Bornes basées sur les ensembles interdits pour le problème d'ordonnancement de projet à moyens limités

T. Garaix, C. Artigues et S. Demassey

Laboratoire d'Informatique de l'Université d'Avignon et des pays de Vaucluse,
CNRS FRE 2487
339 chemin des Meinajariès, BP128 84911 Avignon Cedex 9
thierry.garaix@lia.univ-avignon.fr
christian.artigues@lia.univ-avignon.fr
sophie.demassey@lia.univ-avignon.fr

Résumé Le problème d'ordonnancement de projet à moyens limités revient à planifier l'exécution d'un ensemble de tâches utilisant des ressources disponibles en quantité limitée, de façon à minimiser la durée totale du projet. La résolution de ce problème constitue un véritable défi puisqu'on peut trouver encore dans la littérature des problèmes à 60 tâches non résolus.

La première partie du mémoire propose une analyse de ces instances, l'objectif étant de déterminer les caractéristiques spécifiques des problèmes non résolus. En particulier, la structure des « ensembles minimaux interdits », c'est-à-dire d'ensembles minimaux de tâches ne pouvant être exécutées en parallèle, est étudiée.

Sur la base de cette analyse, la seconde partie du mémoire propose une méthode de calcul de bornes inférieures et supérieures du RCPSP, utilisant une résolution exacte d'une relaxation du problème qui prend en compte seulement une partie des ensembles minimaux interdits. Les bornes ainsi obtenues sont testées sur des instances de la dans la littérature. Plusieurs bornes sont améliorées.

Mots-Clefs. Ordonnancement ; RCPSP ; Ensembles minimaux interdits ; bornes inférieures ; heuristiques ; ensembles critiques.

1 Introduction

Ordonnancer un projet, c'est attribuer aux tâches qui le constituent des dates d'exécution et des ressources à consommer. Le but de cet ordonnancement est d'optimiser un certain critère tel que la minimisation de la durée d'exécution totale du projet ou la minimisation des retards de chaque tâche par rapport à des contraintes de délai...

Dans ce article, nous nous intéressons au problème d'ordonnancement de projet à moyens limités ou RCPSP (Resource Constrained Project Scheduling

Problem) dans le cas de la minimisation de la durée d'exécution du projet (le *makespan*).

Ce problème, fortement combinatoire, est NP-Difficile [BLR83]. Les méthodes exactes existantes ne peuvent s'appliquer qu'à de petites instances (moins de 100 tâches). Des instances du RCPSP de 60 tâches restent toujours ouvertes. La littérature propose de nombreuses méthodes ([KP01]) permettant d'obtenir des bornes inférieures ou supérieures encadrant la valeur d'une solution optimale. Ces méthodes sont majoritairement testées sur les jeux d'instances de la PSPLIB, mises à disposition par Kolish, Sprecher et Drexl [KSD95]. Les principales méthodes heuristiques calculant des bornes supérieures sont recensées et comparées par Kolish et Hartmann dans un état de l'art récent [KH04]. Parmi les plus performantes sur les instances de la PSPLIB, on trouve les méthodes de Valls et al. [VBQ04,VQB04] et de Palpant et al [PAC03]. De nombreuses méthodes de calcul de bornes inférieures ont aussi été développées. Les plus récentes sont basées sur la résolution d'une relaxation linéaire du problème [CN00] parfois précédée d'un prétraitement par propagation de contraintes [BD04,DAM03,BK00] ou d'une relaxation lagrangienne [MSSU03]. Dans ce dernier cas, cette relaxation est ensuite utilisée pour générer une solution réalisable par une heuristique.

Nous présentons le problème dans la section 2. Dans un premier temps, nous analysons les instances KSD de la PSPLIB (section 3). Dans la section 4, nous proposons une méthode originale de calcul de bornes inférieures et supérieures basée sur cette analyse, utilisant une représentation des contraintes de ressources par des ensembles de tâches ne pouvant pas s'exécuter simultanément, et relâchant le problème en prenant en compte progressivement ces ensembles par cardinalité croissante. Une borne inférieure correspond à une solution optimale d'une telle relaxation. De cette relaxation, on déduit également une borne supérieure de manière heuristique. Un des objectifs est d'obtenir de bons résultats sur les problèmes à 60 tâches de la littérature non encore optimalement résolus.

2 Présentation générale du problème d'ordonnancement RCPSP

2.1 Description du problème

Définitions et Notations Le problème du RCPSP a pour données un ensemble de tâches $A = \{A_1, \ldots, A_n\} \cup \{A_0, A_{n+1}\}$ et un ensemble de ressources $R = \{1, \ldots, m\}$. Chaque ressource k possède une disponibilité limitée R_k. Chaque tâche i est définie par sa durée $p_i \in \mathbf{N}$ et sa demande $r_{ik} \in \mathbf{N}$ sur chacune des ressources $k = 1, \ldots, m$. A_0 et A_{n+1} sont des tâches fictives représentant respectivement le début et la fin de l'ordonnancement, de durées et de demandes nulles. Donner un ordonnancement revient à donner à chaque tâche une date de début représentée par la variable de décision $S_i \in \mathbf{N}$. Un ordonnancement est un vecteur $S = (S_0, ..., S_{n+1})$.

On distingue deux types de contraintes. *Les contraintes de précédence* entre les tâches, représentables par un graphe orienté sans circuits $G = (A, E)$ où les

sommets représentent les tâches et où un arc de $(A_i, A_j) \in E$ signifie que A_j ne peut pas commencer avant la fin de A_i. Par convention, un arc relie également A_0 à toutes les tâches sans prédécesseur. De même, un arc relie toute tâche sans successeur au sommet A_{n+1}. Chaque arc $(A_i, A_j) \in E$ étant valué par la durée de A_i, G est appelé graphe *potentiel-tâches*. *Les contraintes de ressources* (dites *cumulatives*) imposent que la somme des demandes des tâches en cours d'exécution à un instant t ne dépasse pas la disponibilité d'une ressource.

Dans le cadre de notre étude les ressources sont renouvelables (les tâches restituent la part demandée à la fin de leur exécution) et de disponibilité fixe, et les tâches sont non préemptives (elles ne peuvent pas être interrompues), de durées fixes et entières. Cette dernière propriété autorise à discrétiser le temps en périodes $t = 0, \ldots, T$ où T désigne l'horizon de l'ordonnancement égal à toute borne supérieure de la durée totale minimale du projet (la somme des durées des tâches par exemple).

Le problème du RCPSP, dans le cadre de la minimisation de la durée totale d'exécution, peut se ramener à la minimisation de la date de début de la dernière tâche (fictive) A_{n+1}, et se modéliser de la façon suivante :

min S_{n+1} s-à :

$$S_i + p_i \leq S_j \quad \forall (A_i, A_j) \in E \qquad (1)$$

$$\sum_{i \in A(t)} r_{ik} \leq R_k \quad \forall\, (k,t)\ \in R \times \{0, \ldots, T\} \qquad (2)$$

$$avec\, A(t) = \{i \in A | S_i \leq t < S_i + p_i\}$$

(1) représente les contraintes de précédence et (2) représente les contraintes de ressources.

Le RCPSP peut être facilement représenté de façon graphique à l'aide du graphe *potentiel-tâches* G défini précédemment. La figure 1 représente un tel graphe pour un RCPSP à 7 tâches réelles et 3 ressources ; où sur chaque sommet (tâche) le *m-uplet* correspondant aux demandes effectuées sur chacune des m ressources est indiqué. La partie droite du schéma représente une solution réalisable sous forme d'un diagramme de Gantt. Le temps est sur l'échelle horizontale. On peut vérifier qu'à aucun moment la capacité des ressources n'est dépassée (échelle verticale) et que les contraintes de précédence sont respectées. On en déduit que **16** est une borne supérieure du problème.

La longueur d'un chemin dans le graphe de précédence peut par exemple donner une borne inférieure de la solution optimale (ex : le chemin *(0,1,7,8)* donne 0+2+7 = **9**.

2.2 Propagation des contraintes temporelles

L'objectif est d'attribuer une date de début à chaque tâche. Pour cela on peut essayer de réduire le domaine (intervalle) des possibilités qui est a priori compris entre 0 et T. Les contraintes de précédence permettent aisément de

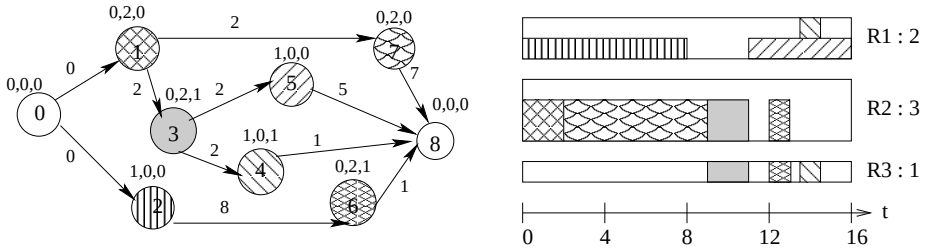


Fig. 1. Représentation graphique du RCPSP

déduire une fenêtre de temps possible pour le début d'une tâche. Par exemple, pour la figure 1, la tâche A_3 doit « attendre »la fin de la tâche A_1 pour pouvoir commencer ($S_3 \geq S_1 + p_1$). La tâche A_3 ne peut ainsi pas commencer avant le temps $t = 0 + p_1 = 2$. De telles déductions peuvent se propager. La longueur du plus long chemin dans le graphe de précédence entre A_0 et A_i correspond ainsi à la date de début au plus tôt de A_i (méthode PERT). Inversement, une date de début au plus tard de i est donnée par T moins la longueur du plus long chemin entre i et $n + 1$.

Ainsi, l'algorithme de *Bellman* (de complexité $O(|E|)$ pour les graphes sans circuits) permet de déduire les fenêtres de temps minimales de toutes les tâches en ne considérant que le graphe *potentiel-tâches* (*i.e.* les contraintes de précédence) et une borne supérieure au problème général (horizon T). Moins la valeur de cette borne supérieure est élevée, plus les fenêtres de temps calculées sont serrées.

Une information plus forte que les fenêtres de temps consiste en une matrice des distances $d(i, j)$ donnant la longueur du plus long chemin dans G entre toute paire de tâches, et ainsi renseignant sur le séquencement relatif de deux tâches.d peut être définie formellement ainsi $d : A \times A \longmapsto \{-T, \dots T\}$ $tq \quad \forall (A_i, A_j) \in A \times A, \quad d(A_i, A_j)$ est égal au chemin le plus long de A_i vers A_j dans G. On a pour $i = 0, \dots, n + 1$, $S_i \in [d(A_0, A_i); T - d(A_i, A_n + 1)]$. Cette distance peut être calculée en $O\left(n^3\right)$ par l'algorithme de *Floyd-Warshall (FW)* à partir d'une borne supérieure. Comme illustré dans l'exemple suivant, de la qualité de cette borne dépend celle des déductions faites. D'autres techniques de propagation de contraintes permettent de réduire davantage les fenêtres de temps en considérant partiellement les ressources [BL00,BK00].

Exemple de réduction de déduction de contrainte de précédence par FW Considérons toujours l'exemple de la figure 1 et plus particulièrement le couple (A_1, A_6). L'ordonnacement de la figure indique que 16 est une borne supérieure. La connaissance d'une telle borne ($T = 16$) nous donne : $S_1 \in [0, 16 - (7 + 2) = 7]$ et $S_6 \in [0 + 8 = 8, 16 - 1 = 15]$. Sachant que $p_1 = 2$, on ne peut pas déduire de containte de précédence entre A_1 et A_6. Par contre une borne supérieure à 15 donnerait $S_1 \in [0, 15 - (7 + 2) = 6]$ et $S_6 \in [0 + 8 = 8, 15 - 1 = 14]$ d'où $S_1 + p_1 \leq S_6$. On a ainsi déduit une nouvelle contrainte de précédence.

3 Analyse des instances de la PSPLIB

Des instances de test du RCPSP regroupées dans la bibliothèque PSPLIB, permettent de tester différentes méthodes de résolution et d'en comparer les résultats (bornes inférieures et supérieures) avec les meilleurs obtenus, ces derniers étant régulièrement mis à jour. Ces instances sont les plus utilisées dans la littérature. Parmi celles nous concernant dénommées **KSD**, on s'intéresse principalement aux instances à 60 tâches (KSD60) car il en existe de non résolues pour lesquelles les bornes ont été fortement améliorées récemment [BD04].

3.1 Description des instances KSD

La PSPLIB propose des jeux de données à 30, 60, 90 et 120 tâches comportant plusieurs centaines d'instances chacun ; 480 pour les 3 premiers et 600 pour le dernier. Toutes les instances possèdent 4 ressources.

Les instances sont générées aléatoirement par séries de 10 à partir de trois indicateurs notés NC, RF, RS, détaillés ci-après, fixés pour chaque série. Les durées des tâches sont générées aléatoirement entre 0 et 10. Les autres caractéristiques (graphe de précédence, valeurs des demandes, disponibilité des ressources) dépendent des paramètres NC, RF, RS, fixés au préalable.

- **NC :** pour *network complexity* paramétrant la densité du graphe en exprimant un nombre moyen de successeurs par tâche. Les valeurs prises sont 1,5 ; 1,8 ; 2,1.
- **RF :** pour *resource factor* paramétrant le nombre moyen de ressources utilisées par tâche. 0,25 ; 0,5 ; 0,75 ; 1 sont les valeurs prises exprimant 1, 2, 3 ou 4 ressources utilisées (sans indication sur la demande sur les ressources).
- **RS :** pour *resource strength* paramétrant la disponibilité des ressources. Il est utilisé dans le calcul de la capacité des ressources (R_k) via la formule $R_k = R_k^{min} + round(RS(R_k^{max} - R_k^{min}))$. Où $round()$ arrondie un réel à l'entier le plus proche ; R_k^{min} est égale à la plus grande demande effectuée par une tâche sur cette ressource $R_k^{min} = \max_{i=1}^{n} r_{ik}$; R_k^{max} correspond à la demande maximale, rencontrée dans un ordonnancement au plus tôt, calculé à partir d'une liste de tâches triées par demande (r_{ik}) décroissante. Une grande valeur du RS indique que la disponibilité de la ressource est grande par rapport aux demandes des tâches. Les valeurs prises sont 0,1 ; 0,2 ; 0,3 ; 0,4 ; 0,5 pour les KSD120 et 0,2 ; 0,5 ; 0,7 ; 1 pour les autres.

Par la suite, les valeurs des indicateurs de génération sont pris comme valeurs effectives, car les variations sont négligeables.

3.2 Motivations et buts de l'analyse

Récemment, des chercheurs [BL00] ont émis certaines critiques sur ces jeux d'instances. Il est en effet intéressant de se demander s'ils sont représentatifs du problème général, de par leur mode de génération. Par exemple, les problèmes

mettant en jeu une ressource critique sont difficilement formulable avec des ressources possédant toutes le même RS. Ceci peut désavantager des méthodes plus efficaces que d'autres dans le cas général mais moins sur ces problèmes particuliers. Utiliser de façon privilégiée ces bancs d'essais peut amener les chercheurs à développer des méthodes trop spécifiques à ces problèmes. De plus, une grande partie des instances sont triviales ; un quart (RS=1) n'étant soumises à aucune contrainte de ressources, elles sont résolues optimalement par une simple méthode PERT ou du chemin critique [SRM65]. Afin de ne pas biaiser leurs résultats statistiques obtenus sur l'ensemble d'un banc d'essai, les chercheurs sont obligés de les regrouper par classes d'instances. Malheureusement, il n'existe pas de classification standard de ces instances autre que celle liée aux paramètres de génération (NC, RF et RS).

Cependant toutes les instances KSD ne sont pas résolues et souvent l'optimum est très mal approché. Un des objectifs de cette analyse est de parvenir à discerner ce qui rend les instances non résolues difficiles et de pouvoir en déduire un ou plusieurs indicateurs relatifs aux problèmes et corrélés à leur difficulté.

3.3 Ensembles Interdits Minimaux (EIM)

Parmi les critiques formulées [BL00], Baptiste et Le Pape émettent l'hypothèse que la caractéristique principale des instances KSD encore irrésolues est de posséder un grand nombre de tâches en disjonction, c'est à dire incompatibles deux à deux vis à vis d'au moins une ressource. Ils distinguent ainsi des instances *hautement disjonctives* à l'inverse des instances *hautement cumulatives*. Nous nous proposons d'étudier ce qu'il en est pour les KSD à 60 tâches à travers le concept d'*Ensembles Interdits Minimaux* qui étend le concept de disjonction. Dans la section suivante, nous rappelons la définition du concept d'EIM. Dans un premier temps nous décrivons la méthode de calcul des EIM puis nous présentons une analyse de la difficulté des problèmes relativement aux EIM.

Définition et exemple Un ensemble de tâches dont l'exécution simultanée est impossible, est dit *interdit*. On le dit *minimal* si chacun de ses sous-ensembles sont exécutables simultanément [1]. Cette définition est dûe à Bartush, Möhring et Radermacher [BMR88]. Les EIM de cardinalité 2 sont appelés *disjonctions*, puis *triplets*, *quadruplets*, etc... pour les EIM de cardinalités supérieures. Parmi ces EIM, nous nous intéressons à ceux liés à la violation d'une contrainte de ressource.

Ainsi, à chaque EIM, on peut associer au moins une contrainte de ressources susceptible d'être violée – si toutes les tâches sont en cours d'exécution à un même instant. Un EIM F est dit résolu par un ordonnancement S si $\exists i, j \in F,\ S_i + p_i \leq S_j$, ce qui revient à imposer une contrainte de précédence entre au moins deux tâches de l'EIM. La contrainte de ressource associée ne peut alors plus être violée. Ainsi, en supposant que l'on arrive à énumérer l'ensemble $\mathcal{F}$ des

[1] on parle aussi d'ensemble critique de tâches

EIM, les contraintes de ressources (2) peuvent être remplacées par (3).

$$\forall F \in \mathcal{F}, \quad \exists \{i,j\} \in F \quad , S_i + p_i \leq S_j \qquad (3)$$

La figure 2 représente les EIM du RCPSP de la figure 1. Ils sont présentés sous forme d'arbre. Chaque noeud est étiqueté par une tâche. Une branche (de la racine à une feuille) donne la composition d'un EIM. On remarque que A_3 et A_6 forment un EIM relativement aux ressources 1 et 3.

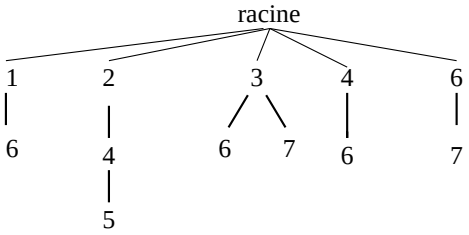


Fig. 2. Arbre représentant les EIM de la figure 1

Méthode d'énumération des EIM Stork et Uetz [SU04] proposent un algorithme d'énumération des EIM en $O(|\mathcal{F}|)$, la meilleure complexité possible. Cet algorithme procède par construction progressive d'un arbre d'EIM (du type de celui de la figure 2). Voici les idées principales de l'algorithme. A chaque instant l'arbre possède des branches (issues de la racine), chacune représentant soit un EIM (elle ne sera plus rallongée), soit un ensemble de tâches exécutables simultanément. A ces dernières, on essaye d'ajouter une nouvelle tâche. Soit le nouvel ensemble est toujours exécutable simultanément et l'on continue à essayer de l'enrichir, soit on vient de créer un ensemble interdit. Si cet ensemble interdit est minimal on stoppe la branche, sinon on essaye une autre tâche. Lorsqu'il n'y a plus de tâches à essayer, on passe à une autre branche.

Du fait du nombre exponentiel de ces EIM (plusieurs millions pour certaines instances KSD60) cette énumération est coûteuse en temps et surtout en espace mémoire (500 MB ne suffisent pas à certaines instances KSD60). Il est important de remarquer que les EIM calculés tiennent compte des contraintes de précédence. En effet, deux tâches reliées par une contrainte de précédence ne peuvent pas appartenir au même EIM. Sinon, celui-ci serait résolu. Or, il existe des contraintes de précédence explicites données par les arcs de G et d'autres, implicites, que l'on peut déduire des contraintes. Stork et Uetz utilisent dans leur algorithme la fermeture transitive du graphe de précédence. Nous proposons d'utiliser la matrice des distances (calculée en $O\left(n^3\right)$ par *Floyd-Warshall*) pour obtenir des contraintes de précédence additionnelles qui vont permettre de supprimer certains EIM (*cf* l'exemple en 2.2 où L'EIM (A_1, A_6) peut être supprimé de l'arbre des EIM). En utilisant *Floyd-Warshall* avec la meilleure borne

supérieure connue incrémentée de 5% (résultats obtenus par les meilleures heuristiques de la littérature [PAC03,VBQ04,VQB04]), nous avons réduit en moyenne de 55% le nombre d'EIM calculés. Le tableau 1 compile les résultats comparés avec et sans *Floyd-Warshall* (respectivement première et deuxième ligne).

Instances	KSD30	KSD60
SU(EIM)	325	180545
FW(EIM)	255	82698

Tab. 1. quantités d'EIM calculés avec différents prétraitements

Pour une étude comparative sur les instances nous nous sommes basés sur les résultats obtenus après une simple fermeture transitive du graphe de précédence.

Analyse des résultats Nous allons observer l'influence des indicateurs de génération (NC, RF, RS) sur les EIM (cardinalité et nombre) ainsi que sur la difficulté des problèmes. On estime cette difficulté en calculant l'écart entre les meilleures bornes inférieures et supérieures connues : $dev = (Bsup - Binf) * 100/Bsup$. Afin de confirmer ce choix de critère de difficulté, nous avons soumis directement les problèmes à la recherche arborescente par défaut d'*IlogScheduler 6.0* (bibliothèque de programmation par contrainte dédiée à l'ordonnancement [BL00,LEP94]) avec un temps de calcul limité à cinq minutes. La qualité des résultats obtenus varie selon les instances de façon similaire aux meilleures bornes connues.

La figure 3 donne le nombre d'EIM $|\mathcal{F}|$ (#FS dans la figure) et leur cardinalité moyenne $avg(|F|) = \sum_{F \in \mathcal{F}} \frac{|F|}{|\mathcal{F}|}$ en fonction des paramètres sur les instances KSD60 (NC, RF, RS). La cardinalité moyenne des EIM et leur quantité croissent en même temps (fig. 3). On peut se dire sans considérer les contraintes spécifiques du problème que ces valeurs sont liées par une relation du type $\mathcal{C}_{|\mathcal{F}|}^{avg(|F|)}$.

Comme on pouvait s'y attendre, plus le graphe de précédence est dense, moins l'on a d' EIM (850 000 en moyenne pour NC=1,5 contre 10 000 pour NC=2,1). Par définition, le RS est directement lié aux contraintes cumulatives. Plus ces contraintes sont serrées (*i.e.* moins le RS est élevé) – capacité des ressources faible par rapport aux demandes – moins le nombre et la cardinalité des EIM sont élevés. Plus une tâche utilise des ressources différentes (RF élevé), plus elle a de « chances »d'appartenir à des EIM, ce qui augmente $|\mathcal{F}|$. Un EIM ne dépendant, en général, que d'une seule ressource cela n'entraîne pas une diminution systématique de $avg(|F|)$.

La figure 4 donne l'écart (noté *dev*) entre la meilleure borne supérieure et la meilleure borne inférieure connues en fonction des paramètres NC, RF et RS.

L'influence du NC sur la difficulté du problème est quasiment nulle. Le RS (un des caractères cumulatifs du problème) est totalement discriminant vis à

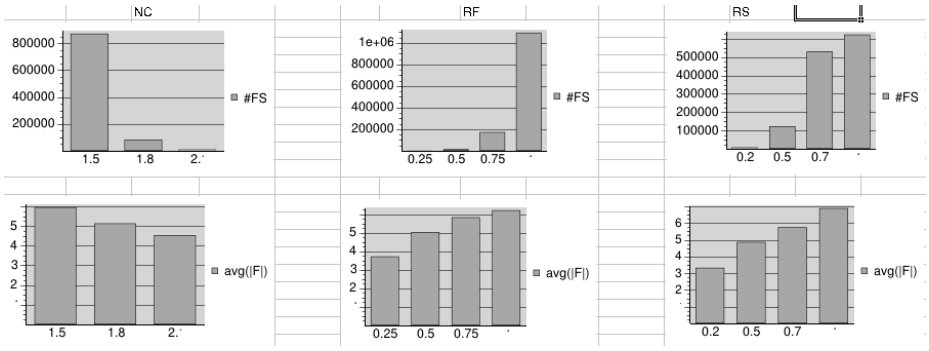


Fig. 3. Nombre d'EIM et leurs cardinalités moyennes par indicateur pour les KSD60

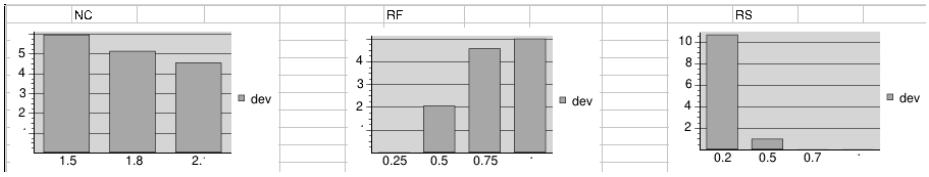


Fig. 4. Ecart (%) entre meilleure Bsup et meilleure Binf par indicateur pour les KSD60

vis de la difficulté ; les problèmes présentant le plus grand écart ont un RS à 0,2. Le recoupement avec la figure 3 montre que les EIM ont alors tendance à être de petite taille (3,2 en moyenne) moins nombreux (7500 en moyenne). Les résultats bruts classés par RF pourraient amener aux conclusions inverses puisque la difficulté croît avec le RF comme le nombre d'EIM, mais de façon beaucoup moins tranchante. En fait, regrouper les instances par RF donne des groupes très hétérogènes. Ce paramètre permet plutôt de distinguer des sous-classes de problèmes pour un RS donné (*cf* la figure 5 qui donne la valeur de l'écart, du nombre d'EIM, et de leur cardinalité moyenne pour les quatres valeurs de RF pour un RS à 0,2). Ces sous-classes se différencient par leur quantité d'EIM mais aussi directement par leur RF qui pourrait être supposé comme critère de difficulté, pas tellement par le fait qu'une instance de RF=1 a beaucoup d'EIM mais plutôt par le fait qu'une tâche effectue des demandes sur beaucoup de ressources, ce qui donne à ces problèmes un caractère *multi-ressource* que nous aborderons plus tard, dans la section 4. La cardinalité moyenne des EIM varie peu et reste proche de 3. Le fait que l'écart (dev) n'évolue pas entre un RF à 0,75 et un RF à 1 alors que la quantité d'EIM triple, semble dévoiler un phénomène de seuil.

Le tableau 2 détaille la nature des EIM (jusqu'aux quintuplets) présents dans les instances regroupées par RS (à gauche) puis par RF pour les plus difficles (à droite). Les quantités d'EIM sont ramenées à des valeurs homogènes par une échelle logarithmique : la quantité q d'EIM de taille m est représentée par

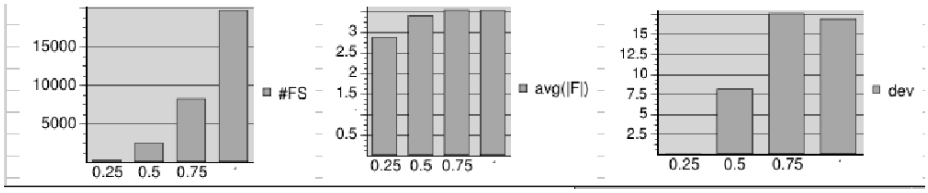
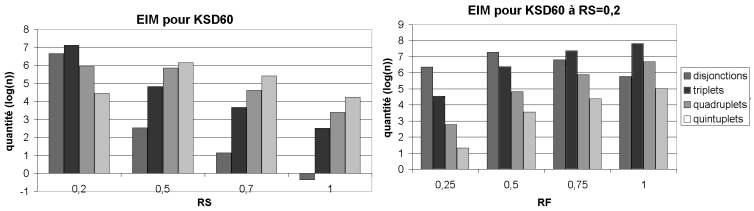


Fig. 5. Statistiques pour les RS=0,2 classées par valeurs de RF

$log_m(q)$. Le tableau 2 indique que les instances les plus difficles (RS=0,2) (fig. 4) sont les seules à être riches en disjonctions et en triplets. On peut se prononcer moins nettement pour les quadruplets et les quintuplets qui ne suivent pas linéairement les valeurs du RS, leur présence seule ne paraît pas être déterminante de la difficulté d'une instance. Néanmoins le tableau 2 *(partie droite)* indique que, pour les instances les plus difficiles, le nombre de quadruplets suit la tendance de RF donc de la difficulté des problèmes (fig. 5).



Tab. 2. Structure des KSD60 en terme d'EIM

Pour les instances KSD, ce sont les problèmes les moins cumulatifs ($|F| = 2$ ou 3) qui semblent être les plus difficiles. Les problèmes cumulatifs de KSD sont donc en général assez bien traités par les méthodes de la littérature qui pourtant sont plutôt orientées vers la résolution des disjonctions. Ceci peut s'expliquer par l'improbabilité de rencontrer des EIM de grande taille dans les solutions calculées sur ces instances sans même avoir considéré ces EIM de grande taille.

En conclusion de cette analyse, il apparaît toutefois que la difficulté des instances KSD à RS=0,2 provienne davantage de la présence conjointe de disjonction et de triplets (voire quadruplets) que sur les disjonctions seules comme cela était supposé dans [BL00]. Ceci peut expliquer d'ailleurs que les méthodes de la littérature, particulièrement au point sur la résolution de problèmes disjonctifs, échouent sur ces instances. Nous rejoignons tout de même les conclusions de Baptiste et Le Pape sur le caractère « faiblement »cumulatif de ces instances, et sur la nécessité de générer de nouvelles instances pour lesquelles les EIM de cardinalité supérieure jouent un rôle primordial.

4 Calcul de bornes inférieures et supérieures par génération de problèmes à m machine

4.1 Principe

D'après l'analyse de la section 3.3, satisfaire les contraintes de disjonctions et de triplets semble être primordial pour espérer résoudre les instances KSD60. Ainsi, l'idée de base de la méthode consiste à résoudre optimalement une relaxation construite à partir d'EIM considérés par cardinalité croissante. Soit $\mathcal{F}_i$ l'ensemble des EIM de cardinalité inférieure ou égale à i ($\mathcal{F}_n = \mathcal{F}$), on note $P(\mathcal{F}_i)$ la relaxation du problème initial relâchant les contraintes (3) d'EIM de cardinalité supérieure à i. Ainsi $P = P(\mathcal{F})$. Tout ordonnancement optimal d'un $P(\mathcal{F}_i)$ donne une borne inférieure de P. La section 4.2 considère la résolution de la relaxation $P(\mathcal{F}_2)$. La section 4.3 considère la résolution de $P(\mathcal{F}_3)$. La section 4.4 décrit une méthode d'obtention d'une borne supérieure à partir des solutions de ces relaxations.

4.2 Relaxation Disjonctive

Idée générale La *relaxation disjonctive* ($P(\mathcal{F}_2)$) consiste à ne considérer que les EIM de cardinalité 2 (disjonctions). Les contraintes peuvent être représentées par un graphe non orienté, le graphe disjonctif, où une arête relie 2 à 2 les tâches appartenant au même EIM. Résoudre $P(\mathcal{F}_2)$ revient à orienter toutes les arêtes du graphe disjonctif de telle sorte que le graphe potentiels-tâches des contraintes de précédence augmenté des arcs ainsi orientés, ait un plus long chemin de longueur minimale. Pour résoudre $P(\mathcal{F}_2)$, plutôt que de travailler directement sur le graphe disjonctif, nous reprenons la modélisation du problème avec les contraintes de ressource de type (2). Toutefois, au lieu de considérer les ressources du problème original P, nous les remplaçons par des ressources unaires (*i.e.* de disponibilité égale à 1) appelées **machines**, déduites des contraintes disjonctives. En effet, on peut montrer que toute clique dans le graphe disjonctif définit une machine sur laquelle doivent être exécutées *sans possibilité de chevauchement* toutes les tâches de la clique. La création d'une machine pour chaque clique maximale permet ainsi d'obtenir exactement le problème $P(\mathcal{F}_2)$. Cette transformation du problème est motivée par l'abondance dans la littérature de travaux concernant les problèmes contenant de telles machines. Notamment, le problème de *job shop* consiste à exécuter N travaux sur M machines, chaque travail étant constitué d'opérations devant passer successivement sur chaque machine selon un ordre propre à chaque travail. Des instances à 100 opérations sont résolues optimalement [BDP96,BJS94,CP94]. Les différences principales entre $P(\mathcal{F}_2)$ et le *job shop* sont d'une part que $P(\mathcal{F}_2)$ a un graphe de précédence moins régulier et d'autre part, que dans $P(\mathcal{F}_2)$ une tâche peut être exécutée simultanément sur plusieurs machines si elle appartient à plusieurs cliques maximales. La figure 6 illustre la génération d'un problème $P(\mathcal{F}_2)$ et d'une solution à partir du RCPSP de la figure 1. On remarque la caractéristique multi-ressources avec

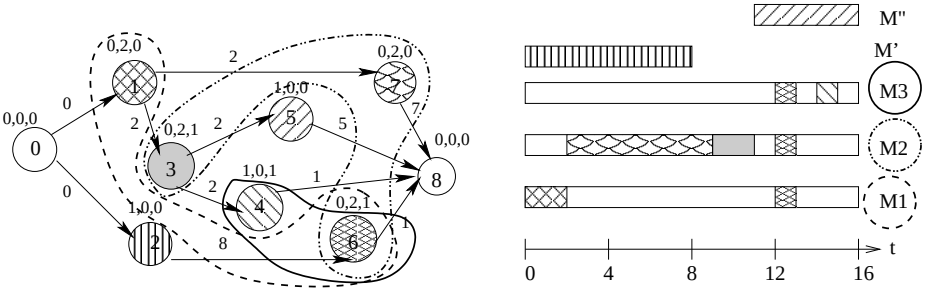


Fig. 6. Problèmes à une machine multi-ressources

la tâche 6 qui utilise les 3 machines. On utilise des machines fictives (M' et M") pour représenter l'exécution des tâches isolées (A_2 et A_5).

Notons enfin que l'idée de génération des machines dans des problèmes d'ordonnacement à déjà été abordée par Carlier et Latapie [CL91] qui génèrent des problèmes à m machines. Plus récemment, de telles machines ont été générées à partir de contraintes disjonctives dans le RCPSP par Carlier, Néron [CN03] qui résolvent une relaxation de $P(\mathcal{F}_2)$ et par Bruker [BK00] et Baptiste [BL00] et Demassey [DAM03] qui génèrent des cliques en nombre limité pour effectuer des propagations de contraintes. L'originalité de notre travail est de chercher à résoudre exactement $P(\mathcal{F}_2)$ en générant si possible toutes les cliques maximales.

Algorithme La méthode implémentée pour résoudre $P(\mathcal{F}_2)$ (algorithme 1) procède de manière destructive [KS99] en imposant à chaque itération un *makespan* égal à une borne inférieure. A chaque valeur *Binf*, on génère les machines puis on résout le problème de satisfaction de contraintes composé des contraintes de précédence, des contraintes de machines et de la contrainte imposant à la durée totale d'exécution (S_{n+1}) d'être inférieure ou égale à *Binf*. De l'obtention d'une solution à ce problème, on déduit que la solution optimale de la relaxation disjonctive est inférieure à *Binf*. Si le problème est irréalisable, alors la solution optimale est strictement supérieure à *Binf*. Les détails de cet algorithme sont présentés dans les trois sections suivantes.

Initialisation de la borne inférieure Toute borne inférieure valide peut servir à initialiser $Binf$. Expérimentalement nous avons utilisé la longueur du chemin critique du graphe de précédence initial ce dernier étant très rapidement calculable.

Génération des machines A chaque valeur *Binf* on associe une relaxation disjonctive associée. Comme cela a été montré dans la première partie (section 2.2), imposer une date de fin au plus tard à un problème d'ordonnancement permet de réduire les fenêtres de temps d'exécution des tâches et d'augmenter les distances entre les tâches. Ceci rend explicites de nouvelles contraintes

Algorithme 1 Algorithme de Résolution de la Relaxation Disjonctive

Initialisation $Binf$
Répéter
  `Poser_Contrainte` $(S_n + 1 \leq Binf)$
  `Réduire_Fenêtres_de_Temps` $(Binf)$
  `Construire_machines`
  `Résoudre_problème_satisfaction_de_Contraintes`
  Si `Temps_Limite_Dépassé` **faire**
    `Sortir`
  fin si
  Si `Solution_trouvée` **faire**
    `Sortir`
  Sinon `Problème_Irréalisable`
  $Binf = Binf + 1$
  Fin si
Fin répéter

de précédence donc diminue (*cf* tab. 1) le nombre de disjonctions à considérer. Ce calcul est effectué par l'algorithme de *Floyd-Warshall* (*étape 4*). Prendre en compte toutes les disjonctions sous forme de cliques dans le graphe de disjonction peut être effectué en considérant toutes les **cliques maximales** de ce graphe (*étape 5*). L'énumération de ces cliques se fait par un algorithme dérivé d'un algorithme sériel [BG92]. Cet algorithme considère un ensemble de cliques potentielles ; pour chacune des tâches t il considère son voisinage (dans le graphe des disjonctions) $\mathcal{N}(t)$; et scinde (si possible) chaque clique potentielle Q en 2 en considérant la différence entre Q et $\mathcal{N}(t)$ d'où deux nouvelles cliques potentielles $Q - \mathcal{N}(t)$ et $\mathcal{N}(t)$. Après avoir parcouru toutes les tâches, les cliques potentielles sont toutes les cliques maximales. Cette énumération, bien que théoriquement exponentielle, est possible en un bref délai (quelques secondes) pour toutes les instances KSD. Pour certaines instances à 120 tâches (RS à 0,1), nous ne parvenons pas à résoudre exactement $P(\mathcal{F}_2)$. Nous proposons donc de relâcher $P(\mathcal{F}_2)$ de façon à se rapprocher d'un *job shop* pour lequel notre implantation est supposée efficace. On cherche à réduire la caractéristique *multi-ressources* des tâches (*mono-ressource* pour le *job shop*), ce qui, dans notre formulation, se traduit en partie par l'appartenance d'une tâche à plusieurs cliques. Nous avons implémenté et testé sur les KSD120 l'algorithme suivant qui permet, une fois la totalité des cliques maximales calculée (ce qui est rapide), de paramétrer la quantité de cliques maximales à considérer.

algorithme de sélection des cliques On associe à chaque tâche un score variable représentant sa fréquence et initialisé à 0. On considére les cliques maximales par ordre décroissant de durée. Pour la clique considérée, on calcule le rapport entre le nombre de tâches la constituant et la somme des scores de ces mêmes tâches. La clique est sélectionnée si et seulement si ce rapport est supérieur à un ratio fixé. Si la clique est sélectionnée, on incrémente le score de chacune des tâches la composant.

Résolution du problème de satisfaction de contraintes (CSP) La résolution du problème de satisfaction de contraintes (*étape 6*) est effectuée par *IlogScheduler 6.0* (capable de résoudre de façon optimale des *job shop* à 100 activités), en adaptant le code fourni pour résoudre le *job shop* [ILO00]. On lui impose un temps limite de calcul. Ce délai dépassé l'algorithme s'arrête (*étape 8*), ce qui peut être le cas pour des problèmes de grande taille et/ou très difficiles. La borne inférieure de la relaxation $Binf$ est alors une borne inférieure valide pour P.

4.3 Ajout des triplets, des problèmes à deux machines

On peut modéliser la relaxation $P(\mathcal{F}_3)$ comportant les triplets interdits par des problèmes à deux machines (fig. 7) où sur chacune d'elle doit s'exécuter au moins une tâche du triplet, une tâche s'exécutant indifférement sur n'importe quelle machine. L'implémentation donne lieu à une ressource de disponibilité 2 sur laquelle chaque tâche du triplet effectue une requête de 1 unité. Il y a donc trois disjonctions (six contraintes de précédence) possibles pour résoudre le triplet. Ceci explique la complexité supérieure de résolution par rapport aux disjonctions. Nous nous sommes donc contentés d'ajouter, dans un premier temps, les triplets violés par la solution optimale de la relaxation disjonctive. Le nouvel algorithme consiste donc à boucler sur le précédent algorithme en résolvant, à chaque boucle la relaxation augmentée de nouveaux triplets violés. Sur la figure 7 les tâches A_2, A_4 et A_5 forment un triplet interdit sur la ressource 1 ($R_1 = 2 < 1 + 1 + 1$). De plus tout couple de tâches prises parmi ces trois là est exécutable en simultané (somme des demandes inférieure ou égale à 2). Le respect de cette contrainte est imposé par les 2 machines M1 et M2, nous avons représenté une résolution possible de ce triplet sur la figure 7.

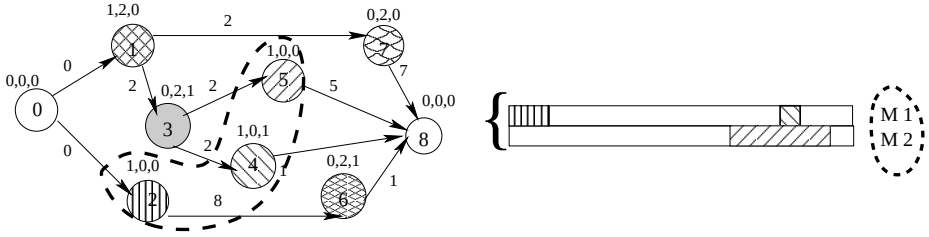


Fig. 7. Exemple de problème à deux machines

4.4 Calcul de bornes supérieures

Une fois la solution optimale de la relaxation $P(\mathcal{F}_2)$ ou $P(\mathcal{F}_3)$ calculée, on utilise un algorithme sériel en $O(m \times n^2)$ [KH04] pour en dériver une solution réalisable du problème initial. On réalise un ordonnancement « au plus tôt »des

tâches prises dans un ordre de priorité. Ce dernier est déterminé par les dates de début des tâches dans la solution de $P(\mathcal{F}_x)$ ($x \in 2, 3$). Dans le cas où $P(\mathcal{F}_x)$ n'est pas résolu dans la limite de temps imposée, la liste des priorités peut être dressée à partir d'une simple solution réalisable de la relaxation calculée auparavant au cours des itérations.

L'algorithme sériel utilisé pour calculer des solutions réalisables de P est rapide. Il paraît intéressant de le lancer plusieurs fois avec des ordres de priorité différents. Afin de rester cohérents avec les contraintes de précédence présentes dans la solution de $P(\mathcal{F}_x)$, nous lançons plusieurs fois l'algorithme sériel en laissant un ordre de priorité aléatoire entre les tâches pouvant débuter à la même date. On estime à environ 1/2 la fréquence d'un tel phénomène, puisqu'il y a potentiellement deux fois plus de dates de début possibles que de tâches ; la valeur des solutions calculées étant voisine du double du nombre de tâches.

Remarquons que Möhring et al. [MSSU03] ont utilisé dans leurs travaux ce genre de méthode pour dériver des solutions réalisables à partir de solutions optimales de relaxations lagrangiennes. Ils proposent d'ailleurs de générer d'autres listes de priorités respectant les contraintes de précédence en ne considérant plus seulement une date de début mais une date (d'α -*completion*) à laquelle une tâche s'est effectuée à α%. Ces listes de priorités pourraient contribuer à améliorer les bornes supérieures calculées ici.

4.5 Résultats

Tous les résultats discutés ont été obtenus dans une limite de temps de 1600 secondes sur un processeur cadencé à 800 MHz.
Le tableau tableau 3 présente les résultats de la résolution de $P(\mathcal{F}_2)$ sur les différentes instances KSD. Pour les instances pour lesquelles on connaît déjà une bonne borne inférieure et une bonne borne supérieure (*i.e.* RS>0,2), notre méthode obtient des résultats proches des meilleurs connus qui sont eux-mêmes proche de l'optimalité. On donne les écarts entre la meilleure borne inférieure connue et celle calculée pour ces instances dans la deuxième colonne du tableau 3. Les instances les plus difficiles (RS $\leq$ 0,2) sont séparées en trois groupes suivant leur RF qui, d'après l'analyse (tab. 2 *(droite)*), croît avec la difficulté des instances. Pour chacun de ces groupes nous calculons l'écart (en %) entre la meilleure borne inférieure connue et celle calculée (D(LB)), le nombre de meilleures bornes inférieures (resp. supérieures) améliorées (LB* (resp. UB*)). Les deux dernières colonnes représentent l'écart (D(CP) en %) entre notre meilleure borne inférieure (resp. supérieure) et la longueur du chemin critique (l'optimum pour les KSD30) sur toutes les instances, pour comparaison avec les autres méthodes de la littérature. Les relaxations disjonctives comportant toutes les cliques maximales ont été résolues optimalement sauf pour les KSD120 avec un RS à 0,1. C'est pour cette raison que la dernière colonne n'est pas renseignée pour les KSD120 ; puisque le calcul d'une borne supérieure s'effectue à partir d'une solution de $P(\mathcal{F}_2)$. La dernière ligne du tableau 3 représente les résultats obtenus sur les KSD120 on ne considérant qu'une partie des disjonctions avec un ratio à 0,2 (cf 4.2). Il est à noter que pour les instances KSD90 et KSD120 les meilleurs

résultats connus sont ceux actuellement accessibles sur le site web de l'université de Kiel qui n'est pas totalement mis à jour. Pour les KSD60 nous avons pu nous procurer des bornes de bien meilleure qualité obtenues par Baptiste et Demassey [BD04]. Notre but principal est de constater l'impact de la résolution totale des contraintes disjonctives, les temps de calcul n'ont donc pas de réelle importance. A ce sujet signalons que les temps de prétraitement (réduction des fenêtres de temps et génération des machines) sont totalement négligeables par rapport à celui de la résolution du problème de satisfaction de contraintes. On a expérimentalement constaté que la qualité des résultats est peu sensible à la limite de temps de calcul imposé, car soit le problème est très rapidement satisfait ou montré irréalisable, soit le temps de résolution explose fortement.

benchs	RS>0,2	RS<=0,2									toutes	
		RF=0,25 ou 0,5			RF=0,75			RF=1			D(CP)	
	D(LB)	D(LB)	LB*	UB*	D(LB)	LB*	UB*	D(LB)	LB*	UB*	LB	UB
KSD30	1,6	1,1	0	0	4,7	0	0	10,2	0	0	$2{,}31^{+}$	$3{,}57^{+}$
KSD60	4,3	2,5	0	6	13,0	0	5	28,6	0	4	4,0	18,7
KSD90	0,1	2,8	15	0	22,6	0	0	32,8	0	0	1,5	19,5
KSD120	3,5	7,5	21	0	30,8	0	1	45,0	0	4	5,5	-
KSD120	3,5	9,0	11	0	30,9	0	0	41,4	0	0	5,0	-

$^{+}$ déviation par rapport à l'optimum

Tab. 3. résultats obtenus avec la relaxation disjonctive

Le tableau 4 montre, sous la même forme que ceux de $P(\mathcal{F}_2)$ (tab. 3), les résultats obtenus en tenant compte des triplets violés, $P(\mathcal{F}_3)$. Résoudre $P(\mathcal{F}_3)$ directement avec notre algorithme est trop coûteux en temps de calcul. On essaye donc de tendre progressivement de $P(\mathcal{F}_2)$ vers $P(\mathcal{F}_3)$ en prenant en compte les triplets violés par la solution obtenue pour la relaxation précédente, comme indiqué en 4.3. Pour 44 des KSD60 et 90 des KSD90, $P(\mathcal{F}_3)$ n'est pas résolue optimalement. Malgré cela on constate une amélioration sensible des résultats. Les points négatifs de l'approche adoptée pour prendre en compte les triplets sont tout de même importants. Le problème de satisfaction de contrainte est trop complexe à résoudre avec notre implémentation. Deux raisons expliquent cela, les contraintes liées au triplets complexifient rapidement la relaxation et notre code n'effectue aucun traitement particulier pour les problèmes à 2 machines. De plus, le temps de calcul global explose puisque l'on relance entièrement un nombre indéterminé de fois la résolution exacte d'une relaxation (cf section 4.3).

Pour les instances les plus difficles (RS$\leq$0,2), la qualité de la borne obtenue en résolvant $P(\mathcal{F}_2)$ se détériore fortement avec l'augmentation du RF. Ceci est cohérent avec l'analyse faite en 3.3 (tab. 2) qui indique que les instances à RF élevé ont des EIM de plus grande taille. Il paraît donc nécessaire de s'intéresser

benchs	RS>0,2	RS<=0,2									toutes	
	RS>0,2	RF=0,25 ou 0,5			RF=0,75			RF=1			D(CP)	
	D(LB)	D(LB)	LB*	UB*	D(LB)	LB*	UB*	D(LB)	LB*	UB*	LB	UB
KSD30	0,6	0,0	0	0	0,6	0	0	3,2	0	0	$0,67^{+}$	$2,16^{+}$
KSD60	0,3	0,1	10	13	7,5	3	9	22,2	0	8	5,6	17,1
KSD90	0,1	0,4	26	0	19,0	0	0	30,3	0	0	2,5	18,0
KSD120	3,3	6,4	35	0	29,8	0	1	44,0	0	4	6,2	-

$^{+}$ déviation par rapport à l'optimum

Tab. 4. résultats obtenus avec la relaxation disjonctive enrichie de triplets

aux contraintes de ressource de cardinalité plus élevée afin d'améliorer la qualité des bornes calculées.

Ceci explique que la comparaison avec les autres méthodes de la littérature ne sont pas globalement favorables à notre méthode. En effet, pour les KSD60, par exemple, les meilleurs bornes supérieures connues se situent autour de 11% au dessus de la borne du chemin critique (CP), contre 17% pour notre méthode. De même, les meilleurs bornes inférieures connues se situent autour de 8% au dessus de CP, alors que notre méthode obtient 5,6%. Ces résultats sont néanmoins encourageants car, pour les instances de faible RF, certaines meilleures bornes inférieures connues ont été améliorées (13 pour les KSD60 et 26 pour les KSD90), ainsi que 30 bornes supérieures pour les KSD60.

5 Conclusion

Dans la section 3 consacrée à l'analyse des instances KSD qui forment le banc d'essai le plus courament utilisé, nous montrons que les paramètres de génération NC, RF et RS permettent assez finement d'estimer la structure d'une instance en terme d'ensemble interdit minimal (EIM)(quantité et cardinalité). Les 3/4 de ces instances sont résolues optimalement (1/2 trivialement). Une des caractéristiques principales des instances non résolues est d'être faiblement cumulatives (EIM de petites tailles) par rapport aux autres. Par contre, les plus difficles parmi celles-ci possèdent les EIM de plus grandes tailles. Il paraît donc intéressant d'une part de développer une méthode spécifique baséee sur ces EIM permettant de résoudre les instances ouvertes et d'autre part de générer des instances difficiles plus cumulatives.

La méthode proposée repose sur la résolution exacte de la relaxation $P(\mathcal{F}_2)$. On constate qu'elle suffit à résoudre la plupart des instances déjà résolues supposées « faciles », ce qui confirme cette première conjecture, et qu'elle est peu efficace sur les instances supposées « difficiles ». Pour ces dernières, la prise en compte des triplets supporte plusieurs voies d'améliorations qui nous donnent bon espoir d'obtenir de meilleurs résultats. Tout d'abord, nous utilisons Ilog-Scheduler 6.0 à partir d'un code adapté de celui de la résolution du *job shop*. Ce

dernier ne tient donc pas compte du caractère multi-ressources de nos relaxations mais il suffit à résoudre exactement $P(\mathcal{F}_2)$ pour les instances à 60 et 90 tâches. La prise en compte des triplets peut être nettement améliorée en sélectionnant mieux (via des heuristiques) des triplets particuliers, comme par exemple les triplets symétriques qui mettent en jeu des tâches en parallèle [BK00]. Enfin, cette méthode peut déboucher sur une méthode exacte de résolution du RCPSP basée sur les EIM. A la vue des résultats, il paraît raisonnable de fermer les instances KSD60 et KSD90 ayant un RF strictement inférieur à 1, en résolvant exactement $P(\mathcal{F}_3)$. Par contre, la résolution des instances les plus dures (RS à 0,2 et RF à 1) semble nécessiter la prise en compte de quadruplets voire de quintuplets.

Notre méthode est basée sur l'analyse des instances KSD, les plus courantes de la littérature. Cela pourrait être source d'information que de la tester sur d'autres instances, comme par exemple celles de Baptiste - Le Pape [BL00] supposées plus cumulatives. Nous pensons aussi que la hiérarchisation des relaxations par cardinalité croissante peut donner une utilité pratique aux solutions des relaxations.

Références

[BD04] Baptiste P. et Demassey S., *Tight LP bounds for resource constrained project scheduling* OR Spectrum 26, 251–262, 2004.

[BDP96] Blazewicz J., Domschke W. et Pesch E., *The Job-Shop scheduling Problem : Conventional and New Solution Techniques.* European Journal of Operational Research, 93(1), 1-33, 1996.

[BG92] Barthelemy P. et Guénoche A., *Trees and proximity representation.* J. Wiley, 1992

[BJS94] Brucker P., Jurish B. et Sievers B., *A Branch & Bound Algorithm for the Job-Shop Scheduling Problem.* Discrete Applied Mathematics, 76, 43-59, 1997.

[BK00] Brucker P. et Knust S., *A linear programming and constraint propagation-based lower bound for the rcpsp* ,European Journal of Operational Research, 127(2), 355-362, 2000.

[BL00] Baptiste P. et Le Pape C., *Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems* Constraints, 5, 119-139, 2000.

[BLR83] Blazewicz J., Lenstra J. et Rinnoy Kan,*Scheduling projects subject to resource constraints : classification and complexity* Discrete Applied Mathematics, 5 :11–24, 1983.

[BMR88] Bartusch M., Möhring R. et Radermacher F., *Scheduling project networks with resource constraints time windows* Annals of Operations Research, 16, 201–240, 1988.

[CL91] Carlier J. et Latapie B., *Une méthode arborescente pour résoudre les problèmes cumulatifs.* Operations Research, 25(3), 311-340, 1994.

[CN00] Carlier J. et Neron E., *A new LP-based lower bound for the cumulative scheduling problem.* European Journal of Operational Research, 127(2), 363–382, 2000.

[CN03] Carlier J. et Neron E., *Computing redundant resources for cumulative scheduling problems* European Journal of Operational Research, A paraiître, 2003.

[CP94] Carlier J. et Pinson E., *Adjustment of Heads and Tails for the Job-Shop Problem.* European Journal of Operational Research, 78, 146-161, 1994.

[DAM03] Demassey S., Artigues C. et Michelon P., *Constraint programming based cutting planes : an application to the rcpsp.* Informs Journal on Computing, (à paraître), 2004.

[ILO00] ILOG, *ILOG Scheduler 5.0 Rerence Manual.* ILOG, 2000.

[KH04] Kolish R. et Hartmann S., *Experimental Investigation of Heuristics for the Resource-Constrained Project Scheduling : An Update.* Rapport Technique, université technique de Munich, 2004.

[KP01] Kolish R.et Padman R., *An integrated survey of deterministic project scheduling.* Omega, 29(3), 249–272, 2001.

[KS99] Klein R. et Scholl A., *Computing Lower Bounds by Destructive Improvement : An Application to resource-Constrained Project Scheduling Problem* European Journal of Operational Research, 112, 332–346, 1999.

[KSD95] Kolisch R., Sprecher A. et Drexl A., *Characterization and generation of a general class of R.C.P.S.P.* Management Science, 41, 1693-1703, 1995.

[LEP94] Le Pape C., *Implementation of resource constraints in ILOG SCHEDULE : A library for development of constraint-based scheduling systems* Intelligent Systems Engineering, 3(2), 55-66, 1994.

[MSSU03] Möhring R. H., Schulz A., Stork F. et Uetz M.,*Solving project scheduling problems by minimum cut computations.* Management Science, 49 :330-350, 2003.

[PAC03] Palpant M., Artigues C. et Michelon P., *Lssepr : Solving the resource-constrained project scheduling problem with large neighbourhood search.* Rapport technique LIA 255, Laboratoire d'Informatique d'Avignon, 2003.

[SRM65] Shaffer L. R., Ritter J. B. et Meyer W. L., *The critical-path method.* Mc Graw Hill, 1965.

[SU04] Stork F. et Uetz M., *On the Generation of Circuits and Minimal Forbidden Sets* Mathematical Programming, (à paraître).

[VBQ04] Valls V., Ballestin F. et Quintanilla M. S., *A population- based approach to the resource-constrained project scheduling problem.* Annals of Operation Research, à paraître, 2004.

[VQB04] Valls V., Quintanilla M. S. et Ballestin F., *Resource-constrained project scheduling : A critical reordering heuristic.* European Journal of Operational Research, à paraître, 2004.