$$LB_{10} = \left\lceil \frac{SPT(m_1) + \sum_{j \in J} p_{1,j}}{m_1} \right\rceil$$

D'où,

$$LB_{11} = \max\{LB_9, LB_{10}\}$$

Dans ce travail, les auteurs utilisent une combinaison entre une méthode Tabou (TS) et un recuit simulé (SA) et ils montrent par des expérimentations que leur méthode donne de bonnes solutions.

Pour le même problème, un résultat important prouvant l'existence d'une méthode d'approximation dont la complexité est polynomiale pour le problème de flow shop hybride à deux étages a été proposé dans [Schuurman et Woeginger, 2000].

Dans [Haouari *et al.*, 2006], les auteurs ont amélioré la borne inférieure LB_{11}. En effet, ils ont remarqué qu'en utilisant la règle *SPT*, on ne tient compte que du rang du job dans la séquence donnée par cette dernière alors que la position d'un job j dépend de la machine et du nombre de jobs qui sont affectés à elle avant j.

Les auteurs ont prouvé que leur nouvelle borne inférieure domine la borne développée dans [Lee *et al.*, 1994]. Une autre borne inférieure basée sur l'idée de relaxation de la contrainte de capacité du premier étage a été développée dans ce travail, ce qui ramène le problème à un problème de machines parallèles au deuxième étage qui est résolu optimalement en utilisant un B&B.

Dans le même travail, les auteurs ont développé une méthode heuristique composée de deux phases. La première phase consiste à ordonnancer les jobs au premier étage en respectant l'ordre croissant des $p_{2,j}$ (LPT_2). La seconde phase consiste à affecter ces jobs en commençant par le job ayant la plus grande $p_{2,j}$ à la première machine disponible. Les auteurs ont proposé une autre approche inspirée de l'heuristique « shifting bottleneck » initialement développée pour la résolution des problèmes de job shop. Leur travail a prouvé son efficacité pour la résolution de ce type de problème.

Notons que ce travail présente les meilleures bornes inférieures connues de la littérature et a été évalué sur différents benchmarks.

2.2.2. Cas du flow shop hybride général

Le flow shop hybride est une généralisation du flow shop traditionnel au cas où plusieurs machines sont disponibles dans chacun des étages pour exécuter les différentes tâches et une généralisation du flow shop hybride à deux étages au cas où le nombre d'étages est supérieur à deux (voir figure 4).

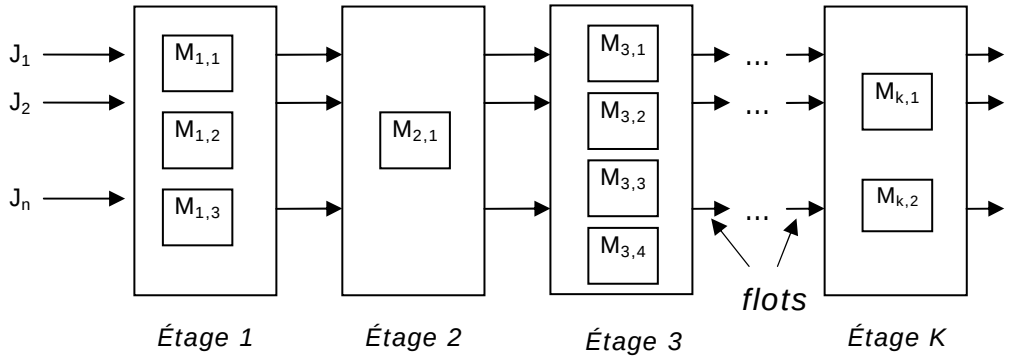


Fig.4. Exemple de flow shop hybride à k étages et n jobs

Le problème du flow shop hybride se définit formellement comme un ensemble $J = \{1,2...,n\}$ de n jobs et un ensemble $E = \{1,2,...,k\}$ de k étages. Chaque étage E_i contient m_i machines parallèles identiques, avec $(i = 1,2,...,k)$. Chaque job j doit être traité par une seule machine de l'étage E_i, sans interruption, pendant $p_{i,j}$ unités de temps. Pour chacun des jobs, l'ordre de passage des opérations dans les k étages est le même. L'objectif est de trouver un ordonnancement faisable minimisant la durée d'exécution totale ou makespan. Ce problème peut être noté *F(P)* | | C_{max}. Comme dans le cas à 2 étages, les problèmes de flow shop hybride de la littérature ne considèrent en général que des machines identiques à chaque étage.

Le flow shop hybride est NP-difficile s'il est constitué d'au moins deux étages et s'il y a au moins deux machines disponibles dans l'un des étages [Gupta, 1988].

Différents mécanismes ont été développés pour résoudre ce problème : des bornes inférieures, des méthodes approchées et des méthodes exactes. Nous en présentons ci-après quelques-uns d'entre eux.

Dans [Brah et Hunsuker, 1991], les auteurs ont proposé la première procédure de Branch and Bound (B&B) pour la résolution optimale de ce problème. Cette méthode repose sur l'énumération des séquences d'opérations associées aux machines pour chaque étage. Ainsi à un nœud

correspond, soit l'affectation d'une opération à la suite des opérations déjà affectées sur la machine en cours de "remplissage", soit le choix d'une nouvelle machine et l'affectation de l'opération sur cette machine libre. Ainsi, l'arbre de recherche construit par cette méthode comporte deux types de nœuds : le premier (T1) correspond à l'affectation d'une tâche sur une machine jusqu'alors inoccupée ; le second (T2) correspond à l'affectation d'une tâche à la suite d'une autre tâche sur la même machine. Cette distinction permet à la fois de fixer les dates de début des tâches, mais également d'affecter les tâches sur les machines. Cinq règles sont énoncées par les auteurs garantissant l'optimalité des solutions trouvées, ainsi que la limitation des solutions redondantes explorées :

- les jobs ne sont ordonnancés qu'une fois par étage ;
- pour un étage k donné, il n'y aura pas plus de nœuds correspondant à une affectation d'une tâche sur une machine libre (T1). Les contraintes de ressources sont donc respectées ;
- les tâches sont affectées "en premier" sur les machines dans l'ordre de leur indice : si à un niveau un job j est affecté en premier sur une machine, alors au cours de l'exploration des niveaux suivants les jobs i tels que $j \prec i$ ne peuvent pas être choisis pour débuter sur une machine ;
- pour chaque partie de l'arborescence construite correspondant à un étage k, il y aura, sur tout chemin allant de la racine à l'une des feuilles, m_k nœuds de type (T1), ce qui correspond au fait que toutes les machines seront utilisées pour minimiser le temps d'achèvement des travaux. Cette règle n'est appliquée que s'il y a plus de jobs à effectuer sur l'étage que de machines disponibles ;
- pour le sous-arbre correspondant à l'étage s, la racine aura $n - m_k + 1$ nœuds fils. Ceci correspond au fait que les n tâches peuvent être choisies pour débuter sur la machine 1. Cependant, la solution dans laquelle une tâche commence sur la machine 1 peut également être construite en faisant débuter cette tâche sur n'importe quelle machine, d'où l'expression $n - m_k + 1$.

Les auteurs ont développé, dans ce travail, deux classes de bornes inférieures calculées au niveau de chaque nœud de l'arborescence. La première borne est liée à la machine alors que la deuxième est liée au job. Les notations dont nous rappelons la signification ici sont celles présentées dans [Kis *et al.*, 2005].

- Y : sous-ensemble des jobs déjà ordonnancés à l'étage k,
- $Sch^{(k)}(Y)$: ordonnancement partiel des jobs Y à l'étage k,
- $C[Sch^{(k)}(Y)]_m$: date de fin d'exécution de l'ordonnancement partiel sur la machine m,

- $AC[Sch^{(k)}(Y)]$: date moyenne de fin d'exécution de tous les jobs ordonnancés à l'étage k, soit :

$$AC[Sch^{(k)}(Y)] = \frac{\sum_{m=1}^{M^{(k)}} C[Sch^{(k)}(Y)]_m}{M^{(k)}} + \frac{\sum_{j \in J-Y} p_j^{(k)}}{M^{(k)}}$$

- $MC[Sch^{(k)}(Y)]$: expression de la date de fin maximale des jobs de Y à l'étage k sur la machine m, soit :

$$MC[Sch^{(k)}(Y)] = \max_{1 \le m \le M^{(s)}} C[Sch^{(k)}(Y)]_m$$

Deux bornes inférieures ont été proposées :

- une borne inférieure basée sur la notion de machine, *LBM*, est définie par :

$$LBM[Sch^{(k)}(Y)] = \begin{cases} AC[Sch^{(k)}(Y)] + \min_{i \in J-Y} \{ \sum_{k'=k+1}^{K} p_i^{(k')} \} \textbf{ si } AC[Sch^{(k)}(Y)] \ge MC[Sch^{(k)}(Y)] \\ MC[Sch^{(k)}(Y)] + \min_{i \in Y} \{ \sum_{k'=k+1}^{K} p_i^{(k')} \} \textbf{ sinon} \end{cases}$$

- une borne inférieure basée sur la notion de job, *LBJ*, est donnée par :

$$LBJ[Sch^{(k)}(Y)] = \min_{1 \le m \le M^{(s)}} \{C[Sch^{(k)}(Y)]_m\} + \min_{i \in J-Y} \{ \sum_{k'=k}^{K} p_i^{(k')} \}$$

Finalement, la composition de (3) et (4), donne une borne inférieure, *LBC*, donnée par :

$$LBC[Sch^{(k)}(Y)] = \max\{LBM[Sch^{(k)}(Y)],\ LBJ[Sch^{(k)}(Y)]\}$$

Dans [Vandevelde, 1994], l'auteur a proposé des bornes inférieures basées sur la relaxation du problème du flow shop hybride en le considérant comme un ensemble de problèmes à machines parallèles. La relaxation consiste à supposer que la capacité des étages est illimitée sauf un, ce qui nous permet de dire qu'il n'y a pas de temps d'attente des jobs pour leur réalisation. En considérant l'étage E_k non relaxé, on obtient un problème à machines parallèles $P|r_j, q_j|C_{\max}$, avec :

$$\begin{cases} r_j = \sum_{i=1}^{k-1} p_{i,j} & \textbf{si } k > 1, \\ r_j = 0, \quad p_j = p_{k,j}, \quad q_j = \sum_{i=k+1}^{k} p_{i,j} & \textbf{si } k < K \\ q_j = 0 & \textbf{sinon} \end{cases}$$

Par conséquent, toute borne inférieure pour le problème à machines parallèles est aussi une borne inférieure pour le problème du flow shop hybride. L'auteur utilise alors les bornes inférieures développées dans [Carlier, 1987] et propose des améliorations de ces dernières. De plus, il a utilisé l'algorithme développé dans [Carlier, 1987] pour calculer une autre borne inférieure. En effet, si L_k est la borne inférieure calculée à l'étage k, alors $L = \max_{k=1,\ldots,K} L_k$ est la borne inférieure retenue pour le problème.

Dans ce même travail, l'auteur a proposé une méthode heuristique permettant d'avoir un ordonnancement admissible qui consiste à commencer l'ordonnancement étage par étage en résolvant le problème à machines parallèles associé à chaque étage en utilisant l'algorithme de [Carlier, 1987].

Le problème à machines parallèles $P\left|r_j, q_j\right|C_{\max}$ associé au premier étage est donné par :

$$\left\{r_j = 0; p_j = p_{1,j}; q_j = \sum_{i=2}^{k} p_{i,j}\right\} \text{ pour } j \in J .$$

Une fois l'ordonnancement du premier étage réalisé, on passe au deuxième étage dont le problème à machines parallèles associé est donné par :

$$\left\{r_j = c_{1,j}; p_j = p_{2,j}; q_j = \sum_{i=3}^{k} p_{i,j}\right\} \text{ pour } j \in J .$$

Et ainsi de suite jusqu'à atteindre le dernier étage E_K.

Le problème associé est :

$$\begin{cases} r_j = c_{k-1,j} & \textbf{si } k > 1 \\ r_j = 0 & \textbf{sinon} \\ p_j = p_{k,j} & \textbf{pour } j \in J \\ q_j = \sum_{i=k+1}^{K} p_{i,j} & \textbf{si } k < K \\ q_j = 0 & \textbf{sinon} \end{cases}$$

Cette heuristique est connue sous le nom *First Center Up* (*FCU*). Puisque le problème de flow shop hybride est symétrique, on peut commencer par le dernier étage et dans ce cas, l'heuristique sera nommée *Last Center Down* (*LCD*).

L'auteur a fait une extension de ces deux heuristiques, en appliquant *FCU* de l'étage E_k à l'étage E_K et *LCD* de E_{k-1} à l'étage E_1. Une alternative consiste à appliquer *LCD* de l'étage E_k à l'étage E_1 et *FCU* de l'étage E_{k+1} à l'étage E_K.

Une étude expérimentale a prouvé que la borne inférieure développée dans ce travail n'est efficace qu'en présence d'un seul étage critique. Dans le cas contraire, elle perd son efficacité.

Dans [Hunsucker et Shah, 1994], les auteurs proposent une étude comparative de six heuristiques pour la résolution du flow shop hybride dans son cas général. Les règles considérées sont : *FIFO* (First In First Out), *LIFO* (Last In First Out), *SPT* (Shortest Processing Time), *LPT* (Longest Processing Time), *MWR* (Most Work Remaining) et *LWR* (Least Work Remaining). Une étude expérimentale importante sur des problèmes générés aléatoirement a montré que la règle *SPT* donne les meilleurs résultats et la règle *LPT* les moins bons.

Dans [Lee et Vairaktarakis, 1994], les auteurs ont proposé une méthode heuristique notée H2 qui utilise une autre méthode, notée H1, développée par les mêmes auteurs pour le problème de flow shop hybride à deux étages (décrite dans le paragraphe précédent). Dans H2, l'heuristique H1 est appliquée successivement aux étages deux à deux : E_{2r-1} et E_{2r} pour $r \in \{1,2,...,K/2\}$. L'ordonnancement résultant est noté S_r. Ensuite, toutes les solutions S_r sont concaténées pour former un ordonnancement final.

Dans [Perregaard, 1995], l'auteur propose un nouveau schéma de branchement qui se rapproche de celui proposé dans [Brah et Hunsucker, 1991], du fait que les séquences de tâches sur les machines ne sont pas construites chronologiquement.

Ce schéma a pour but de déterminer, d'une part, les jobs qui s'exécutent sur la même machine pour un étage donné, et, d'autre part, un ordre s'appliquant aux jobs s'exécutant sur une même machine. Les jobs ordonnés s'exécutant sur une même machine forment une chaîne. Le but est de construire des chaînes de jobs pour chaque étage E_k. Chaque chaîne est par la suite affectée à une machine et les jobs s'exécutent dans l'ordre imposé par la chaîne.

Dans [Leon et Ramamoorthy, 1997], les auteurs se sont intéressés à concevoir des voisinages des données du problème au lieu de générer des voisinages des solutions. En effet, ils ont remarqué qu'une légère variation des données peut fournir une bonne solution du problème initial. Ils ont testé leur approche sur des problèmes réels rencontrés dans l'industrie et ils ont prouvé son efficacité en termes de qualité de solutions.

Pour résoudre le problème de flow shop hybride, [Portmann *et al.*, 1998] ont proposé une amélioration du B&B développé dans [Brah et Hunsucker, 1991]. Différentes heuristiques sont proposées pour calculer des bornes

inférieures initiales. Ces dernières sont améliorées par un algorithme génétique au cours de la résolution. Afin de réduire la taille du domaine de recherche, de nouveaux schémas de branchement ont été proposés pour obtenir une borne machine améliorée :

$$LBM[Sch^{(k)}(Y)]=\begin{cases} AC[Sch^{(k)}(Y)]+\min\limits_{i\in J-Y}\{\sum\limits_{k'=k+1}^{K}p_i^{(k')}\} \ \textbf{si} \ \ AC[Sch^{(k)}(Y)]>MC[Sch^{(k)}(Y)] \\ MC[Sch^{(k)}(Y)]+\min\limits_{i\in Y}\{\sum\limits_{k'=k+1}^{K}p_i^{(k')}\} \ \textbf{si} \ \ AC[Sch^{(k)}(Y)]<MC[Sch^{(k)}(Y)] \\ AC[Sch^{(k)}(Y)]+\max\{\min\limits_{i\in J-Y}\sum\limits_{k'=k+1}^{K}p_i^{(k')},\min\limits_{i\in Y}\sum\limits_{k'=k+1}^{K}p_i^{(k')}\} \ \textbf{sinon} \end{cases}$$

Une étude expérimentale montre que leur approche permet de résoudre des problèmes de petite taille optimalement (n = 10,15 et K=2,3,5).

Le travail de [Riane *et al.*, 1998] s'est intéressé à un problème de flow shop hybride réel qui présente une particularité au cas où $m_1 = 1, m_2 = 2, m_3 = 1$. Les auteurs ont développé deux méthodes heuristiques. La première repose sur la programmation linéaire et la deuxième est basée sur le B&B. Une expérimentation étendue a mené à avoir une excellente performance des heuristiques développées.

Dans [Carlier et Néron, 2000], les auteurs proposent une procédure de B&B non classique, dont le schéma de séparation non chronologique est original et permet de traiter des problèmes de taille importante (jusqu'à 150 tâches). Le schéma de séparation possède l'avantage de ne pas affecter les tâches sur les machines, et ainsi de limiter efficacement la combinatoire du problème. De plus, ce schéma de branchement permet une résolution efficace des problèmes si un étage critique existe. Une nouvelle borne basée sur des dates de disponibilité des machines est également présentée.

Pour la résolution exacte du flow shop hybride, [Néron *et al.*, 2001] ont suggéré une nouvelle procédure de B&B. Elle repose principalement sur l'ajustement systématique des dates de disponibilité et des durées de latence des tâches en utilisant le raisonnement énergétique. Celui-ci permet d'effectuer des raisonnements quantitatifs intégrant les contraintes de temps et de ressources [Lopez, 1991]. Ce raisonnement peut produire des conditions de séquencement en interdisant la localisation d'une tâche sur certains intervalles de temps qui engendrerait un bilan énergétique déficitaire. Plusieurs méthodes, de complexités diverses, utilisant ce raisonnement sont proposées. Les méthodes ainsi conçues améliorent sensiblement les résultats obtenus pour des problèmes n'ayant pas d'étage critique (100% des problèmes de 50 tâches sans étage critique sont résolus). Une étude expérimentale a mis l'accent sur l'importance des méthodes d'ajustements des dates de disponibilité et des durées de latence, même au prix d'une forte complexité.

Dans [Negenman, 2001], l'auteur a proposé une méthode de recherche locale basée sur un recuit simulé (SA), une recherche Tabou (TS) et une méthode connue sous le nom Variable-depth Search (VS) [Kernighan et Lin, 1970 ; Marsland et Björnsson, 2000]. Le principe de ces méthodes est de chercher une solution dans le voisinage d'une solution courante donnée dans le but de l'améliorer. Ce voisinage est constitué de toutes les solutions obtenues en effectuant une seule transition sur la solution courante. Pour ce fait, l'auteur propose d'utiliser des approches conçues pour d'autres types de problèmes d'ordonnancement (et qui seront présentées en détail dans le paragraphe suivant) :

- la recherche à deux niveaux développée dans [Brandimarte, 1993] ;
- l'approche développée dans [Hurink *et al.*, 1994] ;
- l'approche développée dans [Dauzère-Pérès et Paulli, 1994] ;
- l'approche développée dans [Nowicki et Smutnicki, 1996].

Une étude expérimentale basée sur les problèmes générés dans [Vandevelde, 1994] a montré que l'approche d'exploration de voisinages donnée dans [Nowicki et Smutnicki, 1996] fournit les meilleures solutions.

Dans [Engin et Döyen, 2004], les auteurs ont proposé une méthode heuristique basée sur le principe des systèmes immunitaires artificiels (Artificial Immune System ou AIS). C'est une heuristique inspirée par les principes et le fonctionnement du système immunitaire naturel des vertébrés qui exploite les caractéristiques du système immunitaire pour ce qui est de l'apprentissage et de la mémorisation comme moyens de résolution de problèmes.

Les fonctionnements simulés dans les AIS comprennent la sélection clonale, la maturation d'affinité et la théorie des réseaux immunitaires.

2.3. Problèmes de job shop flexible

Un job shop flexible est une extension du problème à cheminements multiples classique. En effet, la résolution de ce problème présente une difficulté supplémentaire dans la mesure où chaque opération nécessite une machine pour être réalisée et cette machine doit être choisie dans un ensemble défini a priori.

Deux cas sont traités dans la littérature : le cas où les machines sont identiques (les durées opératoires sont les mêmes pour toutes les machines pouvant exécuter un job donné) et le cas où les ressources sont non-reliées ; les durées opératoires dépendent de la ressource choisie.

Ce problème peut être décrit comme suit. On dispose d'un ensemble $J = \{1,2,...,n\}$ de n jobs et un ensemble $M = \{1,2,...,m\}$ de m machines. Chaque

job i est composé de n_i opérations $\{O_{i,1}, O_{i,2}, ..., O_{i,n_i}\}$. L'opération j du job i est notée $O_{i,j}$. Chaque opération $O_{i,j}$ doit passer sur une unique machine k sélectionnée parmi un ensemble $M_{i,j} \subseteq M$ ($\forall i, \bigcap_{j=1}^{n_i} M_{i,j}$ peut être différente de l'ensemble vide) pendant une durée $p_{i,j}^k$ de temps et ce, sans interruption. Une machine ne peut exécuter qu'une seule opération à la fois. Pour une opération $O_{i,j}$, sa date de début est notée $S_{i,j}$, sa date de fin $C_{i,j}$ et sa date de début au plus tôt est $r_{i,j}$. Une date de début au plus tôt d'un job est notée r_i et sa date de fin est C_i. Ce problème est noté $JMPM \mid\mid C_{\max}$.

Plusieurs méthodes ont été développées pour résoudre ce type de problème mono- et/ou multi-critère, à savoir des méthodes heuristiques ou exactes. Ces dernières approches ont été proposées pour la résolution des instances de petite taille ou pour des cas particuliers.

La première étude de ce type de problème est celle proposée dans [Brucker et Schlie, 1990]. Les auteurs se sont basés sur une approche géométrique pour développer un algorithme polynomial pour la résolution optimale de ce problème dans le cas de deux jobs $JMPM \mid n = 2 \mid C_{\max}$.

On trouve dans [Jurisch, 1992] une procédure par séparation et évaluation ainsi que des heuristiques pour les problèmes d'ordonnancement d'atelier avec machines multi-usages (plus connue sous le nom « shop scheduling problems with multi-purpose machine »). Les ressources considérées ont une durée identique quelle que soit la ressource. L'auteur a également proposé des bornes inférieures pour le problème.

Dans [Brandimarte, 1993], l'auteur a utilisé une approche hiérarchique pour la résolution du job shop flexible. Il a commencé par la résolution du sous-problème d'affectation de ressources en utilisant des règles de priorité. Les durées opératoires sont variables en fonction des ressources. Une fois ce problème d'affectation résolu, le problème devient un job shop classique. Pour le résoudre, Brandimarte a utilisé une méthode de recherche Tabou. Les expérimentations ont été menées sur des exemples générés aléatoirement. L'heuristique a de plus été appliquée à des problèmes de job shop flexible avec minimisation des retards pondérés.

Dans [Hurink *et al.*, 1994], les auteurs ont proposé une extension du graphe disjonctif pour développer une heuristique de recherche tabou ainsi que deux voisinages permettant de générer une solution voisine basés sur la notion de bloc. Un bloc est une succession (au moins deux) d'opérations

critiques sur une même ressource. En conservant l'admissibilité de l'ordonnancement, le premier voisinage consiste à déplacer une opération d'un bloc avant (ou après) toutes les opérations du bloc. Le second voisinage consiste à changer l'affectation d'une opération du bloc.

Il a été démontré que le deuxième voisinage était connexe, ce qui est une propriété importante pour la convergence des méthodes de recherche locale. Les auteurs proposent, dans le même cadre, une heuristique basée sur les techniques d'insertion afin de construire un ordonnancement.

Des benchmarks ont été générés aléatoirement et les solutions trouvées ont été comparées avec les bornes inférieures proposées dans [Jurisch, 1992]. Les machines considérées sont identiques. Cette approche est caractérisée par le fait que l'affectation des machines aux opérations et le séquencement sur les machines sont traités séparément.

Dans [Dauzère-Pérès et Paulli, 1994], les auteurs se sont intéressés au problème de job shop flexible avec minimisation du makespan et ont proposé des conditions suffisantes permettant d'assurer que le déplacement des opérations n'augmente pas la valeur du makespan. Basée sur ces conditions, une méthode tabou a été développée pour résoudre le problème du job shop une fois l'affectation des machines aux opérations fixée. Les machines considérées sont identiques. Les résultats trouvés par leur méthode sont comparables à ceux présentés dans [Hurink *et al.*, 1994].

Une approche en deux phases a été proposée dans [Paulli, 1995] pour ordonnancer un système de production manufacturier dans lequel le nombre de jobs pouvant être exécutés simultanément est limité. Ainsi, si l'atelier est plein, un nouveau job ne peut être commencé que si l'un des jobs en cours dans l'atelier est terminé. L'objectif est de minimiser le makespan. Les machines considérées sont identiques. Dans la première phase, les machines sont affectées aux opérations en utilisant des règles de priorité, et le problème devient alors un job shop classique. La deuxième phase consiste à représenter le problème résultant par un graphe disjonctif utilisé pour développer une heuristique de recherche tabou permettant de trouver les meilleures séquences d'entrée. Les deux phases sont répétées et une opération appartenant au chemin critique est à chaque fois réaffectée.

Une autre méthode de recherche tabou a été proposée dans [Barnes et Chambers, 1996]. Les auteurs considèrent deux types de voisinages. Le premier consiste à permuter deux opérations critiques adjacentes et le deuxième consiste à réaffecter une opération critique sur toutes les machines pouvant l'exécuter. Ils ont considéré trois exemples de problème de type job shop classique FT10 de [Fisher et Thompson, 1963], LA24 et LA40 de [Lawrence, 1984] et ils ont fait une extension de ces derniers afin d'avoir des problèmes flexibles. Les machines considérées sont identiques. Les résultats

donnent une amélioration du makespan d'environ 10% par rapport aux résultats donnés dans [Chambers *et al.*, 1996].

La méthode proposée dans [Dauzère-Pérès et Paulli, 1997] repose sur une résolution intégrée du problème d'affectation et de séquencement. Basé sur une extension du graphe disjonctif, le voisinage ainsi construit est intégré pour l'exploration de l'espace des solutions dans la recherche tabou. Les résultats obtenus sont meilleurs que ceux donnés dans [Hurink *et al.*, 1994] et dans [Dauzère-Pérès et Paulli, 1994].

Dans [Mastrolilli et Gambardella, 2000], les auteurs proposent une autre approche intégrée basée sur une recherche tabou. Les machines considérées sont identiques. Cette approche utilise le graphe disjonctif pour la représentation des solutions, ainsi que deux types de voisinages (V_1 et V_2). Ces voisinages sont basés sur le déplacement d'une opération dans le graphe disjonctif. Les auteurs ont démontré la connexité d'un seul type de voisinage développé (V_2). La contribution de cette étude est la diminution de la taille du voisinage. Cette approche améliore les solutions proposées dans les études précédentes. Les expérimentations ont montré que, malgré l'absence de connexité du voisinage V_1, ce dernier donne de meilleurs résultats que V_2 grâce à sa vitesse d'exécution. Cette méthode Tabou est parmi celles fournissant actuellement les meilleurs résultats sur les instances connues de job shop flexible.

Un algorithme génétique a été également proposé pour résoudre le problème de job shop flexible dans [Kacem *et al.*, 2002]. Cet algorithme a été testé pour la résolution du problème mono et multicritère. L'objectif de cette méthode est d'optimiser conjointement deux critères classiques : le makespan et la charge des ressources. La première étape de cette approche permet de résoudre le problème d'affectation de ressources et de construire un schéma d'affectation. La deuxième étape est une approche évolutionnaire contrôlée par le modèle des affectations généré dans la première étape. Dans cette approche, les auteurs appliquent des manipulations génétiques avancées pour améliorer la qualité des solutions.

Dans sa thèse, Mati (2002) s'est intéressé à l'étude de la complexité des problèmes d'ordonnancement d'un job shop flexible à deux jobs. Il a montré que le problème de minimisation de toute fonction objectif régulière est NP-difficile au sens faible. Il a ensuite proposé un algorithme polynomial basé sur une approche géométrique pour résoudre le problème et ceci pour n'importe quelle fonction objectif régulière.

Une approche multi-agents a été introduite dans [Ennigrou et Ghedira, 2005] pour la résolution du job shop flexible en se basant également sur une

méthode de recherche Tabou. Le modèle proposé est composé de trois classes d'agents : les agents Job, les agents Ressource responsables de la satisfaction des contraintes et l'agent Interface contenant le noyau de la recherche tabou. Les auteurs ont utilisé deux types de voisinage : la permutation de deux opérations critiques adjacentes exécutées par la même ressource et la réaffectation d'une opération critique sur une autre ressource. Les expérimentations ont été menées sur les problèmes donnés dans [Brandimarte, 1993 ; Hurink *et al.*, 1994].

Xia et Wu (2005) ont développé une autre approche hiérarchique pour la résolution des problèmes de type job shop flexible multicritères. Ils ont utilisé une méthode SPO (Swarm Particle Optimization) pour la résolution du sous-problème d'affectation et le recuit simulé pour la résolution du sous-problème d'ordonnancement. L'objectif de cette méthode hybride est de minimiser le makespan, la charge des ressources et la charge des ressources critiques. Les résultats obtenus par cette approche ont montré que l'algorithme proposé est une approche viable et efficace pour le FJSP multi-critère, particulièrement pour des problèmes de grande taille.

Dans leur travail, Zhang et Gen (2005) ont proposé un algorithme génétique pour la résolution du job shop flexible multicritère. L'objectif est de minimiser le makespan, la charge maximale et la somme des charges de ressources. La méthode a été évaluée via les instances utilisées dans [Kacem *et al.*, 2002] et a donné de meilleures solutions.

Fattahi *et al.* (2007) ont proposé un modèle mathématique pour le problème de job shop flexible et une hybridation de deux méthodes : une méthode de recherche tabou (TS) et un recuit simulé (SA). Sur la base de ces deux méthodes, les auteurs ont développé six algorithmes différents basés sur la combinaison des deux méthodes et des deux approches: intégrée et hiérarchique. Les expérimentations ont permis de comparer les algorithmes entre eux. Il en résulte qu'en général, l'approche hiérarchique donne les meilleures solutions et ce en appliquant la méthode de recherche tabou pour le problème d'affectation et le recuit simulé pour le problème de séquencement.

Gao *et al.* (2007) se sont intéressés au problème de job shop flexible multicritère avec trois objectifs : minimisation du makespan, minimisation de la charge maximale des machines et minimisation de la charge de travail totale. Les machines considérées sont identiques. Un nouvel algorithme génétique croisé avec une méthode de recherche locale (basée sur le « shifting bottleneck ») a été proposé pour la résolution du problème. L'algorithme génétique utilise deux représentations pour décrire des solutions candidates pour le problème de *JMPM*. Il propose des opérateurs de

croisement et de mutation avancés pour s'adapter aux structures de chromosomes et aux caractéristiques du problème. Deux types de voisinage sont utilisés : permutation des opérations et affectation de nouvelles machines pour les opérations critiques. Afin de restreindre le domaine de recherche, la structure de voisinage est ajustée dynamiquement par une recherche locale.

Vilcot (2007) s'est intéressé au problème du job shop flexible multicritère. Pour résoudre ce problème, il a proposé deux recherches Tabou, deux algorithmes génétiques et un algorithme mémétique. Un algorithme mémétique est une hybridation entre un algorithme génétique et une méthode de recherche locale. Ici, l'auteur a proposé de remplacer la mutation de l'algorithme génétique par un algorithme de recherche tabou. Les expérimentations ont permis de comparer les algorithmes entre eux. Il en résulte que la meilleure méthode est l'algorithme génétique avec une population partiellement initialisée par la recherche Tabou basée sur la combinaison linéaire des critères. Néanmoins, dans certains cas, cette méthode est dominée par l'algorithme mémétique. L'auteur a testé les algorithmes développés sur deux combinaisons de critères, la première est l'optimisation du couple $C_{\max}|L_{\max}$ avec $L_{\max}$ le plus grand retard algébrique, la seconde est l'optimisation de $C_{\max}|\sum T$ avec $\sum T$ la somme des retards vrais (ou absolus).

Un algorithme génétique a été proposé pour la résolution du problème $JMPM||C_{\max}$ dans [Ho *et al.*, 2007]. L'originalité de cette approche réside dans l'ajout d'un module d'apprentissage permettant de reconnaître certaines caractéristiques de chromosomes au fil des générations. Cette nouvelle architecture est nommée LEGA (LEarnable Genetic Architecture). En utilisant LEGA, la connaissance extraite des générations précédentes est utilisée pour influencer la diversité et la qualité des résultats suivants. Une grande gamme de données de référence prises de la littérature et certaines instances générées par les auteurs est utilisée pour analyser l'efficacité de LEGA. Des résultats expérimentaux indiquent qu'une instanciation de LEGA appelée GENACE surpasse des approches actuelles utilisant les algorithmes évolutionnaires en termes de temps de calcul et de qualité de solutions.

Pezzella *et al.* (2008) ont également présenté un algorithme génétique pour le problème de job shop flexible. L'algorithme proposé intègre différentes stratégies pour la création de la population initiale, les choix des individus et la reproduction des nouveaux individus. Les expérimentations ont montré que l'intégration de différentes stratégies pour le croisement et la mutation donne de bonnes solutions. Les résultats obtenus sont assez proches de ceux obtenus par la méthode de recherche tabou proposée dans [Mastrollili et Gambardella, 2000].

[Tay et Ho, 2008] ont proposé des règles de priorité tirées de la programmation génétique. Bien que les règles simples aient été largement appliquées en pratique, elles restent d'une faible efficacité en raison d'un manque d'une vue globale. En effet, les règles de priorité composées semblent plus efficaces vu qu'elles reposent sur des expériences humaines. Des résultats expérimentaux montrent que ces dernières règles produites par la structure de programmation génétique surpassent les règles simples et d'autres règles composées choisies de la littérature. Dans ce travail, les auteurs ont considéré plus de cinq grands jeux-tests pour valider leur approche en s'intéressant à la minimisation de trois critères : le makespan, le retard moyen et le temps de flux moyens.

Le travail de [Gao *et al.*, 2008] s'intéresse au problème de job shop flexible multicritère avec trois objectifs : minimisation du makespan, minimisation de la charge maximale des machines et minimisation de la charge de travail totale. Les auteurs proposent ainsi une amélioration de l'algorithme génétique développé dans [Gao *et al.*, 2007]. En effet, pour restreindre l'espace de recherche, les individus donnés par l'algorithme génétique sont améliorés par la méthode de recherche du voisinage par descente (VND), qui implique deux types de mouvements, le déplacement d'une seule opération ou de deux opérations. Déplacer une opération doit supprimer l'opération, lui trouver un autre intervalle de temps, et l'affecter pendant cette période de temps. Pour ce faire, ils ont développé une méthode efficace pour trouver ces intervalles. Afin d'éviter les optima locaux donnés par la première méthode [Gao *et al.*, 2007], ils ont proposé de faire le déplacement de deux opérations simultanément. Une étude expérimentale sur des benchmarks de la littérature montre l'efficacité de leur approche. En effet, l'algorithme proposé est connu pour être un des meilleurs algorithmes pour la résolution de ce type de problème.

2.4. Conclusion

L'objectif majeur de ce chapitre a été de donner une présentation rapide de différentes méthodes utilisées pour la résolution des problèmes d'ordonnancement avec flexibilité des ressources, de type flow shop hybride et job shop flexible.

Il convient de noter qu'il ne s'agit pas ici d'une description exhaustive de l'état de l'art, étant donné qu'il existe de nombreux autres travaux dans la littérature. Cette étude bibliographique permet néanmoins de faire un tour d'horizon assez large des approches de résolution développées et permet de dégager les méthodes actuellement les plus performantes par type de problème.

Chapitre 3. Recherche à base de divergences

Dans ce chapitre, nous présentons les techniques de recherche à base de divergences qui seront utilisées dans notre travail pour la résolution des problèmes d'ordonnancement flexible. Plusieurs techniques développées sont détaillées et illustrées sur des exemples. Les algorithmes de ces méthodes sont également exposés.

3.1. Introduction

Les méthodes de recherche à base de divergences prennent leur source dans la recherche à divergence limitée (« Limited Discrepancy Search » ou LDS) présentée dans [Harvey et Ginsberg, 1995]. Le principe fondamental de la méthode LDS est de proposer une alternative à la technique de branchement classiquement utilisée dans les procédures de Backtracking. Depuis, plusieurs méthodes de recherche à base de divergences ont été conçues. Le point commun entre toutes ces méthodes est de limiter autant que possible les divergences pour l'obtention d'une solution.

La notion de divergence peut être définie comme étant la contradiction des choix d'une heuristique d'instanciation. En effet, une heuristique peut commettre des erreurs au niveau de ses choix ; il se révèle donc parfois judicieux de ne pas la suivre. Cette notion a été proposée initialement pour résoudre des problèmes de satisfaction de contraintes (CSP) à variables binaires et a ensuite été adaptée pour des problèmes d'ordonnancement dont la résolution génère des arborescences de recherche non binaires [Le Pape et Baptiste, 1999].

3.2. Recherche à divergence limitée (LDS)

La méthode LDS s'appuie sur des heuristiques de parcours d'une arborescence. L'objectif principal de cette méthode est d'explorer uniquement une partie de l'espace de recherche en se basant sur des heuristiques pour l'ordre de sélection des variables et pour leur instanciation. Ces heuristiques sont mises au point en vue de guider la recherche vers des régions prometteuses en se basant sur les caractéristiques des problèmes à résoudre ; en ordonnancement, on considère notamment : les dates de disponibilité des tâches, les dates de disponibilité des machines, les durées opératoires au cours de la recherche, etc.

L'idée de la méthode est de calculer, en premier lieu, le chemin de l'arborescence recommandé par l'heuristique, ce qui constitue le chemin à 0 divergence, suivi de tous les chemins qui divergent de l'heuristique en un seul point de décision, ce qui constitue les chemins à une seule divergence, suivi de tous les chemins qui ont 2 divergences et ainsi de suite. Cela revient à dégrader peu à peu les choix des heuristiques dans le but d'améliorer à chaque itération la qualité de l'instanciation pour la rendre admissible (dans le cadre de la résolution des CSP). En autorisant suffisamment de divergences par rapport aux heuristiques de parcours, cette méthode est complète dans le sens où elle revient à parcourir toute l'arborescence.

Une telle stratégie d'exploration se justifie par le fait que les chemins à une seule divergence d'une bonne heuristique sont les plus susceptibles de contenir une solution par rapport à ceux ayant deux divergences. Elle considère alors les chemins de l'arborescence variant peu par rapport au

chemin recommandé par l'heuristique, ceux-ci étant les plus probables de contenir des solutions admissibles.

De ce fait, la méthode LDS exprime l'idée d'un parcours incrémental de l'arborescence avec un nombre limité de divergences. Si elle ne trouve pas de solution pour k divergences, le nombre de divergences augmente et la recherche est alors recommencée avec un nombre de divergences égal à $k+1$. Si k atteint le nombre maximal de divergences possible, alors tous les chemins de l'arborescence sont explorés, ce qui prouve l'absence de solution et rend la méthode complète.

LDS peut aussi produire un algorithme incomplet en fixant $k < k_{\max}$, ce qui permet un développement partiel de l'arbre, restreint aux zones que l'heuristique estime les plus prometteuses. En pratique, la méthode LDS sera d'un très grand intérêt si elle réussit à trouver une solution de bonne qualité avec un nombre minimum de divergences.

Les premiers travaux sur LDS ont uniquement considéré des problèmes à variables binaires. Par convention, les auteurs supposent que l'heuristique privilégie l'exploration du nœud fils gauche. À tout nœud fils gauche correspond alors une valeur de divergence de 0 et à tout nœud fils droit correspond une divergence de valeur 1 de l'heuristique. Le nombre maximum de divergences nécessaires pour l'exploration complète de cette arborescence binaire est égal au nombre de variables du problème.

Considérons une arborescence binaire à trois variables. La trace de la méthode LDS après 4 itérations est donnée par la figure 5. Cette illustration est issue de [Harvey et Ginsberg, 1995]. Sur cette figure, le nombre indiqué en dessous de chaque feuille, à la première ligne, correspond au nombre de divergences qui ont permis d'atteindre cette feuille et à la deuxième ligne, le nombre correspond à l'ordre de visite des feuilles de l'arbre.

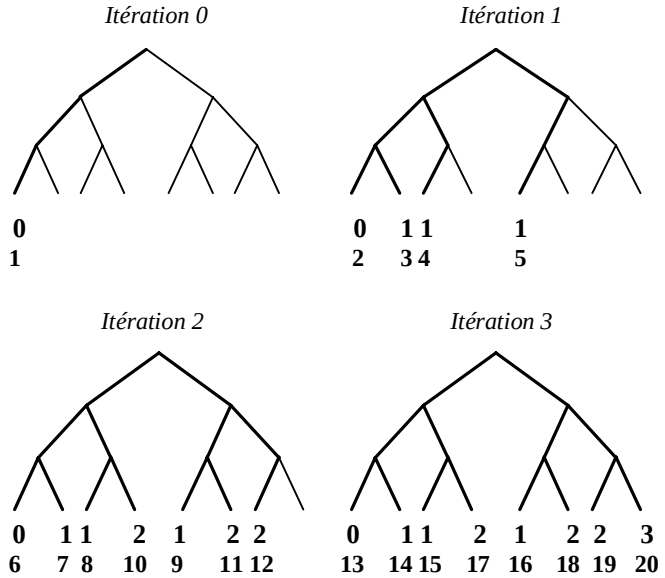


Fig.5. Principe d'exploration d'une arborescence binaire à trois variables par la méthode LDS

Dans cet exemple, les divergences sont appliquées en premier en bas de l'arbre, avec l'intuition de corriger les dernières instanciations (comme pour les méthodes de type Backtrack chronologique).

Le chemin le plus à gauche de l'arborescence correspond au chemin recommandé par l'heuristique (ayant 0 divergence). En revanche, le chemin le plus à droite correspond au chemin obtenu avec le maximum de divergences (3 divergences dans cet exemple). Suivant le principe que l'heuristique utilisée est de bonne qualité, ce dernier chemin devrait ainsi contenir une solution de mauvaise qualité vu que toutes les décisions prises, dans ce cas, sont les opposées directes des décisions recommandées par l'heuristique.

L'algorithme 1 présente le principe de la méthode LDS pour un arbre binaire.

```
LDS (nœud) :
Pour x allant de 0 à h (hauteur de l'arbre) faire
      resultat ←LDS_itération (nœud, x)
      Si resultat ≠ NULL alors retourner resultat Finsi
Fin pour
```

```
LDS_itération (nœud, k) :
// retourner un nœud but dont le chemin s'éloigne
// de k divergences de l'heuristique
Si nœud est un nœud but alors retourner nœud Finsi
a ←successeurs(nœud)
Si a={} alors retourner NULL
Sinon Si k=0 alors
           retourner (LDS_itération (fils_gauche(a),0) )
      Sinon // on effectue une divergence
           resultat ←LDS_itération(fils_droit(a), k-1)
           Si resultat ≠NULL alors retourner (resultat) Finsi
      Finsi
      retourner (LDS_itération(fils_gauche(a), k))
Finsi
```

Algorithme 1. Algorithme LDS pour un arbre binaire

Dans le cas des problèmes non binaires, deux propositions ont été faites pour adapter la notion de divergence (voir figure 6). La première voie considère la valeur recommandée par l'heuristique comme instanciation à 0 divergence et toutes les autres valeurs comme des instanciations à une divergence, cette approche est nommée comptage binaire. La seconde voie affecte à chaque autre valeur (autre que celle recommandée par l'heuristique) une divergence de manière croissante : on parle alors de comptage non binaire [Le Pape et Baptiste, 1999].

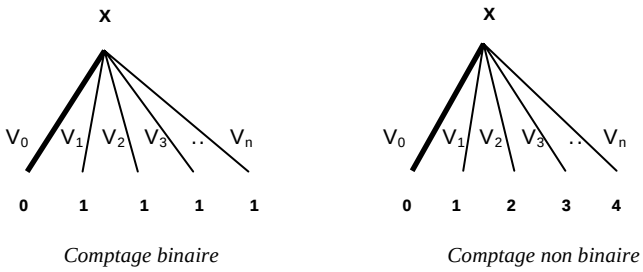


Fig.6. Notion de divergence dans un arbre non binaire

En considérant des problèmes ayant N variables non binaires, la méthode LDS peut être décrite par l'Algorithme 2.

```
k ← 0            -- k est le nombre de divergences
k_max ← N        -- N est le nombre de variables
S_ref ← instantiation_initiale()
-- S_ref est l'instanciation de référence
Tant que Non_Solution(S_ref) et (k < k_max) faire
      k ← k+1
      S_ref' ← Calcul_feuilles(S_ref, k)
      S_ref ← S_ref'
Fin Tant que
```

```
Calcul_feuilles(S_ref, k)
-- Générer l'ensemble des feuilles à k divergences par rapport à S_ref
```

Algorithme 2. Algorithme LDS pour un arbre non binaire

L'inconvénient principal de LDS est la redondance en termes de génération de nœuds. En effet, pour avoir les solutions à k divergences, les solutions de 0 à $k-1$ divergences sont revisitées. Pour éviter ce problème, différentes alternatives présentées ci-dessous ont été développées.

3.3. Méthode à divergence limitée améliorée (ILDS)

La méthode ILDS (« Improved Limited Discrepancy Search ») a été proposée dans [Korf, 1996]. Cette méthode permet à la recherche d'éviter de se diriger vers les feuilles dont le nombre de divergences est inférieur à celui de l'itération courante (voir Algorithme 3).

Ainsi, chaque feuille n'est plus visitée qu'une seule fois comme le présente la figure 7. Comme pour la figure 5, le nombre indiqué en dessous de chaque feuille, à la première ligne, correspond au nombre de divergences qui ont permis d'atteindre cette feuille. A la deuxième ligne, le nombre correspond à l'ordre de visite des feuilles de l'arbre.

En effet, ILDS permet, à k divergences, de ne visiter que les chemins ayant exactement k divergences par rapport à la solution initiale du

problème. Par contre, la redondance subsiste pour les nœuds internes de l'arbre.

```
ILDS_itération (nœud, k, p) :
// retourner un nœud but dont le chemin s'éloigne
// de k divergences de l'heuristique
Si nœud est un nœud but alors retourner nœud Finsi
a ← successeurs(nœud)
Si a={} alors retourner NULL Finsi
Si p>k  alors retourner (ILDS_itération (fils_gauche(a), k, p-1))Finsi
Si k>0  alors retourner (ILDS_itération (fils_droit(a), k-1, p-1))Finsi
```

Algorithme 3. La procédure itération de l'algorithme ILDS pour un arbre binaire

Le principe de cette méthode a été utilisé par toutes les autres améliorations de LDS. Ainsi, nous convenons dans toute la suite du document de substituer le terme « LDS » à « ILDS » (LDS désignera toujours une procédure de type ILDS).

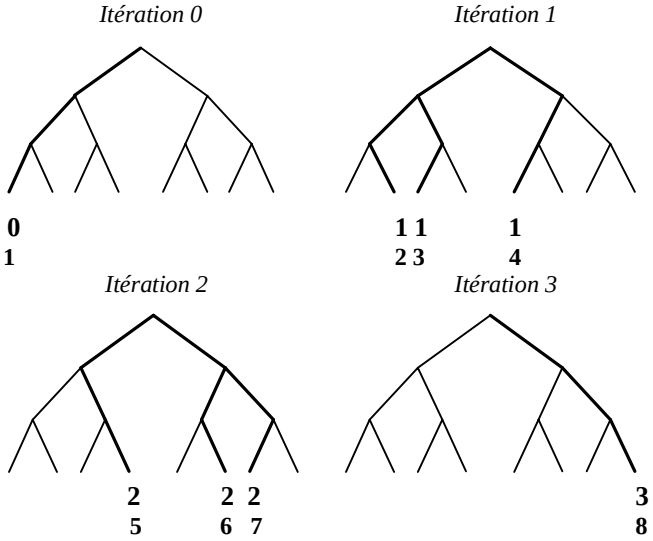


Fig.7. Principe d'exploration d'une arborescence binaire par la méthode ILDS

3.4. Méthode à divergence limitée par la profondeur (DDS)

La méthode à divergence limitée par la profondeur (Depth-bounded Discrepancy Search, DDS) a été proposée dans [Walsh, 1997]. L'idée de l'auteur est de considérer les divergences à un niveau élevé dans l'arbre de recherche plutôt qu'à un niveau bas. Le but est de corriger les premières erreurs dans les instanciations commises par l'heuristique de parcours (voir algorithme 4).

```
DDS_itération (nœud, k, p, d) :
// retourner un nœud but dont le chemin s'éloigne
// de k divergences de l'heuristique
Si nœud est un nœud but alors retourner nœud Finsi
a ←successeurs(nœud)
Si a={} alors retourner NULL Finsi
Si k=0 alors retourner (DDS_itération (fils_gauche(a), 0, p-1, d+1))Finsi
Si k=1 alors retourner (DDS_itération (fils_droit(a),0, p-1, d+1))Finsi
Si k>1 et p>k alors
     resultat ← DDS_itération (fils_gauche(a), k-1, p-1, d+1)
     Si resultat ≠NULL alors retourner (resultat)
     Sinon retourner (DDS_itération(fils_droit(a), k, p-1, d+1))Finsi
Finsi
```

Algorithme 4. La procédure itération de l'algorithme DDS pour un arbre binaire

Cette approche est justifiée par le fait que l'heuristique de choix fait généralement un bon choix dans la partie basse de l'arbre ou plutôt un choix qui n'influence pas trop sur la solution du problème. Ainsi, l'exploration des nœuds terminaux est généralement moins intéressante que celle des nœuds de la partie haute de l'arborescence.

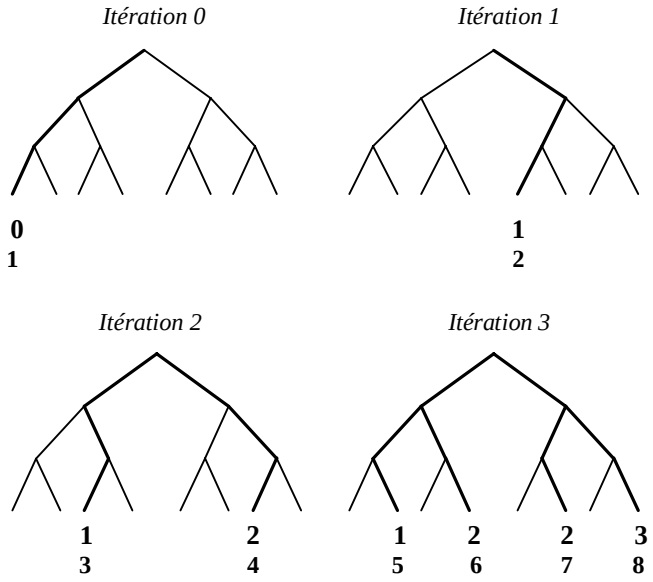


Fig.8. Principe d'exploration d'une arborescence binaire par la méthode DDS

DDS consiste à limiter la profondeur des niveaux à explorer. En effet, cette méthode permet d'incrémenter itérativement une borne sur la profondeur de l'arborescence : à l'itération 0, DDS explore le nœud terminal le plus à gauche. À l'itération *i*, DDS explore les branches obtenues par divergence à une profondeur inférieure ou égale à *i*. Comme pour la méthode ILDS, la méthode DDS n'explore pas plusieurs fois les nœuds terminaux (déjà visités). De plus, à l'itération *i+1*, pour un nœud à une profondeur *i*, DDS explore uniquement son fils droit car la branche gauche a été explorée à une des itérations précédentes.

En conclusion, pour les profondeurs inférieures, DDS explore le fils droit et gauche du nœud tandis que pour les profondeurs supérieures, seule la branche la plus à gauche de ce nœud est explorée. Le parcours de l'arborescence s'arrête lorsque le nombre maximum de divergences est atteint ou lorsque l'arbre soit totalement parcouru (voir figure 8).

```
k ← 0          -- k est le nombre de divergences
k_max ← N      -- N est le nombre de variables
d ← 1          -- d est la profondeur de la variable
S_ref ← instantiation_initiale()
 -- S_ref est l'instanciation de référence
Tant que Non_Solution(S_ref) et (k < k_max) faire
      k ← k+1
      Tant que (d < k) faire
            d←d+1
            -- Générer les feuilles à k divergences par rapport à S_ref
            S_ref' ← Calcul_feuilles(S_ref, k, d)
            -- Arrêt quand une solution est obtenue
      Fin Tant que
      d ← 1
      S_ref' ← Calcul_feuilles(S_ref, k)
      S_ref ← S_ref'
Fin Tant que
```

Algorithme 5. Algorithme DDS pour un arbre non binaire

Dans le cas d'une arborescence non binaire, l'algorithme de cette méthode est exprimé par l'algorithme 5.

Une autre variante de DDS peut être envisagée dans laquelle on limite l'exploration par la profondeur des divergences. On obtient alors une méthode dite DDS tronquée. Ceci permet de générer les branches contenant le même nombre de divergences dans un ordre où celles qui ont des divergences le plus en haut de l'arbre sont générées les premières (voir figure 13).

3.5. Méthode de recherche par profondeur d'abord intercalée (IDFS)

Messeguer et Walsh (1998) ont proposé la méthode Interleaved Depth-First Search (IDFS) qui cherche à explorer des niveaux parallèles le plus haut dans l'arborescence.

En effet, elle parcourt plusieurs sous-arborescences en parallèle (dites actives) sur certains niveaux de l'arbre (dit parallèle). Elle peut être vue comme un parcours en parallèle d'une arborescence en profondeur d'abord (DFS).

IDFS traverse le sous-arbre courant en profondeur jusqu'à ce qu'il atteigne une feuille non encore explorée. S'il atteint une solution optimale, la recherche s'arrête, sinon l'état du sous-arbre courant est sauvegardé pour être réutilisé par la suite.

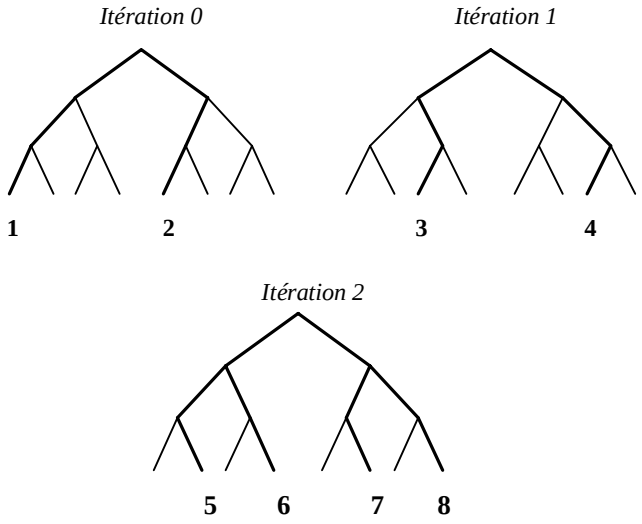


Fig.9. Principe d'exploration d'une arborescence binaire par la méthode Pure IDFS

La méthode s'intéresse donc à l'exploration du niveau parallèle supérieur dans l'arbre, tout en repartant de la racine, pour y sélectionner une autre sous-arborescence active et y répéter le même processus. Deux approches d'IDFS sont proposées :

- *Pure IDFS*, qui peut requérir un espace mémoire exponentiel, considère que tous les niveaux sont parallèles et que pour chaque niveau, tous les sous-arbres sont actifs (voir figure 9).
- *Limited IDFS*, qui s'intéresse à l'exploration à des niveaux parallèles parmi un nombre limité de sous-arbres à chaque niveau parallèle (voir figure 10).

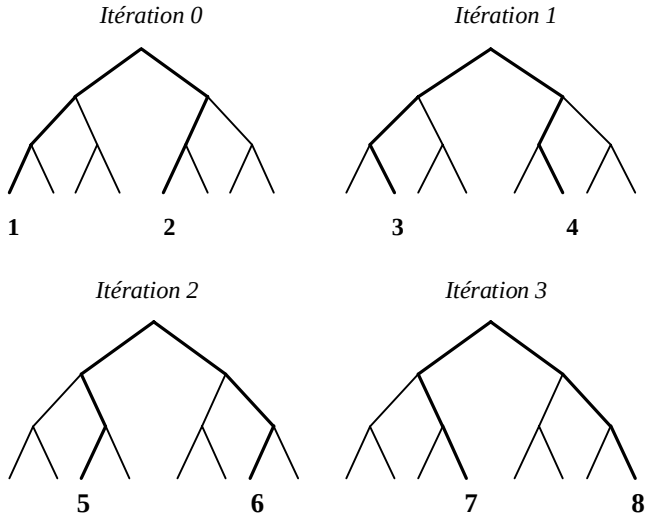


Fig.10. Principe d'exploration d'une arborescence binaire par la méthode Limited IDFS

3.6. Méthode à divergence limitée par profondeur d'abord (DBDFS)

La méthode à divergence limitée par profondeur d'abord (Discrepancy-Bounded Depth First Search, DBDFS) est une autre amélioration de LDS introduite dans [Beck *et al.*, 2000]. Cette méthode, comme son nom l'indique, donne la priorité à la profondeur par rapport à la divergence. Elle peut être considérée comme une combinaison entre DDS et la méthode classique de recherche en profondeur (DFS : Depth First Search).

Ainsi, au lieu de se restreindre à la visite de branches ayant exactement le même nombre de divergences lors de chaque itération de LDS, on examinera les branches ayant leur nombre de divergences appartenant à un intervalle donné qui dépend du numéro de l'itération. Le but est de limiter la redondance dans l'ordre de visite des branches.

Ainsi, la stratégie d'exploration de l'espace de recherche est un peu différente de celle donnée par DDS et par LDS. La figure 11 montre la différence. Le nombre noté en dessous de chaque feuille correspond à l'ordre d'exploration.

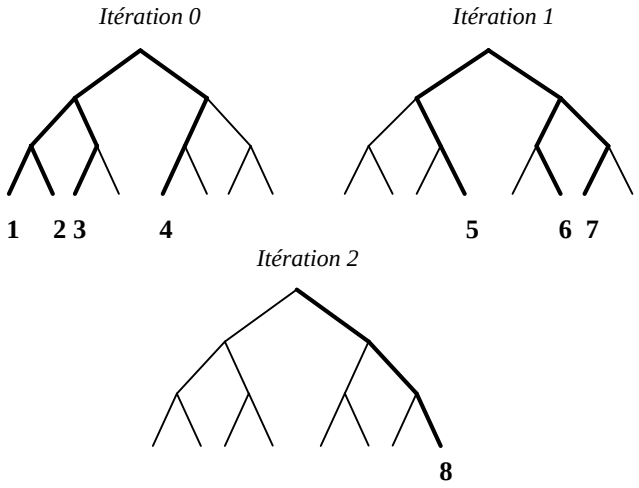


Fig.11. Principe d'exploration d'une arborescence binaire par la méthode DBDFS

3.7. Méthode à divergence limitée inversée (RLDS)

La recherche à divergence limitée inversée (Reverse LDS) proposée dans [Prcovic, 2002] diffère de LDS par le choix de valeur à examiner en premier lors d'une itération de LDS ; le fils droit est examiné avant le fils gauche pour un arbre binaire (voir figure 12).

Le nombre indiqué en dessous de chaque feuille, à la première ligne, correspond au nombre de divergences qui ont permis d'atteindre cette

feuille. A la deuxième ligne, le nombre correspond à l'ordre de visite de ses feuilles.

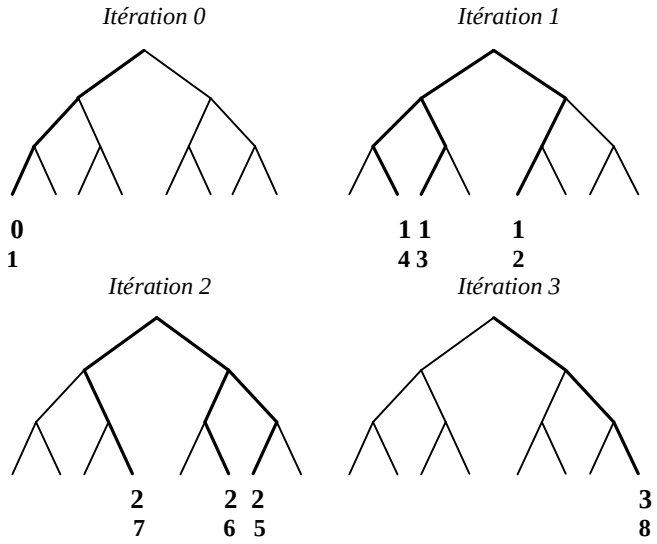


Fig.12. Principe d'exploration d'une arborescence binaire par la méthode RLDS dans l'ordre de génération des feuilles

Ceci permet de générer les branches contenant le même nombre de divergences dans un ordre où celles qui ont des divergences le plus en haut de l'arbre sont générées les premières (algorithme 6).

```
RLDS_itération (nœud, k, p) :
// retourner un nœud but dont le chemin s'éloigne
// de k divergences de l'heuristique
Si nœud est un nœud but alors retourner nœud Finsi
a ← successeurs(nœud)
Si a={} alors retourner NULL
resultat ← NULL
Si k>0  alors
    retourner (RLDS_itération (fils_droit(a), k-1, p-1)) Finsi
Si p>k et resultat = NULL alors
    resultat ← RLDS_itération (fils_gauche(a), k, p-1)
Finsi
```

Algorithme 6. La procédure itération de l'algorithme RLDS pour un arbre binaire

3.8. Méthode à divergence pondérée par sa profondeur (DWDS)

La méthode à divergence pondérée par sa profondeur (Depth-Weighted Discrepancy Search, DWDS) a été proposée dans [Prcovic, 2002]. Plutôt que de limiter le nombre de divergences, elle limite la somme des profondeurs des

divergences. Ainsi, à une somme de profondeur de divergences fixée correspondent des combinaisons de divergences telles que plus les divergences sont profondes, moins elles sont nombreuses (voir algorithme 7).

```
DWDS_itération (nœud, W, p) :
Si nœud est un nœud but alors retourner nœud Finsi
a ← successeurs(nœud)
Si a={} alors retourner NULL
Si decomposable(W, p+1, n) alors
     retourner DWDS_itération(premier(a), W, p+1)
Finsi
Tantque a≠{} faire
    Si decomposable (W-p, p+1, n) alors
       retourner DWDS_itération(premier(a), W-p, p+1)
    Finsi
    a=a\ premier(a)
Fin tantque
```

```
DWDS (nœud) :
Pour x allant de 0 à n.(n+1)/2 faire
      resultat ← DWDS_itération (nœud, x,0)
      Si resultat ≠ NULL alors
         retourner resultat
      Finsi
Fin pour
```

Algorithme 7. L'algorithme DWDS pour un arbre binaire

Pour que DWDS soit efficace, il faut que l'algorithme ne génère que des nœuds susceptibles a priori de mener à des feuilles correspondant à la somme des profondeurs de divergence qui a été fixée. Ceci revient à rejeter tout nœud de l'arbre de recherche qui ne peut pas mener à une solution dont la somme des profondeurs de divergences égale à une constante *W*.

3.9. Méthode par montée de divergences (CDS)

Contrairement aux méthodes à base de divergences que l'on vient de présenter et qui ont été conçues pour la recherche de solutions admissibles, la méthode par montée de divergences (Climbing Discrepancy Search, CDS) est une méthode d'optimisation. La méthode CDS est définie comme une méthode de recherche locale basée sur une adaptation de la notion de divergence afin de trouver une bonne solution pour les problèmes d'optimisation combinatoire [Milano et Roli, 2002].

CDS commence l'exploration à partir d'une solution initiale suggérée par une heuristique donnée. Les nœuds ayant une divergence égale à 1 sont explorés d'abord, puis ceux ayant la divergence 2, et ainsi de suite. Quand une feuille améliorant la fonction objectif est obtenue, la solution de

référence est mise à jour, le nombre de divergences est remis à zéro et le processus d'exploration se déclenche de nouveau (voir algorithme 8).

```
k ← 0            -- k est le nombre de divergences
k_max ← N        -- N est le nombre de variables
S_ref ← solution_initiale()
-- S_ref est la solution de référence
Tant que (k < k_max) faire
      k ← k+1
      -- Générer les feuilles à k divergences par rapport à S_ref
      S_ref' ← Calcul_feuilles(S_ref, k)
      Si Meilleur(S_ref', S_ref) alors
            -- Mise à jour de la solution de référence
            S_ref ← S_ref'
            k ← 0
      Finsi
Fin Tant que
```

Algorithme 8. Algorithme CDS pour un arbre non binaire

La méthode CDS peut être vue comme une méthode de recherche à voisinage variable (« Variable Neighbourhood Search » ou plus précisément Variable Neighbourhood Descent) [Hansen et Mladenovic, 2001]. En effet, les deux méthodes explorent les solutions dans le même ordre. L'intérêt de CDS est que la définition du voisinage d'une solution donnée repose sur le principe de divergence qui permet d'éviter la redondance lors de l'exploration de l'espace de recherche. De ce fait, CDS est vue comme une méthode générique [Milano et Roli, 2002] pouvant intégrer d'autres méthodes de recherche locale comme la recherche Tabou (TS) et le Recuit Simulé (SA).

3.10. Méthode à divergences limitées par apprentissage (YIELDS)

La méthode YIELDS (Yet Improved Limited Discrepancy Search) associe un poids *W*, initialement égal à zéro, à chaque variable [Karoui *et al.*, 2007]. Ce poids est incrémenté à chaque fois qu'un échec est rencontré sur la variable (domaine associé vide rendant impossible son instanciation). Dans les itérations suivantes, cette variable sera privilégiée et sera placée plus haut dans la branche développée par la méthode LDS.

```
YIELDS (X,D,C,k_max,Sol) :
k←0
Sol←NULL
Exceed←Faux
Tant que (Sol = NULL) et (k ≤ k_max) faire
```

```
      Sol ←YIELDS_itération(X,D,C,k,sol)
      k ← k+1
      si (non Exceed) alors
            Exit
      Finsi
Fin tant que
retourner (Sol)
```

```
YIELDS_itération (X,D,C,k,Sol) :
Si (X=∅) alors
      retourner (Sol)
Sinon
   x_i ← First_VariableOrdering (X,W)
   v_i ← First_ValueOrdering (D_i,k)
   si v_i ≠NULL alors
      D'←Update(X\{x_i},D,C,(x_i,v_i))
      I ←YIELDS_iteration(X\{x_i},D',C,k,Sol ∪{(x_i,v_i)})
      Si I≠NULL alors
         Retourner (I)
      Sinon
         si k>0 alors
            D_i←D_i\{v_i}
            Retourner YIELDS_iteration(X,D,C,k-1,Sol)
         Sinon
            W|x_i| ←W|x_i|+1
            Exceed ←Vrai //impossible de diverger
         Finsi
      Finsi
   sinon
      Retourner (NULL)
   Finsi
Finsi
```

Algorithme 9. Algorithme YIELDS pour un arbre binaire

L'intuition des auteurs était qu'en itérant LDS, on retrouve fréquemment la même variable à instancier en premier. Supposons que cette variable soit à l'origine de l'échec et que son instanciation élimine une valeur nécessaire à la solution, il est inutile de développer de nouveau sa branche. De cette manière, on peut éviter la situation d'inconsistance qu'elle peut causer. Le choix de la variable à instancier se base donc initialement sur l'heuristique d'ordre des variables et ensuite, sur le poids des variables. L'ajout du poids peut être considéré comme un apprentissage dynamique qui rectifie les erreurs éventuelles de l'heuristique. De cette manière, on exploite les précédents échecs et on en tire des informations pour les prochaines itérations. Pour accélérer la résolution, les sous-problèmes difficiles et insolubles sont propulsés vers le haut de l'arbre de recherche.

De plus, YIELDS stoppe les incrémentations des divergences quand elle est assurée que le problème est insoluble. En fait, quand une itération avec *k*

divergences autorisées mène vers un échec, il est possible que les itérations avec *k*+1 divergences autorisées ou plus soient inutiles s'ils mènent vers le même échec.

Le principe de YIELDS est exactement celui de LDS : la méthode considère initialement les branches de l'arbre qui cumulent le moins de divergences. La première différence est qu'un poids (initialement identique pour toutes les variables) est associé à chaque variable et à chaque fois qu'une variable échoue à cause de la borne sur le nombre de divergences, son poids est incrémenté pour pouvoir guider les futurs choix de l'heuristique. La seconde différence est que le nombre de divergences n'est plus incrémenté aveuglément jusqu'à atteindre le maximum de divergences possibles de l'arbre de recherche comme avec LDS. YIELDS consomme ainsi moins de divergences que LDS, ILDS ou DDS (voir algorithme 9 appliqué à un problème de type CSP défini par X : ensemble de variables, D : leurs domaines et C : les contraintes).

3.11. Exemple illustratif de ILDS, DDS, DDS tronquée et CDS

Les méthodes ILDS, DDS, DDS tronquée et CDS étant importantes pour la lecture de la suite de ce document, nous illustrons leurs stratégies d'exploration plus en détail.

Considérons un problème se composant de trois variables de décision x_1, x_2 et x_3. Nous donnons la priorité au nœud fils gauche et nous adoptons un mode de comptage binaire des divergences. Une instanciation complète est obtenue après l'instanciation des trois variables. Initialement, l'instanciation de référence S_{ref} est obtenue sans aucune divergence ; on notera par la suite le nombre de divergences pour une instanciation donnée de la manière suivante : $[x_1, x_2, x_3]$ = [0,0,0]. Les instanciations à une divergence par rapport à la solution s_{ref} sont celles ayant un des chiffres $[x_1, x_2, x_3]$ égal à 1 (ex. [0,0,1]). Pour représenter les divergences graphiquement, nous associons à une instanciation définie par l'heuristique un cercle noir et à une divergence un cercle creux. Ainsi, S_{ref} sera représenté par ●●● et une instanciation présentant une divergence au niveau de la troisième variable par ●●○.

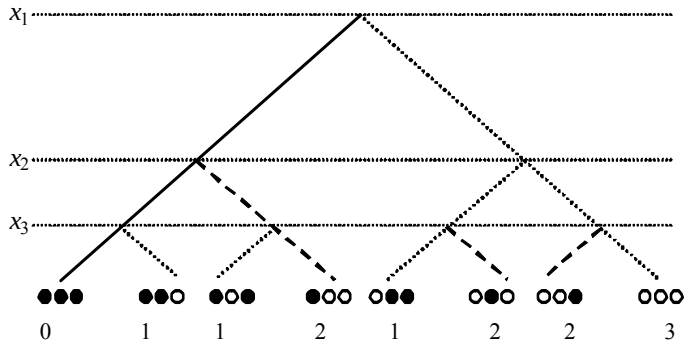
La figure 13 illustre les arbres de recherche obtenus en utilisant ILDS (a), DDS (b), DDS tronquée (c) et CDS (d). Pour les trois méthodes, la recherche commence à partir d'une instanciation de référence S_{ref} obtenue avec $[x_1, x_2, x_3]$ = [0,0,0] ou plutôt ●●●.

La figure 13(a) montre toutes les branches générées par la méthode LDS et qui correspondent aux instanciations obtenues à partir de ●●●. Les numéros en dessous des feuilles indiquent le numéro de l'itération de la méthode LDS ayant permis de découvrir cette instanciation.

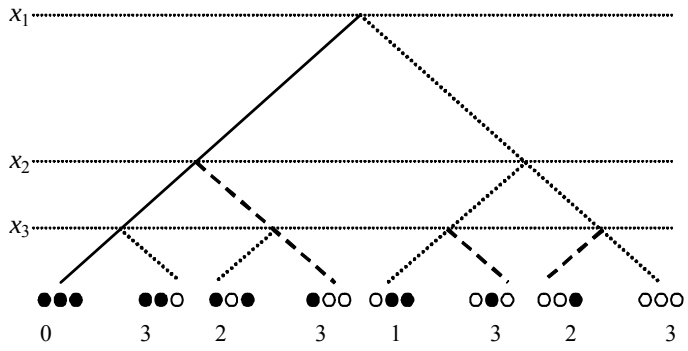
Dans la figure 13(b), les feuilles obtenues par la méthode DDS sont les mêmes que celles obtenues par la méthode LDS mais elles sont découvertes dans un ordre différent puisqu'à chaque itération de la méthode, la profondeur à laquelle on peut effectuer des divergences est limitée.

La figures 13(c) montre la différence entre la méthode DDS et la méthode DDS tronquée à la profondeur d=1 (seules les variables de profondeur inférieur ou égale à 1 sont considérées).

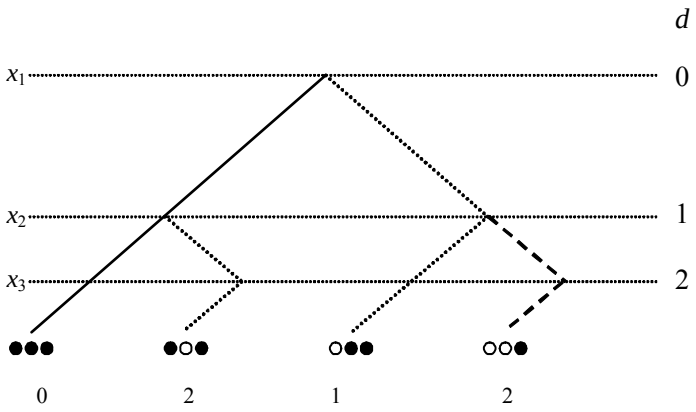
La figure 13(d) explique la stratégie de recherche de la méthode CDS et exprime l'ordre de génération des branches. Pour cette méthode d'optimisation, l'instanciation initiale S_{ref} est une solution et l'on peut évaluer la fonction objectif associée à cette solution. On note f_{ref} cette valeur de la fonction objectif. La première solution à 1 divergence de S_{ref} a une valeur f_1. Puisque f_1 est plus grande que f_{ref} , une deuxième solution d'une valeur f_2 est générée. Son coût est comparé par rapport à f_{ref} et ce processus est répété jusqu'à ce qu'une solution d'une valeur $f_4 \leq f_{ref}$ soit obtenue. Ainsi, cette solution devient la nouvelle solution de référence, f_4 la nouvelle valeur de la fonction objectif et le nombre de divergences est remis à zéro. Par conséquent, la prochaine solution (ayant la valeur f_5) est seulement à une divergence de cette nouvelle solution de référence.



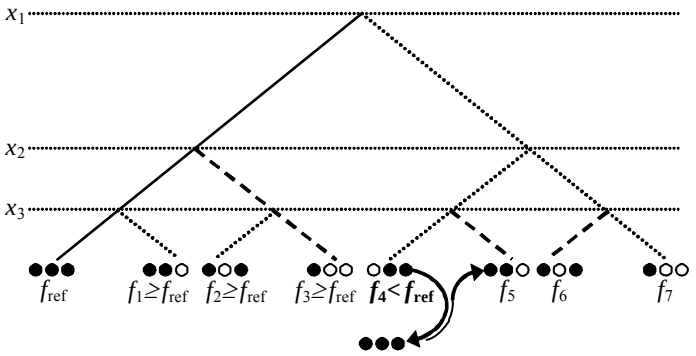
a) LDS



b) DDS



c) DDS tronquée (d=1)



d) CDS

Fig.13. Principe d'exploration d'une arborescence binaire par les méthodes LDS, DDS, DDS-tronquée et CDS

3.12. Conclusion

Ce chapitre a présenté en détail différentes méthodes à base de divergences issues de la littérature. Les caractéristiques des méthodes arborescentes à base de divergences ont été mises en évidence. Elles permettent de sauvegarder l'ordre de parcours des branches explorées, ce qui permet d'orienter l'exploration de l'espace de recherche et évite d'autant plus les redondances. Elles donnent également la possibilité de contrôler l'exploration en limitant les divergences par un seuil donné ou en explorant exhaustivement tout l'espace. De plus, il est facile d'intégrer au sein de ces méthodes à base de divergences des mécanismes de propagation de contraintes afin d'éviter au mieux d'explorer des situations liées à des infaisabilités.

Cette étude nous a permis de concevoir une nouvelle méthode à base de divergences que nous présentons dans le chapitre suivant. Cette dernière n'a pas été conçue pour un seul type de problème d'ordonnancement mais a été adaptée aux différents problèmes d'ordonnancement considérés dans

ce travail. Les deux chapitres suivants exposent le principe de cette nouvelle méthode nommée Climbing Depth-bounded Discrepancy Search (CDDS).

Chapitre 4. Adaptation des méthodes à base de divergences pour le Flow Shop Hybride

Cette partie est consacrée à l'adaptation des méthodes à divergences limitées pour la résolution du Flow-Shop Hybride. Dans un premier temps nous préciserons les caractéristiques des problèmes de Flow-Shop Hybride considérés. Nous présentons une nouvelle méthode à base de divergences nommée Climbing Depth-bounded Discrepancy Search (CDDS) inspirée de différentes autres méthodes reposant sur le même principe de divergences, à savoir : LDS, DDS, CDS. Nous proposons ensuite des adaptations de la méthode CDDS pour la résolution du Flow Shop Hybride dans le cas général et du cas particulier du problème de flow shop hybride à deux étages. Des résultats expérimentaux montrent les atouts et les faiblesses respectifs de chacune des méthodes pour la résolution du Flow-Shop Hybride dans ces deux cas.

Une partie des résultats proposés dans ce chapitre a été présentée dans "Climbing Discrepancy Search for solving the hybrid Flow shop", European Journal of Industrial Engineering, vol.1, N°2, pp.223-243, Juillet 2007.

4.1. Introduction

Les problèmes de Flow Shop Hybride que nous considérons sont soumis aux hypothèses suivantes :

- Les opérations sont non interruptibles. Une fois que l'exécution d'une opération a débuté sur une machine, celle-ci ne peut pas être interrompue. Ainsi, aucune opération ne peut commencer sur cette machine avant la fin de l'opération en cours. De plus, une opération ne peut être changée de machine. Ceci revient à dire que nous cherchons des ordonnancements non préemptifs.
- Les machines sont disjonctives. Elles ne peuvent exécuter qu'une opération à la fois. Ces machines sont disponibles sans restriction du début à la fin de l'ordonnancement. Les machines sont parallèles et identiques, c'est-à-dire que les opérations peuvent être exécutées indifféremment sur l'une ou sur l'autre des machines d'un même étage.
- Les durées des opérations sont entières. Si l'une des durées est rationnelle, il est assez simple de se ramener à une durée entière en multipliant toutes les durées par une constante.
- Nous n'imposons aucune limite sur le nombre d'opérations qui peuvent attendre entre deux étages, les buffers placés à ces endroits sont considérés comme infinis.
- Les temps de transport et/ou de maintenance sont considérés comme nuls, ou éventuellement dans certains cas, ils sont contenus dans la durée d'exécution des tâches.
- Nous n'imposons aucune borne supérieure pour la valeur de date de fin d'exécution totale.

Résoudre un problème d'ordonnancement de type flow shop hybride revient à fixer pour chaque opération de chaque job une date de début et une ressource pouvant l'exécuter. Les dates de début doivent satisfaire à la fois les contraintes de précédence et les contraintes de ressources. Parmi ces ordonnancements réalisables, nous cherchons un ordonnancement faisable minimisant la durée d'exécution totale ou makespan.

4.2. Une nouvelle méthode arborescente à base de divergences et son application au Flow Shop Hybride

4.2.1. Méthode proposée : Climbing Depth-bounded Discrepancy Search (CDDS)

La méthode que nous proposons d'utiliser s'appuie à la fois sur les caractéristiques de la méthode CDS (Climbing Discrepancy Search) afin

d'explorer le voisinage d'une solution de référence et sur les caractéristiques de la méthode DDS tronquée afin de limiter l'exploration des voisinages par la profondeur des divergences.

Nous obtenons alors une nouvelle méthode arborescente basée sur le principe de divergence et dédiée à la résolution de problème d'optimisation comme CDS. Cette nouvelle méthode est appelée ***CDDS*** (*Climbing Depth-bounded Discrepancy Search*) (voir algorithme 10).

```
k ← 0    -- k est le nombre de divergences
kmax ← N -- N est le nombre de variables
Sref ← Solution_Initiale() -- Sref est la solution de référence
Tant que (k < kmax) faire
 k ← k+1
 -- Générer les feuilles à k divergences de Sref
 -- et à une profondeur d de l'arbre avec 1 ≤ d ≤ k
 Sref'← Calcul_feuilles(Sref, k)
 Si Meilleur(Sref', Sref) alors
      -- Mise à jour de la solution courante
      Sref ← Sref'
      k ← 0
 fin si
fin tant que
```

Algorithme 10. Climbing Depth-bounded Discrepancy Search (CDDS)

Ainsi, à partir d'heuristiques fournissant une solution initiale de plus ou moins bonne qualité nous allons explorer son voisinage en utilisant le principe des divergences. Lorsque dans le premier voisinage correspondant à 1 divergence de la solution de référence il n'y a pas de solution de meilleure qualité, CDDS incrémente le nombre de divergences et le processus est ré-itéré. En revanche, dès que dans un voisinage on trouve une solution de meilleure qualité le nombre de divergences est ré-initialisé à 1 et la solution de référence est mise à jour. L'exploration des voisinages redémarre ensuite autour de cette nouvelle solution. Afin de limiter la taille d'un voisinage donné, la méthode CDDS applique le principe de la méthode DDS tronquée en limitant la profondeur des divergences.

Autour de ce principe de CDDS, avec comme objectif de maîtriser la taille des voisinages à explorer, nous proposons de greffer différents mécanismes.

Des calculs de bornes peuvent être introduits à chaque nœud développé afin d'élaguer l'arborescence. La méthode CDDS avec calcul de bornes ne diffère de l'algorithme qu'au niveau de la procédure *Calcul_Feuilles* qui intègre une vérification sur les bornes au niveau de chaque nœud. Pour la résolution d'un problème de minimisation, cette procédure devient *Calcul_Feuilles_Avec_Bornes(S_{ref}, k, UB)*. Elle consiste à chaque nœud développé à évaluer une borne inférieure *LB* et à la comparer avec la valeur d'une borne supérieure *UB* fournie en paramètre de la procédure. Lorsqu'à un nœud donné la valeur de LB est supérieure ou égale à la valeur de *UB*, son développement est stoppé. La variante de CDDS incluant des calculs de

bornes est appelée par la suite CDDS-LB pour un problème de minimisation (respectivement CDDS-UB pour un problème de maximisation).

De plus, nous pouvons intégrer des heuristiques pour guider l'application des divergences. A l'aide de ces heuristiques, les divergences ne seront plus appliquées de manière systématique à l'ensemble des variables du problème mais à certaines variables jugées plus pertinentes car plus susceptibles de conduire à des améliorations du critère que l'on cherche à optimiser. Cette variante de CDDS est appelée CDDS-HDiv par la suite.

Les conditions d'arrêt d'une méthode CDDS peuvent être par exemple un temps CPU, un nombre d'itérations sans amélioration de la solution (ou l'obtention de la solution optimale).

Nous allons voir dans la suite de cette partie, comment appliquer cette méthode CDDS à la résolution du problème de Flow Shop Hybride en définissant les variables de décision de ce problème, une heuristique d'obtention d'une solution initiale, la notion de divergence et une stratégie d'exploration de l'arborescence (c'est-à-dire d'exploration des voisinages).

4.2.2. Variables de décision du problème de Flow Shop Hybride

Afin de résoudre le problème considéré, à chaque étage, il faut sélectionner tout d'abord un job à exécuter, lui allouer une ressource et fixer ensuite sa date de début. On considèrera ainsi deux types de variables : les variables de sélection du job et les variables de sélection de la ressource.

À chaque étage E_i, nous notons X_j le vecteur des variables de sélection des jobs et A_j le vecteur des variables d'allocation des ressources. Ainsi, X_j^i correspond à la sélection du *j*ème job à l'étage i et A_j^i à la ressource qui lui est allouée ($\forall j = 1,...,n$, avec n *le* nombre total de jobs).

À chaque étage E_i, le domaine de la variable X_j^i est {1, 2,…, n}, $\forall j = 1,...,n$; il correspond au choix du job à ordonnancer. Les valeurs prises par les variables X_j^i doivent être toutes différentes. À un étage donné E_i, le domaine des variables A_j^i est {1,…, m_i}, $\forall j = 1,...,n$ et $\forall i = 1,...,k$.

4.2.3. Heuristiques sur l'ordre d'instanciation des variables

Nos variables de sélection de jobs et de ressources seront considérées étage par étage. Pour chaque étage, notre stratégie d'exploration consiste à instancier tout d'abord la variable de sélection de job, puis la variable d'allocation de ressources. Pour ce faire, deux types d'heuristiques d'instanciation des variables sont considérés.

Type 1. Sélection du job. Un choix judicieux de l'ordre d'examen des jobs s'avère d'un très grand intérêt pour l'obtention d'une solution de bonne

qualité. Cette heuristique vise à choisir l'ordre dans lequel les différents jobs vont passer sur un étage donné. Ainsi, à un étage donné, la priorité est donnée au job correspondant à l'opération ayant la plus petite date de début au plus tôt (*EST* : Earliest Start Time). En cas d'égalité entre jobs, nous proposons plusieurs alternatives pour sélectionner le job correspondant à l'opération :

- ayant la plus petite durée d'exécution (*SPT* : Shortest Processing Time) sur l'étage considéré ;
- ayant la plus longue durée d'exécution (*LPT* : Longest Processing Time) sur l'étage considéré ;
- appartenant au job ayant la plus grande durée opératoire (*LJD* : Largest Job Duration) sur tous les étages.

On peut considérer aussi différentes applications de ces quatre règles de priorité sur les différents étages. Ainsi, on peut appliquer au premier étage *SPT* (resp. *LPT / LJD*) et au deuxième étage *LPT* ou *LJD* (resp. *SPT ou LJD / SPT* ou *LPT)* et ainsi de suite. L'idée derrière ces différentes combinaisons est de s'adapter aux diverses configurations des machines dans les différents étages.

Type 2. Allocation des ressources aux jobs. Le job choisi par l'heuristique de Type 1 sera, à la suite, affecté à la ressource offrant au job la plus petite date de fin d'exécution (*ECT* : Earliest Completion Time).

Après toute instanciation des variables de sélection de job et d'allocation des ressources, on applique une technique de propagation de contraintes de type *Forward Checking* [Haralick et Elliot, 1980] afin de mettre à jour la date de début au plus tôt de l'opération successeur pour le job sélectionné et d'actualiser la date de disponibilité de la ressource choisie.

La branche initiale, obtenue par application des différentes heuristiques présentées ci-dessus, est une solution initiale à notre problème de Flow Shop Hybride car la valeur du makespan n'est pas contrainte.

4.2.4. Notion de divergence pour le Flow Shop Hybride

Bien qu'il y ait deux types de variables, nous considérons que les divergences ne peuvent être appliquées que sur les variables de sélection de job. En effet, notre but est de minimiser la durée d'exécution totale, et comme toutes les ressources d'un étage donné sont identiques, les divergences sur les variables de sélection de ressources ne s'avèrent pas d'un grand intérêt par rapport à notre critère.

Par conséquent, faire une divergence consiste à sélectionner un job autre que celui recommandé par l'heuristique. Les variables de sélection des jobs sont *N*-aires. Les divergences sont alors calculées comme suit : la première

valeur choisie par l'heuristique correspond à 0 divergence, toutes les autres valeurs correspondent à 1 divergence (voir figure 14).

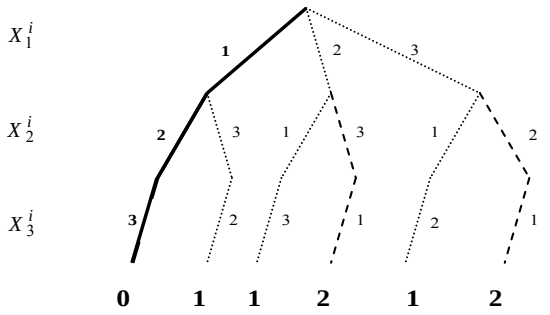


Fig.14. Divergence au niveau de la variable de sélection du job (étage E_i)

Pour obtenir les solutions à ***k+1*** divergences à partir d'une solution à ***k*** divergences directement sans revisiter les solutions de ***0*** à (***k-1***) divergences (principe de ILDS), on utilise un compteur de divergences pour chaque variable. Puis, à partir de chaque solution à ***k*** divergences, on considère la dernière variable ayant une divergence et on applique des divergences sur les variables suivantes. On obtient ainsi les solutions ayant ***k+1*** divergences. Cette stratégie nous permet de générer les solutions sans redondance étant donné que les choix déjà effectués au niveau d'un nœud donné seront inhibés lors du choix suivant.

En considérant l'exemple donné dans la figure 14, on voit que si l'heuristique favorise le choix du job 1, puis le job 2 et ensuite le job 3 (S_{ref}={1,2,3}), les solutions à 1-divergence sont données par les triplets {2,1,3} ; {3,1,2} et {1,3,2} où la valeur soulignée représente la divergence effectuée. Une solution à 2-divergences revient à faire une seule divergence par rapport aux solutions à 1-divergence sans retourner aux solutions déjà générées ; i.e. à partir de la solution {2,1,3} ont obtient la solution {2,3,1} ; à partir de la solution {3,1,2} on obtient la solution {3,2,1} et à partir de la solution {1,3,2}, il n'y a pas de solution à 2 divergences.

4.2.5. Stratégies d'exploration

Pour le problème de Flow Shop Hybride, afin de limiter l'exploration de l'arbre de recherche, il est possible d'intégrer un calcul de bornes inférieures au sein de la méthode CDDS.

Pour cela, nous proposons d'utiliser les bornes inférieures mentionnées dans [Brah and Hunsuker, 1991 ; Portmann ***et al.***, 1998] pour évaluer chaque nœud visité. Ces bornes inférieures ont été présentées au chapitre 2 (voir §2.1.2) et leur application est illustrée dans la partie ci-dessous.

4.2.6. Exemple illustratif

Nous présentons ici un exemple permettant d'illustrer le fonctionnement de cette nouvelle méthode CDDS.

Considérons un HFS de dimension 4×2 (4 jobs et 2 étages). Le premier étage est composé d'une seule machine alors que le second comporte 2 machines. La profondeur autorisée (*d*) est fixée à 2. Les données de cet exemple sont fournies dans le tableau 3.

La figure 16 présente la solution initiale (à 0 divergence) produite par la règle *EST-SPT* appliquée au premier étage.

Tableau 3. *Les durées opératoires d'un problème de dimension 4 × 2*

Jobs	***Etage 1***		***Etage 2***	
1	O_{11}	8	O_{12}	7
2	O_{21}	7	O_{22}	8
3	O_{31}	8	O_{32}	8
4	O_{41}	7	O_{42}	8

Cette dernière donne la séquence suivante : J_2,J_4,J_1,J_3 (l'ordre lexicographique est appliqué en cas d'égalité). La valeur du makespan initial est égale à 38. Celle-ci sera considérée comme une valeur initiale de la borne supérieure associée à ce problème ; UB=38.

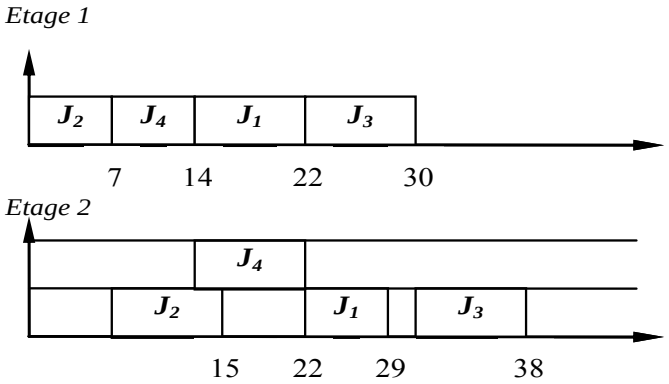


Fig.15. *La solution initiale*

Le premier voisinage de cette première solution est composé des solutions à 1 divergence. Celles-ci sont schématisées sur la figure 16 et correspondent aux solutions suivantes :

1. Au premier niveau (*d=1*):
   - $\boldsymbol{J_4}, J_2, J_1, J_3$
   - $\boldsymbol{J_1}, J_2, J_4, J_3$
   - $\boldsymbol{J_3}, J_2, J_4, J_1$
2. Au second niveau (*d=2*):
   - $J_2, \boldsymbol{J_1}, J_4, J_3$

- J_2, $\boldsymbol{J_3}$, J_4, J_1

Toutes les variables d'allocation de ressources prennent la même valeur, $A_i^1 = M_1$, à l'étage 1, étant donné qu'on a une seule machine pour cet étage (figure 16).

Pour chaque sous-séquence de jobs, l'itération suivante de notre algorithme permet d'ordonnancer tous les jobs et de calculer au niveau de chaque nœud la valeur de la borne inférieure *LB*. Nous utilisons la borne inférieure composée *LBC* décrite au chapitre 2 (voir §2.1.2).

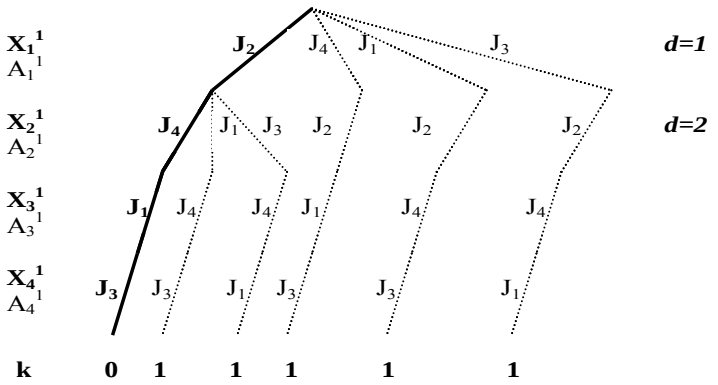


Fig.16. Le voisinage de la solution initiale

Nous commençons par la première séquence {J_4,J_2,J_1,J_3} et nous calculons la borne inférieure à chaque nœud. Considérons par exemple le sous-ensemble Y={J_4,J_2,J_1} ; on aura comme valeur de borne inférieure :

$$ACT[Sch^{(s)}(Y)] = \frac{22+8}{1} = 30$$

et

$$MCT[S^{(1)}(Y)] = 22.$$

Alors,

$$LBM\ [Sch^{(s)}(Y)] = 30 + 8 = 38$$

et

$$LBJ[Sch^{(s)}(Y)] = 22 + 16 = 38.$$

Par conséquent,

$$LBC[Sch^{(s)}(Y)] = \max(38\ ;\ 38) = 38.$$

Cette valeur de borne inférieure est égale à la valeur de la borne supérieure initiale. L'exploration de cette branche de l'arborescence est inutile et on peut alors l'élaguer (voir figure 17). La même stratégie est appliquée pour la seconde séquence {J_1,J_2,J_4,J_3} et nous obtenons au premier nœud LB=38. Donc on arrête aussi l'exploration de cette branche.

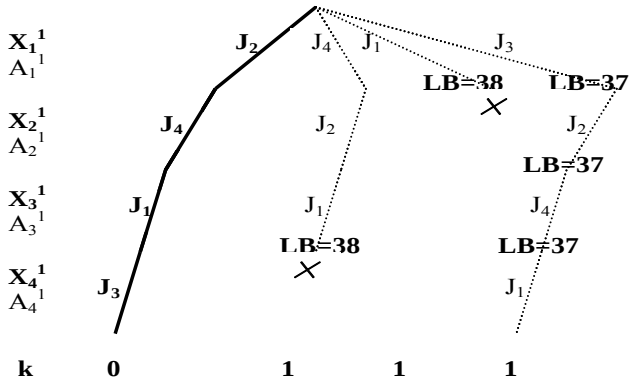


Fig.17. L'exploration du voisinage

La troisième séquence {J_3,J_2,J_4,J_1} donne un ordonnancement de C_{max} = 37 (figure 18). Cette dernière solution sera considérée comme une nouvelle solution de référence (car elle améliore la solution courante) et le nombre de divergences est réinitialisé à 0. Donc, on arrête l'exploration du voisinage de la séquence initiale {J_2,J_4,J_1,J_3} et nous définissons le voisinage de la nouvelle solution. Cette dernière est définie dans la figure 18. La borne supérieure est mise à jour et devient égale à 37 ; UB=37.

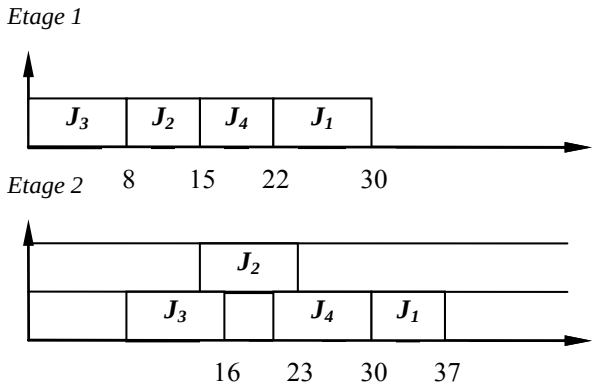


Fig.18. L'ordonnancement de la troisième séquence

Le voisinage à une seule divergence de la nouvelle solution de référence est composé des séquences suivantes :

1. Au premier niveau (*d=1*):
   - $\boldsymbol{J_2},J_3,J_4,J_1$
   - $\boldsymbol{J_4},J_3,J_2,J_1$
   - $\boldsymbol{J_1},J_3,J_2,J_4$
2. Au second niveau (*d=2*):
   - $J_3,\boldsymbol{J_4},J_2,J_1$
   - $J_3,\boldsymbol{J_1},J_2,J_4$

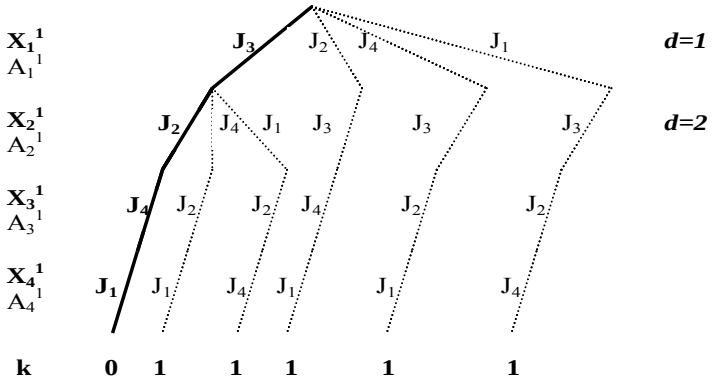


Fig.19. Le voisinage de la nouvelle solution de référence

Le calcul des bornes inférieures au niveau de chaque nœud guidera la recherche vers la région prometteuse de l'arborescence (voir figure 20). Le processus de la recherche est stoppé quand il n'y aura plus de branche à explorer. La meilleure solution trouvée est présentée dans la figure 18.

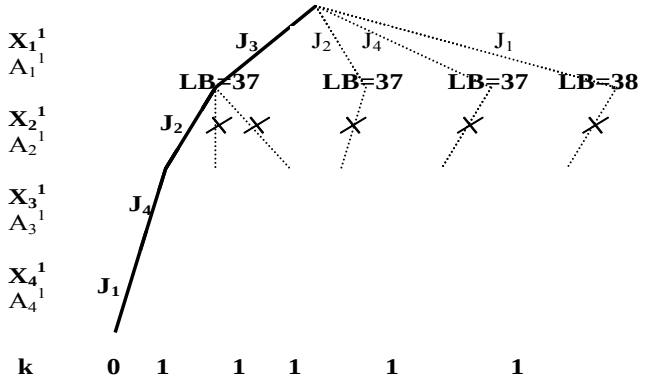


Fig.20. L'exploration du voisinage

4.3. Expérimentations

Toutes les expérimentations ont été réalisées sur un PC Intel Core 2 Duo 2.9 GHz avec 2GB de mémoire vive sur lequel les différents algorithmes ont été implémentés en C.

Nous considérons deux jeux-tests, à savoir les problèmes de [Vignier, 1997] et ceux présentés dans [Carlier et Néron, 2000 ; Néron et Baptiste, 2001] :

- Vdata : Cette classe est constituée de 67 problèmes développés dans [Vignier, 1997]. Elle présente quatre types de configuration :

- Configuration 1 : premier étage à une seule machine (nommé étage critique ou goulet) et avant dernier à 2 machines s'il y a 5 étages ;
- Configuration 2 : dernier étage critique (et le deuxième étage est critique dans le cas d'un problème à 5 étages) ;
- Configuration 3 : pas d'étage critique (trois machines partout) ;
- Configuration 4 : étage du milieu critique.

Chaque configuration contient 6 instances. Le nombre de jobs varie entre 5 et 15 et le nombre de machines par étage varie de 1 à 3.

- Ndata : Cette classe est constituée de 76 problèmes présentés dans [Carlier et Néron, 2000 ; Néron et Baptiste, 2001]. Les tailles de problèmes varient de 10 à 15 jobs, et de 5 à 10 étages, les configurations machines étant plus ou moins équilibrées. Les durées sont tirées aléatoirement dans le même intervalle quel que soit le nombre de machines de l'étage, ce qui implique que la criticité de l'étage dépend directement du nombre de machines dont il dispose pour exécuter les tâches. Cette classe d'instances présente quatre types de configuration :
  - Configuration 1 : étage du milieu critique
  - Configuration 2 : premier étage critique
  - Configuration 3 : étage du milieu à deux machines
  - Configuration 4 : pas d'étage critique (trois machines partout)

4.3.1. Comparaison sur les jeux-tests Vdata

Les deux premiers tableaux que nous présentons comparent l'efficacité de notre nouvelle méthode à base de divergences CDDS-LB avec l'algorithme génétique (AG) proposé dans [Vignier, 1997] et avec la méthode de B&B développée dans [Néron et Baptiste, 2001]. La première colonne indique la configuration (le nombre de machines par étage) et la taille des instances. La deuxième colonne indique le nombre d'instances pour chaque type de problèmes. Les colonnes suivantes présentent le nombre de problèmes non résolus de manière optimale (*Nb_US*), le nombre moyen de nœuds explorés (*nœuds*) et le temps CPU total nécessaire (*CPU*) pour résoudre toutes les instances du type correspondant (en secondes). Le temps maximum alloué à la recherche est de 15 secondes. Cette dernière valeur a été choisie en se basant sur une campagne d'expérimentations et en constatant qu'au-delà de 15 s les résultats ne s'améliorent que très peu.

Nous insistons sur le fait que les temps de calcul ne peuvent être comparés en raison de la disparité entre les machines utilisées (486DX33 pour Vignier et SPARC Enterprise 4000 (Solaris) pour Néron et Carlier) et de la non existence des coefficients de normalisation de Dongarra [Dongarra, 1998] pour ces deux types de machines utilisées.

Tableau 4. *Comparaison des méthodes pour la configuration 13323*

Configuration 13323		AG	B&B		CDDS-LB		
[Jobs, étages]	Nb	Nb_US	Nb_US	Nœuds	Nb_US	Nœuds	CPU
[5, 5]	6	0	0	32	0	61	0.06
[10, 5]	6	4	4	4649	0	10766	1.44
[15, 5]	6	4	4	110	0	4607	60.00

Le tableau 4 montre que, bien que la méthode CDDS-LB explore un nombre important de nœuds, cette dernière surpasse les deux autres méthodes (AG et B&B) au niveau de toutes les instances en termes de qualité de solutions fournies. En effet, elle permet de résoudre 100% des instances considérées de manière optimale en un total de temps CPU très réduit (≈ 1 mn pour les 18 instances) et ce face à 44% des instances pour les deux autres méthodes.

Tableau 5. *Comparaison des méthodes pour la configuration 33231*

Configuration 33231		AG	B&B		CDDS-LB		
[Jobs, étages]	Nb	Nb_US	Nb_US	Nœuds	Nb_US	Nœuds	CPU
[5, 5]	6	6	0	30	0	83	0.06
[10, 5]	6	6	0	11	0	2495	0.30
[15,5]	6	6	1	23	0	60320	72.00

Les conclusions précédentes sont confirmées par les résultats du tableau 5. Nous remarquons que, sur ce type d'instances, notre méthode se montre extrêmement efficace.

La grande majorité des problèmes proposés dans les jeux-tests élaborés par Vignier sont extrêmement faciles à résoudre par CDDS-LB et par les deux autres méthodes (AG et B&B). C'est pour cette raison que nous n'avons pas reporté les résultats sur le reste des instances qui peuvent être résolues optimalement dès la première solution.

Par ailleurs, parmi ceux qui ne sont pas résolus très rapidement, certains semblent plus difficiles (reportés dans les tableaux 4 et 5). En effet, ces problèmes ont une structure très proche de problèmes de 3-partition (trois machines par étage, et des dates de disponibilité et durées de latence quasi-identiques).

D'après ces deux derniers tableaux, on remarque que CDDS-LB arrive à résoudre les deux types d'instances assez rapidement pour lesquelles les deux

autres méthodes semblent impuissantes (sur les instances reportées dans les deux tableaux), quelles que soient les améliorations apportées.

4.3.2. Comparaison sur les jeux-tests Ndata

Dans ce paragraphe, nous comparons, en première étape, les méthodes DDS-tronquée, CDDS et CDDS-LB entre elles. Le temps maximum alloué à la recherche est de 120 secondes. Toutes ces méthodes seront comparées par rapport aux bornes inférieures (LB) développées dans [Néron et Baptiste, 2001].

Dans les tableaux 7 à 10, la première colonne indique le nombre de jobs et le nombre d'étages ainsi que chaque configuration considérée. La deuxième colonne donne le nombre d'instances par configuration. Dans les colonnes qui suivent, on donne pour chaque méthode, le nombre de problèmes non résolus optimalement (*Nb_US*) ainsi que la déviation moyenne par rapport aux bornes inférieures (*dév_moy*). La ligne *#meilleur* récapitule le nombre de fois où chaque méthode a trouvé la solution optimale (C_{max}=*LB*). La ligne *CPU total* rapporte le temps d'exécution total (en secondes) nécessaire pour résoudre les différentes instances d'une configuration donnée à travers trois exécutions.

La déviation de chaque méthode par rapport aux bornes inférieures est utilisée comme critère de comparaison. Cette dernière est calculée comme suit :

$$dév\_moy = \frac{(C_{\max} - LB)}{LB} \times 100\% .$$

Notons que toutes les heuristiques de sélection de job ont été implémentées et testées sur différentes instances. En se reposant sur les résultats fournis par ces dernières (voir tableau 6) au sein de l'algorithme CDDS-LB, seules les heuristiques (*LJD*, *LPT-SPT* et *SPT-LPT*) sont retenues pour le reste des expérimentations. Le pourcentage de déviation par rapport aux bornes inférieures est utilisé comme critère d'évaluation de l'efficacité de chaque heuristique.

Tableau 6. Efficacité des heuristiques de sélection des jobs pour la méthode CDDS-LB

Heuristiques	SPT	LPT	LJD	SPT-LPT	SPT-LJD	LPT-SPT	LPT-LJD	LJD-SPT	LJD-LPT
Dév_moy	4.00	4.93	2.30	2.85	3.23	2.85	3.10	4.00	3.01

Dans les tableaux suivants, nous récapitulons pour chaque instance, la meilleure solution obtenue parmi les trois heuristiques (*LJD*, *LPT-SPT* et *SPT-LPT*), la moyenne de C_{max} notée en dessous entre parenthèses. Les trois dernières

donnent pour l'ensemble des instances la valeur de la déviation moyenne et sa moyenne (notée entre parenthèses), le nombre de fois où chaque méthode a trouvé la meilleure solution et le temps CPU total mis pour la résolution de l'ensemble des instances.

Tableau 7. *Comparaison des trois méthodes pour les problèmes de 10 jobs et 5 étages*

[10-5]	Nb	DDS-tronquée		CDDS		CDDS-LB	
		Nb_US	Dév_moy	Nb_US	Dév_moy	Nb_US	Dév_moy
33133	**5**	0	0.00 (0.45)	0	0.00 (0.45)	0	0.00 (0.45)
13333	**6**	0	0.00 (0.18)	0	0.00 (0.18)	0	0.00 (0.18)
33233	**6**	6	2.62 (3.27)	4	0.95 (1.99)	0	0.00 (1.28)
33333	**6**	6	2.49 (2.67)	3	1.00 (2.09)	0	0.00 (1.25)
Dév_moy		1.28 (1.64)		0.49 (1.18)		0.00 (0.79)	
#meilleur		12		17		24	
CPU total		4860		4336		3240	

Tableau 8. *Comparaison des trois méthodes pour les problèmes de 15 jobs et 5 étages*

[15-5]	Nb	DDS-tronqué		CDDS		CDDS-LB	
		Nb_US	Dév_moy	Nb_US	Dév_moy	Nb_US	Dév_moy
33133	**6**	2	0.35 (0.63)	0	0.00 (0.39)	0	0.00 (0.35)
13333	**6**	1	0.23 (0.41)	0	0.00 (0.25)	0	0.00 (0.22)
33233	**6**	6	7.43 (8.25)	5	3.60 (4.89)	2	0.42 (2.32)
33333	**6**	6	19.97 (20.49)	5	13.93 (14.66)	5	11.86 (13.58)
Dév_moy		6.99 (7.44)		4.38 (5.05)		3.07 (4.12)	
#meilleur		9		14		17	
CPU total		6076		5016		4084	

D'après les tableaux 7 et 8, on remarque que pour ces types de problèmes de taille raisonnable, les trois méthodes arrivent à résoudre la majorité des instances quand l'un des étages est critique. La position de l'étage critique n'influe pas l'efficacité des trois méthodes.

Les meilleurs résultats sont fournis par la méthode CDDS-LB ce qui montre l'intérêt d'intégrer des bornes inférieures au sein de la méthode de résolution. En effet, CDDS-LB arrive à résoudre 71% des problèmes face à 59% pour CDDS

et 38% pour DDS-tronquée. En plus, elle présente une déviation moyenne par rapport aux bornes inférieures de 3.07% face à 4.38% pour CDDS et 6.99% pour DDS-tronquée. L'amélioration de CDDS-LB due à l'intégration des bornes inférieures est considérable, en termes de qualité des solutions et du temps d'exécution total.

Par la suite, nous comparons ces trois méthodes sur les problèmes équilibrés et quasi-équilibrés (voir tableaux 9 et 10).

Tableau 9. *Comparaison des trois méthodes pour les problèmes de 10 jobs et 10 étages*

[10-10]	Nb	DDS		CDDS		CDDS-LB	
		Nb_US	Dév_moy	Nb_US	Dév_moy	Nb_US	Dév_moy
3333133333	**5**	3	0.68 (0.95)	0	0.00 (0.50)	0	0.00 (0.45)
1333333333	**6**	1	0.11 (0.11)	0	0.00 (0.07)	0	0.00 (0.04)
3333233333	**6**	6	10.48 (11.17)	6	9.54 (10.24)	5	8.34 (9.55)
Dév_moy		3.75 (4.07)		3.18 (3.60)		2.78 (3.35)	
#meilleur		7		11		16	
CPU total		4844		3781		2595	

Tableau 10. *Comparaison des trois méthodes pour les problèmes de 15 jobs et 10 étages*

[15-10]	Nb	DDS		CDDS		CDDS-LB	
		Nb_US	Dév_moy	Nb_US	Dév_moy	Nb_US	Dév_moy
3333133333	**6**	2	0.33 (0.33)	0	0.00 (0.17)	0	0.00 (0.17)
1333333333	**6**	0	0.00 (0.00)	0	0.00 (0.00)	0	0.00 (0.00)
Dév_moy		0.17 (0.17)		0.00 (0.08)		0.00 (0.08)	
#meilleur		10		12		12	
CPU total		1472		903		446	

Ces derniers tableaux confirment les conclusions précédentes. En effet, les trois méthodes prouvent leur efficacité pour la résolution des instances ayant des étages critiques. Cependant, l'apport de CDDS-LB pour les problèmes ayant 15 jobs et 10 étages ne semble pas intéressant en termes de qualité de solutions.

On remarque que le temps de résolution reste raisonnable pour ces problèmes de taille assez importante (100 et 150 opérations) qui semblent faciles à résoudre pour les trois méthodes.

Bien que ces derniers problèmes soient de taille plus importante que les précédents (voir tableaux 7 et 8), ceux-ci sont résolus plus rapidement par les différentes méthodes. Ceci peut être dû à des raisons arithmétiques vu la complexité que présentent les instances de configurations 3 et 4. En effet, ces problèmes ont une structure très proche de problèmes de 3-partition (trois machines par étage, et des dates de disponibilité et durées de latence quasi-identiques). Nous notons ces problèmes comme étant difficiles.

Notons également qu'en termes d'amélioration de la valeur initiale du makespan, les trois méthodes prouvent leur contribution (voir tableau 11). En effet, à moins de 120 secondes par instance, la valeur du makespan initiale a été réduite avec DDS-tronquée de 14.7% pour les problèmes difficiles et de 9.7% pour les faciles. Si nous considérons tous les problèmes, le makespan initial a été réduit avec CDDS de 10.6% (8.0% pour les problèmes faciles et 17.9% pour les difficiles). Pour CDDS-LB, la réduction du makespan initial est au sujet de 12%. Ce pourcentage est distribué en 20.2% pour les problèmes difficiles et 8.3% pour les faciles.

Tableau 11. *Efficacité des trois méthodes en termes d'amélioration de la solution initiale*

Méthodes	**problèmes faciles**	**problèmes difficiles**
	amélioration	***amélioration***
DDS-tronquée	9.7 %	14.7 %
CDDS	8.0 %	17.9 %
CDDS-LB	8.3 %	20.2 %

Dans la suite, on se propose de comparer CDDS-LB avec la procédure de Branch and Bound (B&B) développée dans [Néron et Baptiste, 2001] et l'heuristique basée sur les systèmes immunitaires artificiels (AIS) développée dans [Engin et Döyen, 2004] qui, à notre connaissance, fournit les meilleurs résultats pour ce type de problème.

Dans le tableau 12, on présente les meilleures valeurs de makespan données par la méthode CDDS-LB parmi les trois heuristiques de classement de job considérées, les valeurs obtenues pour le B&B ainsi que les valeurs obtenues par la méthode AIS. Leurs déviations par rapport aux bornes inférieures sont données dans les quatre dernières colonnes. Les problèmes en gras sont identifiés comme problèmes difficiles.

Nous pouvons constater que les méthodes CDDS-LB et AIS donnent des solutions optimales pour 61 instances parmi 76. En outre, pour les 15 instances restantes, CDDS-LB surpasse AIS sur 3 instances, alors qu'AIS surpasse CDDS-LB pour seulement une instance.

Tableau 12. *Solutions des problèmes et comparaison de CDDS-LB avec les méthodes B&B et AIS*

Configuration machines	Problème	Cmax			LB de Cmax	% déviation des LBs		
		B&B	CDDS-LB	AIS		B&B	CDDS-LB	AIS
33133	J10c5a2	88	88	88	88	0.0	0.0	0.0
	J10c5a3	117	117	117	117	0.0	0.0	0.0
	J10c5a4	121	121	121	121	0.0	0.0	0.0
	J10c5a5	122	122	122	122	0.0	0.0	0.0
	J10c5a6	110	110	110	110	0.0	0.0	0.0
13333	J10c5b1	130	130	130	130	0.0	0.0	0.0
	J10c5b2	107	107	107	107	0.0	0.0	0.0
	J10c5b3	109	109	109	109	0.0	0.0	0.0
	J10c5b4	122	122	122	122	0.0	0.0	0.0
	J10c5b5	153	153	153	153	0.0	0.0	0.0
	J10c5b6	115	115	115	115	0.0	0.0	0.0
33233	**J10c5c1**	**68**	**68**	**68**	**68**	**0.0**	**0.0**	**0.0**
	J10c5c2	**74**	**74**	**74**	**74**	**0.0**	**0.0**	**0.0**
	J10c5c3	**71**	**71**	**72**	**71**	**0.0**	**0.0**	**1.4**
	J10c5c4	**66**	**66**	**66**	**66**	**0.0**	**0.0**	**0.0**
	J10c5c5	**78**	**78**	**78**	**78**	**0.0**	**0.0**	**0.0**
	J10c5c6	**69**	**69**	**69**	**69**	**0.0**	**0.0**	**0.0**
33333	**J10c5d1**	**66**	**66**	**66**	**66**	**0.0**	**0.0**	**0.0**
	J10c5d2	**73**	**73**	**73**	**73**	**0.0**	**0.0**	**0.0**
	J10c5d3	**64**	**64**	**64**	**64**	**0.0**	**0.0**	**0.0**
	J10c5d4	**70**	**70**	**70**	**70**	**0.0**	**0.0**	**0.0**
	J10c5d5	**66**	**66**	**66**	**66**	**0.0**	**0.0**	**0.0**
	J10c5d6	**62**	**62**	**62**	**62**	**0.0**	**0.0**	**0.0**
3333133333	J10c10a2	158	158	158	158	0.0	0.0	0.0
	J10c10a3	148	148	148	148	0.0	0.0	0.0
	J10c10a4	149	149	149	149	0.0	0.0	0.0
	J10c10a5	148	148	148	148	0.0	0.0	0.0
	J10c10a6	146	146	146	146	0.0	0.0	0.0
1333333333	J10c10b1	163	163	163	163	0.0	0.0	0.0
	J10c10b2	157	157	157	157	0.0	0.0	0.0
	J10c10b3	169	169	169	169	0.0	0.0	0.0
	J10c10b4	159	159	159	159	0.0	0.0	0.0
	J10c10b5	165	165	165	165	0.0	0.0	0.0
	J10c10b6	165	165	165	165	0.0	0.0	0.0
3333233333	J10c10c1	127	115	115	113	12.4	1.8	1.8
	J10c10c2	116	116	119	116	0.0	0.0	2.6
	J10c10c3	133	116	116	98	35.7	18.4	18.4
	J10c10c4	135	120	120	103	31.1	16.5	16.5
	J10c10c5	145	126	126	121	19.8	4.1	4.1
	J10c10c6	112	106	106	97	15.5	9.3	9.3
33133	J15c5a1	178	178	178	178	0.0	0.0	0.0
	J15c5a2	165	165	165	165	0.0	0.0	0.0
	J15c5a3	130	130	130	130	0.0	0.0	0.0
	J15c5a4	156	156	156	156	0.0	0.0	0.0
	J15c5a5	164	164	164	164	0.0	0.0	0.0
	J15c5a6	178	178	178	178	0.0	0.0	0.0
13333	J15c5b1	170	170	170	170	0.0	0.0	0.0
	J15c5b2	152	152	152	152	0.0	0.0	0.0
	J15c5b3	157	157	157	157	0.0	0.0	0.0
	J15c5b4	147	147	147	147	0.0	0.0	0.0
	J15c5b5	166	166	166	166	0.0	0.0	0.0
	J15c5b6	175	175	175	175	0.0	0.0	0.0
33233	**J15c5c1**	**85**	**85**	**85**	**85**	**0.0**	**0.0**	**0.0**
	J15c5c2	**90**	**90**	**91**	**90**	**0.0**	**0.0**	**1.1**

	J15c5c3	**87**	**87**	**87**	**87**	**0.0**	**0.0**	**0.0**
	J15c5c4	**90**	**90**	**89**	**89**	**1.1**	**1.1**	**0.0**
	J15c5c5	**84**	**74**	**74**	**73**	**15.1**	**1.4**	**1.4**
	J15c5c6	**91**	**91**	**91**	**91**	**0.0**	**0.0**	**0.0**
33333	**J15c5d1**	**167**	**167**	**167**	**167**	**0.0**	**0.0**	**0.0**
	J15c5d2	**85**	**84**	**84**	**82**	**3.7**	**2.4**	**2.4**
	J15c5d3	**96**	**83**	**83**	**77**	**24.7**	**7.8**	**7.8**
	J15c5d4	**101**	**84**	**84**	**61**	**65.6**	**37.7**	**37.7**
	J15c5d5	**97**	**80**	**80**	**67**	**44.8**	**19.4**	**19.4**
	J15c5d6	**87**	**82**	**82**	**79**	**10.1**	**3.8**	**3.8**
3333133333	J15c10a1	236	236	236	236	0.0	0.0	0.0
	J15c10a2	200	200	200	200	0.0	0.0	0.0
	J15c10a3	198	198	198	198	0.0	0.0	0.0
	J15c10a4	225	225	225	225	0.0	0.0	0.0
	J15c10a5	183	182	182	182	0.5	0.0	0.0
	J15c10a6	200	200	200	200	0.0	0.0	0.0
1333333333	J15c10b1	222	222	222	222	0.0	0.0	0.0
	J15c10b2	187	187	187	187	0.0	0.0	0.0
	J15c10b3	222	222	222	222	0.0	0.0	0.0
	J15c10b4	221	221	221	221	0.0	0.0	0.0
	J15c10b5	200	200	200	200	0.0	0.0	0.0
	J15c10b6	219	219	219	219	0.0	0.0	0.0
Déviation_moyenne						3.68	**1.62**	1.68

Dans le tableau 13, nous comparons l'efficacité des cinq méthodes (DDS-tronquée, CDDS, CDDS-LB, B&B et AIS) pour les problèmes faciles et difficiles.

Pour les problèmes faciles, les méthodes DDS-tronquée, CDDS et CDDS-LB fournissent de meilleurs résultats que le B&B, mais pour les problèmes difficiles l'algorithme de B&B et la stratégie AIS sont meilleurs que l'algorithme DDS-tronquée. En revanche, nous remarquons que CDDS-LB surpasse les deux méthodes de B&B et AIS sur les deux classes de problème.

Tableau 13. Efficacité des cinq méthodes

Méthodes	**problèmes faciles**	**problèmes difficiles**
	déviation_moyenne	***déviation_moyenne***
B&B	2.21	6.88
AIS	1.01	3.12
DDS-tronquée	1.42	8.01
CDDS	1.10	4.87
CDDS-LB	**0.96**	**3.06**

Si nous considérons tous les problèmes, la déviation moyenne de DDS-tronquée par rapport aux LB est de 3.5%, alors que la moyenne de déviation de B&B est de 3.68%, pour AIS de 1.68% et pour CDDS de 2.29%. Quant à CDDS-LB la moyenne est de 1.627% seulement. Ainsi, en moyenne, CDDS-LB surpasse les autres méthodes mais il faut mentionner qu'AIS présente des résultats assez proches.

4.4. Adaptation de la méthode développée aux problèmes de flow shop hybride à deux étages

La méthode que nous avons proposée pour la résolution du flow shop hybride dans son cas général CDDS-LB (notée dans cette section $CDDS^L$) s'est révélée efficace sur des problèmes de différents types et de tailles pouvant aller jusqu'à 100 et 150 opérations si un étage critique est présent. Ainsi, il nous a semblé logique de l'appliquer pour la résolution du cas particulier du flow shop hybride ayant 2 étages (notée dans ce cas $CDDS^2$). Pour se faire, une adaptation de la méthode au problème considéré est envisagée.

Nous convenons dans toute la suite de ce paragraphe de substituer le terme « CDDS-LB » par « $CDDS^L$ » ($CDDS^L$ désignera toujours une procédure de type CDDS-LB définie dans le cas d'un flow shop hybride à *L* étages).

4.4.1. Heuristiques sur l'ordre d'instanciation des variables

Un choix judicieux de l'ordre d'examen des jobs s'avère d'un très grand intérêt pour l'obtention d'une solution initiale de bonne qualité. Ainsi, nous proposons, d'utiliser dans le premier étage, une extension de la règle de Johnson conçue initialement pour le problème sans flexibilité de ressource (où $m_1=m_2=1$). Pour le second étage, on donne la priorité à l'opération ayant la plus petite date de disponibilité (*EST* : Earliest Start Time). En cas d'égalité, on considère l'opération appartenant au job ayant la plus longue durée opératoire (*LJD* : Largest Job Duration). Ainsi, on a deux phases d'ordonnancement présentées comme suit :

1ère phase. Ordonnancement du premier étage

1. Séquencer les jobs selon la règle de Johnson. Cette dernière donne la priorité au job *i*, si $\min(p_{1,i}, p_{2,j}) \le \min(p_{1,j}, p_{2,i})$. Dans notre cas, on a $m_1 \ge 2$ et $m_2 \ge 2$, donc on applique cette règle au problème à deux machines ayant $\left\{\frac{p_{1,j}}{m_1}, \frac{p_{2,j}}{m_2}, j \in J\right\}$ comme durées opératoires [Lee et Vairaktarakis, 1994]. Ainsi, on divise l'ensemble de jobs *J* en deux sous-ensembles, J_1 et J_2, où $J_1 = \left\{j : \frac{p_{1,j}}{m_1} \le \frac{p_{2,j}}{m_2}\right\}$ et $J_2 = \left\{j : \frac{p_{1,j}}{m_1} > \frac{p_{2,j}}{m_2}\right\}$, on ordonne J_1 dans l'ordre croissant de $\frac{p_{1,j}}{m_1}$ et J_2 dans l'ordre décroissant de $\frac{p_{2,j}}{m_2}$. Puis, on séquence les jobs de J_1 suivis de J_2. Soit $SEQ = J_1.J_2$.
2. Ordonnancer chaque job $j \in SEQ$ sur la première machine disponible. $SEQ = SEQ \setminus \{j\}$,

3. Si $SEQ \neq \phi$, alors aller à l'étape 1.2.

2ème phase. Ordonnancement du second étage

1. Pour chaque job $j \in J$, maintenir les dates de disponibilité de chaque job *j* à $r_j = C_{1,j}$. et soit $SEQ = J$.
2. Ordonnancer tout job *j* disponible ($j \in SEQ$) sur la première machine disponible. En cas d'égalité, on sélectionne l'opération ayant la plus longue durée opératoire. $SEQ = SEQ \setminus \{j\}$,
3. Si $SEQ \neq \phi$, alors aller à l'étape 2.2. Sinon, quitter.

Après toute instanciation des variables de sélection de job et d'allocation des ressources, on applique une technique de propagation de contraintes de type *Forward Checking* [Haralick et Elliot, 1980] afin de mettre à jour la date de début au plus tôt de l'opération successeur pour le job sélectionné et d'actualiser la date de disponibilité de la ressource choisie.

4.4.2. Borne inférieure

Pour améliorer notre recherche de solutions, nous proposons d'utiliser des bornes inférieures conçues pour ce type de problème. Pour cela, nous proposons d'utiliser les bornes inférieures développées dans [M'Hallah et Haouari, 1997] qui peuvent être présentées comme suit :

Soit le sous-ensemble $S \subseteq J$, définissons $I_2(S)$ comme une borne inférieure du temps inoccupé au second étage. Ce laps de temps est une conséquence directe des contraintes de précédence. Si l'on considère $S = J$, $I_2(S)$ est égale à la somme des dates de fin des m_2 jobs de S. Les m_2 jobs considérés sont les jobs ayant les plus petites durées opératoires sur le premier étage. Il est évident que $I_2(S)$ peut être obtenu par application de la règle SPT. Ainsi,

$$LB^2_{SPT}(S) = \left\lceil \frac{I_2(S) + \sum_{j \in S} p_{2,j}}{m_2} \right\rceil$$

définit une borne inférieure. Etant donné que le problème de flow shop hybride est symétrique, on obtient la borne inférieure suivante :

$$LB^1_{SPT}(S) = \left\lceil \frac{I_1(S) + \sum_{j \in S} p_{1,j}}{m_1} \right\rceil$$

Ainsi, la borne inférieure suivante est valide.

$$LB = \max\left(LB^1_{SPT}, LB^2_{SPT}\right)$$

4.4.3. Expérimentations

Nous proposons de tester notre méthode CDDS sur trois types de jeux-tests générés de la même manière que dans [Lee et Vairaktarakis, 1994], à savoir l'ensemble A, B et C.

- ***A*** : Le nombre de jobs est égal à 10, 20, 30, 40, 50, 100 et 150. Les nombres de machines (m_1, m_2) sont : (2, 2), (2, 4), (4, 2), et (4, 4). Les durées opératoires suivent une loi uniforme dans [1, 20] pour le premier étage et dans [1, 40] pour le second. Pour chaque configuration (nombre de jobs et nombre de machines par étage), 20 problèmes sont générés aléatoirement. Cette classe contient ainsi 560 instances.
- ***B*** : Cette classe contient 560 instances générées de la même manière que dans la classe A, mais avec des durées opératoires suivant la loi uniforme dans [1, 40] pour le premier étage et [1, 20] pour le second.
- ***C*** : Cette classe contient 560 instances générées de la même manière que dans la classe A, mais avec des durées opératoires suivant la loi uniforme dans [1, 40] pour les deux étages.

Ainsi, on considère au total 1680 instances. La méthode CDDS a été codée en C et implémentée sur un PC Intel Core 2 Duo 2.9 GHz avec 2GB de mémoire vive. Le temps maximum alloué à la recherche est de 15 secondes. Si la méthode atteint les 15 secondes sans avoir une solution optimale, la meilleure solution trouvée lors des différentes itérations est considérée comme solution finale. On résout les instances et leurs inverses se basant sur la symétrie du problème de flow shop hybride. La profondeur maximale pour l'application des divergences dans CDDS est fixée à 7.

Dans une première étape, nous proposons de comparer la méthode CDDS conçue pour ce type de problème à deux étages (notée $CDDS^2$) avec la méthode CDDS conçue pour le cas général (notée $CDDS^L$). Le temps maximum alloué à la recherche, pour $CDDS^L$ est également de 15 secondes et la profondeur maximale est aussi fixée à 7. Les résultats fournis par chaque algorithme seront comparés avec les bornes inférieures développées dans [Haouari *et al.*, 2006].

Tableau 14. *Performance des deux algorithmes de CDDS sur la classe A*

		$CDDS^2$			$CDDS^L$		
N	(m_1, m_2)	Nb_US	Déviation	CPU	Nb_US	Déviation	CPU
10	(2, 2)	3	0.21	72.40	11	1.20	175.80
	(2, 4)	10	2.47	105.60	17	6.24	256.60
	(4, 2)	0	0.00	1.40	3	0.06	46.20
	(4, 4)	0	0.00	2.60	4	0.12	62.00
20	(2, 2)	2	0.05	40.60	3	0.16	62.20
	(2, 4)	9	0.91	168.40	20	5.79	300.00
	(4, 2)	0	0.00	18.80	4	0.12	75.00
	(4, 4)	4	0.21	92.00	11	1.34	175.40
30	(2, 2)	1	0.02	18.40	7	1.61	117.00
	(2, 4)	6	0.86	137.80	17	5.61	258.40
	(4, 2)	0	0.00	9.00	3	0.12	52.60

	(4, 4)	3	0.07	75.60	8	0.32	135.20
40	(2, 2)	0	0.00	4.80	2	0.04	34.40
	(2, 4)	4	0.21	103.60	8	0.97	135.60
	(4, 2)	0	0.00	19.20	4	0.56	75.40
	(4, 4)	2	0.05	57.80	4	0.12	83.20
50	(2, 2)	0	0.00	8.00	3	0.07	5.80
	(2, 4)	2	0.15	47.40	6	0.70	106.60
	(4, 2)	0	0.00	12.00	4	0.11	69.60
	(4, 4)	2	0.06	78.20	4	0.08	91.20
100	(2, 2)	1	0.05	37.60	4	0.23	75.00
	(2, 4)	2	0.06	73.60	6	0.30	115.80
	(4, 2)	0	0.00	18.20	5	0.09	88.60
	(4, 4)	2	0.02	74.20	6	0.15	116.00
150	(2, 2)	0	0.00	88.00	3	0.30	119.80
	(2, 4)	0	0.00	48.20	2	0.16	73.40
	(4, 2)	0	0.00	105.40	3	0.18	134.60
	(4, 4)	1	0.01	162.00	2	0.05	175.80
Dév_moy			**0.19**			**0.96**	
CPU total			**1681**			**3263**	

Le tableau 14 dresse une comparaison entre les deux algorithmes pour la classe A. Les première et deuxième colonnes présentent les dimensions des instances (nombre de jobs et nombre de machines par étage). La colonne *Nb_US* donne le nombre d'instances non résolues d'une manière optimale ($C_{\max} \neq LB$). Le pourcentage de déviation par rapport aux bornes inférieures est calculé comme suit : $\frac{C_{\max} - LB}{LB} \times 100$. La colonne *CPU* donne le temps d'exécution totale pour chaque classe (20 instances).

D'après le tableau 14, nous observons que l'algorithme *CDDS²* surpasse nettement *CDDS^L* en termes de qualité de solutions et de temps de calcul. Ainsi, il fournit des solutions de bonne qualité au bout de 3 secondes par instance (1681 en total). Nous notons, également, que 90.4% de problèmes ont été résolus de manière optimale par *CDDS²* face à seulement 68.9% pour *CDDS^L*. En outre, si l'on considère toutes les instances, la déviation moyenne de *CDDS²* est strictement inférieure à 0.19% face à 0.96% pour *CDDS^L*.

Tableau 15. Performance des deux algorithmes de CDDS sur la classe B

		CDDS²			*CDDS^L*		
N	(m_1, m_2)	Nb_US	Dév_moy	CPU	Nb_US	Dév_moy	CPU
10	(2, 2)	2	0.09	30.20	5	0.12	86.40
	(2, 4)	0	0.00	8.80	5	0.38	81.60
	(4, 2)	9	1.60	120.60	19	5.92	285.60
	(4, 4)	2	0.29	30.40	7	1.18	114.80
20	(2, 2)	1	0.03	15.80	5	0.08	86.80
	(2, 4)	0	0.00	0.60	3	0.09	45.60
	(4, 2)	5	0.64	98.20	13	5.70	201.80
	(4, 4)	3	0.14	58.20	4	0.15	75.60
30	(2, 2)	0	0.00	0.60	4	0.16	60.40
	(2, 4)	0	0.00	2.00	3	0.12	46.80
	(4, 2)	7	0.93	121.00	11	5.69	172.80
	(4, 4)	2	0.11	49.80	3	0.19	66.20
40	(2, 2)	0	0.00	3.80	2	0.06	33.40
	(2, 4)	0	0.00	1.80	3	0.53	46.60

	(4, 2)	3	0.28	61.20	9	1.01	146.20
	(4, 4)	2	0.05	261.80	4	0.55	164.80
50	(2, 2)	0	0.00	13.00	3	0.56	56.00
	(2, 4)	0	0.00	2.00	2	0.43	31.80
	(4, 2)	8	0.37	144.60	10	0.69	159.00
	(4, 4)	1	0.02	58.80	3	0.06	95.00
100	(2, 2)	0	0.00	10.40	3	0.09	53.80
	(2, 4)	0	0.00	11.00	2	0.09	40.00
	(4, 2)	2	0.03	86.00	7	0.35	133.00
	(4, 4)	2	0.02	66.00	8	0.17	139.80
150	(2, 2)	0	0.00	27.80	3	0.08	68.60
	(2, 4)	0	0.00	18.00	4	0.13	74.40
	(4, 2)	1	0.03	60.60	5	0.22	120.40
	(4, 4)	1	0.01	101.20	3	0.03	131.00
Dév_moy			**0.17**			**0.89**	
CPU total			**1464**			**2818**	

La performance globale de la procédure proposée est confirmée par les résultats obtenus sur le jeu-test de la classe B (voir le tableau 15). Ainsi, ***CDDS²*** fournit des solutions qui dévient des bornes inférieures de 0.17%. Dans cette classe, 91% de problèmes ont été résolus de manière optimale par ***CDDS²***, face à 72.7% pour ***CDDS^L***. En outre, la déviation moyenne par rapport aux bornes inférieures obtenues par ***CDDS^L*** est de 0.89%.

Le tableau 16 rapporte les résultats des deux algorithmes menés sur les instances de la classe C. ***CDDS²*** prouve son efficacité pour la résolution du problème de flow shop hybride à deux étages. En effet, il fournit des solutions qui dévient des bornes inférieures de 0.26% dans un laps de temps de 3 secondes par instance (environ 1769 pour résoudre les 560 instances) ; néanmoins, ***CDDS^L*** obtient des solutions à 0.41% des LB en 7 secondes par instance. Dans cette classe, 90% de problèmes ont été résolus de manière optimale par ***CDDS²*** confronté à 55% pour ***CDDS^L***.

Tableau 16. *Performance des deux algorithmes de CDDS sur la classe C*

		CDDS²			***CDDS^L***		
N	(m_1, m_2)	Nb_US	Dév_moy	CPU	Nb_US	Dév_moy	CPU
10	(2, 2)	3	0.20	47.00	9	0.26	143.60
	(2, 4)	1	0.34	16.20	6	0.45	101.40
	(4, 2)	0	0.00	1.20	5	0.05	76.00
	(4, 4)	9	1.89	135.00	18	1.98	271.60
20	(2, 2)	5	0.39	77.00	11	0.48	172.00
	(2, 4)	0	0.00	1.40	7	0.07	106.00
	(4, 2)	0	0.00	18.20	3	0.23	60.40
	(4, 4)	13	1.19	197.40	20	1.47	300.00
30	(2, 2)	4	0.10	61.40	12	0.63	186.20
	(2, 4)	0	0.00	8.80	8	0.08	125.20
	(4, 2)	1	0.02	23.60	5	0.05	92.80
	(4, 4)	9	1.42	182.00	14	1.95	216.00
40	(2, 2)	3	0.08	47.60	9	0.26	143.80
	(2, 4)	1	0.01	16.20	7	0.14	115.60
	(4, 2)	1	0.01	41.80	6	0.17	119.20
	(4, 4)	5	0.46	107.80	11	0.61	174.80

50	(2, 2)	0	0.00	3.80	6	0.06	92.60
	(2, 4)	0	0.00	5.20	8	0.28	123.20
	(4, 2)	2	0.02	37.60	5	0.05	89.20
	(4, 4)	5	0.84	97.80	12	1.20	187.80
100	(2, 2)	0	0.00	14.60	9	0.16	143.00
	(2, 4)	0	0.00	11.20	11	0.15	170.00
	(4, 2)	2	0.01	68.80	9	0.09	154.00
	(4, 4)	6	0.22	134.60	7	0.32	119.60
150	(2, 2)	1	0.01	85.60	8	0.08	171.40
	(2, 4)	0	0.00	32.40	9	0.09	152.80
	(4, 2)	0	0.00	100.80	7	0.07	170.60
	(4, 4)	7	0.11	193.80	10	0.14	163.80
Dév_moy			**0.26**			**0.41**	
CPU total			**1769**			**4143**	

Dans une seconde étape, nous proposons de dresser une comparaison entre les solutions données par *CDDS²* et les résultats de la méthode de Recherche Tabou (TS) développée dans [Haouari et M'Hallah, 1997]. Les solutions des deux méthodes sont comparées avec les bornes inférieures de [Haouari et M'Hallah, 1997] calculées à la racine (voir le tableau 17). Les valeurs mentionnées dans le tableau 17 sont les déviations moyennes de chaque méthode par rapport aux bornes inférieures.

Le tableau 17 montre que CDDS² surpasse la méthode de recherche tabou sur toutes les instances, sauf pour la sous-classe (4, 2) de la classe B. Cependant, on note que CDDS² est très efficace pour toutes les tailles de problème.

Tableau 17. ***Comparaison de performance entre CDDS² et TS de [Haouari et M'Hallah, 1997]***

N		***classe A***			***classe B***			***classe C***			***Moyenne***
		(2,4)	(4,4)	(4,2)	(2,4)	(4,4)	(4,2)	(2,4)	(4,4)	(4,2)	
20	*CDDS*	0.95	0.26	0.00	0.03	0.14	0.73	0.00	1.48	0.05	0.40
	TS	2.90	1.20	0.35	0.92	5.72	0.13	0.56	3.43	1.22	1.83
30	*CDDS*	0.92	0.10	0.00	0.00	0.11	0.96	0.07	1.45	0.02	0.40
	TS	1.43	0.85	0.06	0.57	3.10	0.05	0.27	1.45	1.46	1.03
40	*CDDS*	0.21	0.05	0.00	0.00	0.05	0.28	0.02	0.46	0.01	0.12
	TS	0.96	0.43	0.12	0.5	1.57	0.12	0.34	1.08	0.89	0.67
50	*CDDS*	0.15	0.06	0.00	0.00	0.02	0.37	0.00	0.88	0.02	0.16
	TS	0.54	0.30	0.02	0.26	1.09	0.04	0.20	0.95	0.42	0.42
100	*CDDS*	0.06	0.02	0.00	0.00	0.02	0.03	0.00	0.22	0.01	0.04
	TS	0.19	0.15	0.02	0.11	0.39	0.01	0.07	0.41	0.18	0.17
Moyenne	*CDDS*	0.46	0.10	0.00	0.01	0.07	0.48	0.02	0.90	0.02	0.22
	TS	1.20	0.59	0.11	0.47	2.37	0.07	0.29	1.46	0.83	0.82

Pour la plupart des problèmes, la procédure CDDS² apporte des solutions optimales. En considérant tous les problèmes, CDDS² atteint la solution optimale dans 85.8% des cas (772 cas sur 900) en un temps de 2.26 secondes

par instance tandis que TS rapporte les solutions optimales à 35% des cas (316 sur 900) en 0.03 seconde par instance. Notons que cette dernière valeur sur le temps de calcul de TS a été calculée utilisant les coefficients de normalisation de [Dongarra, 1998]. Les résultats de TS sont obtenus initialement dans [Haouari et M'Hallah, 1997] en 134 secondes.

Comme le montre la figure 21, la méthode CDDS^2 est plus performante pour les grandes instances. En considérant tous les problèmes et toutes les distributions, CDDS^2 présente une déviation des LB de 0.22% contre 0.82% pour la méthode TS.

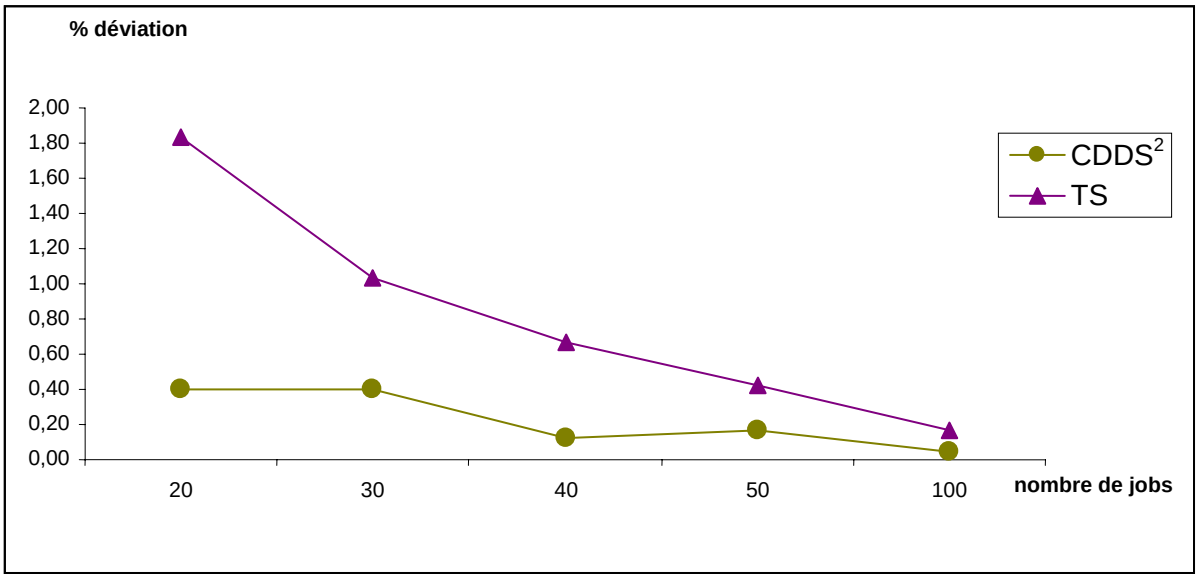


Fig.21. Déviation relative de CDDS^2 et TS pour les différents problèmes

4.5. Conclusion sur le problème de flow shop hybride

La méthode que nous avons conçue (CDDS^L) s'est révélée efficace pour la résolution du problème de Flow Shop Hybride à plusieurs étages et, à notre connaissance, elle fournit les meilleurs résultats pour ce type de problème. L'exploitation des résultats nous a permis d'aboutir aux conclusions suivantes :

- La méthode est efficace sur les problèmes de taille raisonnable (50 opérations) et fournit 100% des solutions optimales, que ces problèmes soient équilibrés ou non.
- Pour des problèmes de plus grande taille, la présence d'un étage critique conditionne nettement la qualité des solutions. En effet, la méthode est plus efficace si un tel étage existe. Dans ce cas, la taille des problèmes pouvant être résolus de manière optimale augmente à 100 et à 150 opérations.
- La position de l'étage critique n'influe pas sur les résultats obtenus. En effet, les résultats obtenus montrent que l'étage critique est traité avec la même efficacité que cet étage soit à la première position ou au milieu.

- L'intégration des bornes inférieures permet de gagner aussi bien en termes de temps de calcul qu'en termes de qualité de solutions. Cependant sur des problèmes de taille importante (15x10), il semble que l'apport des bornes inférieures ne soit pas intéressant en termes de qualité de solutions.

Adaptée aux problèmes à deux étages, CDDS^2 a également prouvé son efficacité. L'intégration de calcul des bornes inférieures spécifiques au sein de la méthode de résolution s'est révélée d'un intérêt indéniable pour les performances de la méthode.

Enfin, la méthode pourrait être adaptée aux problèmes de type job shop flexible et ceci fera l'objet du chapitre suivant.

Chapitre 5. Adaptation de la méthode CDDS pour le Job Shop Flexible

Cette partie est consacrée à l'adaptation de la méthode CDDS aux problèmes de type Job Shop Flexible. Notons que l'utilisation d'une méthode à base de divergences pour aborder les problèmes de job shop flexible est originale. A notre connaissance, cette méthode n'avait pas été utilisée auparavant pour ce genre de problème. Dans un premier temps, nous reviendrons sur les caractéristiques des problèmes de Job Shop Flexible considérés. Ensuite, nous présentons une adaptation de CDDS pour ce type de problème à travers différentes variantes basées sur la notion de bloc permettant de définir des heuristiques pour l'application des divergences. Des résultats expérimentaux montrent les atouts et les faiblesses respectives de chacune de ces variantes pour la résolution des problèmes considérés. Enfin, une comparaison des meilleures variantes de CDDS avec des méthodes connues pour leur performance sur les problèmes de Job Shop Flexible montre la qualité de notre méthode.

Une partie des résultats proposés dans ce chapitre a été présentée dans les conférences internationales : MISTA [2007] (Multidisciplinary International conference on Scheduling: Theory & Applications) à Paris, France et PMS [2008] (international workshop on Project Management and Scheduling) à Istanbul, Turquie.

5.1. Introduction

Les problèmes de Job Shop Flexible que nous considérons sont soumis aux hypothèses suivantes :

- Les opérations sont non interruptibles.
- Les ressources sont disjonctives et non-reliées, c'est-à-dire que les durées opératoires dépendent de la ressource choisie.
- Les durées des opérations sont entières.
- Nous n'imposons aucune limite sur le nombre d'opérations qui peuvent attendre entre deux ressources, les stocks placés à ces endroits sont considérés en quantité infinie.
- Les temps de transport et/ou de maintenance sont considérés comme nuls, ou éventuellement dans certains cas, ils sont contenus dans la durée d'exécution des tâches.
- Nous n'imposons aucune borne supérieure pour la valeur de date de fin d'exécution totale.

Résoudre un problème d'ordonnancement de type job shop flexible revient à fixer pour chaque opération une date de début et une ressource pouvant l'exécuter. Le placement de chaque opération dans le temps et sur les ressources doit satisfaire à la fois les contraintes de précédence et les contraintes de ressources. Les gammes ne sont pas nécessairement linéaires, c'est-à-dire qu'une opération peut avoir plusieurs successeurs et plusieurs prédécesseurs (dans les jeux de données utilisées pour les expérimentations, les gammes sont linéaires, mais elles pourraient ne pas l'être pour notre procédure de résolution). Parmi ces ordonnancements réalisables, nous cherchons un ordonnancement faisable minimisant la durée d'exécution totale ou makespan.

5.2. Adaptation de CDDS pour le problème considéré

5.2.1. Heuristiques sur l'ordre d'instanciation des variables

Afin de résoudre le problème considéré, il faut sélectionner tout d'abord une opération à exécuter, lui allouer une ressource et fixer ensuite sa date de début. On considèrera ainsi deux types de variables : les variables de sélection des opérations et les variables d'allocation de ressources.

La stratégie d'exploration consiste à instancier tout d'abord la variable de sélection d'opération (notées X_i), puis la variable d'allocation de ressources (notées A_i) pour l'opération sélectionnée. Pour ce faire, deux types d'heuristiques d'instanciation des variables sont considérés.

Type 1. Sélection de l'opération. Un choix judicieux de l'ordre d'examen des opérations s'avère d'un très grand intérêt pour l'obtention d'une solution de bonne qualité. Ainsi, la priorité est donnée à l'opération ayant la plus petite date de début au plus tôt (*EST*). En cas d'égalité entre opérations, nous proposons de sélectionner l'opération ayant la plus petite date de fin au plus tard (*EDD*).

Type 2. Allocation des ressources aux opérations. L'opération choisie par l'heuristique de Type 1 sera, à la suite, affectée à la ressource lui offrant la plus petite date de fin d'exécution (*ECT* : Earliest Completion Time).

Après toute instanciation des variables de sélection des opérations et d'allocation des ressources, on applique une technique de propagation de contraintes de type Forward Checking [Haralick et Elliot, 1980] afin de mettre à jour la date de début au plus tôt de l'(des) opération(s) successeur(s) et d'actualiser la date de disponibilité de la ressource choisie.

Dans notre cas, la solution obtenue par l'application des différentes heuristiques présentées ci-dessus, est une solution initiale à notre problème de Job Shop flexible car la valeur du makespan n'est pas contrainte.

5.2.2. Notion de divergence pour le job shop flexible

Dans les problèmes de job shop flexible étudiés, des divergences sont possibles sur les deux types de variables : les variables de sélection de l'opération et les variables d'allocation de la ressource.

En effet, notre but est d'optimiser la valeur du makespan et étant donné que les ressources sont non-reliées, une divergence sur les variables d'affectation peut entraîner une amélioration au niveau du critère considéré. L'ordre des opérations peut bien sûr avoir également un impact sur le temps d'exécution total. Ainsi, faire une divergence consiste à :

- Sélectionner une opération autre que celle recommandée par l'heuristique d'instanciation. Etant donné que les variables de sélection des opérations sont des variables *n*-aires, le nombre de divergences est calculé comme suit : la première valeur suggérée par l'heuristique correspond à 0 divergence, toutes les autres valeurs correspondent à une seule divergence. Considérons par exemple 3 opérations ordonnées (suivant une heuristique donnée) comme suit : O_1, O_2 et O_3. La sélection d'une autre opération que O_1 dans la première position (X_1) correspond à une divergence (voir figure 22).

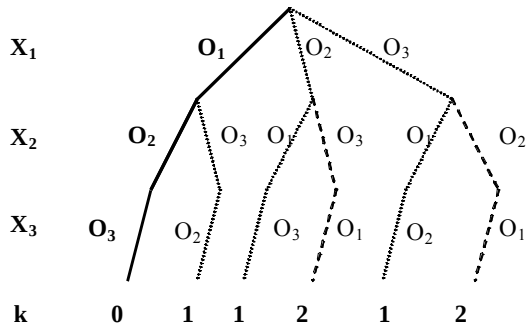


Fig.22. Divergences sur les trois premières variables de sélection des opérations

En considérant cet exemple, on voit que si l'heuristique favorise le choix de l'opération O_1, puis O_2 et ensuite O_3 ($S_{initiale}$={1,2,3}), les solutions à 1-divergence sont données par les triplets {2,1,3}, {3,1,2} et {1,3,2} où la valeur soulignée représente la divergence effectuée. Une solution à 2-divergences revient à faire une seule divergence par rapport aux solutions à 1-divergence sans revenir aux solutions déjà générées ; i.e. à partir de la solution {2,1,3}, on obtient la solution{2,3,1} ; à partir de la solution {3,1,2}, on obtient la solution {3,2,1} et à partir de la solution {1,3,2}, il n'y a pas de solution à 2 divergences.

- Affecter une opération à une ressource autre que celle suggérée par l'heuristique d'instanciation. Le nombre de divergences est calculé comme suit : la première valeur suggérée par l'heuristique correspond à 0 divergence, toutes les autres valeurs correspondent à une seule divergence. Dans ce cas, considérons que O_1 peut être exécutée sur l'une des ressources suivantes {R_2, R_1, R_3} (cet ordre est donné par une heuristique), O_2 par {R_1, R_4} et O_3 par {R_1}. Choisir une autre ressource que R_2 pour la première opération (O_1) consiste à faire une divergence à ce niveau (voir figure 23).

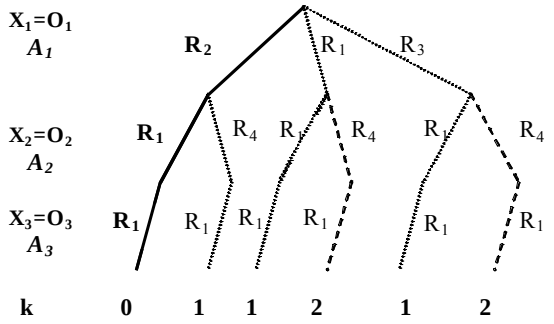


Fig.23. Divergences sur les trois premières variables de sélection de ressources

En considérant cet exemple, on voit que si l'heuristique d'affectation de ressources favorise le choix de la ressource R_2 pour O_1, puis R_1 pour O_2 et ensuite R_1 pour O_3 ($S_{initiale}$={2,1,1}), les solutions à 1-divergence sont données par les triplets {1,1,1}, {3,1,1} et {2,4,1} où la valeur

soulignée représente la divergence effectuée. Une solution à 2-divergences revient à faire une seule divergence par rapport aux solutions à 1-divergence sans revenir aux solutions déjà générées ; i.e. à partir de la solution {1,1,1}, on obtient la solution {1,4,1} ; à partir de la solution {3,1,1}, on obtient la solution {3,4,1} et à partir de la solution {2,4,1}, il n'y a pas de solution à 2-divergences .

Pour obtenir les solutions à *k*+1 divergences à partir d'une solution à *k* divergences directement sans revisiter les solutions de 0 à ($k-1$) divergences (principe de ILDS), on utilise un compteur de divergences pour chaque variable. Puis, à partir de chaque solution à *k* divergences, on considère la dernière variable ayant une divergence et on applique des divergences sur les variables suivantes. On obtient ainsi les solutions ayant *k*+1 divergences. Cette stratégie nous permet de générer les solutions sans redondance étant donné que les choix déjà effectués au niveau d'un nœud donné seront inhibés lors du choix suivant.

5.2.3. Stratégies d'exploration

Pour les problèmes de Job Shop Flexible, nous n'avons pas trouvé dans la littérature de bornes inférieures pour les intégrer à notre méthode CDDS.

Nous proposons alors d'explorer une autre voie d'amélioration de CDDS consistant à déterminer de manière heuristique des variables pertinentes pour application de divergences et des actions à effectuer sur ces variables. Ces heuristiques de guidage de l'application des divergences permettent de définir des voisinages pour une solution de référence.

- Le premier voisinage ***V1*** est défini comme suit :
  - Chaque opération O_i est considérée successivement et pour chacune d'elle, on l'affecte à une autre machine R_i. Si R_i n'est pas disponible à r_i, on déplace l'opération O_j l'occupant après O_i ($O_i \prec O_j$).
  - Réinsérer toute opération O_i, différente de la dernière opération dans la séquence, avant toute autre opération dans la séquence tout en respectant les contraintes de précédence. Seul un réajustement des dates de début des opérations sera pris en compte.
- Le deuxième voisinage ***V2*** est défini comme suit :
  - Chaque opération O_i est considérée successivement et pour chacune d'elle, on l'affecte à une autre machine R_i. Si R_i n'est pas disponible à r_i, on affecte l'opération O_j l'occupant à une autre machine suivant *ECT*.
  - Réinsérer toute opération O_i, différente de la dernière opération dans la séquence, avant toute autre opération dans la séquence tout en respectant les contraintes de précédence. Un réajustement des affectations des opérations successeurs

(dans la séquence et sur la même machine) ainsi que leurs dates de début est effectué en utilisant l'heuristique dynamique *ECT*.

Afin de limiter la taille de ces voisinages, on peut utiliser la notion de bloc [Jurisch, 1992]. Un bloc est une succession d'au moins deux opérations critiques sur une même ressource.

Une propriété intéressante proposée par [Jurisch, 1992] est que si l'on a deux solutions *y* et *y'* avec $C_{max}(y')<C_{max}(y)$, alors au moins une des propriétés suivantes est vérifiée :

- Dans *y'*, au moins une opération d'un bloc *B* de *y* est exécutée sur une autre ressource que dans *y*.
- Dans *y'*, au moins une opération d'un bloc *B* de *y*, différente de la première opération de *B* est exécutée avant la première opération de *B*.
- Dans *y'*, au moins une opération d'un bloc *B* de *y*, différente de la dernière opération de *B* est exécutée après la dernière opération de *B*.

Se basant sur cette propriété, deux voisinages peuvent être considérés :

- Le voisinage ***V3*** consiste à :
  - Changer l'affectation d'une opération O_i d'un bloc *B*, et lui affecter une autre ressource R_i. Si R_i n'est pas disponible à r_i, on déplace l'opération O_j l'occupant après O_i ($O_i \prec O_j$).
  - Déplacer une opération d'un bloc *B*, différente de la première (resp. la dernière) opération de *B*, avant (resp. après) la première (resp. la dernière) opération de *B*. Seul un réajustement des dates de début des opérations sera pris en compte.
- Le voisinage ***V4*** consiste à :
  - Changer l'affectation d'une opération d'un bloc *B*, et lui affecter une autre ressource R_i. Si R_i n'est pas disponible à r_i, on affecte l'opération O_j l'occupant à une autre ressource selon *ECT*.
  - Déplacer une opération d'un bloc *B*, différente de la première (resp. la dernière) opération de B, avant (resp. après) la première (resp. la dernière) opération de *B*. Un réajustement des affectations des opérations successeurs (dans la séquence et sur la même machine) ainsi que leurs dates de début est effectué en utilisant l'heuristique dynamique *ECT*.

Une extension de ces deux voisinages peut être envisagée pour donner deux autres types de voisinage :

- Le voisinage ***V5*** consiste à :

- Changer l'affectation d'une opération O_i d'un bloc *B*, et lui affecter une autre ressource R_i. Si R_i n'est pas disponible à r_i, on déplace l'opération O_j l'occupant après O_i ($O_i \prec O_j$).
- Déplacer une opération d'un bloc *B*, différente de la première (resp. la dernière) opération de *B*, avant (resp. après) toutes les autres opérations de *B*. Seul un réajustement des dates de début des opérations sera pris en compte.

- Le voisinage ***V6*** consiste à :
  - Changer l'affectation d'une opération d'un bloc *B*, et lui affecter une autre ressource R_i. Si R_i n'est pas disponible à r_i, on affecte l'opération O_j l'occupant à une autre ressource selon *ECT*.
  - Déplacer une opération d'un bloc *B*, différente de la première (resp. la dernière) opération de B, avant (resp. après) toutes les autres opérations de *B*. Un réajustement des affectations des opérations successeurs (dans la séquence et sur la même machine) ainsi que leurs dates de début est effectué en utilisant l'heuristique dynamique *ECT*.

On remarque que le voisinage *V5* domine *V3* ($V3 \subseteq V5$) et le voisinage *V6* domine *V4* ($V4 \subseteq V6$).

Nous avons donc défini différentes variantes de la méthode CDDS-HDiv. Chacune de ces variantes est caractérisée par une heuristique d'application des divergences ce qui correspond donc à un voisinage différent exploré autour de la solution de référence.

5.3. Expérimentations

Afin de pouvoir évaluer correctement la méthode CDDS-HDiv adaptée à ce type de problème, nous avons choisi de les tester sur quatre classes d'instances issues de la littérature :

- BRdata : Cette classe est constituée de 10 problèmes décrits par Brandimarte (1993) suivant une distribution uniforme. Le nombre de jobs *n* varie entre 10 et 20, le nombre de machines *M* varie entre 4 et 15, le nombre d'opérations pour chaque job varie entre 5 et 15. La moyenne du nombre de machines par opération (nommée aussi flexibilité et notée *flex.*), varie entre 1.43 et 4.10. Les machines considérées dans cette classe sont non-reliées.

- BCdata : Cette classe est constituée de 21 problèmes définis par Barnes et Chambers (1996) et construits à partir des instances de type job shop classique (mt10, la24, la40) [Fisher et Thompson, 1963 ; Lawrence, 1984]. Le nombre de jobs *n* varie entre 10 et 15, le nombre de machines varie entre 11 et 18, le nombre d'opérations constituant chaque job varie entre 10 et 15 et la moyenne du

nombre de machines par opération varie entre 1.07 et 1.30. Les machines considérées dans cette classe sont identiques.

- DPdata : Cette classe est constituée de 21 problèmes fournis par Dauzère-Pérès et Paulli (1997). Le nombre de jobs *n* varie entre 10 et 20, le nombre de machines varie entre 5 et 10 et la moyenne du nombre de machines par opération varie entre 1.13 et 5.02. Les machines considérées dans cette classe sont de deux types : identiques et non-reliées.

- HUdata : Cette classe est constituée de 129 problèmes présentés par Hurink *et al.* (1994) construits à partir des trois problèmes (mt06, mt10, mt20) de Fisher et Thompson (1963) et de 40 problèmes (la01-la40) de Adams *et al.* (1988). La durée opératoire de chaque opération ne dépend pas de la ressource sur laquelle elle est affectée. Les instances sont découpées en trois sous-ensembles :
  - Edata : instances avec très peu de flexibilité (*flex.* = 1.15 ressources possibles par opération), très proches des problèmes de job shop classique,
  - Rdata : instances avec en moyenne deux machines possibles par opération,
  - Vdata : instances avec en moyenne *M/2* machines possibles par opération (c.-à.-d que la valeur de *flex* est comprise entre 2.50 et 7.50 ressources possibles par opération).

Toutes les expérimentations ont été réalisées sur un PC équipé d'un Intel Core 2 Duo cadencé à 2.9 GHz et ayant 2 GB de mémoire vive.

5.3.1. Comparaison sur les jeux-test de Brandimarte (1993)

Dans cette partie, on compare les différentes variantes de CDDS-HDiv développées à partir de différents voisinages considérés (*V1* à *V6*). On donne ensuite une comparaison entre CDDS-HDiv et d'autres méthodes dont l'efficacité a été avérée pour la résolution de ce type de problèmes. Le temps maximum alloué à la recherche est de 15 secondes. Cette dernière valeur a été choisie en se basant sur une campagne d'expérimentations et en constatant qu'au-delà de 15 s les résultats ne s'améliorent que très peu.

Le premier tableau (tableau 18) que nous présentons compare l'efficacité des différentes variantes de CDDS-HDiv entre elles. Les trois premières colonnes indiquent le nom, la taille des instances ainsi que la flexibilité sur les ressources. La quatrième colonne donne les meilleures bornes inférieures (LB) et supérieures (UB) connues pour chaque problème. Les colonnes suivantes présentent les meilleures valeurs du makespan (C_{max}) obtenues pour toutes les variantes de CDDS-HDiv (*V1* à *V6*). La dernière colonne ($C_{meilleur}$) présente les meilleures solutions données par CDDS parmi toutes les variantes. Les valeurs en gras indiquent que l'algorithme correspondant a trouvé la meilleure valeur du C_{max}. Les valeurs sont en gras souligné lorsque la valeur est

optimale (C_{max}=LB). Les valeurs suivies d'un astérisque indiquent les meilleures valeurs connues dans la littérature. La ligne *#meilleur* récapitule le nombre de fois où chaque variante a trouvé la meilleure valeur.

Sur l'ensemble des instances, la déviation moyenne de chaque variante par rapport aux bornes inférieures est utilisée comme critère de comparaison. Pour une instance donnée, la déviation est calculée comme suit : $dév\_moy = \frac{(C\max - LB)}{LB} \times 100\%$.

Tableau 18. Comparaison des différentes variantes de CDDS-HDiv sur les instances BRdata

instances	n×M	flex.	(LB,UB)	CDDS-HDiv						
				V1	V2	V3	V4	V5	V6	$C_{meilleur}$
MK01	10×6	2.09	(36,42)	**40**	**40**	**40**	**40**	**40**	**40**	40*
MK02	10×6	4.10	(24,32)	**26**	**26**	**26**	**26**	**26**	**26**	26*
MK03	15×8	3.01	(204,211)	**204**	**204**	**204**	**204**	**204**	**204**	**204***
MK04	15×8	1.91	(48,81)	**60**	**60**	**60**	**60**	**60**	**60**	60*
MK05	15×4	1.71	(168,186)	175	175	175	**173**	**173**	**173**	173
MK06	10×15	3.27	(33,86)	63	60	60	59	59	**58**	58*
MK07	20×5	2.83	(133,157)	144	144	**139**	**139**	**139**	**139**	139*
MK08	20×10	1.43	523	**523**	**523**	**523**	**523**	**523**	**523**	**523***
MK09	20×10	2.53	(299,369)	308	308	**307**	**307**	**307**	**307**	307*
MK10	20×15	2.98	(165,296)	212	216	198	**197**	198	198	197*
Déviation_moyenne				*17.92*	*17.26*	*15.76*	*15.27*	*15.33*	*15.03*	*14.98*
#meilleur				*5*	*5*	*7*	*9*	*8*	*9*	*9*

Le tableau 18 montre qu'en utilisant le voisinage ***V6***, on obtient les meilleurs résultats en termes de qualité de solution. En effet, avec ce voisinage les résultats obtenus dévient des bornes inférieures de 15.03% et donne 90% des meilleures solutions fournies par CDDS-HDiv. En utilisant le voisinage *V4*, on atteint également 90% des meilleures solutions de CDDS-HDiv mais avec un écart de 15.27% aux bornes inférieures.

D'après ce tableau, on remarque aussi l'importance de l'utilisation de la notion de bloc pour l'optimisation de la qualité de solutions. En effet, les voisinages utilisant la notion de bloc (***V3***, ***V4, V5, V6***) présentent en moyenne une déviation de 15.34% face aux autres voisinages (***V1***, ***V2***) qui dévient des bornes inférieures de 17.59%.

On constate également que *V5* parvient à dominer *V3* pour toutes les instances en 15 secondes. Cela nous indique que, dans certains cas, *V3* passe beaucoup de temps à diversifier les solutions sans pour autant atteindre les optima. En contrepartie, *V6* domine *V4* pour seulement 9 instances en 15 secondes. On peut en conclure que l'utilisation de l'heuristique dynamique *ECT* permet de diriger la recherche de façon plus efficace.

La différence entre les temps de calcul n'est pas remarquable dans cette classe d'instances, vu que pour 80% des problèmes les méthodes n'ont pas

atteint les solutions optimales et donc parviennent aux 15 secondes allouées. Pour les deux autres instances, on obtient les solutions optimales au niveau de la solution initiale (≈ 0 seconde).

En considérant les meilleures solutions fournies par toutes les variantes proposées de CDDS-HDiv ($C_{meilleur}$), la valeur moyenne du makespan dévie de 14.98% des bornes inférieures en un temps inférieur à 13 secondes par instance.

Dans une seconde étape (tableau 19), on donne une comparaison en termes de qualité de solutions et temps CPU entre la méthode CDDS-HDiv, l'algorithme génétique (GA) de [Pezzella *et al.*, 2008], la méthode de recherche Tabou (TS) développée dans [Mastrolilli et Gambardella, 2000] et l'algorithme génétique (hGA) de [Gao *et al.*, 2008]. Pour chaque méthode, on donne dans la colonne C_{max}, pour chaque instance, la meilleure solution obtenue (parmi les quatre voisinages se basant sur la notion de bloc (*V3*, *V4*, *V5* et *V6*) pour CDDS-HDiv et parmi 5 exécutions pour les autres algorithmes). Dans la même colonne, on donne également entre parenthèses la valeur moyenne du C_{max} sur les différentes applications de chacune des méthodes. La colonne déviation fournit ensuite l'écart entre la valeur trouvée de C_{max} et la borne inférieure, et entre parenthèses la déviation en moyenne sur les différentes exécutions de chacune des méthodes.

Tableau 19. *Comparaison de CDDS-HDiv avec trois autres méthodes (GA, TS, hGA) sur les instances BRdata*

Instance	n×M	flex.	(LB,UB)	CDDS-HDiv		GA		TS		hGA	
				C_{max}	Déviation	C_{max}	Déviation	C_{max}	Déviation	C_{max}	Déviation
MK01	10×6	2.09	(36,42)	**40*** **(40)**	11.11 (11.11)	**40**	11.11	**40** **(40)**	11.11 (11.11)	**40** **(40)**	11.11 (11.11)
MK02	10×6	4.10	(24,32)	**26*** **(26)**	8.33 (8.33)	**26**	8.33	**26** **(26)**	8.33 (8.33)	**26** **(26)**	8.33 (8.33)
MK03	15×8	3.01	(204,211)	**204*** **(204)**	0.00 (0.00)	**204**	0.00	**204** **(204)**	0.00 (0.00)	**204** **(204)**	0.00 (0.00)
MK04	15×8	1.91	(48,81)	**60*** **(60)**	25.00 (25.00)	**60**	25.00	**60** **(60)**	25.00 (25.00)	**60** **(60)**	25.00 (25.00)
MK05	15×4	1.71	(168,186)	173 (173.5)	2.98 (3.27)	173	2.98	173 (173)	2.98 (2.98)	**172*** **(172)**	2.38 (2.38)
MK06	10×15	3.27	(33,86)	**58*** (59)	75.76 (78.79)	63	90.91	**58** (58.4)	75.76 (76.97)	**58** **(58)**	75.76 (75.76)
MK07	20×5	2.83	(133,157)	**139*** **(139)**	4.51 (4.51)	**139**	4.51	144 (147)	8.27 (10.52)	**139** **(139)**	4.51 4.51
MK08	20×10	1.43	523	**523*** **(523)**	0.00 (0.00)	**523**	0.00	**523** **(523)**	0.00 (0.00)	**523** **523**	0.00 (0.00)
MK09	20×10	2.53	(299,369)	**307*** **(307)**	2.68 (2.68)	311	4.01	**307** **(307)**	2.68 (2.68)	**307** **(307)**	2.68 (2.68)
MK10	20×15	2.98	(165,296)	**197*** (197.75)	19.39 (19.85)	212	28.48	198 (199.2)	20.00 (20.73)	**197** **(197)**	19.39 (19.39)
Déviation_moyenne				14.98 (15.34)		17.53		15.41 (15.83)		14.92 (14.92)	
#meilleur				9 (7)		6		7 (6)		10 (10)	
CI-CPU total				480		inconnu		460		566	

La ligne déviation moyenne fournit la moyenne de la déviation sur l'ensemble des instances. La ligne *#meilleur* totalise le nombre de fois où chaque méthode a trouvé la meilleure valeur du makespan. Dans la ligne *CI-CPU* (Computer-Independent CPU) total, on donne le temps de calcul nécessaire à chacune des méthodes (4 voisinages de CDDS-HDiv et 5 exécutions de GA, TS ou hGA). Ces valeurs de temps de calcul sont normalisées selon les coefficients de [Dongarra, 1998].

D'après le tableau 19, on remarque que la méthode CDDS-HDiv donne de meilleurs résultats que l'algorithme génétique (GA) de [Pezzella *et al.*, 2008] et que la méthode de recherche Tabou (TS) de [Mastrolilli et Gambardella, 2000]. Elle reste aussi comparable au meilleur algorithme génétique connu dans la littérature (hGA) de [Gao *et al.*, 2008]. En effet, l'algorithme hGA est meilleur que CDDS-HDiv sur une seule instance (MK05). En considérant toutes les instances, CDDS-HDiv dévie des bornes inférieures de 14.98% face à 14.92% pour l'algorithme génétique de [Gao *et al.*, 2008].

Bien que hGA surpasse CDDS-HDiv en termes de qualité de solutions au niveau d'une seule instance (MK05), CDDS-HDiv est légèrement plus rapide que hGA. En effet, CDDS-HDiv nécessite 480 secondes pour résoudre toutes les instances de cette classe de jeux-test face à 566 secondes pour hGA et 460 secondes pour TS.

5.3.2. Comparaison sur les jeux-test de Barnes et Chambers (1996)

Dans le tableau 20, nous présentons une comparaison entre les différentes approches de CDDS-HDiv développées à partir de différents voisinages considérés (*V1* à *V6*). Le temps maximum alloué à la recherche est de 15 secondes.

Les résultats obtenus en utilisant les deux voisinages *V5* et *V6* sont les plus proches des bornes inférieures. En effet, ces deux approches présentent une déviation de 22.58% face à 30.05% pour *V1*, 28.50% pour *V2*, 22.65% pour *V3* et 22.62% pour *V4*.

En considérant les meilleures solutions fournies par toutes les variantes développées de CDDS-HDiv, cette dernière dévie de 22.54% des bornes inférieures dans le temps imparti de 15 secondes.

Dans cette classe de jeux-test, nous constatons également que *V5* (resp. *V6*) arrive à dominer *V3* (resp. *V4*) pour toutes les instances dans la limite du temps alloué. Notons également que l'utilisation de l'heuristique dynamique (ECT) est efficace aussi pour ce type de problèmes.

Tableau 20. Comparaison des différentes variantes de CDDS-HDiv sur les instances BCdata

instances	n×M	flex.	(LB, UB)	CDDS-HDiv						
				V1	V2	V3	V4	V5	V6	$C_{meilleur}$
mt10x	10×11	1.10	(655, 929)	1037	978	**918**	**918**	**918**	**918**	918*
mt10xx	10×12	1.20	(655, 929)	996	994	**918**	**918**	**918**	**918**	918*
mt10xxx	10×13	1.30	(655, 936)	1002	998	**918**	**918**	**918**	**918**	918*
mt10xy	10×12	1.20	(655, 913)	1008	946	**906**	**906**	**906**	**906**	906
mt10xyz	10×13	1.20	(655, 849)	894	894	851	851	851	**849**	849
mt10c1	10×11	1.10	(655, 927)	1007	1001	930	**928**	**928**	**928**	928
mt10cc	10×12	1.20	(655, 914)	1023	980	912	**910**	911	**910**	910*
setb4x	15×11	1.10	(846, 937)	1001	985	**925**	**925**	**925**	**925**	925*
setb4xx	15×12	1.20	(846, 930)	985	985	**925**	**925**	**925**	**925**	925*
setb4xxx	15×13	1.30	(846, 925)	985	985	**925**	**925**	**925**	**925**	925*
setb4xy	15×12	1.20	(845, 924)	1002	984	**916**	**916**	**916**	**916**	916*
setb4xyz	15×13	1.30	(838, 914)	968	964	908	908	**905**	**905**	905*
setb4c9	15×11	1.10	(857, 924)	979	969	**919**	**919**	**919**	**919**	919
setb4cc	15×12	1.20	(857, 909)	989	989	911	911	**909**	911	909*
seti5x	15×16	1.07	(955, 1218)	1203	1203	**1201**	1203	**1201**	**1201**	1201*
seti5xx	15×17	1.13	(955, 1204)	1199	1199	**1199**	**1199**	**1199**	**1199**	1199*
seti5xxx	15×18	1.20	(955, 1213)	1199	1199	1198	1198	**1197**	**1197**	1197*
seti5xy	15×17	1.13	(955, 1148)	1148	1148	1140	1137	1139	**1136**	1136*
seti5xyz	15×18	1.20	(955, 1127)	1128	1128	**1125**	1126	**1125**	**1125**	1125*
seti5c12	15×16	1.07	(1027,1185)	1180	1180	**1174**	1175	**1174**	1175	1174*
seti5cc	15×17	1.13	(955, 1136)	1166	1166	1138	1137	**1136**	1137	1136*
Déviation_moyenne				*30.05*	*28.50*	*22.65*	*22.62*	*22.58*	*22.58*	***22.54***
#meilleur				*0*	*0*	*11*	*9*	*15*	*14*	*17*

On compare ensuite notre méthode CDDS-HDiv avec l'algorithme génétique (GA) de [Pezzella *et al.*, 2008], la méthode de recherche Tabou (TS) de [Mastrolilli et Gambardella, 2000] et l'algorithme génétique (hGA) de [Gao *et al.*, 2008]. Les éléments de comparaison sont les mêmes que ceux du tableau 19.

Le tableau 21 montre que la méthode CDDS-HDiv surpasse les deux algorithmes génétiques et reste comparable avec les résultats de la méthode de recherche tabou (TS) de Mastrolilli et Gambardella connue comme la meilleure méthode de recherche tabou pour la résolution de ce type de problèmes.

Globalement, CDDS-HDiv fournit 81% des meilleures solutions (17 parmi 21 instances), tandis que TS fournit 86% (18 parmi 21 instances) et hGA fournit 62% (13 parmi 21 instances). Nous notons que l'algorithme TS surpasse CDDS-HDiv au niveau d'une seule instance (mt10xyz).

Mais, CDDS-HDiv reste l'algorithme le plus rapide pour résoudre ces instances; il est environ 2 fois plus rapide que TS et environ 4 fois plus rapide que hGA.

Tableau 21. *Comparaison de CDDS-HDiv avec trois autres méthodes (GA, TS, hGA) sur les instances BCdata*

Instance	*n×M*	*flex.*	(LB,UB)	*CDDS-HDiv*	*GA*	*TS*	*hGA*
mt10x	10×11	1.10	(655, 929)	918* (918)	inconnu	918 (918)	918 (918)
mt10xx	10×12	1.20	(655, 929)	918* (918)	inconnu	918 (918)	918 (918)
mt10xxx	10×13	1.30	(655, 936)	918* (918)	inconnu	918 (918)	918 (918)
mt10xy	10×12	1.20	(655, 913)	906 (906)	inconnu	906 (906)	905* (905)
mt10xyz	10×13	1.20	(655, 849)	849 (850.5)	inconnu	847* (850)	849 (849)
mt10c1	10×11	1.10	(655, 927)	928 (928.5)	inconnu	928 (928)	927* (927.2)
mt10cc	10×12	1.20	(655, 914)	910* (910.75)	inconnu	910 (910)	910 (910)
setb4x	15×11	1.10	(846, 937)	925* (925)	inconnu	925 (925)	925 (931)
setb4xx	15×12	1.20	(846, 930)	925* (925)	inconnu	925 (926.4)	925 (925)
setb4xxx	15×13	1.30	(846, 925)	925* (925)	inconnu	925 (925)	925 (925)
setb4xy	15×12	1.20	(845, 924)	916* (916)	inconnu	916 (916)	916 (916)
setb4xyz	15×13	1.30	(838, 914)	905* (906.5)	inconnu	905 (908.2)	905 (905)
setb4c9	15×11	1.10	(857, 924)	919 (919)	inconnu	919 (919.2)	914* (914)
setb4cc	15×12	1.20	(857, 909)	909* (910.5)	inconnu	909 (911.6)	914 (914)
seti5x	15×16	1.07	(955, 1218)	1201* (1201.5)	inconnu	1201 (1203.6)	1204 (1204)
seti5xx	15×17	1.13	(955, 1204)	1199* (1199)	inconnu	1199 (1200.6)	1202 (1203)
seti5xxx	15×18	1.20	(955, 1213)	1197* (1197.5)	inconnu	1197 (1198.4)	1204 (1204)
seti5xy	15×17	1.13	(955, 1148)	1136* (1138)	inconnu	1136 (1136.4)	1136 (1136.5)
seti5xyz	15×18	1.20	(955, 1127)	1125* (1125.3)	inconnu	1125 (1126.6)	1126 (1126)
seti5c12	15×16	1.07	(1027,1185)	1174* (1174.5)	inconnu	1174 (1174.2)	1175 (1175)
seti5cc	15×17	1.13	(955, 1136)	1136* (1137)	inconnu	1136 (1136.4)	1138 (1138)
dév_moy				22.54 (22.60)	29.56	22.53 (22.63)	22.61 (22.66)
#meilleur				17 (8)	inconnu	18 (7)	13 (10)
CI-CPU total				1260	inconnu	2214	5107

Les expérimentations précédentes nous permettent de conclure que les voisinages se basant sur le principe de bloc fournissent des résultats

nettement meilleurs. On ne s'intéressera donc pour le reste des expérimentations qu'à ces voisinages (soit *V3*, *V4*, *V5* et *V6*).

5.3.3. Comparaison sur les jeux-test de Dauzère-Pérès et Paulli (1997)

Les résultats donnés dans le tableau 22 sur les instances DPdata montrent que CDDS-HDiv surpasse toutes les autres méthodes en termes de déviation moyenne qui est égale à 1.94% face à 7.63% pour GA, 2.01% pour TS et 2.12% pour hGA. De plus, CDDS-HDiv fournit dans 89% des cas les meilleures solutions connues (16 parmi 18 instances), tandis que TS n'en fournit que 72% et l'algorithme hGA seulement 61%.

Tableau 22. *Comparaison entre les quatre variantes de CDDS-HDiv sur les instances DPdata*

Instance	*n×M*	*flex.*	(LB,UB)	*CDDS-HDiv*					*GA*	*TS*	*hGA*
				V3	*V4*	*V5*	*V6*	$C_{meilleur}$			
01a	10×5	1.13	(2505, 2530)	**2518**	2530	2530	2520	2518* (2524.5)	inconnu	2518 (2528)	2518 (2518)
02a	10×5	1.69	(2228, 2244)	**2231**	2244	2232	**2231**	2231* (2234.5)	inconnu	2231 (2234)	2231 (2231)
03a	10×5	2.56	(2228, 2235)	**2229**	2235	2230	2233	2229* (2231.8)	inconnu	2229 (2229.6)	2229 (2229.3)
04a	10×5	1.13	(2503, 2565)	2510	2520	2507	**2503**	**2503*** (2510)	inconnu	2503 (2516.2)	2515 (2518)
05a	10×5	1.69	(2189, 2229)	2220	2219	**2216**	2217	2216* (2218)	inconnu	2216 (2220)	2217 (2218)
06a	10×5	2.56	(2162, 2216)	2199	2214	2201	**2196**	2196* (2202.5)	inconnu	2203 (2206.4)	2196 (2198)
07a	15×8	1.24	(2187, 2408)	2299	**2283**	2293	2307	2283* (2295.5)	inconnu	2283 (2297.6)	2307 (2309.8)
08a	15×8	2.42	(2061, 2093)	**2069**	**2069**	**2069**	**2069**	2069* (2069)	inconnu	2069 (2071.4)	2073 (2076)
09a	15×8	4.03	(2061, 2074)	2069	**2066**	**2066**	**2066**	2066* (2066.8)	inconnu	2066 (2067.4)	2066 (2067)
10a	15×8	1.24	(2178, 2362)	2301	**2291**	2307	2311	2291* (2302.5)	inconnu	2291 (2305.6)	2315 (2315.2)
11a	15×8	2.42	(2017, 2078)	2078	2069	2078	**2063**	2063* (2072)	inconnu	2063 (2065.6)	2071 (2072)
12a	15×8	4.03	(1969, 2047)	2034	2031	2040	2031	2031 (2034)	inconnu	2034 (2038)	2030 (2030.6)
13a	20×10	1.34	(2161, 2302)	**2257**	2265	2260	2259	2257* (2260.3)	inconnu	2260 (2266.2)	2257 (2260)
14a	20×10	2.99	(2161, 2183)	**2167**	2189	2183	2176	2167* (2178.8)	inconnu	2167 (2168)	2167 (2167.6)
15a	20×10	5.02	(2161, 2171)	2167	**2165**	2178	2171	2165* (2170.3)	inconnu	2167 (2167.2)	2165 (2165.4)
16a	20×10	1.34	(2148, 2301)	2259	**2256**	2260	**2256**	2256 (2257.8)	inconnu	2255 (2258.8)	2256 (2258)
17a	20×10	2.99	(2088, 2169)	2143	**2140**	2156	2143	2140* (2145.5)	inconnu	2141 (2144)	2140 (2142)
18a	20×10	5.02	(2057, 2139)	2137	**2127**	2131	2131	2127* (2131.5)	inconnu	2137 (2140.2)	2127 (2130.7)
dév_moy				2.15	2.20	2.27	2.13	**1.94** (2.18)	7.63	2.01 (2.24)	2.12 (2.19)

#meilleur	6	7	3	6	16 (1)	inconnu	13 (0)	11 (2)
CI-CPU total	3600	3600	3600	3596	14396	inconnu	15340	38600

D'autre part, CDDS-HDiv est l'algorithme le plus rapide pour résoudre ces instances ; il est environ 3 fois plus rapide que hGA et de même ordre que TS.

En conclusion, pour cette classe de jeux-test, CDDS-HDiv surpasse toutes les autres méthodes en termes de qualité de solutions et de temps de calcul.

5.3.4. Comparaison sur les jeux-test de Hurink (1994)

Dans cette partie, nous comparons les différentes variantes de CDDS-HDiv entre elles en considérant les différents types d'instances de [Hurink, 1994]. Les tableaux 23, 24 et 25 présentent les déviations de chaque variante par rapport aux bornes inférieures. Ces déviations sont présentées en fonction des tailles des instances.

On peut constater, d'après le tableau 23, que pour les petites instances (M=5 et $N \leq 20$, c'est-à-dire les instances la01 à la15), les voisinages *V5* et *V6* obtiennent de bonnes performances. Cependant, lorsque la taille de l'instance augmente, *V3* et *V4* surpassent les autres voisinages. On peut tout de même noter qu'à aucun moment un voisinage ne domine totalement l'autre sur toutes les instances. En moyenne, *V4* et *V5* présentent une déviation de 2.85%, en deuxième position vient *V3* (2.88%) suivi de *V6* avec 3.14%.

Tableau 23. Comparaison entre les quatre variantes de CDDS-HDiv sur les instances Edata de Hurink

Instances Edata	*n×M*	*CDDS-HDiv*				Dév_moy
		V3	*V4*	*V5*	*V6*	
la01 à la05	10×5	1.64	0.84	0.43	0.00	**0.73**
la06 à la10	15×5	0.14	0.50	0.05	0.07	**0.19**
la11 à la15	20×5	1.58	1.35	0.60	0.93	**1.12**
la16 à la20	10×10	0.49	0.49	0.89	1.58	**0.86**
la21 à la25	15×10	5.70	5.70	5.88	6.67	**5.99**
la26 à la30	20×10	3.96	4.39	4.66	4.85	**4.47**
la31 à la35	30×10	0.42	0.42	0.77	0.85	**0.61**
la36 à la40	15×15	9.10	9.10	9.49	10.12	**9.45**
Dév_moy		**2.88**	**2.85**	**2.85**	**3.14**	**2.93**

Les mêmes conclusions sont confirmées par les résultats des expérimentations sur les jeux-test Rdata de [Hurink, 1994] présentées dans le tableau 24. En effet, *V3* et *V4* présentent de meilleures performances pour les instances de plus grande taille. On peut noter également que l'utilisation de l'heuristique dynamique *ECT* n'est plus efficace avec l'augmentation de la taille des instances.

On peut tout de même noter que la méthode CDDS-HDiv devient de plus en plus performante avec l'augmentation de flexibilité avec les instances de

Rdata. En effet, elle présente un pourcentage de déviation égal à 1.62% par rapport aux bornes inférieures face à 2.93% pour Edata.

Tableau 24. *Comparaison entre les quatre variantes de CDDS-HDiv sur les instances Rdata de Hurink*

Instances Rdata	n×M	CDDS-HDiv				déviation_moyenne
		V3	V4	V5	V6	
la01 à la05	10×5	0.28	0.38	0.34	0.11	**0.28**
la06 à la10	15×5	0.16	0.43	0.07	0.10	**0.19**
la11 à la15	20×5	0.60	0.45	0.27	0.16	**0.37**
la16 à la20	10×10	1.64	1.64	2.02	2.06	**1.84**
la21 à la25	15×10	3.82	3.82	3.95	4.62	**4.05**
la26 à la30	20×10	0.66	0.74	0.84	1.19	**0.86**
la31 à la35	30×10	0.22	0.43	0.29	0.48	**0.36**
la36 à la40	15×15	4.85	4.85	4.96	5.10	**4.94**
Déviation_moyenne		**1.53**	**1.59**	**1.59**	**1.73**	**1.61**

D'après le tableau 25, les performances de chaque variante sont très nettes même pour les instances de grande taille. Le voisinage *V3* fournit les meilleures solutions et présente une déviation de 0.15% par rapport aux bornes inférieures, suivi de *V6* avec 0.16%, puis *V4* et *V5* qui présentent une déviation de 0.21%.

Tableau 25. *Comparaison entre les quatre variantes de CDDS-HDiv sur les instances Vdata de Hurink*

Instances Vdata	n×M	CDDS-HDiv				déviation_moyenne
		V3	V4	V5	V6	
la01 à la05	10×5	0.25	0.20	0.25	0.33	**0.26**
la06 à la10	15×5	0.08	0.08	0.03	0.13	**0.08**
la11 à la15	20×5	0.00	0.00	0.00	0.00	**0.00**
la16 à la20	10×10	0.00	0.00	0.00	0.00	**0.00**
la21 à la25	15×10	0.70	1.11	1.11	0.70	**0.90**
la26 à la30	20×10	0.11	0.27	0.27	0.11	**0.19**
la31 à la35	30×10	0.02	0.06	0.06	0.02	**0.04**
la36 à la40	15×15	0.00	0.00	0.00	0.00	**0.00**
Déviation_moyenne		**0.15**	**0.21**	**0.21**	**0.16**	**0.18**

Une amélioration en termes de qualité de solutions est également notée pour cette classe d'instances qui présente une augmentation de flexibilité qui varie entre 2.50 et 7.50 ressources par opération.

Les temps de calcul sont évidemment du même ordre de grandeur (voir tableau 26) bien qu'en termes de pourcentage *CDDS*-HDiv-*V6* et *CDDS*-HDiv-*V4* atteignent plus de fois le temps maximum alloué (15 secondes par instance). Ceci peut se justifier par le fait que ces deux variantes utilisent une heuristique dynamique qui permet de re-calculer les dates de début et de fin de toutes les opérations et leurs ré-affectations. On constate également que

le temps de calcul augmente fortement pour les grosses instances ayant beaucoup de flexibilité.

En considérant toutes les instances de [Hurink, 1994], et étant donné que les temps de calcul sont du même ordre pour tout type d'instances, on peut conclure que *CDDS*-HDiv-*V3* est globalement plus performant que toutes les autres variantes.

Tableau 26. *Comparaison entre les temps de calcul moyens (en secondes) des variantes de CDDS-HDiv*

Instances	***CDDS-HDiv***			
	V3	***V4***	***V5***	***V6***
Edata de Hurink	446.34	587.38	522.45	565.45
Rdata de Hurink	532.77	598.99	588.24	602.00
Vdata de Hurink	597.7	608.45	596.84	631.24
CI-CPU total	1576.81	1794.82	1707.53	1798.69

Le tableau 27 donne une comparaison de notre méthode CDDS-HDiv avec la méthode de recherche Tabou (TS) de [Mastrolilli et Gambardella, 2000] et l'algorithme génétique (hGA) de [Gao *et al.*, 2008]. Le tableau donne les pourcentages de déviations des valeurs de makespan trouvées par les différentes méthodes par rapport aux meilleures bornes inférieures connues de la littérature.

Tableau 27. *Comparaison de CDDS-HDiv avec deux autres méthodes (TS, hGA) sur les instances de Hurink*

Instance	***n×M***	**Edata (*flex.* = 1.15)**			**Rdata (*flex.* = 2)**			**Vdata (*flex.* ∈ [2.50,7.50])**		
		CDDS-HDiv	***TS***	***hGA***	***CDDS-HDiv***	***TS***	***hGA***	***CDDS-HDiv***	***TS***	***hGA***
mt06/10/20	6×6 10×10 20×5	0.00 (0.05)	0.00 (0.10)	0.00 (0.10)	0.34 (0.47)	0.34 (0.36)	0.34 (0.34)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
la01 – la05	10×5	0.00 (0.73)	0.00 (0.00)	0.00 (0.00)	0.11 (0.28)	0.11 (0.24)	0.07 (0.07)	0.13 (0.23)	0.00 (0.11)	0.00 (0.00)
la06 – la10	15×5	0.00 (0.19)	0.00 (0.00)	0.00 (0.00)	0.03 (0.19)	0.03 (0.08)	0.00 (0.00)	0.00 (0.00)	0.00 (0.03)	0.00 (0.00)
la11 – la15	20×5	0.29 (1.12)	0.29 (0.29)	0.29 (0.29)	0.02 (0.37)	0.02 (0.02)	0.00 (0.00)	0.00 (0.00)	0.00 (0.01)	0.00 (0.00)
la16 – la20	10×10	0.49 (1.15)	0.00 (0.00)	0.02 (0.02)	1.64 (1.90)	1.73 (1.77)	1.64 (1.64)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
la21 – la25	15×10	5.70 (6.27)	5.62 (5.93)	5.60 (5.66)	3.82 (4.26)	3.82 (4.38)	3.57 (3.69)	0.70 (1.01)	0.70 (0.85)	0.60 (0.68)
la26 – la30	20×10	3.96 (4.84)	3.47 (3.76)	3.28 (3.32)	0.66 (0.98)	0.59 (0.76)	0.64 (0.72)	0.11 (0.19)	0.11 (0.18)	0.11 (0.13)
la31 – la35	30×10	0.42 (0.83)	0.30 (0.32)	0.32 (3.32)	0.22 (0.41)	0.09 (0.14)	0.09 (0.12)	0.02 (0.04)	0.01 (0.03)	0.00 (0.00)
la36 – la40	15×15	9.10 (9.88)	8.99 (9.13)	8.82 (8.95)	4.85 (4.97)	3.97 (4.47)	3.86 (3.92)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)

Déviation_moyenne	2.32 (2.78)	2.17 (2.18)	2.13 (2.40)	1.34 (1.54)	1.24 (1.36)	1.19 (1.21)	0.12 (0.16)	0.095 (0.13)	0.082 (0.09)
CI-CPU par classe	2121	103530	inconnu	2322	102990	inconnu	2494	102600	inconnu
CI-CPU total	***CDDS-HDiv***	6937							
	TS	103050							
	hGA	440030							

Le tableau 27 montre que la méthode CDDS-HDiv reste comparable avec les résultats de la méthode de recherche Tabou (TS) de [Mastrolilli et Gambardella, 2000] et l'algorithme génétique de [Gao *et al.*, 2008].

En termes de temps de calcul, on remarque que CDDS-HDiv dépense moins de temps de calcul pour résoudre ce type de jeux-test. En effet, CDDS-HDiv est 15 fois plus rapide que TS et environ 63 fois plus rapide que hGA.

5.4. Conclusion sur le problème de job shop flexible

Dans ce chapitre, nous avons étudié le problème de job shop flexible. Pour le résoudre, nous avons proposé six variantes de la méthode CDDS-HDiv qui diffèrent entre elles par leur stratégie d'exploration des divergences. Nous avons testé ces variantes sur quatre classes de jeux-test, à savoir celles de [Brandimarte, 1993], de [Barnes et Chambers, 1996], de [Dauzère-Pérès et Paulli, 1997] et de [Hurink, 1994].

Les expérimentations ont permis de comparer les variantes entre elles. Il en résulte que *CDDS-V4* et *CDDS-V6* sont, en règle générale, les meilleures variantes en termes de qualité de solutions et de temps de calcul (tableau 28). Néanmoins, dans certains cas, ces variantes sont dominées par *CDDS-V3* (pour les instances *Rdata* et *Vdata* de Hurink).

Bien que ces deux variantes (*CDDS*-HDiv-*V4* et *CDDS*-HDiv-*V6*) fournissent les meilleures solutions, elles nécessitent un peu plus de temps. En effet, elles présentent une déviation moyenne de 7.46% au bout de 45 secondes. Ce résultat n'est pas surprenant vu que l'utilisation de l'heuristique dynamique (ECT) dans la détermination du voisinage nécessite un temps supplémentaire pour calculer les nouvelles dates de début et les nouvelles affectations.

Tableau 28. Récapitulation des résultats fournis par les différentes stratégies de CDDS-HDiv

Instances	***Nb***	***CDDS-V3***		***CDDS-V4***		***CDDS-V5***		***CDDS-V6***	
		Dév	***CPU***	***Dév***	***CPU***	***Dév***	***CPU***	***Dév***	***CPU***
BRdata	10	15.76	120	15.27	120	15.33	120	**15.03**	120
BCdata	21	22.65	315	22.62	315	**22.58**	315	**22.58**	315
DPdata	18	2.15	3600	2.20	3600	2.27	3600	**2.13**	3591
Hurink Edata	43	2.88	446	**2.85**	587	**2.85**	522	3.14	565
Hurink Rdata	43	**1.53**	533	1.59	599	1.59	588	1.73	602
Hurink Vdata	43	**0.15**	598	0.21	608	0.21	597	0.16	631
Moyenne		**7.52**	935.33	**7.46**	971.5	**7.47**	957	**7.46**	970.67

Nous avons également constaté que l'utilisation de l'heuristique dynamique (*ECT*) pour l'exploration du voisinage (CDDS-V4 et CDDS-V6) présente un intérêt pour les instances de petite taille et ayant peu de flexibilité.

Nous avons aussi comparé nos résultats avec d'autres méthodes dont les performances ont été prouvées pour ce type de problème (tableau 29). Les résultats montrent l'efficacité de notre méthode CDDS-HDiv qui fournit de bonnes solutions. Ces dernières surpassent celles de l'algorithme génétique de [Pezzella *et al.*, 2008] sur tous les types d'instances, la méthode de recherche tabou de [Mastrolilli et Gambardella, 2000] sur les instances de [Brandimarte, 1993] et de [Dauzère-Pérès et Paulli, 1997], et l'algorithme génétique hGA sur les instances de [Barnes et Chambers, 1996] et sur les instances de [Dauzère-Pérès et Paulli, 1997].

Tableau 29. Récapitulatif des résultats fournis par CDDS-HDiv et (GA, TS, hGA)

Instances	*Nb*	*CDDS-HDiv*	*GA*	*TS*	*hGA*	*déviation CDDS-HDiv par rapport à*		
						GA	*TS*	*hGA*
BRdata	10	14.98	17.53	15.14	**14.92**	-2.55	-0.16	0.06
BCdata	21	22.54	29.56	**22.53**	22.61	-7.02	0.01	-0.07
DPdata	18	**1.94**	7.63	2.01	2.12	-5.69	-0.07	-0.18
Hurink Edata	43	2.32	6.00	2.17	**2.13**	-3.68	0.15	0.19
Hurink Rdata	43	1.34	4.42	1.24	**1.19**	-3.08	0.10	0.15
Hurink Vdata	43	0.12	2.04	0.095	**0.082**	-1.92	0.03	0.04

Nous avons enfin constaté que la méthode CDDS-HDiv est de plus en plus performante avec l'augmentation de la flexibilité des ressources (instances Hurink Vdata).

Signalons que pour ce type de problème, nous n'avons pas intégré de bornes inférieures performantes car nous n'en avons pas obtenu ni trouvé dans la littérature.

Conclusions et perspectives

Ce document synthétise les travaux de recherche effectués afin de résoudre différents problèmes d'ordonnancement flexible.

Ces travaux sont positionnés au croisement de deux champs thématiques. Le premier concerne l'amélioration des performances des systèmes de production discrets et de la résolution de problèmes d'ordonnancement de nature flexible. Le second champ de recherche attenant à ces travaux est celui des méthodes à base de divergence qui appartiennent au domaine général des méthodes arborescentes. Ce domaine a connu un essor important ces dernières années consécutif au développement de la première méthode connue sous le nom de LDS et développée par Harvey et Ginsberg en 1995.

Il n'en demeure pas moins, et ceci ressort des recherches qui ont été menés dans ce travail mais aussi dans des travaux connexes, que l'exploitation de ces méthodes comme outils d'optimisation ne peut se faire par une simple application du l'algorithme initial de la méthode LDS. En effet, pour espérer atteindre des performances de bonne qualité sur des instances de taille importante, il est impératif d'adapter et d'améliorer cette méthode canonique.

Une nouvelle méthode à base de divergence (CDDS), couplant deux méthodes existantes, CDS et DDS, a été proposée dans ce travail. Cette méthode est enrichie par deux mécanismes permettant de limiter l'application des divergences et donc de restreindre l'espace de recherche. Ces deux mécanismes sont l'utilisation de bornes et l'utilisation d'heuristique d'application des divergences.

La méthode CDDS que nous avons proposée a ensuite été adaptée pour résoudre trois types de problèmes d'ordonnancement flexible : le flow shop hybride à plusieurs étages, puis le cas particulier à deux étages et enfin le job shop flexible. Les différents mécanismes de CDDS ont été évalués et testés sur différents jeux de données tirés de la littérature.

L'exploitation des résultats nous a permis d'aboutir aux conclusions générales suivantes :

- la méthode CDDS additionnée d'un mécanisme de calcul de bornes est efficace sur les problèmes de type flow shop hybride à plusieurs étages. Elle fournit, à notre connaissance, les meilleures

solutions de la littérature, que ces problèmes soient équilibrés ou non ;

- la présence d'un étage critique conditionne nettement la qualité des solutions. En effet, la méthode est plus efficace si un tel étage existe ;
- la position de l'étage critique n'influe pas sur les résultats obtenus ;
- l'intégration de bornes inférieures permet de gagner aussi bien en termes de temps de calcul qu'en termes de qualité de solutions ;
- adaptée aux problèmes à deux étages, CDDS avec calcul de bornes a également prouvé son efficacité en termes de qualité de solutions ainsi qu'en termes de temps d'exécution ;
- l'intégration du calcul de bornes inférieures spécifiques à chaque type de problème au sein de la méthode de résolution s'est révélée d'un grand intérêt pour les performances de la méthode ;
- CDDS est de plus en plus performante avec l'augmentation de la flexibilité des ressources quel que soit le type de problème considéré ;
- adaptée au job shop flexible, CDDS couplée avec des heuristiques sur l'application des divergences fournit de très bonnes solutions. Elle obtient en particulier les meilleurs résultats pour les instances de Dauzère-Pérès et Paulli (1997). Pour les autres jeux de données, elle est très proche des meilleures méthodes connues pour la résolution de ce type de problème.

L'intégration de calcul de bornes pour la résolution du problème de flow shop hybride s'étant avéré particulièrement performante, une des perspectives de ce travail consiste, pour le cas du problème de type job shop flexible, à proposer des bornes inférieures et à les intégrer au sein de la méthode CDDS afin d'évaluer leur apport pour la résolution du job shop flexible.

Par ailleurs, les mécanismes de propagation de contraintes utilisés dans notre méthode CDDS sont très simples. L'étude et l'intégration au sein de notre méthode de mécanismes de propagation dédiées aux problèmes d'ordonnancement flexible pourraient également contribuer à son amélioration en réduisant la taille de l'espace de recherche. Il serait notamment profitable d'appliquer les principes du raisonnement énergétique qui a déjà fait ses preuves sur les problèmes d'ordonnancement à machines parallèles et problèmes de flow shop hybride.

Une autre voie de recherche pourra également être considérée est celle qui consiste à considérer les problèmes de job shop flexible à gammes non linéaires, connus sous le nom de problèmes d'atelier généralisés (generalized shop floor problems).

Une autre voie de recherche intéressante consiste enfin à essayer d'adapter ces méthodes pour les problèmes d'ordonnancement

multicritères. En effet, un critère comme le makespan considéré de manière unique a souvent peu de sens dans des cas concrets.

Bibliographie

Barnes J.W., Chambers J.B. (1996). "Flexible job shop scheduling by tabu search", Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09, http://www.cs.utexas.edu/users/jbc/.

Beck J.C., Perron L., (2000). "Discrepancy-Bounded Depth First Search"**,** Proceedings of the second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CPAIOR'00), Paderborn (Germany).

Brah S.A., Hunsucker J.L., (1991). "Branch and bound method for the flow shop with multiple processors", European Journal of Operational Research 51, 88-99.

Brandimarte P.J., (1993). "Routing and scheduling in a flexible job shop by tabu search", Annals of Operations of Research 41, 157-183.

Brucker P., (1988). "An efficient algorithm for the job-shop problem with two jobs", Computing 40, 353-359.

Brucker P., (2004). "Scheduling Algorithms". Springer, Berlin.

Brucker P., Schlie R., (1990). "Job shop scheduling with multi-purpose machines", Computing 45, 369–375.

Carlier J., (1987). "Scheduling jobs with release dates and tails on identical machines to minimize the makespan", European Journal of Operational Research 29, 298-306.

Carlier J., Neron E., (2000). "An exact method for solving the multi-processor flow-shop", RAIRO Operations Research 34, 1-25.

Chambers J.B., Barnes J.W., (1996). "New tabu search results for the job shop scheduling problem". Technical Report Series ORP 96-06, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin.

Dauzère-Pérès S., Paulli, J., (1994). "Solving the general job-shop scheduling problem. Management". Report Series, vol. 182. Erasmus University Rotterdam, Rotterdam School of Management, Rotterdam.

Dauzère-Pérès S., Paulli J., (1997). "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search". Annals of Operations Resarch, 70, 281–306.

Dechter R., (2003). "Constraint processing", Morgan Kaufmann, San Francisco.

Engin O., Döyen A., (2004), "A new approach to solve hybrid flow shop scheduling problems by artificial immune system", Future Generation Computer Systems 20, 1083-1095.

Esquirol P., Lopez P., (1999). "L'ordonnancement", Economica, Collection Gestion, Série : "Production et techniques quantitatives appliquées à la gestion", Paris, 1999, ISBN 2-7178-3798-1.

Fattahi P., Saidi Mehrabad M., Jolai F., (2007). "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems", Journal of Intelligence and Manufacturing, 18:331-342.

Fisher H., Thompson G.L., (1963). "Probabilistic learning combinations of local job-shop scheduling rules". In Industrial Scheduling, J.F. Muth and G.L. Thompson (eds.), Prentice-Hall, Englewood Cliffs NJ. 225-251.

Garey M. R., Johnson D. S., (1979). "Computers and intractability: a guide to the theory of NP-Completeness". W.H. Preeman and Company.

Gao J., Gen M., Sun L., Zhao X., (2007). "A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems", Computers and Industrial Engineering 53, 149-162.

Gao J., Sun L., Gen M., (2008). "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems", Computers & Operations Research. New York: Sep 2008. Vol. 35, Iss. 9; p. 2892.

Gilmore P.C., Gomory R.E. (1964). "Sequencing a one-state Variable Machine: A Solvable Case of the Traveling Salesman Problem". Operations Research, 12, 655-679.

Graham, R.-L., Lawler, E.-L., Lenstra, J.-K., Rinnooy Kan, A.-H.-G. (1979). "Optimization and approximation in deterministic sequencing and scheduling : a survey". Annals of Discrete Mathematics, 5 :287–326.

Guinet A., Solomon. M.M., Kedia P.K., Dussauchoy A., (1996). "A computational study of heuristics for two-stage flexible flowshop", International Journal of Production Research 34, 1399-1415.

Gupta J.N.D., (1988). "two-stage hybrid flow shop scheduling problem", Journal of Operational Research Society 39, 359-364.

Gupta J.N.D., Hariri A.M.A., Potts C.N., (1997). "Scheduling a two-stage hybrid flow shop with parallel machines at the first stage", Annals of Operations Research 69, 171-191.

Gupta J.N.D., Tunc E. A., (1991). "Schedules for a two stage hybrid flow shop with parallel machines at the second stage", International Journal of Production Research 29, 1489-1502.

Hansen P., Mladenovic N., (2001). "Variable neighborhood search: principles and applications". European Journal of Operational Research, 130:449–467.

Haralick R., Elliot G., (1980). "Increasing tree search efficiency for constraint satisfaction problems" , Artificial Intelligence, 14 :263-313.

Harvey, W.-D., Ginsberg, M.-L., (1995). "Limited discrepancy search". Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), 607–613.

Haouari M., Hidri L., Gharbi A., (2006). " Optimal scheduling of a two-stage hybrid flow shop", Mathematical Methods of Operational Research 64, 107-124.

Haouari M., M'Hallah R., (1997). "Heuristic algorithms for the two-stage hybrid flow shop problem", Operations Research Letters 21, 43-53.

Ho, N. B., Tay, J. C., and Lai, E. M.-K., (2007). "An effective architecture for learning and evolving flexible job-shop schedules". European Journal of Operational Research, 179(2), 316–333.

Hoogeveen J.A., Lenstra J.K., Veltman B. (1996). "*Preemptive scheduling in a two-stage multiprocessor Flow shop is NP-Hard*", European Journal of Operational Research 89, 172-175.

Hunsucker J.L., Shah J.R., (1994). "Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment". European Journal of Operational Research 72, 102-114.

Hurink J., Jurisch B., Thole M. (1994). "Tabu search for the job-shop scheduling problem with multi-purpose machines", OR Spektrum 15, 205-215.

Jackson J.R., (1956). "An Extension of Johnson's Results on Job Lot Scheduling". Naval Research Logistic Quarterly, 1, 61-68.

Johnson S. M. (1954). "Optimal two and three stage production schedules with set-up time included". Naval Research Logistics Quarterly, 1:61-68.

Jurisch B. (1992). "Scheduling Jobs in Shops with Multi-Purpose Machines". Thèse de doctorat.

Kacem I., Hammadi S., Borne P., (2002). "Approach by localization and multiobjective evolu-tionary optimization for flexible job shop scheduling problems", IEEE Transactions on Systems, Man and Cybernetics, Part C 32(1), 408–419.

Karoui W., Huguet M.J., Lopez P., Naanaa W., (2007). "YIELDS: A yet improved limited discrepancy search for CSPs", 4th International Conference, CPAIOR 2007, Bruxelles, pp.99-111.

Kernighan, B.W., Lin, S., (1970). "An efficient heuristic procedure for partitioning graphs". Bell System Technical Journal 49, 291-307.

Kis T., Pesch E., (2005). "A review of exact solution methods for the non-preemptive multiprocessor flowshop problem", European Journal of Operational Research 164, 592-608.

Korf R.-E., (1996). "Improved limited discrepancy search". In Proceedings of the 13th AAAI/IAAI, 286–291, Portland, Oregon.

Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G., Shmoys, D. B., (1989). "Sequencing and scheduling: algorithms and complexity". Technical Report NFI 11.89/03, Eindhoven University of Technology, Eindhoven, The Netherlands.

Lawrence S., (1984). "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques". Graduate School of Industrial Administration, Carnegie-Mellon University.

Le Pape C. and Baptiste, Ph. (1999). "Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem". Journal of Heuristics, 5 :305–325.

Lee C.Y., Vairaktarakis G.L., (1994). "Minimizing makespan in hybrid flow shop", Operations Research Letters 16, 149-158.

Leon V.J., Ramamoorthy B., (1997). "An adaptable problem space based search method for flexible flow line scheduling", IIE Transactions 29, 115-125.

Lopez P.,(1991). "Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources". Thèse de Doctorat, Université Paul Sabatier, Toulouse.

Marsland T.A., Björnsson Y., (2000). "Variable Depth Search", Advances in Computer Games 9, University of Maastricht p. 5-20.

Mastrolilli M., Gambardella L. M., (2000). "Effective neighbourhood functions for the flexible job shop problem", Journal of Scheduling 3, 3–20.

Mati Y., (2002). "Les Problèmes d'Ordonnancement dans les Systèmes de Production Automatisés : Modèles, Complexité et Approches de Résolution", thèse de doctorat, Université de Metz.

Messeguer P., Walsh T., (1998). "Interleaved and discrepancy based search". ECAI 98.

Milano M., Roli A., (2002). "On the relation between complete and incomplete search: an informal discussion". Proceedings CPAIOR'02, p.237–250, Le Croisic, France.

Negenman E.G., (2001). "Local search algorithms for the multiprocessor flow shop scheduling problem", European Journal of Operational Research 128, 147-158.

Néron E., Baptiste Ph., Gupta JND., (2001). "Solving hybrid flow shop problem using the energetic reasoning and global operations", Omega 29, 501-511.

Nowicki E., Smutnicki C., (1996). "A fast taboo search algorithm for the permutation flow-shop problem", European Journal of Operational Research 91, 160-175.

Paulli J., (1995). "A hierarchical approach for the FMS scheduling problem". European Journal of Operational Research, 86, 32-42.

Perregaard M., (1995). "Branch-and-Bound methods for the multi-Processor Job Shop and Flow Shop scheduling problems", Master's Thesis, Datalogisk institut Kubenhavns Universitet.

Pezzella F., Morganti G., Ciaschetti G., (2007). "A genetic algorithm for the flexible job-shop scheduling problem", Computers & Operations Research, Article in Press, Corrected Proof, http://dx.doi.org/10.1016/j.cor.2007.02.014.

Pinedo M., (1995). "Scheduling - Theory, Algorithms, and Systems". Prentice Hall, Englewood Cliffs.

Prcovic N., (2002). "Quelques variants de LDS". JNPC'02, 195-208.

Portmann M.C., Vignier A., Dardilhac. D., Dezalay D., (1998). "Branch and bound crossed with GA to solve hybrid flowshops", European Journal of Operational Research 107 (2), 389-400.

Riane F., Artiba A., Elmaghraby S.E. (1998). "A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan", European Journal of Operational Research 109, 321-329.

Sotskov Y.N., (1985). "Optimal servicing two jobs with regular criterion". Automation of Designing Process, Minsk, 86-95.

Schuurman P., Woeginger G.J., (2000). "A polynomial time approximation scheme for the two-stage multiprocessor fow shop problem", Theoretical Computer Science 237, 105-122.

Tanaev V. S., Sotskov Y. N., Strusevich V. A., (1994). "Scheduling Theory. Multi-Stage Systems". Kluwer, The Netherlands.

Tay, J.C., Ho, N.B., (2008). "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems". Computers & Industrial Engineering, 54(3), 453–473.

Uetake T., Tsubone H., Ohba M., (1995). "A production scheduling system in a hybrid flow shop", International Journal of Production Economics 41, 395-398.

Vandevelde A., (1994). "Minimizing the makespan in a multiprocessor flow shop", Thèse de doctorat, Eindhoven University of Technology, the Netherlands.

Vignier A., (1997). "Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachines (flow shop hybride)", PhD Thesis, University of Tours, France.

Vilcot G., (2007). "Algorithmes approchés pour des problèmes d'ordonnancement multicritère de type job shop flexible et job shop multiressource", Thèse de doctorat, université François-Rabelais, Tours, France.

Walsh T., (1997). "Depth-bounded discrepancy search". Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI).

Xia W., Wu Z., (2005). "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems", Computers and Industrial Engineering, 48-2, 409-425.

Zhang, H. and Gen, M. (2005). "Multistage-based genetic algorithm for flexible job-shop scheduling problem". Journal of Complexity International, 11, 223–232.

Résumé

Au cours de ces dernières années, les problèmes d'ordonnancement flexible ont largement attiré l'attention des chercheurs dans le domaine de la recherche opérationnelle. Ces problèmes présentent une difficulté supplémentaire du fait qu'une opération peut être exécutée par une ou plusieurs ressources devant être choisie(s) parmi d'autres candidates. L'objectif étant alors d'affecter et de séquencer les opérations sur les ressources en minimisant la durée d'exécution totale ou *makespan*. Dans cette étude, nous proposons de résoudre trois types de problèmes d'ordonnancement flexible : le flow shop hybride à plusieurs étages, à deux étages et le job shop flexible, en utilisant les méthodes arborescentes à base de divergences. Une étude expérimentale exhaustive a prouvé l'efficacité des différentes approches proposées pour les différents types de problèmes.

Mots clés : Ordonnancement, flexibilité des ressources, méthodes arborescentes à divergences, job shop flexible, flow shop hybride.

Abstract

Nowadays, the flexible scheduling problems drew attention of researchers in the field of operational research. These problems introduce an additional difficulty due to the fact that an operation can be carried out by one or several machines to be chosen among other candidates. The objective is to assign and to sequence the operations on the resources so that the total duration of the schedule is minimized. In this study, we offer to solve three flexible problems: the hybrid flow shop on several stages, the two-stage hybrid flow shop, and the flexible job shop problem, by using tree methods based on discrepancies. An exhaustive experimental study proved the efficiency of the different approaches developed for considered problems.

Key words: Scheduling, resource flexibility, discrepancy-based search, flexible job shop, hybrid flow shop.