N° d'ordre : 02/2011 - D/MT

الجمهوريـــة الجـــزائريـــة الديمقراطيـــــة الشعبيـــة

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté de Mathématiques

وزارة التعليم العالي والبحث العلمي

جامعة هواري بومدين للعلوم والتكنولوجيا

كليـــة الرياضيات

THÈSE

Présentée pour l'obtention du grade de **DOCTORAT**

En **MATHEMATIQUES**
Spécialité : **Recherche Opérationnelle**

Par : **BENDRAOUCHE Mohamed**

Thème :

ORDONNANCEMENT SUR MACHINES IDENTIQUES AVEC GRAPHE DE CONCORDANCE

Soutenue publiquement le 11 Décembre 2011, devant le jury composé de :

M. KHELLADI Abdelkader, Professeur, USTHB.	Président
M. BOUDHAR Mourad, Professeur, USTHB.	Directeur de thèse
M. BERRACHEDI Abdelhafid , Professeur, USTHB.	Examinateur
M. CHERGUI Mohamed EL Amine, MC A, USTHB.	Examinateur
M. FINKE Gerd, Professeur, UJF- Grenoble, France.	Examinateur
M. OULAMARA Ammar, MC-HDR, EMNancy, France.	Examinateur

(بسم الله الرحمن الرحيم)

Remerciements

En tout premier lieu, je tiens à exprimer ma gratitude et ma reconnaissance à Monsieur Mourad Boudhar, mon directeur de thèse, Professeur à l'U.S.T.H.B pour m'avoir proposé ce sujet, guidé dans mes recherches et dirigé cette thèse. Je le remercie également pour la confiance qu'il m'a témoigné, pour ses conseils fructueux, sa patience, sa disponibilité, son soutien continu et pour son aide précieuse. Je lui suis redevable de m'avoir aidé à mieux comprendre les problèmes d'ordonnancement et la théorie de la complexité.

Mes sincères remerciements s'adressent aux personnes qui m'ont fait l'honneur de faire partie du Jury de ma thèse.

à Monsieur Abdelkader Khelladi, Professeur à l'USTHB, qui m'a fait l'honneur de présider ce Jury.
à Messieurs Abdelhafid Berrachedi (Professeur à l'USTHB), Mohamed El Amine Chergui (Maitre de Conférence à l'USTHB), Gerd Finke (Professeur à l'UJF-Grenoble, France et qu'il soit le bienvenu en Algérie) et Ammar Oulamara (Maitre de Conférences HDR, EMNancy, France) pour avoir accepté de juger ce travail et d'être membres de ce Jury.

Je remercie mes collègues de l'USTHB : Haned Amina, Miloud Mihoubi qui m'a toujours encouragé et surtout Wafaa Labbi qui m'a beaucoup aidé à l'écriture de ma thèse en Latex.

Enfin, je remercie bien sûr celle qui a fait de moi ce que je suis, celle grâce à qui tant d'années d'études ont été possibles, celle envers qui j'ai une dette imprescriptible : ma Mère. Un grand merci à ma femme Lila qui me supporte depuis bien des années et qui m'a toujours encouragé, à sa mère, ainsi qu'à mes enfants Soumia, Billel et Noussaiba. Sans oublier ma sœur Djamila et son mari Kaddour, et mes neveux : Djamel, Mohamed, Mohamed Amine et mes nièces Amel, Hadjer et Akila. Enfin, ce travail est dédié à la mémoire de mon père Abdelkader et mon frère Madani et ma fille Roufaida qu'Allah les recueillera dans son Paradis (El Djenna).

Résumé

Le travail présenté dans cette thèse, traite le problème d'ordonnancement de tâches en mode non préemptif, sur des machines identiques. Chaque tâche a une durée de traitement et une date de disponibilité et l'objectif est de minimiser la date de fin de traitement de l'ordonnancement (makespan). Nous supposons que seules certaines tâches spécifiques, peuvent être ordonnancées simultanément sur deux machines différentes. Ce problème est NP-difficile. Nous représentons ces contraintes par un graphe appelé graphe de concordance et nous considérons plusieurs cas possibles de graphes : graphes bipartis, complémentaires de graphes bipartis et graphes scindés. Nous montrons que certains sous problèmes sont NP-difficiles et que d'autres sont polynomiaux, en particulier nous montrons la NP-difficulté d'un problème ouvert proposé dans la littérature, qui est le cas de deux machines avec au plus 3 durées de traitement possibles. Pour la résolution des sous problèmes polynomiaux cités, nous proposons des algorithmes polynomiaux exacts. Quant au problème dans le cas où toutes les tâches sont disponibles à l'instant 0, nous proposons des heuristiques de type algorithmes de liste avec des expérimentations numériques.

Mots clés : Ordonnancement, machines identiques, graphe de concordance, makespan, heuristique, algorithme de liste, complexité.

Abstract

In this thesis, we consider the problem of scheduling jobs non-preemptively on identical machines subject to constraints that only some specific jobs can be scheduled concurrently on different machines. We suppose that each job has a processing time and a release date and the aim is to minimise the makespan. This problem is NP-hard. We represent the constraints above by a graph called an agreement graph and consider various possible cases of graphs : bipartite graphs, complement of bipartite graphs and split graphs. We prove that some subproblems are NP-hard and others are polynomial, in particular we prove the NP-hardness of an open problem in the literature, which is the case of two machines with at most 3 possible processing times. For the polynomial subproblems cited, we propose exact polynomial algorithms for their solution. We propose list algorithm type heuristics with empirical results for the problem when all the jobs are ready at time zero.

Keywords : Scheduling, identical machines, agreement graph, makespan, heuristic, complexity, list scheduling.

Table des matières

Liste des tableaux

Table des figures

Introduction

L'ordonnancement est un domaine d'investigation qui a connu un développement considérable ces dernières années, tant par les nombreux problèmes identifiés que par l'utilisation et l'élaboration de techniques de résolution. Parmi les problèmes d'optimisation combinatoires, les problèmes d'ordonnancement sont probablement les plus liés au monde industriel puisqu'il s'agit de concilier, de façon optimale, des ressources limitées avec des tâches dans le temps.

Dans le problème d'ordonnacement sur des machines parallèles, un ordonnancement consiste à déterminer :

- le placement des tâches sur les machines.
- les dates d'exécution de ces tâches.

Dans le problème classique d'ordonnancement sur des machines parallèles identiques, on ne tient pas compte de l'exécution simultanée des tâches sur les machines. Par contre dans beaucoup de cas, il existe des contraintes entre les tâches dites contraintes de conflit : certaines d'entre elles ne peuvent pas être exécutées simultanément sur deux machines différentes parcequ'elles partagent une même ressource. Ce genre de problème est appelé problème d'ordonnancement avec conflit.

Dans la littérature deux problèmes fondamentaux d'ordonnancement avec conflit on été étudiés. Le premier est le problème d'ordonnancement avec exclusion mutuelle introduit par Baker et Coffman (Mutual Exclusion Scheduling) et le second problème appelé "Scheduling With Conflicts" étudié récement par Even et al.. Dans ces derniers, les contraintes de conflit ont été modélisées par un graphe appelé graphe de conflit où deux tâches adjacentes de ce graphe ne peuvent pas être ordonnancées simultanément sur deux machines différentes.

Lors de la recherche d'un problème ouvert posé par Even et al., nous avons trouvé une grande difficulté dans la manipulation des preuves, particulièrement dans les réductions polynomiales. Une des motivations de notre travail est la modélisation de ces contraintes de conflit par le complémentaire du graphe de conflit, que nous avons appelé graphe de

concordance. Notre problème est donc une version équivalente au problème d'ordonnancement avec conflit, utilisant le graphe de concordance. Ceci nous a été de grande aide et nous a donné une bonne vision pour la manipulation de tous les résultats obtenus.

Cette thèse est constituée de sept chapitres et est organisée comme suit. Dans le premier chapitre nous présentons les principaux termes et notions utilisés tout au long de cette thèse. Au premier lieu, nous donnons quelques notions de base de la théorie des graphes. Ensuite nous présentons les problèmes d'ordonnancement, leurs domaines d'application ainsi que leur classification.

Le second chapitre est composé de deux parties. Dans la première partie, nous donnons une introduction à la théorie de la complexité et dans la seconde nous présentons une introduction aux problèmes d'optimisation combinatoires ainsi que les méthodes classiques de résolution.

Le chapitre 3 est consacré aux définitions, notations du problème général ainsi que de tous les sous problèmes traités dans cette thèse. Nous exposons dans ce chapitre un état de l'art du problème traité et nous citons les principaux résultats connus dans la littérature. Aussi nous proposons une modélisation mathématique du problème, sous forme d'un programme linéaire en variables bivalentes et réelles.

Quant au quatrième chapitre nous étudions la complexité du problème dans le cas des graphes bipartis, en particulier nous montrons la NP-difficulté d'un problème ouvert posé dans la littérature, qui est le cas de deux machines avec au plus trois durées de traitement possibles.

Dans le chapitre 5 est considéré la complexité du problème dans le cas où le graphe de concordance est le complémentaire d'un graphe biparti. Le sixième chapitre étudie la complexité du problème dans le cas des graphes scindés. Dans chacun des trois derniers chapitres cités, nous montrons que certains problèmes sont NP-difficiles et que d'autres sont polynomiaux. Pour ces derniers nous proposons des algorithmes polynomiaux exacts pour leur résolution. Dans le dernier chapitre, nous proposons des heuristiques de type algorithmes de liste pour le problème dans le cas où toutes les tâches sont diponibles à l'instant 0. Tous ces algorithmes ont été comparés expérimentalement sur des instances générées aléatoirement.

Enfin, dans la conclusion nous dressons un tableau récapitulatif des principaux résultats présentés dans cette thèse, et nous proposons des perspectives.

Chapitre 1

Graphes et Problèmes d'ordonnancement

Dans ce chapitre, nous abordons les notions principales utilisées dans cette thèse. D'abord nous présentons quelques notions de base de la théorie des graphes, en se basant surtout sur les définitions des types de graphes étudiés. Dans le second paragraphe, nous rappelons la notion de réseaux et le problème du flot maximum. Ensuite, dans le dernier paragraphe, nous présentons une introduction aux problèmes d'ordonnancement où nous donnons leurs notations ainsi que leurs classifications.

1.1 Notions de base de la théorie des graphes

Un graphe G (sans boucles) est défini par la donnée d'un ensemble fini V, appelé ensemble de sommets et d'un ensemble E de paires non ordonnées de sommets distincts de V, dites arêtes. On note ce graphe par $G = (V, E)$.

Soit $G = (V, E)$ un graphe, si $e = \{x, y\}$ est une arête de G, e est représentée par une ligne. On dira dans ce cas que les deux sommets x et y sont reliés ou adjacents. L'arête e a pour extrémités x et y.
Une arête e est dite incidente à un sommet x si ce sommet est une extrémité de cette arête.
Un sommet y est dit un voisin de x si x et y sont adjacents. Si x est un sommet de G, le voisinage de x est l'ensemble $V(x)$ formé par tous les voisins de x. Un sommet de G est dit isolé si son voisinage est un ensemble vide. Le degré $d(x)$ de x est la cardinalité du voisinage de x : $d(x) = |V(x)|$.

Exemple 1.1. $V = \{v_1, v_2, ..., v_6\}$, $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_5\}, \{v_2, v_4\}, \{v_4, v_5\}\}$
le sommet v_6 *est un sommet isolé*

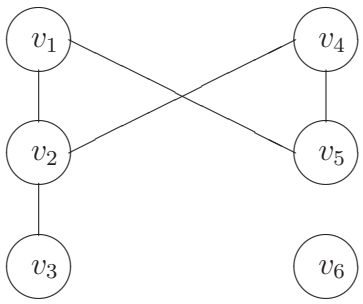

FIGURE 1.1 – Graphe $G = (V, E)$ de l'exemple 1.1

Un graphe G =(V,E) est dit isomorphe à un graphe $G' = (V', E')$ s'il existe une bijection $f : V \longrightarrow V'$ tel que $\{x, y\} \in E$ si et seulement si $\{f(x), f(y)\} \in E'$.

Le complémentaire de G est le graphe noté $\overline{G} = (V, \overline{E})$ tel que $\{x, y\} \in \overline{E}$ si et seulement si $\{x, y\} \notin E$.

Le sous-graphe de G induit par un sous-ensemble $V' \subseteq V$ est le graphe $G' = (V', E')$ tel que E' est l'ensemble des arêtes de G ayant les deux extémités dans V'. Un couplage dans un graphe $G = (V, E)$ est un sous-ensemble d'arêtes E' tel que deux arêtes de E' ne soient pas incidentes au même sommet.

Une chaîne de longueur l, est une séquence de sommets $x_0, x_1, ...x_l$ tels que les sommets x_{i-1} et x_i sont reliés par une arête, pour tout i $(i = \overline{1, l})$, x_0 et x_l sont appelés les extrémités de la chaine. Un cycle est une chaine fermée (les extrémités coïncident) dont toutes les arêtes sont distinctes. Une corde dans un cycle est une arête joignant deux sommets non consécutifs de ce cycle.

Un graphe est dit complet si toute paire de sommets est une arête. Soit G un graphe, une clique de G est un sous-ensemble de sommets de G, deux-à-deux adjacents. Le graphe complet sur n sommets est généralement noté K_n, le graphe K_3 est communément appelé un triangle.

Un stable de G est un sous-ensemble de sommets deux à deux non-adjacents.

Un graphe $G = (V, E)$ est dit biparti, si l'ensemble V de ses sommets peut être partitionné en deux stables S_1 et S_2. Un tel graphe est noté $G = (S_1; S_2, E)$.

Soit G est un graphe, $\omega(G)$ désigne la taille d'une clique maximum du graphe G. Par analogie, La stabilité $\alpha(G)$ dénote la taille d'un stable maximum de G.

1.1.1 Graphes parfaits et quelques sous classes

Une k-coloration d'un graphe $G = (V, E)$ est une partition de V en k classes $S_1, S_2, ..., S_k$, les sommets de la classe S_i étant coloriés avec la couleur i ($i = \overline{1, k}$) de sorte que deux sommets adjacents n'aient pas la même couleur. Notons qu'une k-coloration d'un graphe $G = (V, E)$ correspond à une partition de G en k stables. Le nombre chromatique $\chi(G)$ d'un graphe G, est le nombre minimum k pour lequel G admet une k-coloration.

Définition 1.1. *Un graphe $G = (V, E)$ est dit parfait si et seulement si pour tout sous-graphe induit H de G, on a : $\omega(H) = \chi(H)$*

Notons que les graphes bipartis sont des graphes parfaits. Parmi les graphes parfaits classiques on distingue deux grandes classes [36] :

1. **Les graphes triangulés :** Un graphe est dit triangulé si tout cycle de longueur supérieure strictement à 3 admet une corde. On les appelle "chordal graphs" dans la littérature Anglophone. Deux sous-classes classiques des graphes triangulés sont :

   - **Les graphes scindés :** Un graphe $G = (V, E)$ est dit scindé, si l'ensemble V de ses sommets peut être partitionné en deux sous-ensembles S et K tels que S est un stable et K est une clique. Un tel graphe est noté $G = (S; K, E)$.

   - **Les graphes d'intervalles :** Un graphe $G = (V, E)$ est dit un graphe d'intervalles si on peut représenter ses sommets par des intervalles de sorte que deux sommets sont reliés par une arête si et seulement si les intervalles correspondants s'intersectent.
     Un cas particulier des graphes d'intervalles, sont les graphes d'intervalles propres : Un graphe est un graphe d'intervalle propre s'il admet une représentation par des intervalles dans laquelle aucun intervalle n'en contient un autre.

2. **Graphes de comparabilité :** Un graphe est dit de comparabilité si ses arêtes peuvent être transitivement orientées. Une orientation transitive des arêtes satisfait

la condition suivante : s'il existe un arc d'un sommet x vers un sommet y et un arc allant de y vers un sommet z, alors il existe aussi un arc allant de x vers z.

Le livre de Golumbic [36] constitue une excellente introduction aux graphes parfaits.

1.1.2 Réseaux et problème du flot maximum

Définition 1.2. *Un réseau noté $R = (V, U, c)$, est un graphe orienté $G = (V, U)$ (chaque arête a une orientation, appelé arc) muni d'une fonction $c : U \longrightarrow \mathbb{R}$ tel que le nombre $c(u)$ est appelé la capacité de l'arc u.*

Définition 1.3. *Soit $R = (V, U, c)$ un réseau dans lequel un arc spécial $u_r = (p, s)$ a été distingué. Le sommet s est dit le sommet source et p est le sommet puits, l'arc u_r est dit arc de retour. Pour tout sommet $x \in V$, on note $w^+(x)$ l'ensemble des arcs sortant de x, $w^-(x)$ l'ensemble des arcs entrant en x. c est une fonction $c : U \longrightarrow \overline{\mathbb{R}_+}$, qui à chaque arc u, associe sa capacité $c(u)$ avec $c(u_r) = +\infty$. Le problème du flot maximum de s à p sur le réseau R consiste à chercher une fonction $f : U \longrightarrow \mathbb{R}$ vérifiant :*

1. $\sum_{u \in w^+(x)} f(u) = \sum_{u \in w^-(x)} f(u)$ *quel que soit* $x \in V$
2. $0 \leq f(u) \leq c(u)$ *quel que soit* $u \in U$
3. $f(u_r)$ *soit maximum*

Pour tout $u \in U$, $f(u)$ s'appelle le flux sur l'arc u. La condition 1 exprime que f est un flot sur R, elle est appelée Loi de conservation de flot : pour tout sommet x, la somme des flux entrant en x est égale à la somme des flux sortant de x. Le flot f vérifiant la condition 2 est dit réalisable. La quantité $f(u_r) = \sum_{u \in w^+(s)} f(u) - \sum_{u \in w^-(s) \setminus \{u_r\}} f(u)$, est appelée la valeur du flot f. $f(u_r)$ est aussi égal à $\sum_{u \in w^-(p)} f(u) - \sum_{u \in w^+(p) \setminus \{u_r\}} f(u)$. En d'autres termes, il s'agit de trouver un flot réalisable sur le réseau R de valeur maximale. $\overline{\mathbb{R}_+} = \mathbb{R}_+ \cup \{+\infty\}$.

Formulation en Programmation Linéaire

Considérons la matrice d'incidence A aux arcs du graphe $G = (V, U)$. Si nous considérons f comme étant un vecteur, dans la théorie des graphes on montre que f est un flot sur le réseau R si et seulement si le produit matriciel $Af = 0$. Par conséquent le problème du flot maximum de s à p s'écrit :

$$\begin{cases} Max\ Z = f(u_r) \\ Af = 0 \\ 0 \leq f \leq c \end{cases}$$

Dans ce qui suit on présentera l'algorithme de Ford et Fulkerson pour sa résolution. Pour une bonne compréhension des problèmes de flots voir [48].

Algorithme de Ford et Fulkerson

Le principe de cet algorithme est le suivant : on part d'un flot réalisable sur R, par exemple un flot nul. On marque le sommet s, puis on utilise les processus de marquage suivants :

- **Marquage direct** : s'il existe un arc $u = (x, y)$ tel que x est marqué, y non marqué et que $f(u) < c(u)$, on marque le sommet y.

- **Marquage indirect** : s'il existe un arc $u = (y, x)$ tel que x est marqué, y non marqué et que $f(u) > 0$, on marque le sommet y.

Deux cas sont alors possibles :

cas 1 : on arrive à marquer le sommet p. Soit C la chaine d'extrémités s et p par laquelle le marquage de p a été effectué et soit C^+ l'ensemble des arcs de C traversés quand on parcourt C de s à p et C^- l'ensemble des arcs de C traversés quand on parcourt C de p à s. Soit Γ le cycle formé par la chaine C et l'arc u_r. On calcule les nombres $\delta_1 = \min_{u \in C^+}\{c(u) - f(u)\}$, $\delta_2 = \min_{u \in C^-}\{f(u)\}$ puis $\delta = min\{\delta_1, \delta_2\}$. On détermine alors un nouveau flot f' défini par :

$$f'(u) = \begin{cases} f(u) + \delta & \text{si } u \in C^+ \cup \{u_r\} \\ f(u) - \delta & \text{si } u \in C^- \\ f(u) & \text{si } u \notin \Gamma \end{cases}$$

Comme dans ce cas le marquage de p a été possible donc $\delta > 0$ et par conséquent le flot f' améliore le flot f. On répète cette procédure en partant du nouveau flot f'.

cas 2 : on n'arrive pas à marquer p , alors d'après le théorème de la coupe minimum [48], le flot courant est maximum.

1.2 Problèmes d'ordonnancement

La définition suivante est due à Carlier et Chrétienne [21] :

Définition 1.4. *Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant les ressources aux tâches et en fixant leurs dates d'exécution*

Les problèmes d'ordonnancement apparaissent dans de nombreux domaines : informatique (tâches : jobs ; ressources : processeurs), chantiers (suivi de projets), l'industrie (problèmes d'atelier, gestion de production), administration, écoles et universités (personnel, emplois du temps).

Les éléments essentiels d'un problème d'ordonnancement sont : les tâches, les ressources, les contraintes et les objectifs.

1.2.1 Tâches

Une tâche est définie par un ensemble d'opérations qui doivent être exécutées. Les tâches peuvent représenter des pièces mais également des projets, des programmes ou des activités. Une tâche notée J_i peut être définie par ses caractéristiques temporelles :

- p_i : la durée de traitement de la tâche J_i.
- r_i : la date de disponibilité de la tâche J_i.
- d_i : la date échue de la tâche J_i.

1.2.2 Ressources

Une ressource est un moyen matériel (machine) ou humain intervenant dans la réalisation d'une tâche.
Il existe deux principaux types de ressources :

les ressources renouvelables

Elles sont disponibles en quantité constante tout au long de l'exécution des tâches. Les ressources renouvelables usuelles sont les machines, les processeurs, le personnel etc.

Les ressources consommables

Une ressource est consommable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches restant à exécuter, c'est le cas des matières premières et de l'énergie par exemple.

1.2.3 Contraintes

Les contraintes représentent les limitations imposées par l'environnement. Dans la plupart des problèmes d'ordonnancement les tâches à exécuter sont soumises à des contraintes qu'il faut satisfaire au moment de la recherche d'une solution optimale, on distingue trois principales types de contraintes :

- **Les contraintes potentielles :** Ceux sont les contraintes de localisation temporelle par exemple " la tache i doit précéder la tache j " ou la tache i doit être achevée avant telle date.

– **Les contraintes disjonctives :** Lorsque deux tâches ne peuvent pas être exécutées en même temps, on dit qu'il y a une contrainte disjonctive que doivent satisfaire ces deux tâches.

– **Les contraintes cumulatives :** Elles concernent l'évolution dans le temps du volume total des moyens humains ou matériels consacrés à l'exécution des tâches.

1.2.4 Ordonnancement

Un ordonnancement constitue une solution au problème d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps et vise à satisfaire un ou plusieurs objectifs. Les variables intervenant le plus souvent dans un ordonnancement sont :

– t_i : la date de début de la tâche J_i.
– C_i : la date de fin de traitement de la tâche J_i.
– $l_i = C_i - d_i$: tardivité de la tâche J_i.
– $t_i = \max\{l_i, 0\}$: retard de la tâche J_i.
– U_i : indicateur de retard de la tâche J_i, $U_i = \begin{cases} 1 & \text{si } C_i > d_i \\ 0 & \text{sinon} \end{cases}$
– $F_i = C_i - r_i$: le temps de flot (flow time).

Selon les problèmes, les tâches peuvent être exécutées par morceaux, le mode de traitement est dit préemptif. Il est dit non préemptif si celles-ci doivent être exécutées sans interruption.

1.2.5 Critères

Dans un problème d'ordonnancement on cherche un ordonnancement minimisant une fonction objectif, appelé critère. Les critères que l'on peut vouloir optimiser sont nombreux, dont on trouve : $X_{max} = \max\limits_{1 \leq i \leq n}\{X_i\}$, $\sum X_i = \sum\limits_{i=1}^{n} X_i$, $\sum w_i X_i = \sum\limits_{i=1}^{n} w_i X_i$ avec $X \in \{C, F, L, U\}$.

– C_{max} : date de fin de traitement de l'ensemble des tâches (makespan).

– $\sum C_i$: la somme des dates de fin de traitement.

– $\sum w_i C_i$: la somme pondérée des dates de fin de traitement.

– D_{max} : le grand retard.

– L_{max} : la grande tardivité.

– F_{max} : le grand temps de flow.

– $\sum U_i$: le nombre de tâches en retard.

– $\sum w_i U_i$: la somme pondérée du nombre de tâches en retard.

– d'autres critères peuvent être définis.

Dans cette thèse, le critère étudié est le makespan.

1.2.6 Diagramme de Gantt

Les ordonnancements sont généralement représentés par des diagrammes dits de Gantt, voir l'exemple de la figure 1.2. Ceux ci indiquent, selon une échelle temporelle donnée, l'occupation des machines par les différentes tâches, les temps morts (parties hachurées) dus, par exemple aux éventuelles indisponibilités des tâches ou contraintes potentielles.

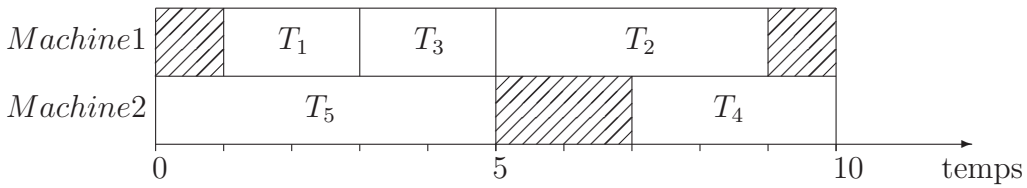

FIGURE 1.2 – Exemple de diagramme de Gantt

1.2.7 Ordonnancement d'atelier

Dans les problèmes d'ordonnancement d'atelier, les ressources sont les machines, ne pouvant réaliser qu'une tâche à la fois, celle-ci pouvant être composée d'une ou plusieurs opérations. Parmi les problèmes fréquemment rencontrés dans la littérature, on trouve les problèmes à une machine unique et les problèmes multi-machines. Dans le second cas on distingue les problèmes à machines parallèles et ceux à machines dédiées. Dans le premier cas chaque tâche est réduite à une seule opération : il faut alors trouver une séquence de tâches sur la machine unique. Dans l'autre cas, on distinguera d'abord les problèmes à machines parallèles, puis ceux à machines dédiées.

1.2.6.1 Problème à une machine unique

Chaque tâche est composée d'une seule opération. Toutes les tâches à réaliser sont alors exécutées par une seule machine.

1.2.6.2 Problème à machines parallèles

Ce problème se subdivise en trois classes selon les vitesses d'exécution des machines :

- **Machines identiques :** Toutes les machines ont la même vitesse de traitement des tâches quel que soit la tâche.

- **Machines uniformes :** Chaque machine a sa propre vitesse qui ne dépend pas des tâches à exécuter.

- **Machines générales :** Contrairement au cas précédent, la vitesse de traitement des tâches dépend de la machine et de la tâche à exécuter.

1.2.6.3 Le problème à machines dédiées (spécialisées)

Elles sont spécialisées à l'exécution de certaines opérations. Dans cette catégorie, chaque tâche est constituée de plusieurs opérations. En fonction du mode de passage des opérations sur les différentes machines, trois ateliers spécialisés sont différenciés :

- **Le problème à cheminement unique (Flow shop) :** Les tâches doivent être traitées par toutes les machines, et l'ordre de passage des opérations sur ces machines est le même pour toutes les tâches.

- **Le problème à cheminements multiples (Job shop) :** Chaque tâche est traitée par un sous ensemble de machines et l'ordre de passage des opérations de chaque tâche est propre à cette tâche et est spécifié à l'avance.

- **Le problème à cheminement quelconque (Open shop) :** Les tâches doivent être traitées par toutes les machines, et l'ordre de passage des opérations sur ces machines est quelconque.

1.3 Classification et notations

Pour des raisons de clarté et de compréhension, il est important d'adopter un descripteur qui soit basé sur l'existant et homogénéisant les notations actuelles.

La notation $(\alpha|\beta|\gamma)$ introduite par Graham et al. [38] est couramment utilisée pour distinguer les différents problèmes d'ordonnancement entre eux et les classifier.

Champ α

Renseigne les informations relatives aux ressources, à savoir le nombre de machines, type de machines, et type d'atelier. Il est composé de deux sous champs et désigné par $\alpha_1\alpha_2$:

- $\alpha_1 \in \{1, P, Q, R, F, O, J\}$
  1 : machine unique.
  P : machines parallèles identiques.
  Q : machines parallèles uniformes.
  R : machines parallèles générales.
  F : flow shop.
  O : open shop.
  J : job shop.

- $\alpha_2 \in\{\phi$, m $\}$ $(\alpha_1 \neq 1)$
  Lorsque $\alpha_2 = \phi$, le nombre de machines est variable, tandis que lorsque $\alpha_2 = m$, le nombre de machines est fixé à m.

Champ β

Permet d'obtenir des informations sur les contraintes prises en compte même sur les tâches qui doivent être ordonnancées. Il est composé de 5 sous champs $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$:

- $\beta_1 \in \{\phi, pmtn\}$
  $\beta_1 = \phi$: la préemption des tâches n'est pas autorisée.
  $\beta_1 =$ pmtn : la préemption est autorisé.

- $\beta_2 \in \{\phi$, prec, tree, chain$\}$
  $\beta_2 = \phi$: pas de contraintes de précédence.
  $\beta_2 =$ prec : les contraintes de précédence sont quelconques.
  $\beta_2 =$ tree : les contraintes de précédence sont données sous forme d'arborescence.
  $\beta_2 =$ chain : les contraintes de précédence sont données sous forme de chaînes.

- $\beta_3 \in\{\phi, r_i\}$
  $\beta_3 = \phi$: toutes les dates de disponibilité sont nulles.
  $\beta_3 = r_i$: il existe des dates de disponibilité pour les tâches.

– $\beta_4 \in \{\phi,\ p_i = p,\ \underline{p} \leq p_i \leq \overline{p}\ \}$
$\beta_4 = \phi$: les temps de traitement des tâches sont quelconques.
$\beta_4 = p_i = p$: les temps de traitement des tâches sont égaux à p.
$\beta_4 = \underline{p} \leq p_i \leq \overline{p}$: le temps de traitement de chaque tâche est compris entre $\underline{p}$ et $\overline{p}$.

– $\beta_5 \in \{\phi,\ d_i,\ \tilde{d}_i\ \}$
$\beta_5 = \phi$: il n'y a pas des dates échues pour les tâches, ou il existe des dates échues pour les tâches dans le cas ou le critère d'optimisation est en fonction de ces derniers.
$\beta_5 = d_i$: il existe des dates échues pour les tâches.
$\beta_5 = \tilde{d}_i$: il existe une date limite pour chaque tâche.

Champ γ

Le troisième champ γ décrit le critère d'optimalité. On cherche à minimiser γ avec $\gamma \in \{\ C_{max},\ \sum C_i,\ \sum w_i C_i,\ D_{max},\ L_{max},\ F_{max},\ \sum U_i,\ \sum w_i U_i ...\}$.

1.3.1 Exemples

Exemple 1.2. *Le problème $1|r_i|C_{max}$ est le problème d'ordonnancement de tâches indépendantes sur une seule machine avec des durées de traitement et des dates de disponibilité arbitraires, l'objectif est la minimisation du makespan.*

Exemple 1.3. *Le problème $P2||C_{max}$ est le problème d'ordonnancement de tâches indépendantes sur deux machines parallèles identiques avec des durées de traitement arbitraires.*

Exemple 1.4. *Le problème $P|tree, pmtn|C_{max}$ est le problème d'ordonnancement de tâches indépendantes et préemptives sur un nombre arbitraire de machines parallèles identiques avec des contraintes de précédence données sous forme d'arborescence, l'objectif est de minimiser le makespan .*

Chapitre 2

Théorie de la complexité et optimisation combinatoire

Dans ce chapitre nous introduisons les notions de problèmes, d'algorithmes et leurs complexités ainsi que la complexité des problèmes. Le lecteur intéressé par de plus amples informations pourra consulter le livre de base de Garey and Johnson [32]. Nous exposons ensuite la notion de problèmes d'optimisations combinatoires et les principales méthodes de résolution.

2.1 Problèmes, algorithmes et complexité

Un problème est une question générale à laquelle on veut apporter une réponse (ou une solution), possédant généralement des paramètres. Il est formalisé par (i) une description de tous ses paramètres formels dont les valeurs ne sont pas spécifiées et (ii) une indication précisant les propriétés qui doivent être vérifiées par la réponse (ou la solution).

Une instance d'un problème est obtenue en attribuant une valeur à chacun de ses paramètres. Résoudre un problème c'est donner une solution à toutes ses instances. Cette résolution s'effectue par la définition d'un algorithme, c'est-à-dire une suite d'opérations élémentaires à effectuer.

Parmi les problèmes qui nous intéressent, il existe généralement deux grands types :

– **Les problèmes de décisions :** ceux sont les problèmes posant une question dont la réponse est "oui" ou "non". Ils sont formalisés par deux composantes : une instance générique précisant les paramètres formels du problème et une question.

  Exemple : **Partition :** Instance : n entiers positifs $a_1, a_2, ..., a_n$. Question : existe-t-il

un sous-ensemble $J \subseteq I = \{1, 2, ..., n\}$ tel que $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$?

– **Les problèmes d'optimisation :** Ce sont les problèmes cherchant à optimiser (maximiser ou minimiser) une certaine valeur.

Exemple : **"Le problème du voyageur de commerce" :** Etant donné un ensemble de n villes et pour chaque paire de villes $\{v_i, v_j\}$, la distance de la ville v_i à la ville v_j. Trouver un parcours fermé passant par toutes les villes (une et une seule fois) minimisant la distance totale parcourue.

2.1.1 Complexité d'un algorithme, efficacité et taille d'entrée

Un algorithme qui résout un problème prend en entrée une instance I, dite une entrée du problème. D'autre part pour implémenter cet algorithme sur un ordinateur, cette instance est codée sous forme de chaines binaires de 0 et 1 (codage binaire). La longueur de cette chaine définit la taille de l'entrée notée $|I|$.
La complexité d'un algorithme est une fonction $T(|I|)$, qui est égale au nombre maximum d'opérations élémentaires requises par l'exécution de l'algorithme sur une entrée de taille $|I|$ (on considère donc le pire des cas possibles).

L'efficacité d'un algorithme pour un problème se caractérise par sa complexité : on dira qu'un algorithme est polynomial ou efficace si sa complexité est en $O(|I|^k)$ pour une certaine constante k ($|I|$ étant la taille d'entrée). En revanche, un algorithme dont la complexité ne peut pas être bornée polynomialement est qualifié d'exponentiel, par exemple des algorithmes de complexité en $O(2^{|I|})$ et en $O(|I|!)$.

La complexité d'un algorithme va dépendre non seulement de l'algorithme lui même mais aussi de la manière dont est codée l'entrée. Toutefois, d'après la définition précédente un algorithme efficace pour un codage raisonnable de l'entrée (le restera pour tout codage qui lui est polynomialement équivalent (c'est à dire que la taille par rapport au le premier codage est majorée par une fonction polynomiale de celle du second, et inversement).
Ainsi, dans la théorie de la complexité on peut remplacer la taille d'entrée (comme elle a été définie ci-dessus par les chaines de 0 et 1) par un paramètre "intuitif" qui, souvent et par abus de language représentera **la taille d'entrée**. Voici quelques exemples :

– si l'entrée est un graphe $G = (V, E)$: tel que $|V| = n$, $|E| = m$ la taille d'entrée peut être soit n, soit m ou $n + m$.
– si l'entrée est une matrice mxn : la taille d'entrée est le maximum, la somme ou le produit de m et n.

– si l'entrée est un polynôme : la taille d'entrée est son degré ou le nombre de coefficients.

Lors du calcul de la complexité d'un algorithme, on cherche un ordre de grandeur plutôt que le nombre exact. On utilise pour ce faire, la notation de Landau O. Si la taille d'entrée est n, un algorithme dont la complexité est $T(n) = 2n^3 + 5n^2 + 4$ est en $O(n^3)$.

A titre d'exemple, la complexité de l'algorithme classique du produit de deux matrices carrées d'ordre n se calcule comme suit : il existe n^2 éléments à calculer. Or chacun de ces éléments nécessite n multiplications et $n - 1$ additions, ce qui fait $2n - 1$ opérations élémentaires. Par conséquent la complexité de cet algorithme est en $O(n^3)$.

2.1.2 Classes de complexité des problèmes

La classification usuelle des problèmes se fait en considérant leurs complexités. Celle-ci correspond à la complexité de l'algorithme le plus efficace pour résoudre ce problème. A cause des difficultés rencontrés, la théorie de la complexité ne traite fondamentalement que les problèmes de décision, car leur formalisme Oui-Non, Vrai-Faux a permis de les étudier avec les outils de la logique mathématique. Cependant on étend la notion de complexité aux problèmes d'optimisation. En effet il est facile de transformer un problème d'optimisation en problème de décision. Par exemple chercher à optimiser une certaine valeur v revient à traiter un problème de décision qui consiste à comparer v à un certain k, puis en générant un certain nombre de valeurs k, on peut déterminer la valeur optimale (exemple par l'utilisation de l'algorithme de dichotomie).

Les trois classes de complexité de problèmes les plus courantes sont P, NP et NP-Complet :

La classe P (Polynomial)

Cette classe se compose de tous les problèmes de décision qui sont résolus par des algorithmes polynomiaux par rapport à la taille d'entrée.

La classe NP (Nondeterministic Polynomial)

Elle regroupe tous les problèmes de décision qui peuvent être résolus en temps polynomial par des algorithmes non déterministes. Un algorithme est dit non déterministe s'il comporte des instructions de choix. Pour ces algorithmes, si à chaque instruction, le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés l'algorithme devient déterministe et son temps de calcul devient exponentiel. De

façon informelle, un problème de décision appartient à NP si on peut vérifier en un temps polynomial, qu'une solution proposée (ou devinée) permet d'affirmer que la réponse est "oui".

Remarque 2.1. *Les algorithmes polynomiaux sont évidemment des cas particuliers des algorithmes non déterministes. Alors tout problème de décision de la classe P, appartient également à la classe NP, d'où* $P \subseteq NP$.

2.1.3 Classe des problèmes NP-Complets

Parmi les problèmes de la classe NP, il existe une large classe de problèmes : les problèmes NP-Complets, qui sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre. C'est-à-dire s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de la classe NP. Les problèmes NP-Complets sont les problèmes les plus difficiles de la classe NP. Afin de décrire cette classe d'équivalence, définissons tout d'abord la transformation (réduction) polynomiale entre deux problèmes.

Définition 2.1. *Soient* D_1 *et* D_2 *deux problèmes de décision. Une transformation polynomiale de* D_1 *vers* D_2 *peut être vue comme une fonction* f *de l'ensemble des instances de* D_1 *vers l'ensemble des instances de* D_2 *qui satisfait les deux conditions suivantes :*

1. f *est calculable par un algorithme polynomial*
2. *Pour toute instance* I *de* D_1, I *a pour réponse "oui" (pour* D_1*) si et seulement si* $f(I)$ *a pour réponse "oui" (pour* D_2*).*

Si une telle transformation existe, on dira que D_1 se transforme (réduit) polynomialement en D_2, et on écrit $D_1 \propto D_2$.

D'une manière équivalente, on dit que D_1 se transforme (réduit) polynomialement en D_2, s'il existe un algorithme de résolution de D_1, qui fait appel à un algorithme de résolution de D_2 et qui est polynomial lorsque la résolution de D_2 est comptabilisée comme une opération élémentaire.

On en déduit que : si $D_1 \propto D_2$ et s'il existe un algorithme polynomial pour résoudre D_2, alors il existe un algorithme polynomial pour résoudre D_1.

Définition 2.2. *Un problème de décision* D *est NP-Complet si :*

1. D *est dans NP*

2. Tout problème D' de NP se réduit polynomialement à D

La classe des problèmes de décision NP peut être partitionnée en 3 sous classes : les problèmes de la classe P, les problèmes NP-Complets et les autres, voir figure 2.1.

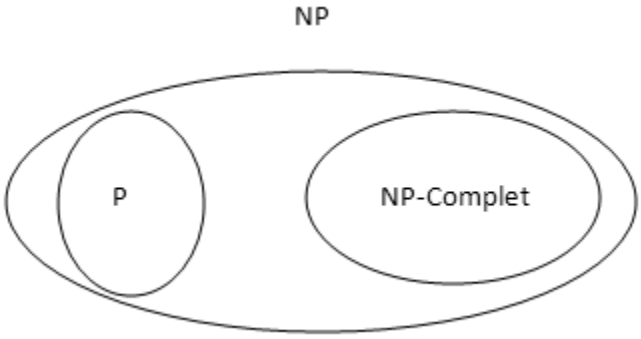

FIGURE 2.1 – Géographie de la classe NP

Le premier problème qui a été prouvé NP-Complet par S.A. Cook en 1970, est le problème de Satisfiabilité (SAT), qui est un problème de Logique. Pour le définir, on considère un ensemble U de variables booléennes qui peuvent recevoir deux valeurs possibles "vrai" ou "faux" ; à chaque variable u est associée la variable complémentée $\overline{u}$ qui est vraie quand u est fausse et réciproquement. Une clause de U est un ensemble de variables complémentées ou non, qui prend la valeur "vrai" dés qu'une de ses variables prend la valeur "vrai".

Le problème SAT :
Instance : Un ensemble U de variables booléennes et une collection C de clauses sur U.
Question : Existe-t-il une affectation en "vrai" et "faux" des variables telle que toutes les clauses de C prennent la valeur "vrai" ?

Si la réponse est "oui" on dit que C est satisfiable. Exemple : Pour l'instance $I : U = \{u_1, u_2\}$, $C = \{\{u_1, \overline{u_2}\}, \{\overline{u_1}, u_2\}\}$, la réponse est oui, il suffit de prendre valeur "vrai" pour u_1 et pour u_2. Par contre pour l'instance $I' : U = \{u_1, u_2\}$, $C' = \{\{u_1, u_2\}, \{u_1, \overline{u_2}\}, \{\overline{u_1}\}\}$, la réponse est non.

2.2 Prouver la NP-complétude d'un problème

La démonstration d'appartenance d'un problème de décision à la classe des problèmes NP-Complets est très importante car, une fois cette démonstration est faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce

problème.

Montrer qu'un problème de décision D est NP-Complet consiste en les trois étapes suivantes :

- Montrer que D appartient à NP
- Choisir D', un problème NP-Complet
- Construire une transformation polynomiale f de D' vers D

2.2.1 Quelques problèmes NP-Complets de base

Nous citons six problèmes NP-Complets dont la NP-complétude n'a pu être démontrée que grâce au problème de Satisfiabilité. Ces problèmes sont considérés comme des problèmes de base et sont exposés par Garey et Johnson, dans [32], qui a enrichi par la suite la littérature concernant la complexité des problèmes.

3-SAT(3-satisfiabilité) : Etant donné un ensemble U de variables booléennes et une collection C de m clauses sur U, contenant chacune 3 variables. Existe-t-il une affectation en "vrai" et "faux" des variables telle que toutes les clauses de C prennent la valeur "vrai" ?

3-DM(3-Dimensional Matching) : Etant donné un ensemble M de triplets, $M \subseteq X \times Y \times Z$ où X, Y, Z sont des ensembles mutuellement disjoints et de même cardinalité q. Existe-t-il un sous ensemble $M' \subseteq M$ tel que $|M'| = q$ et les triplets de M' sont deux à deux disjoints ?

Recouvrement : Etant donné un graphe $G = (V, E)$ et un entier k. Existe-t-il $X \subseteq V$ tel que $|X| \leq k$ et toute arête de G a au moins une de ses extrémités dans X ?

Clique : Etant donné un graphe $G = (V, E)$ et un entier k. Existe-t-il $X \subseteq V$ tel que $|X| \geq k$ et tous les sommets de X sont deux à deux adjacents ?

Cycle Hamiltonien : Etant donné un graphe $G = (V, E)$. Existe-t-il un cycle passant une et une seule fois par tous les sommets de G ?

Partition : Etant donné n entiers positifs $a_1, a_2, ..., a_n$. Existe-t-il un sous ensemble $J \subseteq I = \{1, 2, ..., n\}$ tel que $\sum_{i \in J} a_i = \sum_{i \in I \setminus J} a_i$?

La figure 2.2 montre les différentes réductions polynomiales utilisées pour prouver la NP-Complétude de ces six problèmes.

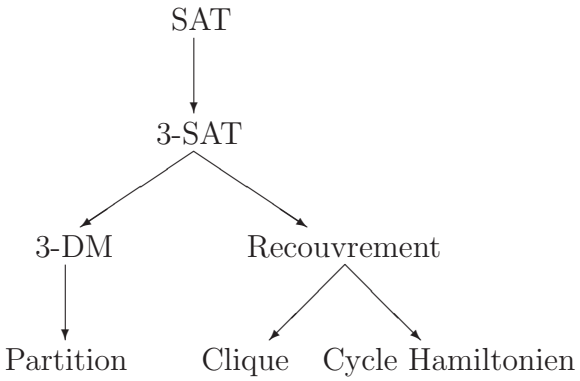

FIGURE 2.2 – Enchaînement des preuves de la NP-complétude.

2.2.2 Conjecture fondamentale P $\neq$ NP

On a vu dans la remarque précédente que $P \subseteq NP$, mais on ne sait pas si P est égale ou non à NP. S'il s'avérait que $P = NP$, alors on pourrait résoudre tous les problèmes de la classe NP en un temps polynomial. Or, les problèmes NP-Complets sont très fréquents et pour aucun d'entre eux on a réussi à trouver un algorithme polynomial le résolvant. La communauté scientifique pense que $P \neq NP$, mais ce n'est pas prouvé et que cette conjecture reste un problème ouvert depuis 1971.

2.2.3 NP-complétude au sens fort

On rappelle qu'un algorithme résolvant un problème de décision (D) est dit polynomial si sa complexité est en $O(|I|^k)$, où $|I|$ représente la taille d'entrée, pour un codage raisonnable, ou d'une manière équivalente que sa complexité est bornée par un polynôme en $|I|$. Un algorithme est dit pseudo-polynomial si sa complexité est bornée par une fonction polynomiale de deux variables $|I|$ et max(I), où max(I) est la valeur du plus grand nombre dans I. Par exemple en ordonnancement, un algorithme est dit polynomial si sa complexité est bornée par un polynôme du nombre de tâches, tandis que si sa complexité est bornée par un polynôme de deux variables : le nombre de tâches et la durée de traitement maximale, il sera dit pseudo-polynomial.

Quand un problème est NP-complet, il n'est pas raisonnable d'espérer de construire un algorithme polynomial le résolvant, mais on peut trouver dans certains cas des algorithmes pseudo-polynomiaux, comme celui de Partition. Dans d'autres cas, il sera difficile de trouver un algorithme pseudo-polynomial qu'un algorithme polynomial, on parlera alors des problèmes NP-Complets au sens fort.

Définition 2.3. *Un problème de décision D est dit NP-Complet au sens fort si :*

1. *D est NP-Complet*

2. *L'existence d'un algorithme pseudo-polynomial pour le résoudre entraine l'existence d'un algorithme polynomial.*

Exemple 2.1. *Le problème suivant est NP-Complet au sens fort [32].*

3-Partition : *Etant donné un ensemble A à $3m$ éléments, une borne entière B et une taille a_i pour chaque élément $a \in A$, telles que $\frac{B}{4} < a_i < \frac{B}{2}$ et $\sum_{a \in A} a_i = mB$. Existe-t-il m sous ensembles disjoints $S_1, ..., S_m$ tels que pour tout j $(j = 1, 2, ..., m) : \sum_{a \in S_j} a_i = B$?*

Un problème NP-Complet dont les paramètres formels sont bornés est évidemment NP-Complet au sens fort.

2.3 Problèmes d'optimisation combinatoire

Définition 2.4. *Un Problème d'Optimisation Combinatoire (POC en abrégé) est défini par la donnée d'un ensemble fini D de solutions réalisables et une fonction objectif $f : D \to \mathbb{R}$ où on cherche un élément $x_0 \in D$ tel que $f(x_0) = \min_{x \in D} f(x)$.*

Ce problème s'appelle un problème de minimisation et est noté $\begin{cases} Min f(x) \\ x \in D \end{cases}$
Le problème consistant à chercher un élément maximum au lieu d'un élément minimum est de même nature puisque $\max_{x \in D} f(x) = -\min_{x \in D}(-f(x))$.

2.3.1 Exemples de problèmes d'optimisation combinatoire

Problème du voyageur de commerce :

Etant donné un graphe avec des poids associés aux arêtes. Déterminer un cycle fermé qui passe exactement une fois par chaque sommet du graphe de façon à minimiser le poids total des arêtes.

Problème du sac à dos :

Un randonneur doit décider de l'équipement qu'il veut emporter pour sa prochaine expédition. Pour des raisons évidentes, il veut limiter le poids total des objets emportés et s'est fixé une borne supérieure de B Kg à ne pas dépasser. Chacun des n objets i possède un poids p_i et une utilité c_i. Déterminer une sélection dont le poids total ne dépasse pas B et qui maximise l'utilité totale des objets sélectionnés. Si pour chaque objet i $(i = \overline{1, n})$, on définit une variable de décision binaire x_i tel que :

$$x_i = \begin{cases} 1 & \text{si l'objet i est retenu} \\ 0 & \text{sinon} \end{cases}$$

Le problème est de déterminer une sélection optimale, ce qui revient à résoudre le programme linéaire en nombres entiers suivant :

$$\begin{cases} MaxZ = \sum_{i=1}^{n} c_i x_i \\ \sum_{i=1}^{n} p_i x_i \leq B \\ x_i \in \{0,1\},\ i = \overline{1,n} \end{cases}$$

Problème de coloration minimum :

Etant donné un graphe $G = (V, E)$, le problème consiste à affecter une couleur à chacun des sommets du graphe G de manière à minimiser le nombre total des couleurs utilisées, tout en assurant que deux sommets adjacents aient des couleurs différentes.

Problème du flot maximum :

Etant donné un réseau $R = (V, U, c)$ et deux sommets s, p de R, déterminer un flot réalisable de s à p dans le réseau R, de valeur totale maximale.

Problème du couplage maximum :

Etant donné un graphe $G = (V, E)$, déterminer un sous-ensemble d'arêtes $E' \subseteq E$ de cardinalité maximum tel que deux arêtes quelconques de E' ne soient pas adjacentes.

Ordonnancement avec des dates d'arrivées :

Etant donné n tâches indépendantes et non morcelables où chaque tâche T_i a un durée de traitement p_i et une date d'arrivée r_i. Déterminer un ordonnancement de ces tâches sur une machine minimisant la date de fin traitement.

Programmation Linéaire (sur un domaine borné) :

La programmation linéaire (PL) consiste à minimiser une fonction linéaire de n variables à m contraintes linéaires. Plus formellement, elle consiste à résoudre le programme linéaire suivant, où A est une matrice mxn, c un n-vecteur et b est un m-vecteur :

$$\begin{cases} MinZ = cx \\ Ax \leq b \\ x \geq 0 \end{cases}$$

Ce problème est résoluble par l'algorithme du Simplexe. Celui-ci consiste à énumérer intelligemment les solutions dites de base, qui correspondent aux sommets du polytope $P = \{x \in \mathbb{R}^n / Ax \leq b, x \geq 0\}$. La solution optimale se trouve parmi ces solutions de base qui sont en nombre fini. Ceci explique le fait d'inclure la Programmation Linéaire dans les problèmes d'optimisation Combinatoire.

2.3.2 Problèmes NP-difficiles

D'abord notons qu'à chaque problème d'optimisation combinatoire on peut lui associer un problème de décision, par exemple le problème de décision associé au problème $\begin{cases} Minf(x) \\ x \in D \end{cases}$ est le suivant : étant donné un ensemble fini D, une fonction objectif f définie sur D et une borne K , existe t-il $x \in D$ tel que $f(x) \leq K$?

Si l'on dispose d'un algorithme polynomial pour un POC, alors on dispose d'un algorithme polynomial pour le problème de décision associé, et vice versa.

La difficulté d'un problème d'optimisation combinatoire réside dans le fait que la solution doit être cherchée dans un ensemble de très grande cardinalité.

Définition 2.5. *Un problème d'optimisation combinatoire est dit NP-difficile (respectivement au sens fort) si le problème de Décision associé est NP-Complet (respectivement au sens fort).*

Par exemple le problème d'optimisation combinatoire associé à Deux Processeurs, est NP-difficile

Deux Processeurs : Etant donné n tâches $J_1, J_2, ..., J_n$ indépendantes et non morcelables telles que chaque tâche J_i possède une durée de traitement p_i, et un nombre k . Existe-t-il un ordonnancement de ces tâches sur deux processeurs de durée inférieure ou égale à k ?

Remarque 2.2.

Deux problèmes d'optimisation seront dits polynomialement équivalents si le problème de décision associé à l'un peut se réduire polynomialement à celui de l'autre et vice versa.

2.3.3 Méthodes classiques de résolution

Trouver une solution optimale dans un ensemble fini D est un problème facile à comprendre : il suffit d'énumérer toutes les solutions $x \in D$ et retenir la meilleure. Cependant

en pratique l'énumération de toutes les solutions peut prendre trop de temps, or le temps de recherche de la solution optimale est un facteur déterminant dans la résolution du problème.

Nous avons vu qu'il existe des problèmes de complexité différentes. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Pour les problèmes NP-Complet (ou NP-difficiles), l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes de résolution : méthodes exactes ou heuristiques sont largement utilisées pour appréhender ces problèmes. On distingue principalement trois classes de méthodes de résolution en fonction des propriétés des solutions qu'elles fournissent :

– Les méthodes exactes
– Les heuristiques constructives
– Les méthodes amélioratrices

2.3.3.1 Méthodes exactes

Ceux sont des algorithmes qui fournissent une solution optimale au prix d'un temps de calcul souvent important. Ces méthodes se caractérisent par un temps de calcul exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petite taille. Trois familles de méthodes exactes sont à distinguer : la résolution à partir d'une modélisation analytique, la programmation dynamique et la méthode de séparation et évaluation.

Modélisation analytique

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également de le résoudre. L'idéal est de modéliser le problème sous la forme d'un programme linéaire dont les variables sont réelles. Dans ce cas, il existe un nombre de solveurs permettant de le résoudre. Les modèles deviennent plus difficiles à résoudre si on utilise des variables entières, et lorsque le modèle n'est pas linéaire.

La programmation dynamique

Cette méthode d'énumération implicite est applicable uniquement si le problème est décomposable en phases, et si le critère possède certaines propriétés. En général, à partir d'un ordonnancement partiel, une formulation donne la valeur du critère comme une fonction récurrente, on trouve la valeur optimale en énumérant un ensemble d'ordonnancement partiel, qu'on complète à chaque étape par une opération (décision). Grâce à cette formulation, on réduit l'énumération.

Procédures par séparation et évaluation

Cette méthode est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. La séparation consiste à décomposer l'ensemble des solutions en plusieurs sous-ensembles qui sont décomposés à leur tour selon une démarche itérative. Ce processus peut se visualiser sous la forme d'un arbre d'énumération ; les nœuds de l'arbre correspondent aux sous-ensembles et les feuilles correspondent à des solutions réalisables. Pour accélérer la recherche de la solution optimale en évitant l'exploration inutile de certains nœuds, on utilise une procédure d'évaluation qui calcule, à chaque niveau de l'arbre, la valeur de chaque nœud et permet ainsi de décider du nœud à décomposer. Par exemple, dans le cas d'un problème de minimisation, cela revient à calculer la borne inférieure pour tous les nœuds fils du niveau considéré et la descente dans l'arbre est poursuivie avec le nœud donnant la borne inférieure. Par ailleurs, une borne supérieure de la solution optimale est calculée et est utilisée pour éviter l'exploration de nœuds dont la valeur de la borne inférieure est supérieure à la valeur de la borne supérieure. Cette borne supérieure est réactualisée lorsqu'une solution réalisable de valeur inférieure est atteinte.

Enfin des remontées et descentes par d'autres nœuds de l'arbre sont effectuées tant que la borne calculée est inférieure ou égale à la valeur actualisée de la borne supérieure.
Ainsi, l'exploration de certaines branches de l'arbre est évitée, ce qui permet de ne pas énumérer toutes les solutions réalisables.

2.3.3.2 Méthodes heuristiques constructives

Pour des problèmes NP-difficiles de grande taille, les méthodes exactes sont peu envisageables à cause de leur temps de calcul. Il est alors possible d'utiliser des méthodes approximatives qui donnent des solutions certes sous optimales, mais en un temps de calcul raisonnable. La performance de telles méthodes est généralement calculée par le rapport entre la valeur de la solution fournie et la valeur de la solution optimale. Si la solution optimale est non calculable, il est possible d'étudier expérimentalement le comportement de l'heuristique en comparant ses performances soit à celles d'autres heuristiques, soit à des bornes inférieures de la solution optimale.

Ces méthodes par construction progressive sont des méthodes itératives, où à chaque itération, une solution partielle est complétée. La plupart de ces méthodes sont des algorithmes gloutons car elles considèrent les éléments (travaux ou processeurs) dans un certain ordre sans jamais remettre en question un choix une fois qu'il a été effectué. Ces méthodes permettent de trouver une solution rapidement.

Un premier type de telles méthodes consiste à établir une liste initiale qui donnera l'ordre

de prise en compte des éléments. Cette liste construite à priori, à partir d'un critère bien défini, la deuxième phase de l'algorithme se réduit à considérer les éléments dans l'ordre de la liste construite pour l'ordonnancement.

Un deuxième type de méthode consiste à choisir, au cours de la construction, l'affectation d'un travail à une machine en utilisant des règles de priorité.

2.3.3.3 Méthodes amélioratrices

Ces méthodes sont initialisées par une solution réalisable, calculée soit aléatoirement soit à l'aide de l'une des heuristiques constructives. Elles recherchent à chaque itération, une amélioration de la solution courante par des modifications locales. Cet examen se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. L'utilisation de ces heuristiques itératives suppose que l'on puisse définir, pour toute solution S, un voisinage de solution N(S) contenant les solutions voisines. En général, le voisinage d'une solution est généré en appliquant plusieurs fois et de façon différente une petite transformation. Ce voisinage est ensuite évalué et comparé à la solution courante. Nous présentons ci-dessous les méthodes amélioratrices les plus connues :

Méthode de déscente

Cette méthode se caractérise par le fait qu'elle ne considère, à chaque itération, que des solutions évoluant dans le sens de l'amélioration. Cette méthode ne conduit pas en général au minimum absolu mais seulement au minimum local qui constitue la meilleure des solutions accessibles compte tenu de la solution initiale. Pour améliorer l'efficacité, on peut l'appliquer plusieurs fois avec des conditions initiales différentes et retenir comme solution finale le meilleur des minimums locaux obtenus, cette procédure augmente sensiblement le temps de calcul de l'algorithme et ne garantit pas de trouver la configuration optimale.

Recuit simulé

Cette méthode est due aux physiciens Kirkpatrick, Gellat et Vecchi, elle s'inspire des méthodes de simulation de Métropolies en mécanique statique. Les inconvénients du recuit simulé résident d'une part dans les réglages, comme la gestion de la décroissance par palier de la température. D'autre part, les temps de calcul peuvent devenir selon les cas très importants. Par contre, les méthodes de recuit simulé ont l'avantage d'être souples et rapidement implémentables lorsque l'on veut résoudre des problèmes d'optimisation combinatoire, le plus souvent de grande taille. Cette méthode a l'intérêt d'admettre une preuve de la convergence, aussi lorsque certaines conditions sont vérifiées, on a la garantie d'obtenir la solution optimale.

Recherche tabou

La principale particularité de la méthode tient dans la mise en œuvre de mécanismes inspirés de la mémoire humaine dont le but est de tirer les leçons du passé.
Le principe de base de la méthode tabou est simple, à partir d'une solution initiale quelconque, la méthode tabou engendre une succession de solutions ou configurations qui doivent aboutir à une solution optimale. A chaque itération le mécanisme de passage d'une configuration (s) à la suivante (t) est le suivant :

- On construit l'ensemble des voisins de s, c'est-à-dire l'ensemble des configurations accessibles en un seul mouvement élémentaire à partir de s, soit V(s) l'ensemble de ces voisins.
- On évalue la fonction objectif f du problème pour chacune des configurations appartenant à V(s). La configuration t, qui succède à s dans la chaîne construite par tabou est la configuration de V(s) pour laquelle f prend la valeur minimale.

La procédure ne fonctionne généralement pas car il y a un risque de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui provoque un cyclage. Pour éviter ce phénomène on tient à jour à chaque itération une liste taboue de mouvements interdits, cette liste circulaire contient m mouvements inverses. La recherche du successeur de la configuration courante se restreint alors aux voisins de s qui peuvent être atteints sans utiliser de mouvement de la liste tabou. La procédure est stoppée dès qu'on a effectué un nombre donné d'itérations sans améliorer la meilleure solution courante.

Algorithmes génétiques

Les algorithmes génétiques sont des techniques de recherche inspirées de l'évolution biologique des espèces et basées sur une imitation des phénomènes d'adaptation des êtres vivants. On part d'une population initiale sur laquelle des opérations de reproduction vont être réalisées dans l'objectif d'exploiter au mieux les caractéristiques et les propriétés de cette population. Un algorithme génétique consiste à faire évoluer progressivement, par génération successive la composition de cette population en maintenant sa taille constante.

Chapitre 3

Définition du problème, notations et état de l'art

Le début de ce chapitre est consacré aux définitions et notations du problème général et des sous problèmes traités dans cette thèse. Nous proposons une modélisation mathématique du problème général sous la forme d'un programme linéaire en variables bivalentes et réelles. Ensuite nous citons quelques problèmes liés dans la littérature et montrons les liens existant avec ces derniers. Un état de l'art et des applications de notre problème sont aussi présentés.

3.1 Définition du problème traité

Dans ce travail, nous considérons le problème d'ordonnancement d'un ensemble de n tâches, $V = \{J_1, J_2\ , .., J_n\}$ sur m machines parallèles identiques. On suppose qu'à tout instant une machine n'est allouée qu'à au plus une tâche et chaque tâche n'est traitée que par au plus une machine. Chaque tâche J_i $(i = 1, 2, ..., n)$ possède une durée de traitement p_i et une date de disponibilité r_i. Le mode de traitement des tâches est non-préemptif, c'est-à-dire l'interruption des tâches n'est pas autorisée.

On suppose qu'il existe une relation de concordance entre les tâches : deux tâches sont concordantes (ou en concordance) si leurs intervalles de traitement peuvent s'intersecter, c'est-à-dire si elles peuvent être traitées simultanément sur deux machines différentes. Cette relation est donnée par un graphe $G = (V, E)$ où V est l'ensemble des tâches et une paire de tâches est dans E si et seulement si ces deux tâches sont concordantes. Ce graphe est appelé graphe de concordance. Les contraintes induites par la relation de concordance, ou d'une manière équivalente par le graphe de concordance sont appelées contraintes de concordance.

Un ordonnancement de tâches est dit réalisable, s'il respecte les contraintes de concordance, les durées de traitement et les dates de disponibilité des tâches. Le problème consiste alors à trouver un ordonnancement réalisable qui minimise la date de fin de traitement de l'ensemble des tâches (makespan).

3.1.1 Notations et classifications

Dans la littérature [6], nous avons utilisé le terme "agreement graph" qui est le synonyme du terme "graphe de concordance". Afin de faciliter la description et la classification des problèmes étudiés dans cette thèse, on adoptera le formalisme de Graham et al. [38] à trois champs $\alpha|\beta|\gamma$, qui a été exposé dans le chapitre 1.

Champ α

Ce champ est composé de deux sous champs et est désigné par $\alpha_1\alpha_2$:

- $\alpha_1 = P$
  P : machines parallèles identiques.

- $\alpha_2 \in\{\phi$, m $\}$
  Lorsque $\alpha_2 = \phi$, le nombre de machines est arbitraire, tandis que lorsque $\alpha_2 = m$, le nombre de machines est fixé et égal à m.

Champ β

Ce champ est composé de trois sous champs $\beta = \beta_1\beta_2\beta_3$:

- $\beta_1 \in \{\phi, AgreeG = (V, E), AgreeG = (S_1, S_2; E), AgreeG = (S, K; E), AgreeG = (K_1, K_2; E)\}$
  $\beta_1 = \phi$: le graphe de concordance est complet.
  $\beta_1 =$ "$AgreeG = (V, E)$" : le graphe de concordance est un graphe arbitraire.
  $\beta_1 =$ "$AgreeG = (S_1, S_2; E)$" : le graphe de concordance est un graphe biparti arbitraire .
  $\beta_1 =$ "$AgreeG = (S, K; E)$" : le graphe de concordance est un graphe scindé arbitraire.
  $\beta_1 =$ "$AgreeG = (K_1, K_2; E)$" : le graphe de concordance est le complémentaire d'un graphe biparti arbitraire.

- $\beta_2 \in \{\phi, r_i\}$
  $\beta_2 = \phi$: toutes les dates de disponibilité sont nulles.
  $\beta_2 = r_i$: il existe des dates de disponibilité arbitraires pour les tâches

- $\beta_3 \in \{\phi, "p_i = 1", "p_i = p"\}$
  $\beta_3 = \phi$: les durées de traitement des tâches sont arbitraires.
  $\beta_3 = "p_i = 1"$: les durées de traitement des tâches sont unitaires.
  $\beta_3 = "p_i = p"$: les durées de traitement des tâches sont toutes égales à p.

Champ γ

Le troisième champ $\gamma = C_{\max}$ décrit le critère d'optimalité, qui est la minimisation de la date de fin de traitement de l'ensemble des tâches $C_{\max}$ (makespan).

Selon ces notations notre problème général est noté : $\boldsymbol{P|AgreeG = (V, E), r_i|C_{\max}}$.

Ce problème est NP-difficile puisqu'il contient le problème classique $P||C_{max}$, qui est défini dans l'exemple suivant.

3.1.2 Problèmes particuliers

Exemples : Le problème classique $P||C_{max}$ est le problème d'ordonnancement de tâches indépendantes sur un nombre arbitraire de machines parallèles identiques avec des durées de traitement arbitraires, dont le but est de minimiser le makespan.

$Pm|AgreeG = (S_1, S_2; E), p_i = 1|C_{max}$ est le problème d'ordonnancement de tâches sur un nombre de machines parallèles identiques, fixé et égal à m, dont les durées de traitement sont unitaires et le graphe de concordance est un graphe biparti arbitraire, l'objectif est de minimiser le makespan.

3.2 Motivation et applications

Notre problème peut être rencontré dans les problèmes d'ordonnancement sous contraintes de ressources non-partageables. Parmi les applications on cite celles de Baker et Coffman [2] présentées dans l'équilibrage de charge pour les calculs parallèles, d'autres sont mentionnées dans le contrôle de la circulation aux intersections, allocation des fréquences dans les réseaux cellulaires et la gestion des sessions dans les réseaux locaux (voir [40]). Motivés par un problème d'allocation d'opérations à des processeurs Bodlaender et Jansen [14] ont établi une série de résultats sur ce problème.

Initialement, notre problème a été motivé par les problèmes suivants :

3.2.1 Application 1 : Problème à ressources limitées

Dans un atelier, n tâches $J_1, J_2, .., J_n$ doivent être exécutées par m ouvriers. Chaque tâche J_i nécessite pour son traitement un temps p_i et un sous-ensemble de ressources $R_i \subseteq R$ où $R = \{rs_1, rs_2, ..., rs_k\}$ est l'ensemble des ressources disponibles (par exemple : une pelle, une pioche, un chariot, etc.). L'objectif est d'ordonnancer ces tâches dans un temps minimum.

Modélisation :
Si on regarde les ouvriers comme étant des machines, on peut associer le graphe de concordance $G = (V, E)$ tels que :
$V = \{J_1, J_2, ...J_n\}$, $\{J_i, J_j\} \in E \iff R_i \cap R_j = \varnothing$.

Ce problème s'écrit : $\boldsymbol{Pm|AgreeG = (V, E)|C_{\max}}$

Exemple 3.1. *Considérons l'exemple suivant : $n = 5, m = 2$, $R = \{rs_1, rs_2, rs_3\}$. Les demandes en ressources et les durées de traitement sont données dans la table 3.1, le graphe de concordance est représenté sur la figure 3.1. La figure 3.2 est le diagramme de Gantt représentant un ordonnancement réalisable pour l'exemple 3.1, qui est optimal.*

J_i	J_1	J_2	J_3	J_4	J_5
p_i	1	2	1	3	1
R_i	$\{rs_1, rs_3\}$	$\{rs_3\}$	$\emptyset$	$\{rs_1\}$	$\{rs_2, rs_3\}$

TABLE 3.1 – Durées de traitement et demandes en ressources de l'exemple 3.1

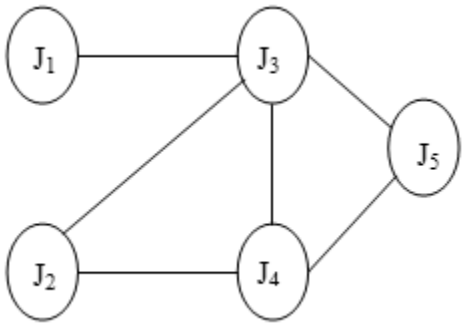

FIGURE 3.1 – Le graphe de concordance de l'exemple 3.1

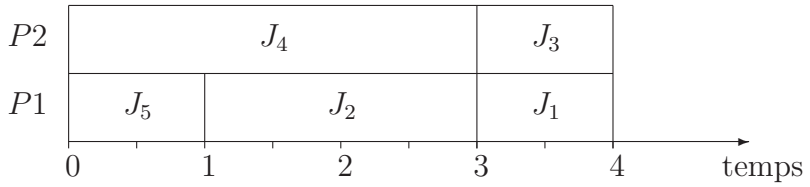

FIGURE 3.2 – Diagramme de Gantt de l'exemple 3.1

3.2.2 Application 2 : Planning des examens (Université)

Dans une Université (système LMD), n examens $T_1, \ldots, T_n$ doivent se dérouler à la fin du semestre. Chaque examen doit être pris par un ensemble d'étudiants. Les durées des examens sont de 1 heure ou de 2 heures. Les examens se déroulent dans des salles, dont le nombre est égal à m. Aussi nous supposons que les classes ont une grande capacité de sorte que n'importe quel examen peut être déroulé dans n'importe quelle salle. Le but est de minimiser la durée totale de l'ensemble des examens.

Modélisation :
Les examens sont représentés par des tâches $(T_1, T_2, ..., T_n)$ tandis que les salles sont représentées par des machines $P1, P2, ..., Pm$.
Le graphe de concordance $G = (V, E)$ est défini par : $V = \{T_1, T_2, ...T_n\}$ et une paire $\{T_i, T_j\} \in E \iff$ il n'existe aucun étudiant qui passe les deux examens T_i et T_j à la fois. Ce problème s'écrit alors $\boldsymbol{Pm|AgreeG = (V, E), p_i \in \{1, 2\}|C_{\max}}$.

3.2.3 Application 3 : Planification d'horaires de travail

Une autre application dûe à F. Gardi [30] est la suivante :
Soient $T_1, T_2, ..., T_n$ n tâches telles que chaque tâche T_i doit être exécutée dans un intervalle de temps $[a_i, b_i]$. La règlementation impose pas plus de m tâches par employer. Sachant que les tâches affectées à un employer ne doivent pas se chevaucher, trouver un planning pour l'entreprise utilisant un nombre minimum d'employés.

Modélisation :
A chaque tâche T_i, $i = \overline{1, n}$, du problème de planification d'horaires, on fait correspondre une tâche T'_i, du problème avec graphe de concordance, avec une durée égale à 1. Le graphe de concordance $G = (V, E)$ est tel que $V = \{T'_1, T'_2, ...T'_n\}$ et $\{T'_i, T'_j\} \in E \iff$ les deux tâches T_i et T_j ne se chevauchent pas, c'est-à-dire que leurs intervalles d'exécution ne s'intersectent pas. Ce nouveau problème s'écrit alors $\boldsymbol{Pm|AgreeG = (V, E), p_i = 1|C_{\max}}$ et le nombre minimum d'employers est égal à la valeur optimale du $C_{\max}$.

3.3 Modélisationn mathématique du problème

Considérons le problème $Pm|AgreeG = (V, E), r_i|C_{\max}$. Soient $p_1, p_2, ..., p_n$ et $r_1, r_2, ..., r_n$ les durées de traitement et les dates de diponibilité des tâches $J_1, J_2, ..., J_n$ respectivement. Soient $P1, P2, ..., Pm$ les machines traitant ces tâches. Pour la modélisation mathématique de ce problème, nous avons utilisé la variable $C_{\max}$ et trois types de variables qui sont définies comme suit : pour toute tâche J_i, on associe la variable t_i qui représente la date de début de traitement de la tache J_i $(i = \overline{1, n})$. Le deuxième type de variables sont les variables bivalentes X_{ik} définies par :

$$X_{ik} = \begin{cases} 1 & \text{si la tâche } J_i \text{ est ordonnancée sur la machine } Pk \\ 0 & \text{sinon} \end{cases}$$

et ceci pour tout couple (i, k) tel que $i = \overline{1, n}$, $k = \overline{1, m}$. Ces deux types de variables nous permettent de définir le premier type de contraintes : $\sum_{k=1}^{m} X_{ik} = 1$
Ces contraintes expriment le fait que chaque tâche doit être affectée à une seule machine. Le troisième type de variables sont les variables bivalentes Z_{ij} telles que :

$$Z_{ij} = \begin{cases} 1 & \text{si } t_i \leq t_j \\ 0 & \text{sinon} \end{cases}$$

pour tout couple (i, j) tel que $i \neq j$, $i, j = \overline{1, n}$. Soit M une trés grande valeur positive, qu'on peut estimer, par exemple, à $\sum_{i=1}^{n} p_i$. Les variables Z_{ij} peuvent être décrites par les contraintes suivantes : $Z_{ij} + Z_{ji} = 1$ pour $i < j$; $i, j = \overline{1, n}$
$t_j - t_i \leq M\ Z_{ij}$ pour $i \neq j;\ i, j = \overline{1, n}$
$Z_{ij} \in \{0, 1\}$ pour $i, j = \overline{1, n}$

Les contraintes de non chevauchement : expriment le fait qu'une machine ne peut traiter plus d'une tâche à la fois. Autrement, dit si deux tâches J_i et J_j sont affectées à la même machine P_k et que J_i commence avant J_j, alors $t_j \geq t_i + p_i$. Ceci est équivalent à dire que si $X_{ik} = 1$ et $X_{jk} = 1$, alors $t_j \geq t_i + p_i$.

Ces contraintes de non chevauchement s'écrivent :
$t_i + p_i - t_j \leq M\ (1 - Z_{ij} + 2 - X_{ik} - X_{jk})$ $i \neq j$ et $i, j = \overline{1, n}$ et $k = \overline{1, m}$.
Les contraintes respectant les dates de disponibilité des tâches sont : $t_i \geq r_i$. Les contraintes de concordance sont définies par :
$t_j - t_i \geq p_i$ ou $t_i - t_j \geq p_j$ pour toute paire de tâches J_i, J_j telle que $\{J_i, J_j\} \notin E$. Si nous considérons la matrice d'adjacence du graphe $G : a = (a_{ij})$, alors ces contraintes sont équivalentes au système de contraintes $t_j - t_i \geq (1 - a_{ij})p_i$ ou $t_i - t_j \geq (1 - a_{ij})p_j$ pour $i \neq j;\ i, j = \overline{1, n}$. Ces contraintes disjonctives peuvent être écrites sous forme conjonctive par :

$$\begin{cases} t_j - t_i \geq (1 - a_{ij})p_i + M(Z_{ij} - 1) & i \neq j;\ i, j = \overline{1, n} \\ \\ t_i - t_j \geq (1 - a_{ij})p_j - MZ_{ij} & i \neq j;\ i, j = \overline{1, n} \end{cases}$$

Les contraintes définissant le makespan sont modélisées comme suit : $t_i + p_i \leq C_{max}$, $i = \overline{1, n}$. Autres contraintes sur les variables sont :

$t_i \geq 0$, $i = \overline{1, n}$; $X_{ik} \in \{0, 1\}$ pour $i = \overline{1, n}$, $k = \overline{1, m}$

Le problème $Pm|AgreeG = (V, E), r_i|C_{\max}$ est modélisé comme ci-dessous.

$$\begin{cases} MinZ = C_{max} & & \\ \sum_{k=1}^{m} X_{ik} = 1 & i = \overline{1, n} & (1) \\ Z_{ij} + Z_{ji} = 1 & i < j;\ i, j = \overline{1, n} & (2) \\ t_j - t_i \leq M\ Z_{ij} & i \neq j;\ i, j = \overline{1, n} & (3) \\ t_i + p_i - t_j \leq M\ (3 - Z_{ij} - X_{ik} - X_{jk}) & (i \neq j);\ i, j = \overline{1, n};\ k = \overline{1, m} & (4) \\ t_i \geq r_i & i = \overline{1, n} & (5) \\ t_j - t_i \geq (1 - a_{ij})p_i + M(Z_{ij} - 1) & i, j = \overline{1, n} & (6) \\ t_i - t_j \geq (1 - a_{ij})p_j - MZ_{ij} & i, j = \overline{1, n} & (7) \\ t_i + p_i \leq C_{max} & i = \overline{1, n} & (8) \\ t_i \geq 0 & i = \overline{1, n} & (9) \\ Z_{ij} \in \{0, 1\} & (i \neq j);\ i, j = \overline{1, n} & (10) \\ X_{ik} \in \{0, 1\} & i = \overline{1, n};\ k = \overline{1, m} & (11) \\ C_{max} \geq 0 & & (12) \end{cases}$$

3.4 Etat de l'art

Notons que la donnée du graphe de concordance est équivalente à la donnée du complémentaire de ce graphe. G. Even et al. [28] ont considéré une version équivalente à notre

problème en utilisant le complémentaire du graphe de concordance, qu'ils l'ont appelé graphe de conflit, dans le cas où le nombre de machines m est fixé. Dans la littérature cette version est connue sous le nom : "Scheduling With Conflicts (S.W.C). Dans ce qui suit, nous rappelons la définition du problème S.W.C, qui a été donnée par G. Even et al. dans [28].

3.4.1 Problème d'ordonnancement avec conflit (S.W.C)

Le problème d'ordonnancement avec conflit (S.W.C) a été étudié par Even et al. [28] et est défini comme suit. Il existe un ensemble de m machines parallèles identiques. Les données consistent en un ensemble J de n tâches (jobs en anglais) avec leurs durées de traitement $\{p_j\}$ $(j \in J)$, qui doivent être éxécutées sur ces machines. On suppose qu'il existe un graphe de conflit $G = (J, E)$ entre les tâches où chaque arête représente une paire de tâches qui ne peuvent pas être ordonnancées simultanément sur deux machines différentes (tâches en conflit). Un ordonnancement est une affectation d'intervalles de temps aux tâches, sur les m machines vérifiant les conditions suivantes : (i) chaque tâche j est affecté un intervalle de temps de longueur p_j sur une machine ; (ii) les intervalles sur la même machine ne se chevauchent pas ; et (iii) les intervalles de temps affectés aux tâches en conflit ne se chevauchent pas. Le makespan d'un ordonnancement est la plus grande extrémité finale d'un intervalle de temps assigné à une tâche. L'objectif est de minimiser le makespan.

Un cas plus général que l'ordonnancement avec conflit (i.e. le cas où les conflits entre les tâches peuvent ne pas être représentées par un graphe) a été étudié par Garey et Graham [33]. Les auteurs de [28] ont présenté un algorithme glouton pour le problème S.W.C (appelé dans la littérature Anglophone G.M.E : Greedy Mutual Exclusion) déduit des travaux de Garey et Graham [33]. Ces mêmes auteurs ont analysé le makespan de cet algorithme et ont prouvé que cet algorithme est une $(m+1)/2$-approximation pour le problème S.W.C et que cette borne est atteinte.

D'après cette définition, notons que les problèmes $Pm|AgreeG = (V, E)|C_{max}$ et le problème S.W.C avec graphe de conflit $\overline{G}$ sont équivalents. Comme l'opération de passage d'un graphe à son complémentaire peut être effectuée polynomialement, donc du point de vue complexité nous déduisons que les problèmes $Pm|AgreeG = (V, E)|C_{max}$ et S.W.C sont polynomialement équivalents.

Dans le cas où le nombre de machines m est égal à 2 et que les durées de traitement $p_i \in \{1, 2\}$, les auteurs [28] ont montré que le problème S.W.C est polynomial et ont proposé un algorithme polynomial pour sa résolution, basé sur la détermination d'un cou-

plage maximum dans un graphe auxilliaire bien approprié.
Nous déduisons que le problème $P2|AgreeG = (V, E), p_i \in \{1, 2\}|C_{max}$ est polynomial.
Lorsque les durées de traitement $p_i \in \{1, 2, 3, 4\}$, les auteurs de [28] ont établi que ce problème est NP-difficile.

Quant au cas de deux machines mais avec $p_i \in \{1, 2, 3\}$, les mêmes auteurs ont proposé, dans la même référence ce problème comme étant un problème ouvert parmi quatres et ont donné une 4/3-approximation pour ce problème.
Ce problème ouvert est polynomialement équivalent au problème $P2|AgreeG = (V, E), p_i \in \{1, 2, 3\}|C_{max}$.

3.4.2 Ordonnancement avec exclusion mutuelle (M.E.S) : cas $p_i = 1$ et $r_i = 0$

Dans le cas des durées de traitement unitaires avec dates de disponibilités nulles, notre problème est noté $P|AgreeG = (V, E), p_i = 1|C_{max}$.
En termes de la théorie des graphes, ce problème est équivalent au problème suivant : Etant donné un nombre m et un graphe (qui est le graphe de concordance), trouver une partition minimum de ce graphe en cliques, chacune de taille inférieure ou égale à m.

D'autre part dans la littérature, ce problème est équivalent au problème d'ordonnancement avec exclusion mutuelle, en anglais : Mutual Exclusion Scheduling (M.E.S). Ce dernier a été introduit par Baker et Coffman [2] et est défini comme suit : n tâches avec des durées de traitement unitaires doivent être exécutées sur m processeurs identiques en un minimum de temps, sous contraintes que certaines tâches ne peuvent pas être exécutées simultanément parcequ'elles partagent une même ressource. De telles tâches sont dites en conflit.

Si on représente ces contraintes par un graphe de conflit G tel que chaque sommet du graphe correspond à une tâche et une arête entre chaque paire de tâches en conflit, en termes de la théorie des graphes ce problème peut être formulé comme suit : Etant donné un graphe G et un entier m, le problème est de trouver une coloration minimum de G telle chaque couleur est utilisée au maximum m fois.

D'après ce qui a été dit, notons que les problèmes $Pm|AgreeG = (V, E)|C_{max}$ et M.E.S avec graphe de conflit $\overline{G}$ sont équivalents et du point de vue complexité, les problèmes $P|AgreeG = (V, E)|C_{max}$ et M.E.S sont polynomialement équivalents .

Vu l'importance de leurs applications industrielles, plusieurs variantes du problème M.E.S ont été considérées dans la littérature. Le lecteur intéressé à ce genre de problèmes peut consulter l'article de Krarup et De Werra [43] , recueil récent de Blazewicz et al. [8], Baker et Coffman [2], Jansen [39] et Gardi [30], [31].

Dans ce qui suit, nous citons les principaux résultats de complexité connus pour le problème M.E.S pour ceraines classes de graphes et par équivalence ces résultats sont aussi des résultats pour notre problème $P/AgreeG = (V, E), p_i = 1/C_{max}$, en passant aux complémentaires. Nous citons ci-dessous quelques résultats connus du problème M.E.S :

Problème M.E.S	Condition	Référence
graphes bipartis	$m \geq 4$,m fixé	[14]
cographes	$m \geq 4$,m fixé	[14]
graphes d'intervalles	$m \geq 4$,m fixé	[14]
graphes triangulés	$m \geq 3$,m fixé	[24]
complémentaire de graphes de comparabilité	$m \geq 3$,m fixé	[39]
complémentaire de graphes adjoints	$m \geq 3$,m fixé	[23]
graphes de permutation	$m \geq 6$,m fixé	[39]

Table 3.2 – Problèmes NP-difficiles

Problème M.E.S	Complexité	Référence
graphes scindés	Polynomial	[14],[45]
forêts	Polynomial	[2],[42]
arbres	Polynomial	[2],[42]
collection de cliques disjointes	Polynomial	[26]
graphes bipartis	Polynomial pour m fixé	[41]
complémentaire de graphes fortement triangulés	Polynomial	[25]
graphes de largeur arbre bornée	Polynomial Polynomial	[13]
graphes adjoints	Polynomial pour m fixé	[1]
cographes	Polynomial pour m fixé	[45]
graphes d'intervalles propres	Polynomial	[31]

Table 3.3 – Problèmes Polynomiaux

Pour $m = 3$, le problème M.E.S est équivalent au problème de la partition minimum en triangles du complémentaire du graphe de conflit [34]. Pour $m = 2$, ce problème est

équivalent au problème du couplage maximum dans le complémentaire du graphe de conflit [28], ou d'une manière équivalente notre problème $P2/AgreeG = (V, E), p_i = 1/C_{max}$ est équivalent au problème du couplage maximum dans le graphe de concordance.

3.4.3 Ordonnancement sur une machine à traitement par batchs : cas $p_i = 1$ et r_i arbitraires

Dans [18] M. Boudhar et G. Finke ont étudié le problème d'ordonnancement suivant : n tâches indépendantes $T_1, T_2, ..., T_n$ doivent être exécutées sur une machine à traitement par batch $B1$ sachant qu'il existe une relation de compatibilité entre les tâches, où deux taches sont compatibles si elles peuvent être traitées simultanément dans un même batch. Cette relation est représentée par un graphe $G = (V, E)$ appelé graphe de compatibilité, où V représente l'ensemble des tâches et une paire de tâches est dans E si et seulement si ces dernières sont compatibles. Chaque tâche T_i a une durée de traitement p_i et une date de disponibilité r_i respectivement. La durée de traitement d'un batch est égal au maximum des durées de traitement des tâches appartenant à celui ci. Toutes les tâches d'un même batch commencent leur traitement à une même date et terminent à une même date. Il est aussi supposé que la capacité de la machine à traitement par batch peut être finie (elle peut traiter b tâches à la fois) ou infinie (elle peut traiter un nombre illimité de tâches à la fois). L'interruption des tâches n'est pas autorisée. Le problème consiste à chercher une partition $B_1, B_2, ..., B_q$ de l'ensemble des tâches, en batchs, ainsi que leurs dates de début d'exécution telles que $|B_j| \leq b$ pour $j = 1, 2, ..., q$ et minimisant la date de fin de traitement de l'ensemble des batchs (le makespan). Ce problème est noté : $B1|G = (V, E), b, r_i|C_{max}$ où G représente le graphe de compatibilité.

Notons que dans cette notation b est arbitraire, par contre si b est une constante égale à k, le problème est noté : $B1|G = (V, E), b = k, r_i|C_{max}$. Quand les dates de disponibilités sont nulles, le problème est noté $B1|G = (V, E), b|C_{max}$. Lorsque les durées de traitement des tâches sont unitaires, le problème est noté $B1|G = (V, E), b, r_i, p_i = 1|C_{max}$.

Dans ce qui suit nous donnons la première relation qui lie le problème d'ordonnancement sur une machine à traitement par batchs et notre problème.

Proposition 3.1. *Les problèmes* $Pm|AgreeG = (V, E), r_i, p_i = 1|C_{max}$ *et* $B1|G = (V, E), b = m, r_i, p_i = 1|C_{max}$ *sont équivalents.*

Preuve. Soient (P) et (P') les problèmes $Pm/AgreeG = (V, E), r_i, p_i = 1/C_{max}$ et $B1|G = (V, E), b = m, r_i, p_i = 1|C_{max}$ respectivement.

Supposons qu'il existe un ordonnancement réalisable σ pour le problème (P). Pour tout instant t tel qu'il existe au moins une tâche ordonnancée (par rapport à σ) à t, on associe le batch $B_t = \{$tâches ordonnancées par σ à l'instant $t\}$. Les batchs ainsi définis

forment une partition de l'ensemble des tâches. En ordonnançant chacun de ces batchs B_t à l'instant t, on obtient un ordonnancement réalisable σ' pour le problème (P') avec un makespan $C_{max}(\sigma') = C_{max}(\sigma)$. Comme illustration voir figure 3.3 , qui représente les ordonnancements σ et σ' dans le cas de deux machines, tel qu'ils exitent deux tâches J_t^1 et J_t^2 ordonnancées à l'instant t, par rapport à l'ordonnancement σ.

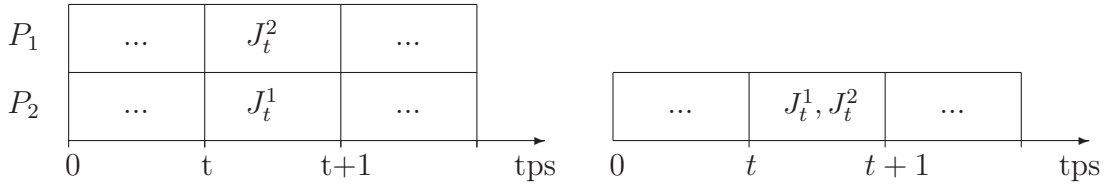

FIGURE 3.3 – Les ordonnancements σ et σ', $B_t = \{J_t^1, J_t^2\}$

Réciproquement, supposons que (P') admet un ordonnancement réalisable σ'. Cet ordonnancement est défini par une partition des tâches en batchs $B_1, B_2, ..., B_q$ vérifiant $|B_j| \leq m$ pour tout $j = 1, 2, ..., q$, avec leurs dates d'exécution $t_1, t_2, ..., t_q$ respectivement. La valeur de la fonction objectif en σ' est $C_{max}(\sigma')$. A partir de σ', on construit un ordonnancement réalisable σ pour le problème (P) comme suit : pour tout j $(j \in \{1, 2, ..., q\})$, on ordonnance les tâches de B_j à l'instant t_j. L'ordonnancement σ ainsi obtenu est réalisable du fait que σ' l'est aussi. D'autre part il est clair que la valeur de la fonction objectif pour (P), en σ est $C_{max}(\sigma) = C_{max}(\sigma')$. ■

En se basant sur la démonstration précédante, on montre du point de vue complexité, que les problèmes $P|AgreeG = (V, E)|C_{max}$ et $B1|G = (V, E), m|C_{max}$ sont polynomialement équivalents. En particulier, pour le cas des durées de traitement unitaires, on en déduit que les trois problèmes $P|AgreeG = (V, E), p_i = 1|C_{max}$, $B1|G = (V, E), m, p_i = 1|C_{max}$ et le problème M.E.S sont deux à deux polynomialement équivalents.

D'aprés les équivalences entre les problèmes, nous donnons d'autres résultats connus sur notre problème, dûs en grande totalité à M. Boudhar et G. Finke. D'abord nous citons les problèmes NP-difficiles.

Problème	Complexité	Référence
$P2\|AgreeG = (V, E), r_i, p_i = 1\|C_{\max}$	NP-difficile	[18]
$P\|AgreeG = (V, E), m \geq n, p_i = 1\|C_{\max}$	NP-difficile	[18]
$P2\|AgreeG = (S_1, S_2; E), r_i, p_i = 1\|C_{\max}$	NP-difficile au sens fort	[20]
$P\|AgreeG = (K_1, K_2; E), m \geq n, r_i, p_i = 1\|C_{\max}$	NP-difficile	[19]
$P\|AgreeG = G(K_b^1, ..., K^s; E), r_i, p_i = 1\|C_{max}$ où G est un graphe ayant une partition de s cliques avec m sommets	NP-difficile	[18]

TABLE 3.4 – Problèmes NP-difficiles

Problème	Complexité	Référence
$P2\|AgreeG = (V, E), p_i = 1\|C_{max}$	Polynomial en O($n^{2.5}$)	[18]
$P2\|AgreeG = (S_1, S_2; E), r_{S_1}, r_{S_2} = 0, p_i = 1\|C_{max}$	Polynomial en O($n^{2.5}logn$)	[20]
$P2\|AgreeG = (S_1, S_2; S_1 \times S_2), r_i, p_i = 1\|C_{max}$	Polynomial en O($nlogn$)	[20]
$P\|AgreeG = (S, K; E), p_i = 1\|C_{max}$	Polynomial en O(n^3)	[18],[16]
$P\|AgreeG = (S, K; E), m \geq n, p_i = 1\|C_{max}$	Polynomial en O(n)	[19],[16]
$P\|AgreeG = INT, p_i = 1\|C_{max}$	Polynomial en O(n^2)	[29]
$P\|AgreeG = (S, K; S \times K), m \geq n, r_i, p_i = 1\|C_{max}$	Polynomial en O($nlogn$)	[19]
$P\|AgreeG = (V, E), m \geq n, p_i = 1\|C_{max}$ où G est un graphe arc circulaire	Polynomial en O(n^3)	[18]
$P\|AgreeG = (V, E), m \geq n, p_i = 1\|C_{max}$ où G est un graphe triangulé	Polynomial en O(n^2)	[18]
$P\|G = (V, E), m \geq n, p_i = 1\|C_{max}$ où G est un graphe de comparabilité	Polynomial en O(n^3)	[18]

TABLE 3.5 – Problèmes Polynomiaux

3.4.4 Problème d'ordonnancement à contraintes de ressources discrètes

Dans la référence [8], Blazewicz et al. ont considéré ce problème, qui est défini comme suit : il s'agit d'ordonnancer un ensemble de n tâches $T_1, T_2, ..., T_n$ non-préemptives sur m machines parallèles identiques. Outre que les machines, on suppose qu'il existe un ensemble de s types de ressources additionnelles $\Re = \{R_1, R_2, ..., R_s\}$, qui sont disponibles en quantitées $m_1, m_2, ..., m_s$ unités respectivement. Chaque tâche T_i nécessite pour son traitement une machine et des quantitées de ressources spécifiques. Chaque tâche T_i ($i = \overline{1, n}$) est donc caractérisée par les paramètres suivants :

1. demande en ressource $R(T_i) = [R_1(T_i), R_2(T_i), ..., R_s(T_i)]$ où $0 \leq R_k(T_i) \leq m_k$ pour tout k ($k = \overline{1, s}$) et représente le nombre d'unités de la ressource R_k nécessaires pour le traitement de la tâche T_i.
2. la durée de traitement p_i de la tâche T_i.
3. la date de disponibilité r_i de la tâche T_i.

De plus, on suppose que toute tâche doit être assignée de toutes les ressources exigées en sa date de début, qui seront retournées une fois que la tâche aurait terminé son traitement.

Dans la littérature [8] ce problème est noté avec la notation $P|\beta, ...|C_{\max}$, où le paramètre β est défini comme suit :
$\beta \in \{\phi, res\lambda\sigma\rho\}$, caractérise les ressources additionnelles.
$\beta = \phi$ correspond à la non-existence des contraintes de ressources.
$\beta = res\lambda\sigma\rho$ correspond à l'existence des contraintes de ressources spécifiques :
$\lambda, \sigma, \rho \in \{., k\}$ représentent le nombre de types de ressources additionnelles, les disponibilités en ressources et les demandes en ressources respectivement. Si $\lambda, \sigma, \rho = .$ ceci signifie que le nombre de types de ressources additionnelles, les disponibilités en ressources et les demandes en ressources respectivement sont arbitraires.
Si $\lambda, \sigma, \rho = k$ ceci signifie que le nombre de types de ressources additionnelles, les disponibilités en ressources sont toutes égales à k et les demandes en ressources de chaque tâche sont inférieures ou égales à k unités.

Notons que le problème d'ordonnancement à contraintes de ressources $P|res.11, r_i|C_{\max}$ peut se réduire polynomialement au problème $P|AgreeG = (V, E), r_i|C_{\max}$ et vice versa, ils sont donc plynomialement équivalents.

Quelques résultats de complexité connus dans la littérature :

Problème	Complexité	Référence
$P2\|res\ 1.., p_i = 1\|C_{\max}$	$O(nlogn)$	[8]
$P2\|res\ 1.., r_i, p_i = 1\|C_{\max}$	NP-difficile au sens fort	[9]
$P2\|res\ ..., p_i = 1\|C_{\max}$	$O(n^{2.5})$	[35]
$P2\|res\ .11, r_i, p_i = 1\|C_{\max}$	NP-difficile au sens fort	[9]
$P3\|res\ 1.., p_i = 1\|C_{\max}$	NP-difficile au sens fort	[35]
$P3\|res\ .11, r_i, p_i = 1\|C_{\max}$	NP-difficile au sens fort	[10]
$P\|res\ 1.1, r_i, p_i = 1\|C_{\max}$	$O(n)$	[11]
$P\|res\ spr, p_i = 1\|C_{\max}$	$O(n)$	[12]

TABLE 3.6 – Résultats de complexité connus

3.5 Inéxistence d'un algorithme absolu

Un algorithme fournissant une approximation absolue (ou algorithme absolu) est une heuristique donnant, pour toute instance du problème, une solution approchée dont l'écart par rapport à la solution optimale est borné par une constante. Dans le résultat suivant, on montre qu'il n'existe pas d'algorithmes absolus pour le problème général traité à moins que $P = NP$.

Théorème 3.1. *Si $P \neq NP$ alors il n'existe pas d'algorithme polynomial fournissant une approximation absolue pour le problème général $P|AgreeG = (V, E), r_i|C_{\max}$.*

Preuve. On procède par contradiction en utilisant un changement d'échelle. Supposons dans le cas contraire qu'un tel algorithme, disant A existe, fournissant une approximation absolue d'ordre K c-à-d $|A(I) - OPT(I)| \leq K$ pour toute instance I. Soit I une instance du problème général, représentée par la donnée d'un ensemble de n tâches $J_1, J_2, ...J_n$ avec leurs durées de traitement respectives $p_1, p_2, ..., p_n$ et le graphe de concordance G. Nous construisons une nouvelle instance I' pour le problème comme suit : $J'_1, J'_2, ..., J'_n$ sont n tâches en correspondance avec les tâches $J_1, J_2, ..., J_n$ de sorte que le graphe de concordance G' est isomorphe au graphe G. Pour tout $i = 1, 2, ..., n$, la durée de traitement de la tâche J'_i est $p'_i = (K + 1)p_i$.

Dans la suite, on montre qu'à tout ordonnancement réalisable σ de l'instance I, correspond un ordonnancement réalisable σ' de l'instance I' et vice-versa tel que $C_{\max}(\sigma') = (K + 1)C_{\max}(\sigma)$ et en particulier $OPT(I') = (K + 1)OPT(I)$.

En effet, soit σ un ordonnancement réalisable pour I et soient t_i et c_i les dates de début et fin de traitement respectivement pour toute tâche J_i, par rapport à σ. On construit l'ordonnancement σ' de I' comme suit : toute tâche J'_i est traitée à la date $t'_i = (K + 1)t_i$ sur la même machine sur laquelle la tâche J_i a été traitée. Montrons que σ' est un ordonnancement réalisable pour I'. Pour cela , soit $\{J'_i, J'_j\}$ une paire de tâches qui se chevauchent sur un certain intervalle, par rapport à l'ordonnancement σ'. Ceci est équivalent à dire que $t'_j < c'_i$ et $t'_i < c'_j$, où t'_k et c'_k représentent la date de début et de fin de traitement respectivement de la tâche J'_k par rapport à σ'. En divisant par $K + 1$, il en résulte que leurs tâches correspondantes respectives J_i et J_j de l'instance I vérifient aussi les relations $t_j < c_i$ et $t_i < c_j$. Ceci implique que les tâches J_i et J_j se chevauchent par rapport à l'ordonnancement σ. D'après la réalisabilité de σ, les tâches J_i et J_j sont alors concordantes pour l'instance I. Ceci veut dire que ces dernières sont adjacentes dans G. Comme G' est isomorphe à G, on en déduit que les tâches J'_i et J'_j sont adjacentes dans G', donc concordantes par rapport à l'instance I' et par conséquent l'ordonnancement σ' est réalisable.

Quant à la relation $C_{\max}(\sigma') = (K+1)C_{\max}(\sigma)$, soit J'_l une tâche de G' telle que $C_{\max}(\sigma') =$

c'_l. Soit J_l la tâche correspondante à J'_l. On a : $c'_l = t'_l + p'_l = (K+1)t_l + (K+1)p_l = (K+1)c_l$. On affirme que $C_{\max}(\sigma) = c_l$. En effet, il est clair que $C_{\max}(\sigma) \geq c_l$ mais si $C_{\max}(\sigma) > c_l$, alors la tâche J_k de G dont la date de fin de traitement égale à $C_{\max}(\sigma)$ vérifiera $c_k > c_l$, donnant $c'_k > c'_l$ et ceci contredit le fait que $C_{\max}(\sigma') = c'_l$. Il en résulte que $C_{\max}(\sigma) = c_l$ et par conséquent $C_{\max}(\sigma') = (K+1)C_{\max}(\sigma)$.

D'autre part on peut facilement montrer, comme on a fait précédemment qu'un ordonnancement réalisable σ est optimal pour l'instance I si et seulement si l'ordonnancement correspondant σ' pour l'instance I' est optimal et que $OPT(I') = (K+1)OPT(I)$.

Réciproquement, on peut montrer de la même manière qu'à tout ordonnancement réalisable σ' de l'instance I' correspond un ordonnancement réalisable σ de I satisfaisant les relations $C_{\max}(\sigma') = (K+1)C_{\max}(\sigma)$ et $OPT(I') = (K+1)OPT(I)$.

Maintenant, on exécute l'algorithme A pour l'instance I' et soient σ'_A l'ordonnancement réalisable obtenu et σ_A l'ordonnancement réalisable correspondant pour l'instance I. Clairement on a $|A(I') - OPT(I')| \leq K$, mais $A(I') = C_{\max}(\sigma'_A) = (K+1)C_{\max}(\sigma'_A)$ donc $|(K+1)C_{\max}(\sigma_A) - (K+1)OPT(I)| \leq K$, ce qui implique que $|C_{\max}(\sigma_A) - OPT(I)| \leq \frac{K}{K+1} < 1$.
Comme $C_{\max}(\sigma_A)$ et $OPT(I)$ sont des entiers, on a alors $C_{\max}(\sigma_A) = OPT(I)$. Il s'en suit que σ_A est un ordonnancement réalisable optimal pour l'instance I et donc le problème général peut être résolu polynomialement par l'algorithme A, contredisant la supposition que $P \neq NP$. ■

Chapitre 4

Graphe biparti

Dans ce chapitre, nous considérons le cas où le graphe de concordance est un graphe biparti arbitraire, noté $G = (S_1, S_2; E)$ et le nombre de machines est égal à 2.
Dans la littérature et plus exactement dans la référence [28], Even et al. ont posé un problème ouvert qui est la complexité du problème d'ordonnancement avec conflit S.W.C avec deux machines et au plus 3 durées de traitement possibles : $p_i \in \{1, 2, 3\}$, pour un graphe de conflit arbitraire . Du point de vue complexité, ce problème ouvert est polynomialement équivalent au problème $P2|AgreeG = (V, E), p_i \in \{1, 2, 3\}|C_{max}$. Dans le premier paragraphe, nous montrons que ce problème ouvert est NP-difficile, même quand le graphe de concordance est biparti. Nous montrons, dans le second paragraphe que le problème est NP-difficile dans le cas de deux machines avec $p_i \in \{1, 2\}$ et $r_i \in \{0, r\}$ et nous présentons dans le dernier paragraphe un sous problème polynomial, avec un exemple illustratif.

4.1 Cas $p_i \in \{1, 2, 3\}$

Théorème 4.1. *[6] Le problème $P2|AgreeG = (S_1, S_2; E), p_i \in \{1, 2, 3\}|C_{max}$ est NP-difficile.*

Preuve. Considérons le problème 3-Dimensional Matching Problem(3-DM) :
Instance : un ensemble M de triplets tel que $M \subseteq X \times Y \times Z$ où X, Y, Z sont mutuellement disjoints de même cardinalité q.
Question : existe-t-il un sous ensemble $M' \subseteq M$ tel que $|M'| = q$ et les triplets de M' sont deux à deux disjoints ?
Ce problème de décision est NP-complet (Garey and Johnson [32]).
On démontre que ce problème se réduit polynomialement au problème (P) suivant :
Instance : l'ensemble des tâches est $V = V_M \cup V_Y \cup V_Z \cup V_D \cup V_Q \cup V_R$ défini comme suit. Les tâches de V_M sont en correspondance avec les éléments de M de sorte qu'à chaque élément $(x, y, z) \in M$, lui correspond une tâche $J(x, y, z)$ de V_M. Les tâches de V_Y et

V_Z sont en correspondance avec les éléments des ensembles Y et Z respectivement, de façon que chaque élément $y \in Y$ correspond à une tâche J_y de V_Y et chaque élément $z \in Z$ correspond à une tâche J_z de V_Z, donnant $|V_Y| = |V_Z| = q$. Supposons que $X = \{x_1, x_2, ..., x_q\}$, donc pour chaque i $(i = 1, 2, ..., q)$ soit $M_i = \{(x, y, z) \in M : x = x_i\}$ de sorte que $\bigcup_{i=1}^{q} M_i = M$ et soit V_{M_i} le sous ensemble de V_M correspondant à M_i. Pour tout i $(i = 1, 2, ..., q)$, à l'ensemble des tâches V_{M_i} correspond un ensemble V_{D_i} , aussi l'ensemble V_{D_i} à son tour, lui correspond un ensemble V_{Q_i} et finalement à l'ensemble V_{Q_i} est associé un ensemble V_{R_i} de manière que $|V_{D_i}| = |V_{Q_i}| = |V_{R_i}| = |M_i| - 1$. Les ensembles V_D, V_Q et V_R sont tel que : $V_D = \bigcup_{i=1}^{q} V_{D_i}$, $V_Q = \bigcup_{i=1}^{q} V_{Q_i}$, $V_R = \bigcup_{i=1}^{q} V_{R_i}$. On peut voir que $|V_D| = |V_Q| = |V_R| = |M| - q$. Le graphe de concordance $G = (V, E)$ est défini de la manière suivante : les ensembles V_Y et V_Z, ainsi que les ensembles V_{M_i}, V_{D_i}, V_{Q_i} et V_{R_i} forment chacun un stable dans le graphe G pour tout i $(i = 1, 2, ..., q)$. Toute tâche $J(x, y, z)$ de V_M correspondant à l'élément (x, y, z) de M est adjacente à deux tâches J_y et J_z correspondant à y et z respectivement ($J_y \in V_Y$ et $J_z \in V_Z$). Pour tout i $(i = 1, 2, ..., q)$, toutes les tâches de V_{M_i} sont reliées à toutes les tâches de V_{D_i}. Pour tout i $(i = 1, 2, ..., q)$ les tâches de V_{D_i} sont en correspondance biunivoque avec celles de V_{Q_i}, ce qui veut dire que chaque tâche de V_{D_i} est adjacente seulement à sa tâche correspondante dans V_{Q_i}. De la même manière les tâches de V_{Q_i} sont en correspondance biunivoque avec celles de V_{R_i}, voir figure 4.1 pour illustration.

Les durées de traitements des tâches des deux ensembles V_M et V_Q sont égales à 2, celles des ensembles V_R, V_Y et V_Z sont égales à 1. Les durées de traitements des tâches de l'ensemble V_D valent 3.

Notons que le graphe de concordance G est biparti puisqu'il peut être écrit $G = (S_1, S_2; E)$ où $S_1 = V_M \cup V_Q$, $S_2 = V_Y \cup V_Z \cup V_D \cup V_R$ sont deux stables formant une partition de l'ensemble des sommets de G.

Question : existe-t-il un ordonnancement réalisable σ avec un makespan $C_{\max}(\sigma) \leq 4|M| - 2q$?

Supposons que 3-DM admet une solution et considérons l'ordonnancement σ défini comme suit : soit $V_{M'} = \{J_1, J_2, ..., J_q\}$, le sous ensemble de tâches de V_M correspondant à M' tels que $J_i \in V_{M_i}$ $(i = 1, 2, ..., q)$. On ordonnance ces tâches ainsi que leurs tâches correspondantes des ensembles V_Y et V_Z, dans l'intervalle de temps $[0,\ 2q]$ comme indiqué sur la figure 4.2. Pour tout i $(i = 1, 2, ..., q)$ soit $V_i = V_{D_i} \cup V_{Q_i} \cup V_{R_i} \cup V_{M_i}$. Noter que $(\bigcup_{i=1}^{q} V_i) \cup V_Y \cup V_Z = V$. Soient $V_{R_i} = \{R_{i1}, R_{i2}, ..., R_{i(|M_i|-1)}\}$, $V_{Q_i} = \{Q_{i1}, Q_{i2}, ..., Q_{i(|M_i|-1)}\}$ et $V_{D_i} = \{D_{i1}, D_{i2}, ..., D_{i(|M_i|-1)}\}$ où la tâche R_{ik} correspond à la tâche Q_{ik} et la tâche Q_{ik} correspond à la tâche D_{ik} pour tous les k $(k \in \{1, 2, ..., |M_i| - 1\})$.

Pour tout $i \in \{1, 2, ..., q\}$, on ordonnance les tâches de V_i de la manière suivante. Rappelons qu'une tâche de V_i, qui est la tâche J_i de V_{M_i} a été déjà ordonnancée avec les tâches correspondant à M' comme indiqué sur la figure 4.2. On ordonnance le reste des tâches de

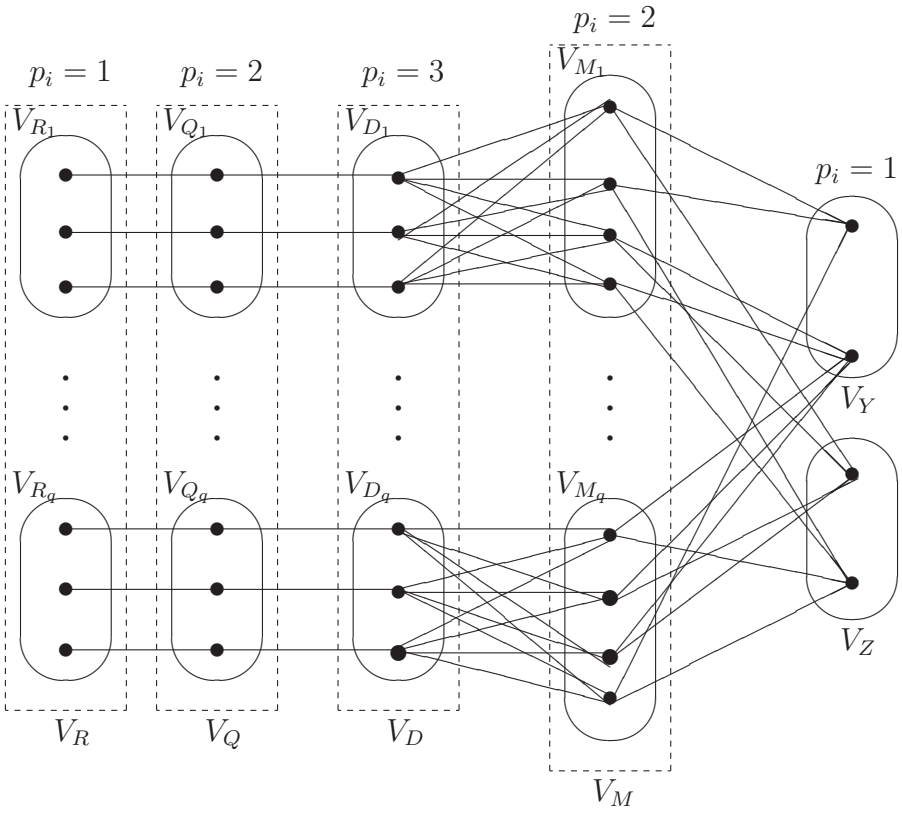

FIGURE 4.1 – Le graphe de concordance $G = (V, E)$

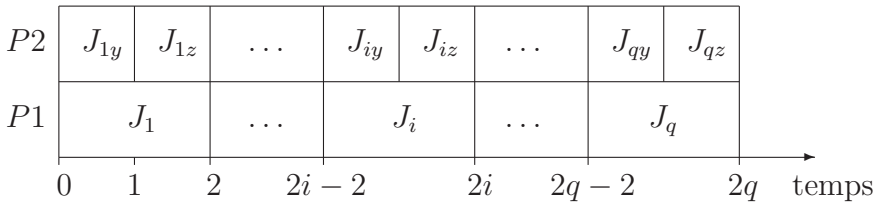

FIGURE 4.2 – Ordonnancement des tâches de $V_{M'}$ avec leurs tâches correspondantes

l'ensemble V_i en $(|M_i| - 1)$ intervalles de temps de longueur 4 comme le montre la figure 4.3. Ceci veut dire que pour tout k $(k = 1, ..., (|M_i| - 1))$, les deux tâches Q_{ik} et R_{ik} sont ordonnancées à la même date sur les différentes machines, la tâche D_{ik} est ordonnancée immédiatement après la tâche R_{ik} sur la même machine où la tâche D_{ik} est traitée. Une tâche de V_{M_i} (différente de J_i) est ordonnancée immédiatement après la tâche Q_{ik} sur la même machine traitant la tâche Q_{ik}. Pour obtenir un ordonnancement global des tâches de $V_i \backslash \{J_i\}$, on ordonnance les tâches de tous les quadruplets de la forme $\{Q_{ik}, R_{ik}, D_{ik},$ une tâche de V_i (différente de J_i)$\}$ successivement comme indiqué sur la figure 4.3, sans temps morts $(k = 1, ..., (|M_i| - 1))$. En ordonnançant les tâches des ensembles $V_i \backslash \{J_i\}$ successivement, de la façon juste décrite et en y joignant l'ordonnancement des tâches de $V_{M'}$ et leurs tâches correspondantes de V_Y et V_Z déjà ordonnancées, on obtient un

ordonnancement σ avec un $C_{max} = \sum_{i=1}^{q} 4\ (|M_i| - 1) + 2q = 4|M| - 2q$, qui est une solution du problème (P).

Inversement, supposons qu'il existe un ordonnancement réalisable σ pour le problème (P), représenté par un diagramme de Gantt. En comptant la somme des durées de traitement de toutes les tâches du graphe de concordance G, on peut vérifier que cette somme vaut $8|M| - 4q$, ceci veut dire que tout l'espace du diagramme de Gantt représentant σ est occupé. Par construction du graphe de concordance G, $(|M_i| - 1)$ quadruplets de la forme $\{Q_{ik}, R_{ik}, D_{ik},$ une tâche de l'ensemble $V_{M_i}\}$ devaient être ordonnancées comme indiqué sur la figure 4.3 ou comme sur sa configuration symétrique, pour tout i $(i = 1, 2, ..., q)$. L'espace occupé par toutes ces tâches est égal à $2[\sum_{i=1}^{q} 4\ (|M_i| - 1)] = 8|M| - 8q$. L'espace restant est alors égal à $(8|M| - 4q) - (8|M| - 8q) = 4q$, qui devait être occupé par q tâches de G_M et les $2q$ tâches des ensembles V_Y et V_Z.

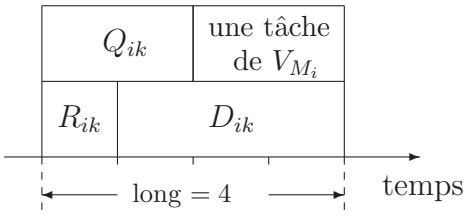

FIGURE 4.3 – Ordonnancement des tâches de V_i

Par construction du graphe de concordance G, ces $3q$ tâches doivent être ordonnancées comme indiqué sur la figure 4.2. Ceci correspond à un sous ensemble $M' \subseteq M$ avec q éléments, produisant donc une solution de 3-DM. Il est clair que la réduction utilisée est polynomiale et que $(P) \in$ NP, ce qui complète la preuve du Théorème 4.1. ■

4.2 Cas $p_i \in \{1, 2\},\ r_i \in \{0, r\}$

Dans cette section, nous supposons que les dates de disponibilité des tâches peuvent prendre deux valeurs possibles 0 et r. Il a été prouvé par Even et al. [28], que le problème avec 2 machines et deux durées de traitement : $p_i \in \{1, 2\}$, est polynomial. Dans le résultat suivant, nous montrons qu'en ajoutant la condition $r_i \in \{0, r\}$, avec r arbitraire, le problème devient NP-difficile même pour les graphes bipartis.

Théorème 4.2. *[6] Le problème $P2/AgreeG = (S_1, S_2; E), p_i \in \{1, 2\}, r_i \in \{0, r\}|C_{max}$ est NP-difficile.*

Preuve. On montre que 3-Dimensional Matching Problem (3-DM) peut être réduit polynomialement au problème de décision suivant (P) :
Instance : l'ensemble des tâches est $V = V_M \cup V_Y \cup V_Z \cup V_D$ où les ensembles V_M, V_Y, V_Z

et V_D comme dans le cas $p_i \in \{1, 2, 3\}$, considéré précédemment. Les ensembles V_{M_i} et le graphe de concordance G sont aussi définis de la même manière comme dans le cas $p_i \in \{1, 2, 3\}$. Les durées de traitement des tâches de V_M et celles de V_D sont égales à 2, celles de V_Y et V_Z valent 1. Les dates de disponibilité des tâches de G sont nulles sauf celles de V_D qui sont égales à $2q$,voir figure 4.4.

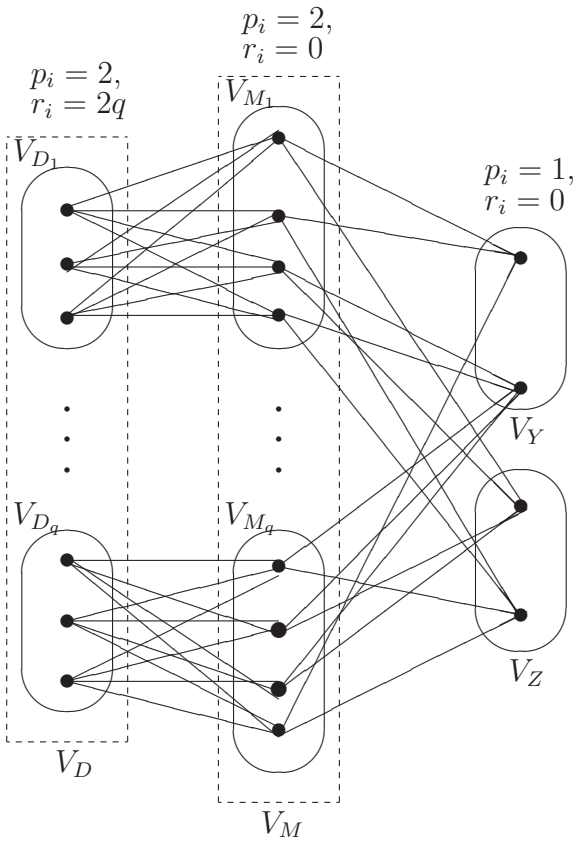

FIGURE 4.4 – Le graphe de concordance $G = (V, E)$

Notons que le graphe de concordance G est un graphe biparti puisqu'il peut être écrit $G = (S_1, S_2; E)$ où $S_1 = V_M$ et $S_2 = V_Y \cup V_Z \cup V_D$ sont deux stables formant une partition de l'ensemble de sommets de G.

Question : existe t- il- un ordonnancement réalisable σ avec un makespan $C_{\max}(\sigma) \leq 2\,|M|$?

Supposons que 3-DM a une solution et considérons l'ordonnancement σ défini comme suit : soit $V_{M'} = \{J_1, J_2, .., J_q\}$: le sous ensemble de tâches de V_M correspondant à M' tels que $J_i \in V_{M_i}$ $(i = 1, 2, ..., q)$. On ordonnance ces tâches J_i, ainsi que leurs tâches correspondantes J_{iy} et J_{iz} des ensembles V_Y et V_Z respectivement, dans l'intervalle de temps $[0,\ 2q]$ comme indiqué sur la figure 4.2.

On ordonnance les autres $|M| - q$ tâches de V_M et les tâches de V_D dans l'intervalle de temps $[2q,\ 2|M|]$ en q groupes comme suit : pour tout i $(i = 1, 2, ..., q)$ on ordonnance les tâches de $V_{M_i} \backslash \{J_i\}$ sur la machine $P1$, puis opposé sur $P2$ on ordonnance les tâches de

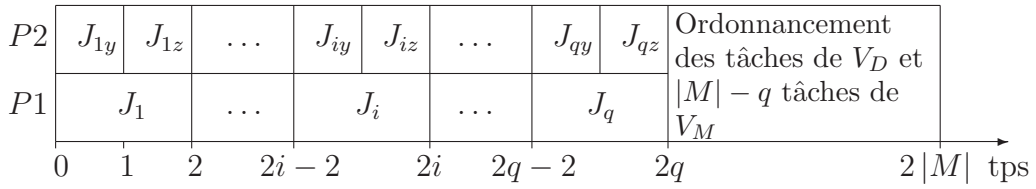

FIGURE 4.5 – L'ordonnancement σ

V_{D_i} (par exemple successivement), voir figure 4.5. L'ordonnancement σ qu'on vient juste de construire, est une solution du problème (P).

Réciproquement, supposons qu'il existe ordonnancement réalisable σ pour le problème (P) tel que $C_{\max}(\sigma) \leq 2|M|$. Les tâches de V_D forment un stable dans le graphe de concordance G de cardinalité $|M|-q$. Ayant chacune une durée de traitement égale à 2 et une date de disponibilité de valeur $2q$, elles devaient être ordonnancées dans l'intervalle de temps $[2q,\ 2|M|]$. Considérons le diagramme de Gantt représentant σ. La somme totale des durées de traitement des tâches de G est égale à $4|M|$, ceci implique que l'espace de ce diagramme est entièrement occupé. Puisque pour tout i $(i = 1, 2, ..., q)$ les seules tâches concordantes avec les tâches de V_{D_i} sont celles de V_{M_i} et que les durées de traitement des tâches des deux ensembles V_D et V_M valent 2, alors $|M|-q$ tâches de V_M devaient être ordonnancées parallèlement aux tâches de V_D dans l'intervalle de temps $[2q,\ 2|M|]$, voir figure 4.5. Donc les tâches restantes sont q tâches de V_M, disant $J_1, J_2, .., J_q$ appartenant à $V_{M_1}, V_{M_2}, ..., V_{M_q}$ respectivement, qui devaient être été ordonnancées avec les tâches des ensembles V_Y et V_Z comme le montre la même figure 4.5. Ceci correspond à un sous ensemble $M' \subseteq M$ avec q éléments produisant une solution de 3-DM. Il est facile de vérifier que la réduction utilisée est polynomiale et que $(P) \in$ NP, complétant la preuve du Théorème 4.2. ■

4.3 Durées de traitement unitaires

D'après le Théorème 4.1, le problème est NP-difficile pour deux machines avec graphe de concordance biparti et durées de traitement arbitraires. Dans le résultat suivant on montre que ce dernier devient polynomial quand les durées de traitement des tâches de S_1 sont égales à 1 et celles de S_2 sont arbitraires. Ce problème est noté $P2/AgreeG = (S_1, S_2; E), p_{S_1} = 1|C_{max}$.

Supposons que $S_1 = \{J_1, J_2, ..., J_{|S_1|}\}$ et $S_2 = \{J_{|S_1|+1}, J_{|S_1|+2}, ..., J_{|S_1|+|S_2|}\}$. Pour toute tâche $J_{|S_1|+i}$ de S_2, on note sa durée de traitement par p_i.
Considerons l'algorithme suivant dont le principe est : d'abord, on ordonnance les tâches de S_2 successivement sur $P1$ et puis on ordonnance sur $P2$, le maximum de tâches uni-

taires (qui sont les tâches de S_1) en parallèle avec celles de S_2. Les tâches restantes de S_1 sont ordonnancées successivement après les tâches de S_2.

Algorithme 1
Début

1. Construire le réseau $R = (V, U, c)$ comme suit : $V = \{s, p\} \cup S_1 \cup S_2$,
   $U = U_1 \cup U_2 \cup U_3 \cup \{u_r\}$ où $u_r = (p, s)$ est un arc de retour, $U_1 = \{(s, J_i) : J_i \in S_1\}$,
   $U_2 = \{(J_i, J_j) \in S_1 \times S_2 : \{J_i, J_j\} \in E\}, U_3 = \{(J_j, p) : J_j \in S_2\}$
2. Construire le vecteur arc capacité c comme suit :
   si $(s, J_i) \in U_1 \Longrightarrow c(s, J_i) = 1$, si $(J_i, J_j) \in U_2 \Longrightarrow c(J_i, J_j) = 1$,
   $c(u_r) = +\infty$, si $(J_{|S_1|+i}, p) \in U_3 \Longrightarrow c(J_{|S_1|+i}, p) = p_i$.
3. Déterminer un flot réalisable maximum f^* de s à p, dans le réseau R.
4. Ordonnancer les tâches de S_2 successivement dans l'intervalle $[0, \sum_{i=1}^{|S_2|} p_i]$, sur $P1$.
5. Ordonnancer les tâches J_i de S_1 tel que $f^*(J_i, J_{|S_1|+1}) = 1$ successivement et parallèlement à la tâche $J_{|S_1|+1}$ dans l'intervalle $[0, p_1]$ sur $P2$ et les tâches J_i de S_1 tel que $f^*(J_i, J_{|S_1|+2}) = 1$ successivement et parallèlement à la tâche $J_{|S_1|+2}$ dans l'intervalle $[p_1, p_1 + p_2]$ sur $P2$ et ainsi de suite.
6. Ordonnancer les tâches restantes de S_1 successivement dans l'intervalle de temps $[\sum_{i=1}^{|S_2|} p_i, \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)\ ]$.

Fin.

On donne la forme du réseau $R = (V, U, c)$ dans
dans le cas $|S_1| = |S_2| = 3$.

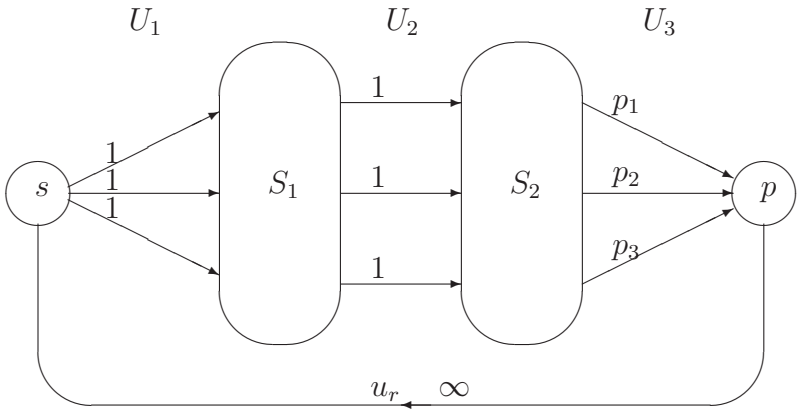

FIGURE 4.6 – Le réseau $R = (V, U, c)$

Remarque 4.1. *Si u est un arc de R, et f est un flot dans R, alors $f(u)$ représente la valeur du flot f sur l'arc u dans R.*

Théorème 4.3. *[6] Le problème $P2/AgreeG = (S_1, S_2; E), p_{S_1} = 1|C_{max}$ est polynomial. Algorithme 1 le résout avec une complexité en $O(n^3)$ si dans l'étape 3 on utilise l'algorithme de J. Cheriyan et S.N. Maheshwari (Cheriyan and Maheshwari [22]) pour obtenir un flot maximum.*

Preuve. Il est clair que Algorithme 1 retourne un ordonnancement réalisable σ^* pour le problème $P2/AgreeG = (S_1, S_2; E), p_{S_1} = 1|C_{max}$ avec un makespan $C_{\max}(\sigma^*) = \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)$ comme représenté par le diagramme de Gantt de la figure 4.7, où $\alpha^* = \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)$. Montrons maintenant que tout ordonnancement réalisable σ vérifie $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Soit σ un ordonnancement réalisable arbitraire représenté par un diagramme de Gantt. On décale les tâches de S_2 ainsi que leurs tâches correspondantes opposées de S_1 au côté gauche le plus extrême de ce diagramme de sorte que toutes ces tâches seront placées dans l'intervalle de temps $[0, \sum_{i=1}^{|S_2|} p_i]$. On place les tâches de S_2 sur $P1$, leurs tâches correspondantes opposées de S_1 sur la machine $P2$. Ensuite en plaçant le reste des tâches de S_1 successivement et bout à bout sur $P1$, après l'instant $\sum_{i=1}^{|S_2|} p_i$, on obtient un ordonnancement réalisable σ_1 du problème. On peut imaginer σ_1 comme l'ordonnancement représenté par un diagramme de Gantt similaire à celui correspondant à σ^* de la figure 4.7. L'ordonnancement σ_1 induit un flot f_1 sur le réseau R tel que $C_{\max}(\sigma_1) = \sum_{i=1}^{|S_2|} p_i + |S_1| - f_1(u_r)$. Noter que dans l'ordonnancement σ_1, il n'existe aucun temps mort commun aux deux machines et donc $C_{\max}(\sigma) \geq C_{\max}(\sigma_1)$. Puisque f^* est un flot maximum dans R, il s'en suit que $f_1(u_r) \leq f^*(u_r)$. D'autre part $C_{\max}(\sigma^*) = \sum_{i=1}^{|S_2|} p_i + |S_1| - f^*(u_r)$, alors $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. σ étant arbitraire, on en déduit que σ^* est optimal.

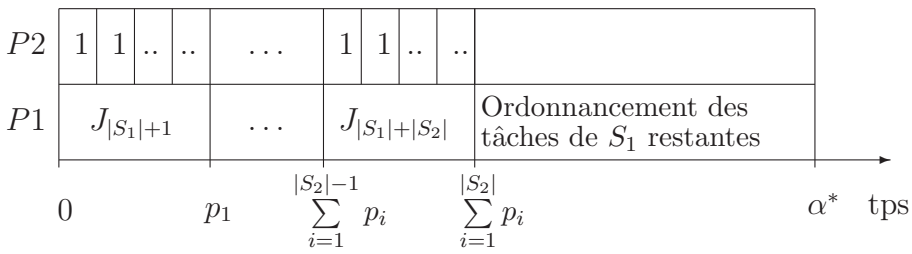

FIGURE 4.7 – Le diagramme de Gantt représentant σ^*

La complexité de l'Algorithme 1 : la construction de R est de l'ordre $O(|S_1|\,|S_2|)$ $= O(n^2)$. La détermination d'un flot maximum dans le réseau R est polynomial, donc

le problème $P2/AgreeG = (S_1, S_2; E), p_{S_1} = 1|C_{max}$ est polynomial. En utilisant l'algorithme de J. Cheriyan et S.N.Maheshwari [22], l'étape 3 peut être implémentée en $O(n^2\sqrt{|U|})$. Comme $|U| = O(|S_1||S_2|) = O(n^2)$, donc l'étape 3 peut être réalisée en $O(n^3)$. Il s'en suit que la complexité de l'algorithme Algorithme 1 est en $O(n^3)$. ■

Exemple 4.1. *Considérons l'ordonnancement de six tâches $J_1, J_2, ..., J_6$ tels que : le graphe de concordance est biparti $G = (S_1, S_2; E)$ où $S_1 = \{J_1, J_2, J_3\}$ et $S_2 = \{J_4, J_5, J_6\}$. Les durées de traitements des tâches sont données par le tableau suivant :*

J_i	J_1	J_2	J_3	J_4	J_5	J_6
p_i	1	1	1	2	3	4

TABLE 4.1 – Durées de traitement de l'exemple 4.1

Le graphe de concordance G est représenté par la figure 4.8.

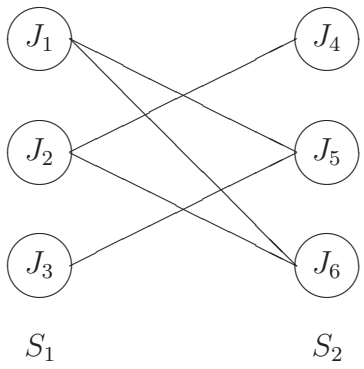

FIGURE 4.8 – Le graphe de concordance de l'exemple 4.1, $G = (S_1, S_2; E)$

Déroulement de l'Algorithme1 :

Le réseau R correspondant à cet exemple est donné sur la figure 4.9.
Première itération : on part du flot nul f, on marque le sommet s et essayons de marquer p.
Il existe un arc de s à J_1 tel que $f(s, J_1) = 0 < c(s, J_1) = 1$. Donc on marque le sommet J_1 par le marquage direct.
Il existe un arc de J_1 à J_5 tel que $f(J_1, J_5) = 0 < c(J_1, J_5) = 1$, on marque le sommet J_5.
Il existe un arc de J_5 à p tel que $f(J_5, p) = 0 < c(J_5, p) = 3$, on marque le sommet p. La chaine augmentante est $C_1 = \{(s, J_1), (J_1, J_5), (J_5, p)$.
$\delta_1 = min\{1, 1, 3\} = 1$, le nouveau flot réalisable f_1 est tel que : $f_1(s, J_1) = 1, f_1(J_1, J_5) =$

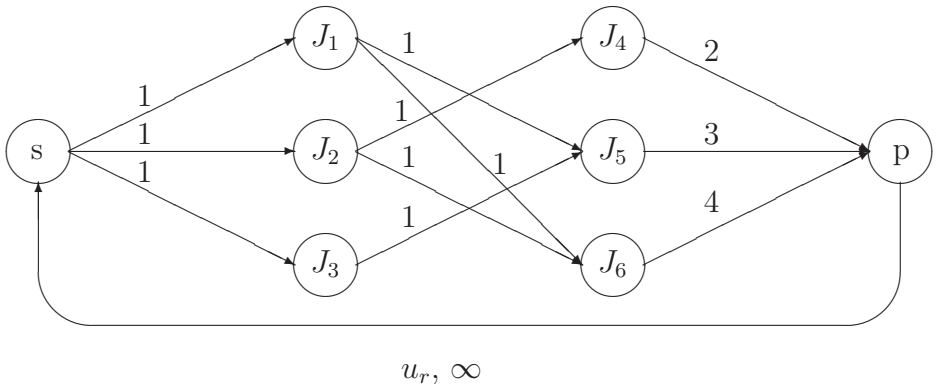

FIGURE 4.9 – Le réseau $R = (V, U, c)$ de l'exemple 4.1

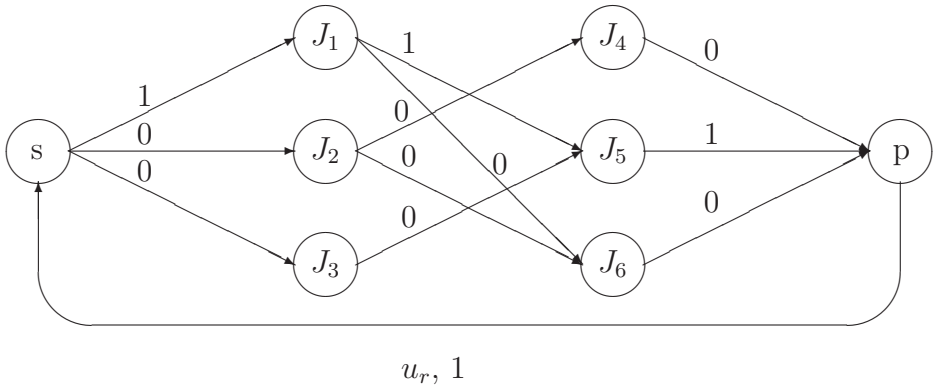

FIGURE 4.10 – Représentation du flot f_1

$1, f_1(J_5, p) = 1, f_1(u_r) = 1$ et pour tout arc $u \notin C_1$, $f_1(u) = f_(u)$. Ce flot f_1 est représenté sur la figure 4.10.

Deuxième itération : on part du flot f_1 représenté sur la figure 4.10, on marque le sommet s et essayons de marquer p.

L'arc (s, J_1) est tel que $f(s, J_1) = c(s, J_1) = 1$. Donc cet arc est saturé , on ne peut pas marquer J_1.
Il existe un arc de s à J_2 tel que $f(s, J_2) = 0 < c(s, J_2) = 1$, on marque le sommet J_2. Il existe un arc de J_2 à J_6 tel que $f(J_2, J_6) = 0 < c(J_2, J_6) = 1$, on marque le sommet J_6 par le marquage direct. La chaine augmentante est $C_2 = \{(s, J_2), (J_2, J_6), (J_6, p)\}$.
$\delta_2 = min\{1, 1, 4\} = 1$, le nouveau flot réalisable f_2 est tel que : $f_2(s, J_2) = 1, f_2(J_2, J_6) = 1, f_2(J_6, p) = 1, f_2(u_r) = 2$ et pour tout arc $u \notin C_2$, $f_2(u) = f_1(u)$. Ce flot f_2 est représenté sur la figure 4.11.

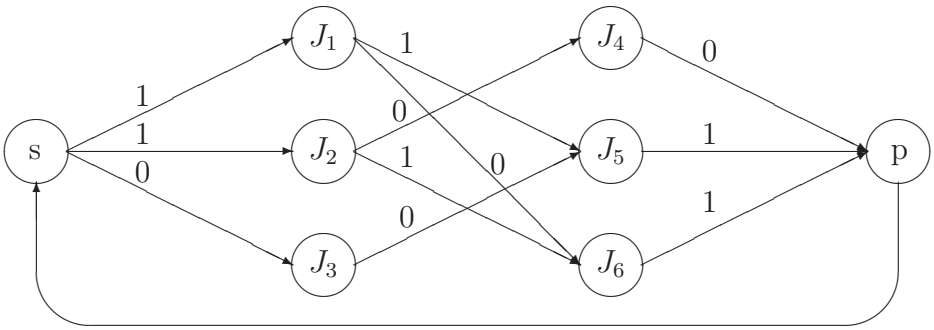

FIGURE 4.11 – Représentation du flot f_2

Toisième itération : on part du flot f_2, on marque le sommet s et essayons de marquer p.
Il existe un arc de s à J_3 tel que $f(s, J_3) = 0 < c(s, J_3) = 1$. Donc on marque le sommet J_3 par le marquage direct.
Il existe un arc de J_3 à J_5 tel que $f(J_3, J_5) = 0 < c(J_3, J_5) = 1$, on marque le sommet J_5.
Il existe un arc de J_5 à p tel que $f(J_5, p) = 1 < c(J_5, p) = 3$, on marque le sommet p. La chaine augmentante est $C_3 = \{(s, J_3), (J_3, J_5), (J_5, p)\}$.
$\delta_3 = min\{1, 1, 2\} = 1$, le nouveau flot réalisable f_3 est tel que : $f_3(s, J_3) = 1, f_3(J_3, J_5) = 2, f_3(J_5, p) = 1, f_3(u_r) = 2 + 1 = 3$ et pour tout arc $u \notin C_3$, $f_3(u) = f_2(u)$. Ce flot f_3 est représenté sur la figure 4.12.

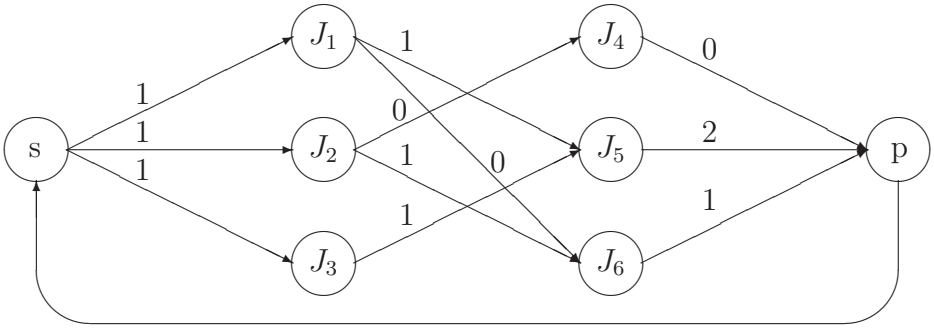

FIGURE 4.12 – Représentation du flot f_3

Quatrième itération : on part du flot f_3, on marque le sommet s et essayons de marquer p.
On voit qu'on ne peut plus faire d'autres marquages, par le marquage indirect. Comme tous les arcs sortant de s sont saturés, donc on ne peut pas faire d'autres marquage par le marquage direct, par conséquent on ne peut pas marquer le sommet p. D'après le

Théorème de la coupe minimum [48], le flot courant f_3 donné dans la figure 4.12 est un flot maximum de s à p , dans le réseau initial R de la figure 4.9.

Ordonnancement des tâches de S_2 : On ordonnance les tâches de S_2 successivement sur la machine $P1$ comme indiqué sur la figure 4.13.

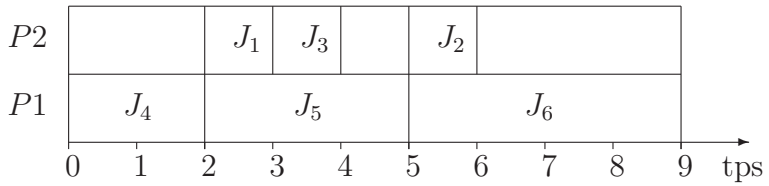

FIGURE 4.13 – L'ordonnancement obtenu de l'exemple 4.1

Ordonnancement des tâches de S_1 : Le flot maximum est le flot f_3 représenté sur la figure 4.12. On voit qu'il n'existe pas de tâches J_i de S_1 telle que : $f_3(J_i, J_4) = 1$. Les tâches J_i de S_1 telle que : $f_3(J_i, J_5) = 1$ sont les tâches J_1 et J_3. On ordonnance ces deux dernières sur la machine $P2$, au dessus de la tâche J_5, voir figure 4.13. Les tâches J_i de S_1 telle que : $f_3(J_i, J_6) = 1$ sont la tâche J_2. On ordonnance cette tâche sur la machine $P2$, au dessus de la tâche J_6, voir figure 4.13.

L'application de l'algorithme est terminée, l'ordonnancement obtenu est celui de la figure 4.13, il est optimal pour l'exemple 4.1.

Chapitre 5

Complémentaire d'un graphe biparti

Dans ce chapitre, nous considérons le problème dans le cas où le graphe de concordance est le complémentaire d'un graphe biparti avec des durées de traitement unitaires. Un tel graphe sera noté $G = (K_1, K_2; E)$, où K_1 et K_2 sont deux cliques formant une partition de l'ensemble des tâches de G. D'après les équivalences des problèmes étudiées au chapitre 3, ce problème est polynomialement équivalent au problème d'ordonnancement avec exclusion mutuelle pour les graphes bipartis. Comme ce dernier est NP-difficile (Bodlaender and Jansen [14]), il en est de même pour le problème $P|AgreeG = (K1, K_2; E), p_i = 1|C_{max}$. Dans l'absence des contraintes sur les machines (cas $m \geq n$), ce dernier est équivalent au problème de la coloration minimum pour les graphes bipartis arbitraires qui est polynomial.
Si de plus, les dates de disponibilités sont arbitraires, M. Boudhar et K. Khelladi [19] ont montré que le problème $P|AgreeG = (K_1, K_2; E), r_i, p_i = 1|C_{max}$ est NP-difficile au sens fort. En s'inspirant de la même démonstration, nous améliorons ce résultat et on montre dans le résultat suivant que ce dernier est NP-difficile même si on se restreint seulement à trois dates de disponibilités possibles. Dans le second paragraphe, nous présentons des sous problèmes polynomiaux dans le cas où les dates de disponibilités sont arbitraires impairs, ainsi que lorsque ces dernières sont arbitraires paires. Un exemple illustratif est présenté dans le cas de dates de disponibilités impaires.

5.1 Cas de dates de disponibilité $r_i \in \{0, 1, 2\}$

Théorème 5.1. *Le problème $P|AgreeG = (K_1, K_2; E), p_i = 1, r_i \in \{0, 1, 2\}|C_{max}$ est NP-difficile même pour $m \geq n$.*

Preuve. Considérons le problème 1-Precoloring Extension problem pour les graphes bipartis (1-PrExt Biparti) : **Instance :** un graphe biparti $G = (S_1 \cup S_2, E)$ et 3 sommets $v_1, v_2, v_3 \in S_1$. **Question :** existe t-il une 3-coloration de G tel que v_i reçoit la couleur i, pour tout i $(i = 1, 2, 3)$?

Le problème 1-PrExt Biparti est NP-complet (Bodlaender et al. [15]). On montre qu'il peut être réduit polynomialement au problème suivant D' :
instance : soit $m \geq n$ et soit $G' = (V', E')$ le graphe composé de $\overline{G} = (K_1, K_2; \overline{E})$ où K_1 et K_2 sont deux cliques qui correspondent à S_1 et S_2 respectivement, et de trois tâches u_1, u_2, u_3 de sorte que $V' = K_1 \cup K_2 \cup \{u_1, u_2, u_3\}$. L'ensemble des arêtes E' est défini comme suit : pour tout i $(i = 1, 2, 3)$, u_i est adjacent à v_i et à toutes les tâches de $K_1 \backslash \{v_1, v_2, v_3\}$, K_1 et $K_2 \cup \{u_1, u_2, u_3\}$ sont deux cliques de G' rendant ce dernier le complémentaire d'un graphe biparti voir figure 5.1 (a).

Les durées de traitement de toutes les tâches de G' sont égales à 1. $r_{v_i} = r_{u_i} = i - 1$ pout tout i $(i = 1, 2, 3)$, les dates de disponibilité pour le reste des tâches sont nulles.
Question : existe t-il un ordonnancement réalisable σ avec makespan $C_{\max}(\sigma) \leq 3$?.

Supposons qu'il existe une 3-coloration (C_1, C_2, C_3) de G telle que $v_i \in C_i$ pour tout i $(i = 1, 2, 3)$. On construit l'ordonnancement σ comme suit : les tâches de C_i et la tâche u_i sont ordonnancées à la date $t_i = i - 1$ pour tout i $(i = 1, 2, 3)$. L'ordonnancement σ est représenté dans figure 5.1 (b). On peut vérifier que σ est un ordonnancement réalisable pour le problème D' satisfaisant $C_{\max}(\sigma) \leq 3$. Réciproquement supposons qu'il existe un ordonnancement réalisable σ pour le problème D' tel que $C_{\max}(\sigma) \leq 3$, donc u_3 et v_3 doivent être été ordonnancé dans le même intervalle $[2, 3]$. Par construction, les tâches u_i et v_i doivent être été ordonnancé dans $[i - 1, i]$ pour tout i $(i = 1, 2)$. Pour chaque i $(i = 1, 2, 3)$ soit C_i= {tâches (différentes de u_i), ordonnancées à la date $i - 1$}. On peut vérifier que (C_1, C_2, C_3) est une 3-coloration de G telle que $v_i \in C_i$ pour tout i $(i = 1, 2, 3)$. On peut aussi vérifier que la réduction utilisée est polynomiale et que $D' \in NP$, complétant la preuve de ce théorème. ■

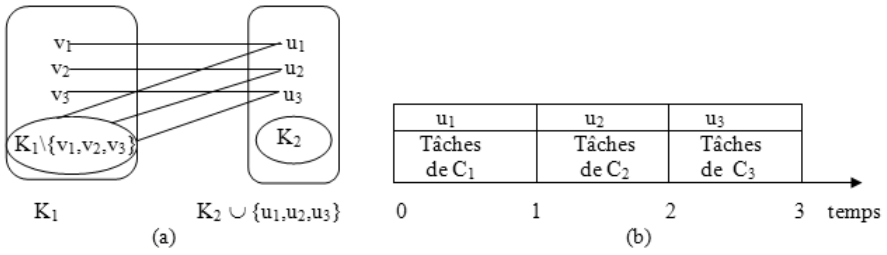

FIGURE 5.1 – (a) Le graphe de concordance G' (b) L'ordonnancement σ

5.2 Dates de disponibilités impaires

D'après le résultat précédent, on en déduit que le problème $P|AgreeG = (K_1, K_2; E), r_i, p_i = 1|C_{max}$ est NP-difficile, dans le cas des dates de disponibilité arbitraires. Par contre

dans le résultat suivant, nous montrons que le problème est polynomial si on se restreint aux dates de disponibilité arbitraires impaires. Notons que la condition $m \geq n$ veut dire que le problème est sans contrainte de machines, donc on a besoin de se concentrer seulement sur les dates de début de traitement des tâches. Ceci implique qu'il existe toujours des machines libres pour traiter les tâches. Considérons l'algorithme suivant dans lequel pour toute tâche J_i, t_i représente sa date de début et supposons que $K_1 = \{J_1, J_2, .., J_s\}$ et $K_2 = \{J_{s+1}, J_{s+2}, .., J_n\}$.

Algorithme 2
Début
Pour $i := 1$ à s
Faire - $t_i := r_i$
- Ordonnancer J_i sur Pi à l'instant t_i
Fait ;
Pour $k := s + 1$ à n
Faire Si toutes les tâches de K_1 ordonnancées à $t = r_k$ sont concordantes avec J_k
Alors - $t_k := r_k$
- Ordonnancer J_k sur Pk à l'instant t_k
FinSi
- $t_k := r_k + 1$
- Ordonnancer J_k sur Pk à l'instant t_k
Fait ;
Fin.

Théorème 5.2. *[4], [5] L'algorithme 2 résout le problème $P/AgreeG = (K_1, K_2; E)$, r_i impaires, $p_i = 1/C_{max}$ avec $m \geq n$, polynomialement en $O(n^2)$.*

Preuve. Soit τ l'ordonnancement obtenu par l'algorithme Algorithme 2 et soit $C^*_{\max}$ la valeur optimale du makespan. On peut vérifier que τ est réalisable, montrons qu'il est optimal. Soit $L = \max_{1 \leq i \leq n} \{t_i\}$, donc $C_{\max}(\tau) = L + 1$ et soit J_k une tâche vérifiant $L = t_k$.

Deux cas sont alors possibles : cas1 : L est pair. Puisque t_k est pair, par construction de l'algorithme $J_k \in K_2$. D'autre part J_k doit être été ordonnancée par l'instruction qui se trouve juste après le FinSi de la deuxième boucle "Pour" de l'algorithme et on a $t_k = r_k + 1$. Aussi, il doit exister une tâche $J_i \in K_1$ ordonnancée à la date t_i telle que $t_i = r_k$ et n'est pas en concordance avec J_k. Comme $J_i \in K_1$ donc $t_i = r_i$ et par suite $r_i = r_k$. Puisque les tâches J_i et J_k sont non concordantes et ayantt les mêmes dates de disponibilité, donc $C^*_{\max} \geq r_k + 2 = t_k + 1 = L + 1 = C_{\max}(\tau)$.
cas2 : L est impair. t_k est impair, alors J_k n'a pas été ordonnancée par l'instruction qui se

trouve juste après le FinSi de la deuxième boucle "Pour" et donc $t_k = r_k$. Ceci implique que $C^*_{\max} \geq r_k + 1 = L + 1 = C_{\max}(\tau)$, par conséquent l'ordonnancement τ est optimal.

Complexité de l'Algorithme 2 : la première boucle "Pour" est implémentée en $O(s)$ au pire des cas, la deuxième boucle "Pour" nécessite $s(n-s)$ itérations. Alors la complexité de cet algorithme est en $O(n^2)$. ■
En utilisant le même argument que celui du résultat précédent, on obtient un résultat analogue pour le cas où les dates de disponibilités sont arbitraires et paires.

Corollaire 5.1. *Le problème $P|AgreeG = (K_1, K_2; E), r_i$ paires, $p_i = 1|C_{max}$ avec $m \geq n$, est polynomial et peut être résolu polynomialement en $O(n^2)$.*

Exemple 5.1. *Considérons l'ordonnancement de six tâches $J_1, J_2, ..., J_6$ sur six machines $P1, P2, ..., P6$ tels que : le graphe de concordance est le complémentaire d'un graphe biparti $G = (K_1, K2; E)$ où $K_1 = \{J_1, J_2, J_3\}$ et $K_2 = \{J_4, J_5, J_6\}$. Les durées de traitements des tâches sont unitaires et les dates de disponibilités sont données par le tableau suivant :*

J_i	J_1	J_2	J_3	J_4	J_5	J_6
r_i	5	1	5	1	4	4

TABLE 5.1 – Dates de disponibilité de l'exemple 5.1

Le graphe de concordance pour l'exemple 5.1 est donné par la figure 5.2.

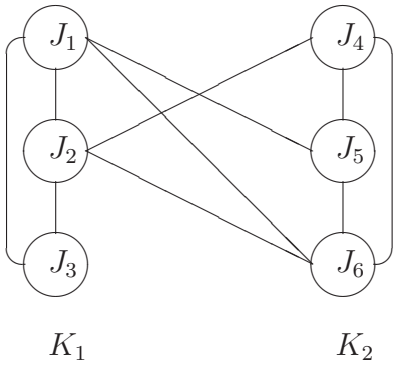

FIGURE 5.2 – Graphe de concordance de l'exemple 5.1

Application de l'Algorithme 2 à l'exemple 5.1

Dans la première boucle "Pour" on calcule les dates de début de traitement : $t_1 = 5$, $t_2 = 1$, $t_3 = 5$, des tâches de la clique K_1. Ensuite on ordonnance la tâche J_i sur la machine Pi à l'instant t_i pour tout i $(i = 1, 2, 3)$.
Ordonnancements des tâches de la clique K_2 :
$k = 4$, la tâche J_4 n'est pas en concordance avec toutes les tâches de la clique K_1, donc sa date de début de traitement est $t_4 = r_4 + 1 = 1 + 1 = 2$, puis on ordonnance J_4 sur $P4$ à l'instant t_4.
Dans la 2^{eme} boucle "Pour" $k = 5$, la tâche J_5 n'est pas en concordance avec toutes les tâches de la clique K_1, donc sa date de début de traitement est $t_5 = r_5 + 1 = 3 + 1 = 4$, puis on ordonnance J_5 sur $P5$ à l'instant t_5.
De même pour la tâche J_6, on trouve $t_6 = 4$, donc on ordonnance J_6 sur $P6$ à l'instant t_6.
L'ordonnancement final obtenu est donné par le diagramme de Gantt de la figure 5.3.
La valeur optimale du makespan est donc égale à 6.

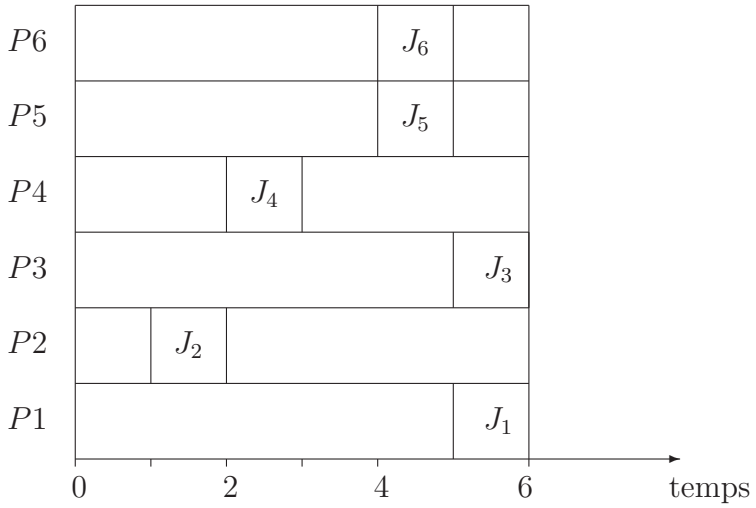

FIGURE 5.3 – L'ordonnancement obtenu de l'exemple 5.1

Chapitre 6

Graphe scindé

Dans ce chapitre, on suppose que le nombre de machines est égal à deux et que le graphe de concordance est un graphe scindé arbitraire. Un tel graphe est noté $G = (K; S, E)$, où K est une clique et S est un stable formant une partition de l'ensemble des sommets de G.

Dans le premier paragraphe, on montre que ce problème est NP-difficile lorsque les durées de traitements sont arbitraires et que toutes les tâches sont disponibles à l'instant 0. Nous présentons, dans le second paragraphe un sous problème NP-difficile et un sous problème polynomial dans le dernier paragraphe.

6.1 Durées de traitement arbitraires et dates de disponibilités nulles

Théorème 6.1. *[7] Le problème $P2|AgreeG = (K, S; E)|C_{max}$ est NP-difficile.*

Preuve. Rappelons le problème Deux-processeurs : Instance : n tâches $J_1, J_2, ..., J_n$ avec durées de traitement $p_1, p_2, ..., p_n$ respectivement et un nombre k. Question : existe -t-il un ordonnancement σ de ces tâches sur ces deux processeurs avec un makespan $C_{\max}(\sigma) \leq k$?
On fait une réduction de Deux Processeurs au problème de décision suivant (D) :
Instance : $n+1$ tâches : $J_1, J_2, ..., J_n$ avec durées de traitement $p_1, p_2, ..., p_n$ respectivement et une tâche J_{n+1} avec une durée de traitement $p_{n+1} = 1$, un nombre $k' = k+1$. Le graphe de concordance $G = (V, E)$ est tel que $V = \{J_1, J_2, ..., J_n, J_{n+1}\}$ avec $K = \{J_1, J_2, ..., J_n\}$ qui est une clique de G et la tâche J_{n+1} est un sommet isolé formant un stable de G, voir figure 6.1 (a).
Question : existe t-il un ordonnancement réalisable σ' avec un makespan $C_{\max}(\sigma') \leq k'$?
Supposons que Deux-Processeurs a une solution, donc il existe un ordonnancement σ des tâches $J_1, J_2, ..., J_n$ avec un makespan $C_{\max}(\sigma) \leq k$. En ordonnançant la tâche J_{n+1} à la date k sur la machine $P1$, on obtient un ordonnancement σ' avec un makespan

$C_{\max}(\sigma') \leq k'$? voir figure 6.1 (b).

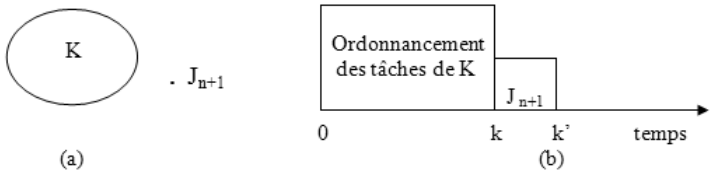

FIGURE 6.1 – (a) Le graphe de concordance $G = (V, E)$ (b) L'ordonnancement σ'

Inversement, supposons qu'il existe un ordonnancement σ' avec un makespan $C_{\max}(\sigma') \leq k'$. Puisque la tâche J_{n+1} n'est en concordance avec aucune tâche de l'ensemble $\{J_1, J_2, ..., J_n\}$, donc en translatant la tâche J_{n+1} à l'instant k, on obtient un nouveau ordonnancement réalisable σ" pour le problème (D). Cet ordonnancement induit un ordonnancement σ des tâches $J_1, J_2, ..., J_n$ avec $C_{\max}(\sigma) \leq k$, qui est une solution de Deux-Processeurs. Notons que par construction le graphe G est un graphe scindé. On peut vérifier que la réduction utilisée est polynomiale et que $(D) \in$ NP, ce qui achève la preuve du Théorème 6.1. ■

6.2 Cas $p_i \in \{1, 2\},\ r_i \in \{0, r\}$

Dans ce sous-paragraphe nous considérons le cas de deux machines, les durées de traitement appartiennent à {1,2} et que les dates de disponibilités peuvent prendre que deux valeurs possibles 0 et r. Il a été déjà prouvé par Even et al. [28] que dans le cas de deux machines et $p_i \in \{1, 2\}$, le problème est polynomial pour un graphe de concordance arbitraire. Dans le résultat suivant, nous montrons que ce dernier devient NP-difficile quand on ajoute la condition $r_i \in \{0, r\}$ et ceci même pour les graphes scindés.

Théorème 6.2. *[7] Le problème* $P2|AgreeG = (K, S; E),\ r_i \in \{0, r\}, p_i \in \{1, 2\}|C_{max}$ *est NP-difficile.*

Preuve. On montre que 3-Dimensional Matching Problem (3-DM) peut être réduit polynomialement au problème de décision suivant (P) :
Instance : l'ensemble des tâches est $V = V_M \cup V_Y \cup V_Z \cup V_D$ où les ensembles V_M, V_Y, V_Z et V_D sont définies comme dans le cas des graphes bipartis, considérés précédemment. Les ensembles V_{M_i} et le graphe de concordance G sont aussi définis de la même manière comme dans le cas $p_i \in \{1, 2, 3\}$ de sorte que les tâches de V_M forment une clique de G et que les tâches de $V_Y \cup V_Z \cup V_D$ forment un stable de G, rendant le graphe G un graphe scindé. Les durées de traitement des tâches de V_M et celles de V_D sont égales à 2, celles de V_Y et V_Z sont égales à 1. Les dates de disponibilité de G sont toutes nulles à l'exception des celles de V_D qui valent $2q$, voir figure 6.2.

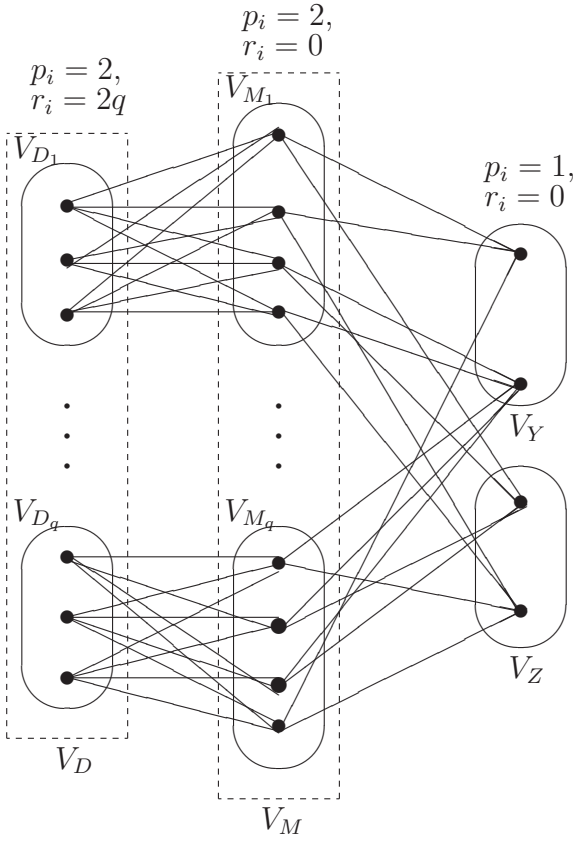

FIGURE 6.2 – Le graphe de concordance $G = (V, E)$

Question : existe t-il un ordonnancement réalisable σ de makespan $C_{\max}(\sigma) \leq 2\,|M|$? Supposons que 3-DM a une solution et considérons l'ordonnancement σ défini comme suit : soit $V_{M'} = \{J_1, J_2, .., J_q\}$: le sous-ensemble des tâches de V_M correspondant à M' tel que $J_i \in V_{M_i}$ $(i = 1, 2, ..., q)$. On ordonnance ces tâches J_i ainsi que leurs tâches correspondantes J_{iy} et J_{iz} des ensembles V_Y et V_Z respectivement, dans l'intervalle de temps $[0,\ 2q]$ comme indiqué sur la figure 6.3.

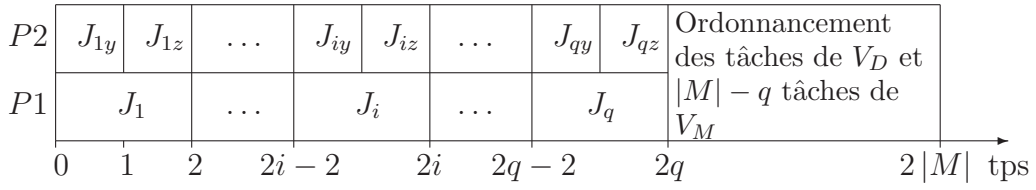

FIGURE 6.3 – L'ordonnancement σ

On ordonnance les autres $|M| - q$ tâches de V_M et les tâches de V_D dans l'intervalle de temps $[2q,\ 2|M|]$ en q groupes comme suit : pour tout i $(i = 1, 2, ..., q)$ on ordonnance les tâches de $V_{M_i} \backslash \{J_i\}$ sur la machine $P1$, ensuite opposé sur $P2$ on ordonnance les tâches de V_{D_i} (par exemple successivement), voir figure 6.3. L'ordonnancement σ qui vient d'être obtenu, est une solution du problème (P).

Réciproquement, supposons qu'il existe une solution pour le problème (P), donc il existe un ordonnancement réalisable σ tel que $C_{\max}(\sigma) \leq 2\,|M|$. Les tâches de V_D forment un stable de taille $|M| - q$ dans le graphe de concordance G. Ayant chacune une durée de traitement égale à 2 et une date de disponibilité de $2q$, elles devaient être ordonnancées dans l'intervalle de temps $[2q,\ 2|M|]$. Considérons le diagramme de Gantt représentant σ. La somme totale des durées de traitement des tâches de G est $4|M|$, ce qui implique l'espace de ce diagramme a été occupé entièrement. Puisque pour chaque i $(i = 1, 2, ..., q)$ les seules tâches adjacentes aux tâches de V_{D_i} sont celles de V_{M_i} et que les durées de traitement des tâches des deux ensembles V_D et V_M valent 2, alors $|M| - q$ tâches de V_M devaient être ordonnancées, parallèlement aux tâches de V_D dans l'intervalle de temps $[2q,\ 2|M|]$, voir figure 6.3. Par conséquent les tâches restantes sont q tâches de V_M, disant $J_1, J_2, .., J_q$ appartenant à $V_{M_1}, V_{M_2}, ..., V_{M_q}$ respectivement, et les $2q$ tâches de $V_Y \cup V_Z$. Les tâches de $V_Y \cup V_Z$ forment un stable de taille $2q$, donc chacune d'elles devait occuper un intervalle de longueur unité, couvrant l'intervalle de temps $[0, 2q]$. De plus, il n'existe aucune paire de tâches parmi $J_1, J_2, .., J_q$ ayant été ordonnancées en parallèle puisque autrement les tâches de $V_Y \cup V_Z$ n'auraient pu être ordonnancées. On en déduit que les tâches $J_1, J_2, .., J_q$ et celles de $V_Y \cup V_Z$ devaient être ordonnancées comme indiqué dans la figure 6.3. Ceci correspond à un sous-ensemble $M' \subseteq\ M$ avec q éléments produisant une solution de 3-DM. Il est facile de voir que la réduction utilisée est polynomiale et que (P) $\in$ NP, achevant donc la preuve du Théorème 6.2. ■

6.3 Durées de traitements unitaires

Dans ce paragraphe, on suppose nous supposons que le nombre de machines est égal à deux et que le graphe de concordance est un graphe scindé toujours, noté par $G = (K, S; E)$ tel que K est une clique et S est un stable formant une partition de l'ensemble des sommets de G. D'après le Théorème 6.1 ce problème est NP-difficile lorsque les durées de traitement sont arbitraires. Dans le résultat suivant, on montre que ce dernier devient polynomial quand les durées de traitement de la clique K sont égales à 1 et que celles de S sont arbitraires.

Supposons que $K = \{J_1, J_2, ..., J_{|K|}\}$ et $S = \{J_{|K|+1}, J_{|K|+2}, ..., J_{|K|+|S|}\}$. Pour toute tâche $J_{|K|+i}$ de S, on note sa durée de traitement par p_i.
Nous considérons l'algorithme suivant dont le principe est : au début, on ordonnance les tâches de S successivement sur $P1$, puis on ordonnance sur $P2$, le maximum de tâches unitaires (i.e. les tâches de K) en parallèle avec celles de S. Les tâches restantes de K sont ordonnancées successivement et alternativement sur les deux machines, après les tâches de S.

Algorithme 1
Début

1. Construire le réseau $R = (V, U, c)$ comme suit : $V = \{s, p\} \cup K \cup S$,
   $U = U_1 \cup U_2 \cup U_3 \cup \{u_r\}$ où $u_r = (p, s)$ est un arc de retour, $U_1 = \{(s, J_i) : J_i \in K\}$,
   $U_2 = \{(J_i, J_j) \in K \times S : \{J_i, J_j\} \in E\}, U_3 = \{(J_j, p) : J_j \in S\}$
2. Construire le vecteur arc capacité c comme suit :
   si $(s, J_i) \in U_1 \Longrightarrow c(s, J_i) = 1$, si $(J_i, J_j) \in U_2 \Longrightarrow c(J_i, J_j) = 1$,
   $c(u_r) = +\infty$, if $(J_{|K|+i}, p) \in U_3 \Longrightarrow c(J_{|K|+i}, p) = p_i$.
3. Déterminer un flot réalisable maximum f^* de s à p, dans le réseau R.
4. Ordonnancer les tâches de S, successivement dans l'intervalle de temps $[0, \sum_{i=1}^{|S|} p_i]$, sur $P1$.
5. Ordonnancer les tâches J_i de K tel que $f^*(J_i, J_{|K|+1}) = 1$ successivement et parallèlement à la tâche $J_{|K|+1}$ dans l'intervalle $[0, p_1]$ sur $P2$ et les tâches J_i de K tel que $f^*(J_i, J_{|K|+2}) = 1$ successivement et parallèlement à la tâche $J_{|K|+2}$ dans l'intervalle $[p_1, p_1 + p_2]$ sur $P2$ et ainsi de suite.
6. Ordonnancer les tâches restantes de K, successivement et alternativement sur les machines, dans l'intervalle de temps $[\sum_{i=1}^{|S|} p_i, \sum_{i=1}^{|S|} p_i + \lceil \frac{|K| - f^*(u_r)}{2} \rceil]$.

Fin.

On donne la forme du réseau $R = (V, U, c)$ dans la figure 6.4 dans le cas $|K| = |S| = 3$.

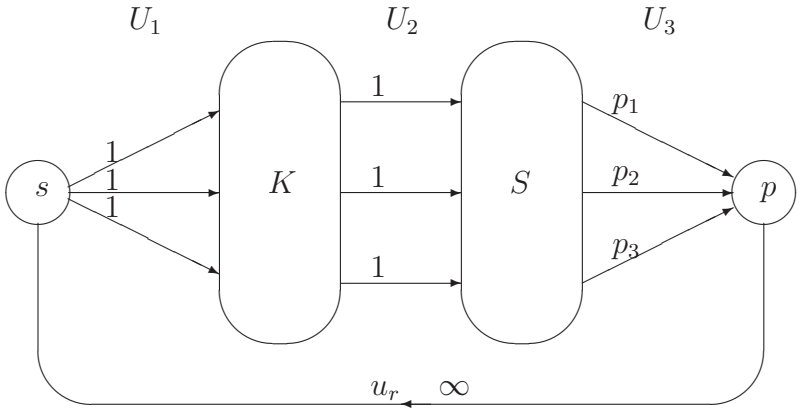

FIGURE 6.4 – Le réseau $R = (V, U, c)$

Remarque 6.1. *1- $f(u)$ est le flux sur l'arc u.*
2- Dans le réseau R on n'a pas tenu compte des arêtes de la clique K.

Théorème 6.3. *[7] Le problème $P2|AgreeG = (K, S; E), p_K = 1|C_{max}$ est polynomial. Algorithme 1 le résout en $O(n^3)$ si on utilise dans l'étape 3, l'algorithme de J. Cheriyan et S.N. Maheshwari [22] pour obtenir un flot maximum.*

Preuve. Il est clair que l'Algorithme 1 retourne un ordonnancement réalisable σ^* pour le problème en main avec makespan $C_{\max}(\sigma^*) = \sum_{i=1}^{|S|} p_i + \lceil \frac{|K|-f^*(u_r)}{2} \rceil$ comme représenté par le diagramme de Gantt de la figure 6.5, où $\alpha^* = \sum_{i=1}^{|S|} p_i + \lceil \frac{|K|-f^*(u_r)}{2} \rceil$. Montrons maintenant que tout ordonnancement réalisable σ vérifie $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Soit σ un ordonnancement réalisable arbitraire représenté par un diagramme de Gantt. On décale les tâches de S ainsi que leurs tâches opposées de K correspondantes au côté gauche le plus extrême de ce diagramme de sorte que toutes ces tâches seront placées dans l'intervalle de temps $[0, \sum_{i=1}^{|S|} p_i]$. On peut s'arranger de sorte que les tâches de S seront placées sur $P1$, leurs tâches correspondantes opposées de K sur $P2$. Ensuite en plaçant le reste des tâches de K successivement et alternativement sur les machines, après l'instant $\sum_{i=1}^{|S|} p_i$, on obtient un ordonnancement réalisable σ_1 du problème. On peut imaginer σ_1 comme l'ordonnancement représenté par un diagramme de Gantt similaire à celui correspondant à σ^* de la figure 6.5. L'ordonnancement σ_1 induit un flot f_1 sur le réseau R tel que $C_{\max}(\sigma_1) = \sum_{i=1}^{|S|} p_i + \lceil \frac{|K|-f_1(u_r)}{2} \rceil$. Noter que dans l'ordonnancement σ_1, il n'existe aucun temps mort commun aux deux machines et donc $C_{\max}(\sigma) \geq C_{\max}(\sigma_1)$. Puisque f^* est un flot maximum dans R, il s'en suit que $f_1(u_r) \leq f^*(u_r)$. D'autre part $C_{\max}(\sigma^*) = \sum_{i=1}^{|S|} p_i + \lceil \frac{|K|-f^*(u_r)}{2} \rceil$, donc $C_{\max}(\sigma) \geq C_{\max}(\sigma^*)$. Comme σ est arbitraire, alors σ^* est optimal.

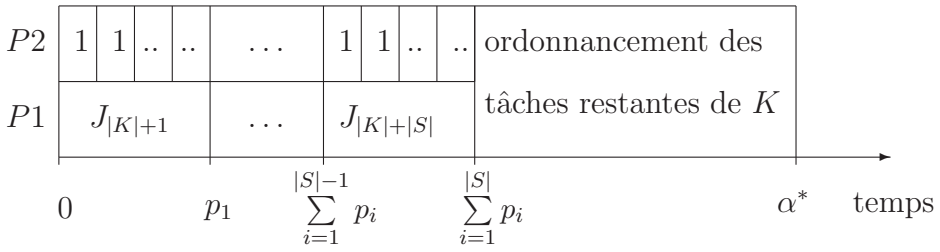

FIGURE 6.5 – Le diagramme de Gantt représentant σ^*

Complexité de l'algorithme Algorithme 1 : la construction de R peut être implémentée dans le pire des cas en $O(|K|\,|S|) = O(n^2)$. La détermination d'un flot maximum dans le réseau R est polynomial, donc le problème $P2|AgreeG = (K, S; E), p_K = 1|C_{max}$ est polynomial. En utilisant l'algorithme de J. Cheriyan et S.N. Maheshwari, l'étape 3 peut être implémentée faite en $O(n^2\sqrt{|U|})$. Comme $|U| = O(|K|\,|S|) = O(n^2)$, donc l'étape 3 peut être exécutée en $O(n^3)$. On en déduit que la complexité de l'Algorithme 1 est en $O(n^3)$. Ceci termine la preuve du Théorème 6.3. ■

Exemple 6.1. *Considérons l'ordonnancement de six tâches $J_1, J_2, ..., J_7$ telles que : le graphe de concordance est $G = (S_1, S2; E)$ où $K = \{J_1, J_2, J_3, J_7\}$, $S = \{J_4, J_5, J_6\}$ et est représenté dans la figure 6.6.*

J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7
p_i	1	1	1	2	3	4	1

TABLE 6.1 – Durées de traitement de l'exemple 6.1

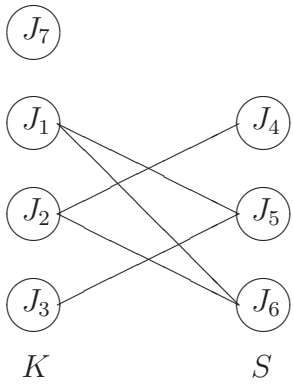

FIGURE 6.6 – Le graphe de concordance de l'exemple 6.1

Application de l'algorithme 1 : Le réseau correspondant est donné par la figure suivante :

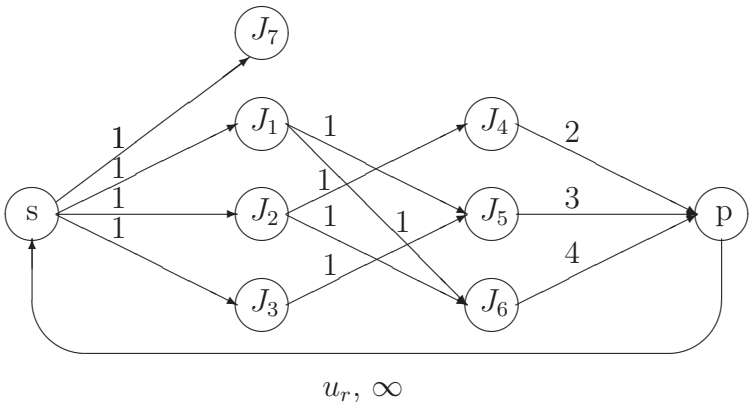

FIGURE 6.7 – Le réseau $R = (V, E, c)$ de l'exemple 6.1

Aprés application de l'algorithme de Ford et Fulkerson, nous avons obtenu le flot maximum représenté par la figure 6.8. Les valeurs portées sur les arcs représentent des flux c-à-d les valeurs du flot sur les arcs.

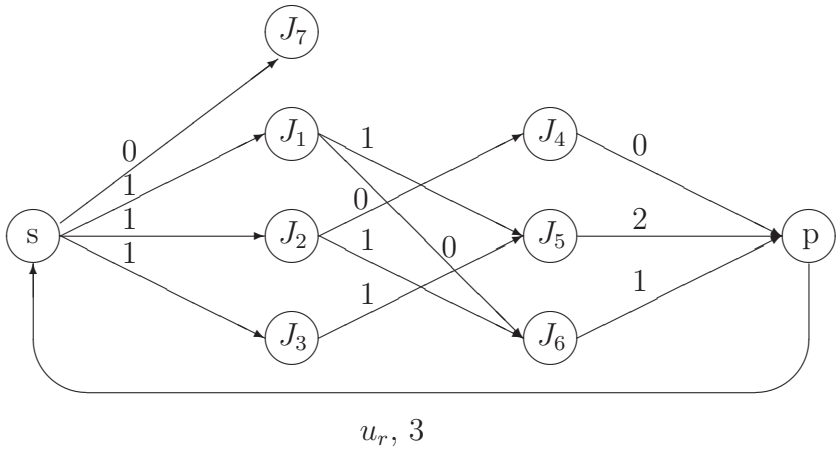

FIGURE 6.8 – Représentation du flot maximum de l'exemple 6.1

L'ordonnancement optimal obtenu par application de l'Algorithme 1 est représenté par le diagramme de Gantt de la figure 6.9, dont la valeur du makespan est $C^*_{max} = 10$.

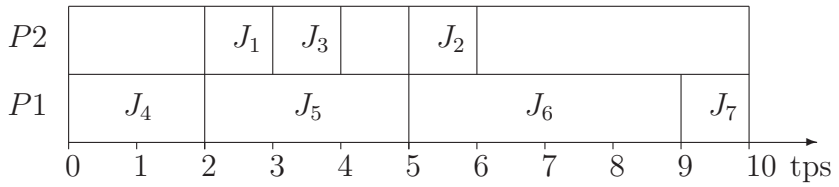

FIGURE 6.9 – L'ordonnancement obtenu de l'exemple 6.1

Chapitre 7

Heuristiques et expérimentations numériques

Dans ce chapitre, nous proposons des heuristiques de type algorithmes de liste pour le problème avec dates de disponibilités nulles, qui est le problème : $P|AgreeG = (V, E)|C_{max}$. Toutes ces heuristiques sont générées par un algorithme de liste, basé sur l'approche utilisée par R.L. Graham [37] pour le problème $P||C_{\max}$. D'abord, nous donnons quelques bornes inférieures de la valeur optimale du makespan, pour ce problème.

7.1 Bornes inférieures

Quand le graphe de concordance G est complet, le problème ci-dessus est équivalent au problème $P||C_{\max}$. Ce cas correspond à l'absence des contraintes de concordance, donc la première borne inferieure de la valeur optimale du makespan pour le problème $P|AgreeG = (V, E)|C_{max}$ est la borne classique du problème $P||C_{\max}$ qui est :
$LB_0 = \max\{\lceil(\sum_{j=1}^{n} p_j)/m\rceil, \max_{1\leq j\leq n}\{p_j\}\}$.

Bornes basées sur le graphe de concordance : Rappelons d'abord le problème du stable de poids maximum (Maximum Weighted Independent Set, M.W.I.S). Un graphe $G = (V, E)$ est dit valué si à chaque sommet est associé un poids non négative $w(v)$, un tel graphe est noté (G, w). Le problème M.W.I.S est défini par : étant donné un graphe valué (G, w), le problème consiste à trouver un stable de poids maximum dans (G, w). Quand tous les poids sont tous égaux à 1, le problème est appelé problème du stable maximum (Maximum Independent Set problem, M.I.S en abrégé) qui est NP-difficile (Lovasz [46]) et par conséquent le problème M.W.I.S l'est aussi. Le poids d'un stable de poids maximum dans un graphe valué (G, w) est noté $ind(G, w)$.

Considérons le problème avec dates de disponibilités nulles. Etant donné une instance

de ce problème avec un graphe de concordance $G = (V, E)$ et p le vecteur représentant les durées de traitements des tâches de cette instance. Si on regarde ces durées de traitement comme étant les poids pour le graphe $G = (V, E)$, on obtient le graphe valué (G, p).

Puisque les tâches non adjacentes doivent être ordonnancées dans des intervalles de temps disjoints, pour tout ordonnancement réalisable, alors toute borne inférieure pour $ind(G, p)$ est aussi une borne inférieure pour $Opt(G)$. Basé sur ce fait, on donnera trois bornes inférieures pour $Opt(G)$.

Les auteurs Sakai et al. de la référence [47]) ont étudié le problème M.W.I.S et ont présenté trois algorithmes gloutons pour ce problème, notamment GWMIN, GWMAX et GWMIN2. Pour un graphe valué donné (G, w), ces algorithmes retournent chacun, un stable de (G, w).

Maintenant, considérons le problème $P|AgreeG = (V, E)|C_{max}$ dans lequel les durées de traitement sont représentées par le vecteur p. En appliquant les trois algorithmes cités précédemment pour le graphe valué (G, p), on obtient trois stables de (G, p) dont les poids sont trois bornes inférieures, disant LB_1, LB_2 et LB_3 respectivement pour la valeur optimale du makespan $Opt(G)$.

La borne inferieure globale LB : D'après les implémentations, on a constaté que la borne inferieure LB_2 est faible par rapport aux deux autres bornes, par conséquent nous avons considéré seulement les bornes LB_1 et LB_3. En combinant avec la borne LB_0, on obtient une borne inferieure globale pour $Opt(G)$ donnée par :

$LB = Max\{LB_0, LB_1, LB_3\}$.

7.2 Les heuristiques proposées

Nous présentons d'abord l'algorithme suivant qui est un algorithme de liste, appelé LS-Algorithme à partir duquel on dérive toutes les heuristiques proposées.
Soit V l'ensemble des tâches et notons par U l'ensemble des tâches non ordonnancées. Si J_j est une tâche non ordonnancée, on dira que J_j est disponible à un instant t s'il existe une machine libre à cet instant et si les tâches déjà ordonnancées et ayant une partie de leur traitement dans l'intervalle de temps $[t, t + p_j]$, sont toutes concordantes avec J_j.

Algorithme LS-Algorithme
Début
Choisir une liste de priorité des tâches ;
$U := V$;
$t := 0$;
Tantque $U \neq \emptyset$
Faire
Soit $\overline{U}$ l'ensemble des tâches disponibles à l'instant t
Si $(\overline{U} \neq \emptyset)$ **alors**
Choisir une tâche non ordonnancée J_j de plus grande priorité et l'ordonnancer, sur une machine, à l'instant t ;
$U := U \setminus \{J_j\}$;
Sinon
Déterminer le plus petit instant t où une tâche de U devient disponible ;
finSi
Fait ;
Fin.

Pour déterminer le plus petit instant t où une tâche de U devient disponible, on calcule dans l'ordre croissant les dates de fin de traitement des machines.

Définition 7.1. *Le nombre de concordance d'une tâche J_j est le nombre de tâches concordantes avec J_j.*

Nous avons considéré neuf listes de priorité qui sont définies comme suit. Si les tâches sont rangées selon l'ordre croissant (respectivement décroissant) des nombres de concordance, les listes obtenues sont appelées $Liste1$ et $Liste2$ respectivement. $Liste3$ est définie par la procédure suivante :

Procédure Liste 3 ;
Début
1- J_1 est une tâche ayant le plus grand nombre de concordance ;
2- J_j $(j = 2, ..., n)$ est une tâche ayant le plus grand nombre de concordance dans l'ensemble des tâches $V \setminus \{J_1, ..., J_{j-1}\}$;
Fin.

Si dans la Procédure Liste3, on choisit une tâche J_j avec le plus petit nombre de concordance, on obtient $Liste4$. $Liste5$ est une permutation aléatoire des nombres $1, 2, 3, ..., n$. Soit maintenant la procédure produisant $Liste6$.

Procédure Liste 6 ;
Début
1- J_1 est une tâche ayant le plus grand nombre de concordance ;
2- J_j $(j = 2, ..., n)$ est une tâche ayant le plus grand nombre de concordance avec les tâches de l'ensemble $\{J_1, ..., J_{j-1}\}$;
Fin.

*Liste*7 est obtenue de la procédure Liste6 en remplaçant le terme plus grand par plus petit. Dans la *Liste*8 les tâches sont rangées selon la règle Shortest Processing Time (SPT). Finalement *Liste*9 est basée sur le rangement des tâches selon la règle Longest Processing Time (LPT).

Chaque liste de priorité des tâches choisie au début de l'algorithme LS-Algorithme génère une heuristique pour le problème $P|AgreeG = (V, E)|C_{max}$.

Comme nous avons neuf listes notamment *Liste*1, *Liste*2, ..., t *Liste*9, on obtient alors neuf heuristiques appelées $H1, H2, ...H9$ respectivement.

Donnons une illustration de l'application de l'algorithme LS-Algorithme avec l'exemple suivant :

Exemple 7.1. *Considérons le cas* $m = 2$, $n = 5$ *avec le graphe de concordance et les durées de traitement des tâches données comme suit :*

J_i	J_1	J_2	J_3	J_4	J_5
p_i	2	1	2	4	3

TABLE 7.1 – Durées de traitement de l'exemple 7.1

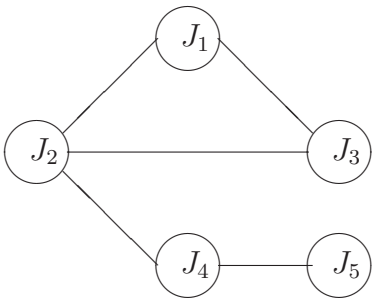

FIGURE 7.1 – Graphe de concordance de l'exemple 7.1

On a donc $V = \{J_1, J_2, J_3, J_4, J_5\}$. Choisissons par exemple la liste $L = \{J_4, J_3, J_2, J_1, J_5\}$

Première itération :

$U = \{J_1, J_2, J_3, J_4, J_5\}$
$t = 0$
$U \neq \emptyset$; $\overline{U} = U$ car toutes les tâches sont disponibles au début
$\overline{U} \neq \emptyset$ on choisit la tâche J_4 car c'est la tâche
de plus grande priorité et on l'ordonnance sur la première machine libre, qui est $P1$ voir Diagramme1
Le nouveau ensemble U devient : $U = \{J_1, J_2, J_3, J_5\}$; $U \neq \emptyset$ donc on passe à la deuxième itération.

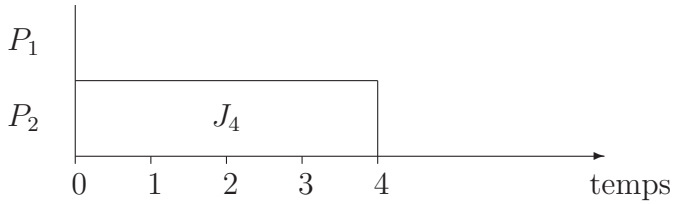

FIGURE 7.2 – Diagramme 1

Deuxième itération :

L'ensemble des tâches non ordonnancées $U = \{J_1, J_2, J_3, J_5\}$
Cherchons l'ensemble $\overline{U}$ des tâches disponibles à $t = 0$:
On ne peut pas affecter J_1 à la machine $P2$ à l'instant $t = 0$, car elle n'est pas adjacente avec la tâche J_4, par conséquent J_1 n'est pas disponible à $t = 0$.
J_2 est disponible à $t = 0$; J_3 Non ; J_5 oui, donc $\overline{U} = \{J_1, J_5\}$
$\overline{U} \neq \emptyset$, J_2 est la tâche de plus grande priorité, on la choisit
On ordonnance J_2 à $t = 0$ sur la première machine libre , qui est $P2$, voir Diagramme 2
U devient $U = \{J_1, J_3, J_5\}$;$U \neq \emptyset$ on passe donc à la troisième itération

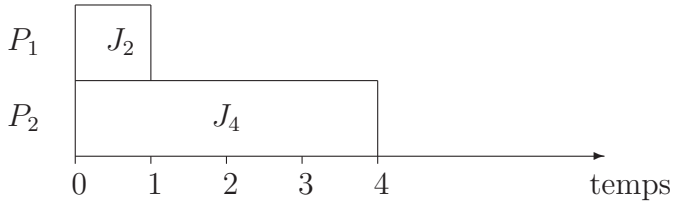

FIGURE 7.3 – Diagramme 2

Troisième itération :

L'ensemble des tâches non ordonnancées $U = \{J_1, J_3, J_5\}$
Cherchons l'ensemble $\overline{U}$ des tâches disponibles à $t = 0$:
J_1 n'est pas disponible à $t = 0$ car J_1 et J_4 ne sont pas adjacentes
J_3 n'est pas disponible à $t = 0$ car J_3 et J_4 ne sont pas adjacentes
J_5 n'est pas disponible à $t = 0$ car J_5 et J_2 ne sont pas adjacentes
aucune tâche n'est disponible à $t = 0$, donc $\overline{U} = \emptyset$, on passe à l'étape 9 de LS-Algorithme
On détermine le plus petit instant t où une tâche de U devient disponible
La machine $P2$ est libre à l'instant $t = 1$, qui est le 1er instant remarquable
existe t-il des tâches disponibles à cet instant ?
Les tâches J_1 et J_3 ne sont pas disponibles, tandis que la tâche J_5 l'est.
Donc "le 1er instant où une tâche de U devient disponible" est 1 , on fait donc
$t = 1$ et on passe à l'itération suivante

Quatrième itération :

On a $t = 1$,$U \neq \emptyset$, on cherche $\overline{U}$ à $t = 1$, on trouve $\overline{U} = \{J_5\}$
$\overline{U} \neq \emptyset$, comme $\overline{U}$ comporte uniquement J_5, on ordonnance
alors J_5 sur la 1ère machine libre, soit $P2$, voir Diagramme 3.
L'ensemble des tâches non ordonnancées devient $U = \{J_1, J_3\}$
$\overline{U} \neq \emptyset$, cherchons l'ensemble $\overline{U}$ des tâches disponibles à $t = 1$:
J_1 n'est pas disponible à $t = 1$ car J_1 et J_4 ne sont pas adjacentes
J_3 n'est pas disponible à $t = 1$ car J_3 et J_4 ne sont pas adjacentes
donc $\overline{U} = \emptyset$, on passe à l'étape 9 de l'algorithme LS-Algorithme
le 1er instant remarquable est $t = 4$, existe t-il des tâches disponibles à cet instant ? oui, il en existe deux qui sont taches J_1 et J_3
donc le 1er instant où une tâche du U "actuel" est disponible est l'instant 4 , on fait donc
$t = 4$ et on passe à l'itération suivante

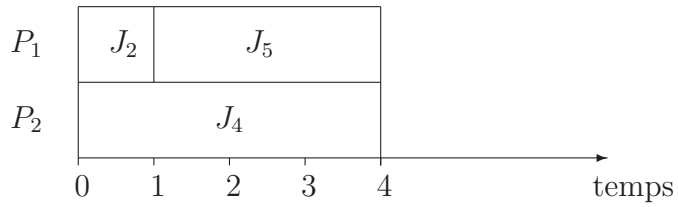

FIGURE 7.4 – Diagramme 3

Cinquième itération :

à $t = 4$ on a $\overline{U} = \{J_1, J_3\}$

$\overline{U} \neq \emptyset$, on choisit J_3 qui a la plus grande priorité et on l'ordonnance sur $P1$, qui est le 1er processeur libre, voir Diagramme 4
le nouveau U est $U = \{J_1\}$ et on passe à l'itération suivante.

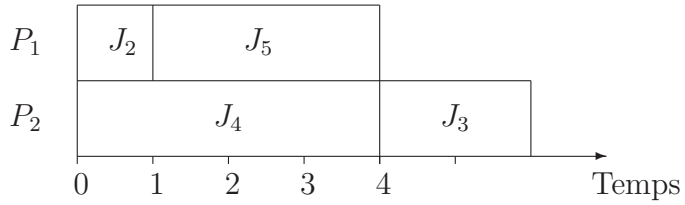

FIGURE 7.5 – Diagramme 4

Sixième itération :

comme J_1 est adjacente avec J_3 donc à $t = 4$ on a $\overline{U} = \{J_1\}$
comme c'est la seule, on la choisit et on l'ordonnance sur la machine $P2$ à $t = 4$, voir Diagramme 5

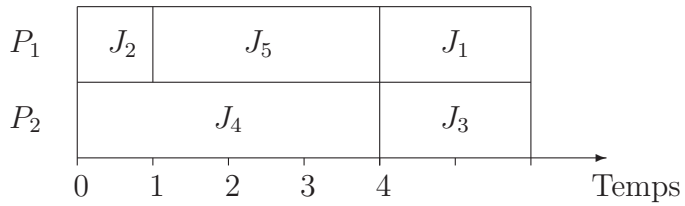

FIGURE 7.6 – Diagramme 5

Remarquons que cet ordonnancement obtenu est optimal.

7.3 Expérimentations numériques

Toutes les heuristiques ont été implémentées sur un Pentium IV PC 3.4 GHZ et 2 GB Ram avec le logiciel Matlab 7.0. Autres que les paramètres m et n, nous avons utilisé le paramètre d qui représente la densité (en pourcentage) du graphe de concordance. Toutes les instances ont été générées aléatoirement selon la loi uniforme et sont classées en quatre classes : m2Classe, m5Classe, m10Classe and m20Classe correspondant à $m = 2$, $m = 5$, $m = 10$ et $m = 20$ respectivement. m2Classe et m5Classe sont chacune associée avec sept valeurs de n : $n \in \{10, 20, 50, 100, 250, 500, 1000\}$. m10Classe correspond à sept valeurs de n : $n \in \{20, 50, 100, 250, 350, 500, 1000\}$ et m20Classe est associée avec sept valeurs de n : $n \in \{30, 50, 100, 250, 350, 500, 1000\}$. Les instances de chaque classe sont construites comme suit : pour chaque paire $\{m, n\}$ (n différent de 1000), nous avons généré trois ensembles de 100 instances chacun correspondant à densité faible($d = 10$), densité moyenne

($d = 50$) et densité forte ($d = 90$) respectivement. Chacun de ces ensembles est composé de quatre sous-ensembles de 25 instances chacun, tels que 25 instances avec $p_i \in \{1, 2, ..., 10\}$, 25 instances avec $p_i \in \{1, 2, ..., 20\}$, 25 instances avec $p_i \in \{1, 2, ..., 50\}$ et 25 instances avec $p_i \in \{50, 51, ..., 100\}$. Pour chaque paire $\{m, n = 1000\}$, on a fait la même chose, sauf que nous avons généré trois ensembles de 40 instances (au lieu de 100 instances), chacun se composant de 4 sous-ensembles de 10 instances (au lieu de 25 instances). Le nombre d'instances de chaque classe est égal à $(6 \times 100 \times 3) + (3 \times 40) = 1920$. Par conséquent le nombre total des instances générées est égal à $4 \times 1920 = 7680$. Pour toute heuristique H, nous avons considéré sa déviation de la borne inférieure globale LB donnée dans le sous-chapitre précédent, qui est égale à $(C_{max}(H) - LB)/LB$, où $C_{max}(H)$ est le makespan obtenu par H.

Observations et analyse : Après une vaste comparaison expérimentale de toutes les neuf heuristiques, la première observation est que les quatre heuristiques $H1$, $H4$, $H7$ et $H9$ sont significativement les meilleures, en terme du meilleur makespan trouvé. Alors, nous avons comparé seulement ces quatre heuristiques. Les résultats expérimentaux obtenus pour les quatre classes m2Classe, m5Classe, m10Classe et m20Classe d'instances sont résumés dans les tableaux : tables 7.2, 7.3, 7.4 et 7.5 respectivement.

Notons que dans ces tableaux, pour commodité les résultats correspondant à l'heuristique $H7$ ont été supprimés à cause des mauvaises performances qu'elle a donné, comparée avec les trois autres heuristiques. Dans chacun de ces tableaux, la première ligne BestCmax représente le nombre de fois (en pourcentage) pour lequel l'heuristique correspondante donne le meilleur makespan. Les lignes MD (Maximum Deviation) et AD (Average Deviation) représentent les déviations maximale et moyenne de la borne inferieure LB, respectivement. La dernière ligne AT (Average Time) représente le temps CPU moyen en secondes pour chaque heuristique. La table 7.6 reporte les performances globales de chacune des trois heuristiques pour les différentes classes d'instances. La table 7.7 représente les performances globales de chacune des heuristiques sur l'ensemble de toutes les instances.

Les observations principales qui peuvent être tirées sont comme suit. Quand le critère d'évaluation est celui du meilleur makespan trouvé, globalement $H1$ est meilleure pour les instances à densité faible tandis que $H9$ est meilleure dans les instances à forte et moyenne densité.

Quelques exceptions peuvent être constatées : $H4$ est plus performante dans les cas $m = 2$ avec une densité faible et $m = 5$ avec une densité moyenne. L'heuristique $H9$ a donné de bonnes solutions pour $n = 1000$ avec densité faible. Par rapport à la déviation maximale, globalement l'heuristique $H9$ est la plus performante pour les instances ayant un graphe

de concordance moins dense, par contre que $H1$ est légèrement meilleure dans les cas des densité moyenne et forte. Si la déviation moyenne ou le temps moyen est considéré comme un critère d'évaluation, globalement l'heuristique $H1$ est légèrement meilleure, quelque soit la densité.

TABLE 7.2 – Résultats expérimentaux pour $m = 2$

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
n=10									
BestCmax	98	97	94	50	52	34	22	22	75
MD	0.141	0.141	0.133	0.302	0.250	0.361	0.174	0.160	0.123
AD	0.002	0.002	0.004	0.069	0.070	0.110	0.051	0.052	0.016
AT	0.007	0.006	0.006	0.002	0.004	0.003	0.002	0.003	0.006
n=20									
BestCmax	72	67	35	33	34	33	21	20	79
MD	0.163	0.141	0.158	0.110	0.086	0.144	0.096	0.079	0.065
AD	0.026	0.0026	0.047	0.033	0.027	0.036	0.030	0.029	0.005
AT	0.019	0.021	0.019	0.016	0.016	0.016	0.017	0.017	0.016
n=50									
BestCmax	52	31	6	29	23	52	18	9	97
MD	0.191	0.183	0.287	0.041	0.033	0.050	0.036	0.033	0.027
AD	0.087	0.094	0.142	0.012	0.011	0.008	0.010	0.013	0.001
AT	0.099	0.100	0.100	0.096	0.098	0.095	0.103	0.103	0.102
n=100									
BestCmax	41.000	54	4	10	24	79	18	18	97
MD	0.092	0.073	0.111	0.019	0.020	0.027	0.018	0.015	0.014
AD	0.025	0.022	0.061	0.007	0.006	0.003	0.005	0.005	0.000
AT	0.391	0.391	0.401	0.411	0.413	0.415	0.439	0.439	0.440
n=250									
BestCmax	29	75	0	18	16	82	12	7	99
MD	0.015	0.007	0.044	0.007	0.006	0.006	0.007	0.007	0.005
AD	0.004	0.002	0.014	0.002	0.002	0.001	0.002	0.002	0.000
AT	3.028	3.028	3.042	3.209	3.212	3.249	3.429	3.429	3.442
n=500									
BestCmax	27	66	9	15	13	89	8	8	100
MD	0.006	0.003	0.019	0.004	0.004	0.005	0.004	0.004	0.000
AD	0.002	0.001	0.004	0.001	0.001	0.000	0.001	0.001	0.000
AT	16.239	16.245	16.265	17.309	17.313	17.556	18.385	18.385	18.455
n=1000									
BestCmax	22.5	42.5	40	12.5	15	97.5	10	12.5	100
MD	0.003	0.001	0.008	0.002	0.002	0.001	0.002	0.002	0.000
AD	0.001	0.001	0.002	0.001	0.001	0.000	0.001	0.001	0.000
AT	101.910	101.390	102.570	104.700	104.700	106.660	110.960	110.930	111.420

TABLE 7.3 – Résultats expérimentaux pour $m = 5$

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
n=10									
BestCmax	100	99	90	86	78	69	90	77	67
MD	0.009	0.025	0.171	0.326	0.326	0.348	0.364	0.364	0.433
AD	0.000	0.000	0.006	0.033	0.039	0.052	0.042	0.058	0.096
AT	0.006	0.006	0.006	0.005	0.006	0.006	0.002	0.005	0.005
n=20									
BestCmax	74	63	42	53	47	22	30	27	48
MD	0.140	0.140	0.183	0.511	0.617	0.553	0.337	0.415	0.342
AD	0.018	0.022	0.037	0.156	0.165	0.203	0.144	0.146	0.131
AT	0.024	0.023	0.023	0.021	0.021	0.021	0.016	0.016	0.017
n=50									
BestCmax	45	24	8	42	29	14	14	12	75
MD	0.283	0.289	0.304	0.566	0.581	0.598	0.143	0.130	0.155
AD	0.095	0.107	0.140	0.357	0.382	0.397	0.060	0.055	0.029
AT	0.143	0.142	0.142	0.111	0.113	0.113	0.099	0.100	0.099
n=100									
BestCmax	46	32	6	51	10	12	10	10	81
MD	0.515	0.511	0.605	0.344	0.395	0.326	0.067	0.059	0.068
AD	0.289	0.296	0.333	0.188	0.213	0.224	0.028	0.027	0.010
AT	0.610	0.612	0.614	0.431	0.433	0.451	0.431	0.431	0.431
n=250									
BestCmax	44	30.000	19	40	54	2	6	9	84
MD	0.997	1.012	1.077	0.089	0.090	0.109	0.023	0.023	0.027
AD	0.780	0.790	0.805	0.041	0.037	0.069	0.012	0.010	0.003
AT	4.820	4.819	4.804	3.125	3.124	3.205	3.318	3.318	3.343
n=500									
BestCmax	55	21	19	30	59	3	1	2	94
MD	0.852	0.870	0.860	0.027	0.017	0.050	0.013	0.014	0.007
AD	0.787	0.796	0.801	0.009	0.006	0.021	0.006	0.006	0.000
AT	25.702	25.779	25.511	16.698	16.706	16.783	17.883	17.878	18.049
n=1000									
BestCmax	37	27.5	37.5	10	40	50	2.5	0	95
MD	0.669	0.665	0.644	0.009	0.005	0.023	0.006	0.006	0.003
AD	0.613	0.616	0.617	0.003	0.002	0.006	0.003	0.003	0.000
AT	154.290	154.360	152.860	100.710	100.770	100.590	110.240	110.540	111.770

TABLE 7.4 – Résultats expérimentaux pour $m = 10$

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
n=20									
BestCmax	75	68	48	44	39	37	84	76	49
MD	0.099	0.099	0.163	0.776	0.776	0.613	0.571	0.571	0.720
AD	0.015	0.018	0.032	0.178	0.188	0.191	0.090	0.102	0.155
AT	0.024	0.024	0.024	0.021	0.022	0.022	0.018	0.017	0.019
n=50									
BestCmax	40	21	17	31	24	24	38	30	12
MD	0.285	0.294	0.313	1.041	1.025	1.109	0.509	0.522	0.509
AD	0.101	0.111	0.138	0.507	0.513	0.527	0.191	0.227	0.277
AT	0.143	0.142	0.144	0.138	0.138	0.138	0.098	0.096	0.100
n=100									
BestCmax	47	28	8	32	24	19	44	42	20
MD	0.532	0.497	0.536	1.260	1.386	1.298	0.263	0.224	0.229
AD	0.298	0.306	0.339	0.915	0.934	0.948	0.083	0.082	0.103
AT	0.612	0.614	0.618	0.588	0.590	0.587	0.414	0.413	0.418
n=250									
BestCmax	42	37	14	28	17	43	14	20	62
MD	1.085	1.126	1.190	0.936	0.929	0.926	0.120	0.048	0.086
AD	0.785	0.791	0.811	1.0.786	0.797	00.771	0.030	0.027	0.021
AT	4.868	4.873	4.896	4.195	4.218	4.274	3.248	3.250	3.245
n=350									
BestCmax	47	27	19	20.000	19	47	5.000	14	75
MD	1.439	1.512	1.544	0.829	0.786	0.746	0.069	0.033	0.043
AD	1.067	1.076	1.087	0.665	0.663	0.652	0.023	0.019	0.011
AT	10.873	10.885	10.967	8.974	8.986	9.344	7.243	7.264	7.242
n=500									
BestCmax	41	32	21	28	17	46	10	12	75
MD	1.926	1.956	2.064	0.642	0.639	0.595	0.027	0.028	0.040
AD	1.453	1.459	1.468	0.550	0.557	0.541	0.014	0.014	0.006
AT	26.603	26.602	26.825	20.678	20.750	21.981	17.526	17.518	17.549
n=1000									
BestCmax	22.5	30	42.5	27.5	20	47.5	7.5	12.5	75
MD	2.285	2.334	2.251	0.419	0.402	0.398	0.013	0.012	0.013
AD	2.199	2.202	2.202	0.372	0.376	0.368	0.008	0.007	0.002
AT	158.020	158.440	159.270	113.930	113.760	124.790	105.610	105.390	106.080

TABLE 7.5 – Résultats expérimentaux pour $m = 20$

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
n=30									
BestCmax	63	52	21	32	35	21	62	54	34
MD	0.119	0.135	0.180	0.662	0.771	0.735	0.678	0.700	0.967
AD	0.030	0.035	0.061	0.267	0.263	0.292	0.191	0.205	0.246
AT	0.052	0.053	0.053	0.049	0.049	0.049	0.042	0.041	0.043
n=50									
BestCmax	53	26	7	30.000	29	20	53.000	45	12
MD	0.256	0.209	0.328	1.053	0.976	1.062	0.964	0.964	1.031
AD	0.093	0.104	0.136	0.482	0.486	0.504	0.319	0.343	0.419
AT	0.144	0.144	0.145	0.140	0.139	0.140	0.120	0.119	0.118
n=100									
BestCmax	43	33	9	30.000	17	27	39.000	25	19
MD	0.537	0.556	0.643	1.573	1.530	1.526	0.983	0.893	1.208
AD	0.301	0.311	0.346	0.916	0.941	0.922	0.573	0.589	0.629
AT	0.621	0.619	0.625	0.615	0.610	0.614	0.454	0.454	0.455
n=250									
BestCmax	51	29	16	23.000	22	30	39.000	30	13
MD	1.107	1.109	1.176	2.485	2.526	2.542	0.411	0.425	0.448
AD	0.800	0.807	0.827	1.905	1.899	1.890	0.243	0.250	0.289
AT	4.913	4.915	4.946	4.814	4.815	4.841	3.243	3.239	3.351
n=350									
BestCmax	39	30	23	19.000	18	41	48.000	22	15
MD	1.446	1.465	1.508	2.420	2.474	2.305	0.294	0.338	0.295
AD	1.076	1.082	1.092	2.155	2.173	2.124	0.164	0.180	0.196
AT	10.975	10.977	11.061	10.439	10.457	10.540	7.189	7.176	7.356
n=500									
BestCmax	35	33	30	25	20	46	45	35	10
MD	1.903	1.907	1.959	2.119	2.135	2.089	0.223	0.190	0.231
AD	1.461	1.467	1.471	1.958	1.959	1.932	0.097	0.107	0.125
AT	26.581	26.515	26.718	24.273	24.245	24.687	17.221	17.207	17.463
n=1000									
BestCmax	0	25	75	25	25	50	25	50	0
MD	2.961	2.946	3.056	1.731	1.669	1.615	0.046	0.033	0.078
AD	2.513	2.520	2.511	1.610	1.608	1.554	0.037	0.022	0.048
AT	159.520	160.040	161.060	135.820	134.280	141.970	103.580	103.560	103.710

TABLE 7.6 – Performances globales pour les différentes classes

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
BestCmax									
m2Class	48.786	61.786	26.857	23.829	25.286	66.643	15.571	13.786	92.429
m5Class	57.357	42.357	31.643	44.571	45.286	24.571	21.929	19.571	77.714
m10Class	44.929	34.714	24.214	30.071	22.857	37.643	28.929	29.5	52.571
m20Class	40.571	32.571	25.857	26.286	23.714	33.571	44.429	37.286	14.714
Max-Dev									
m2Class	0.191	0.183	0.287	0.302	0.250	0.361	0.174	0.160	0.123
m5Class	0.997	1.010	1.077	0.566	0.617	0.598	0.364	0.415	0.430
m10Class	2.285	2.330	2.251	1.260	1.386	1.298	0.571	0.571	0.720
m20Class	2.961	2.950	3.056	2.485	2.526	2.542	0.983	0.964	1.210
Av-Dev									
m2Class	0.021	0.021	0.039	0.018	0.017	0.023	0.014	0.015	0.003
m5Class	0.369	0.380	0.391	0.112	0.121	0.139	0.042	0.044	0.040
m10Class	0.845	0.850	0.0868	0.568	0.576	0.571	0.063	0.069	0.080
m20Class	0.896	0.900	0.921	1.328	1.333	1.317	0.232	0.242	0.280
Av-Time									
m2Class	17.385	17.312	17.486	17.963	17.965	18.285	19.048	19.044	19.126
m5Class	25.912	25.933	25.681	16.916	16.926	16.917	18.443	18.486	18.686
m10Class	28.735	28.797	28.963	21.218	21.209	23.019	19.165	19.135	19.236
m20Class	28.972	29.038	29.230	25.164	24.942	26.120	18.835	18.829	18.928

TABLE 7.7 – Performances globales sur toutes les instances

	densité faible			densité moyenne			densité forte		
	H1	**H4**	**H9**	**H1**	**H4**	**H9**	**H1**	**H4**	**H9**
Global BestCmax	47.911	42.857	27.143	31.214	29.286	40.607	27.714	25.036	59.357
Global Max-Dev	2.961	2.946	1.668	1.153	1.195	1.200	0.523	0.528	0.620
Global Aver-Dev	0.533	0.538	0.555	0.506	0.512	0.512	0.088	0.092	0.101
Global Aver-Time	25.251	25.270	25.340	20.315	20.261	21.085	18.873	18.874	18.994

Conclusion

Dans cette thèse, nous avons considéré le problème d'ordonnancement de tâches sur des machines parallèles identiques sous contraintes de concordance de tâches, représentées par un graphe, appelé graphe de concordance. Nous avons étudié et analysé plusieurs classes de graphes : les graphes bipartis, les complémentaires de graphes bipartis et les graphes scindés. Pour chacune de ces classes nous avons établi des résultats de type NP-difficulté et des résultats polynomiaux. Ainsi nous avons présenté plusieurs algorithmes polynomiaux exacts pour résoudre de nombreux sous-problèmes polynomiaux. En particulier, nous avons montré la NP-difficulté d'un problème ouvert posé dans la litérature qui est le cas de deux machines avec au plus trois durées de traitement possibles.

Pour la résolution du problème dans le cas où toutes les tâches sont disponibles à l'instant 0, nous avons proposé des heuristiques de type algorithmes de liste. Tous ces algorithmes ont été testés et comparés sur des instances générées aléatoirement.

Dans le cas des durées de traitement unitaires, nous avons montré le lien existant entre notre problème et certains problèmes déja étudiés dans la littérature : le problème d'ordonnacement avec exclusion mutuelle, le problème ordonnancement avec conflit (S.W.C), le problème d'ordonnancement avec exclusion mutuelle (M.E.S), le probléme d'ordonnancement sur une machine à traitement par batchs et le problème d'ordonnancement à contraintes de ressources.

Le tableau suivant résume les principaux types de problèmes étudiés et leurs complexités. A travers cette thèse nous avons montré qu'en utilisant le graphe de concordance au lieu du graphe de conflit, on peut faciliter l'étude de tels problèmes et avoir une bonne vision quant aux démonstrations des résultats de complexité.

Beaucoup de problèmes ont été étudiés dans cette thèse, certains sont difficiles, d'autres sont faciles (au sens de la complexité) et plusieurs algorithmes ont été proposés, toutefois beaucoup de questions restent en attente : améliorer les algorithmes existants, étudier d'autres sous-problèmes et proposer d'autres approches de résolution. Les résultats obtenus ouvrent la voie à de nouvelles pistes de recherches intéréssantes :

Problème	Complexité	Paragraphe
$P2\|AgreeG = (S_1, S_2; E), p_i \in \{1, 2, 3\}\|C_{max}$	NP-difficile	4.1
$P2\|AgreeG = (S_1, S_2; E), r_i \in \{0, r\}$, $p_i \in \{1, 2\}\|C_{max}$	NP-difficile	4.2
$P2\|AgreeG = (S_1, S_2; E), p_{S_1} = 1\|C_{max}$	Polynomial en $O(n^3)$	4.3
$P\|AgreeG = (K_1, K_2; E), r_i \in \{0, 1, 2\}$, $p_i = 1, \|C_{max}$	NP-difficile pour $m \geq n$	5.1
$P\|AgreeG = (K_1, K_2; E), p_i = 1\|C_{max}$	Polynomial en $O(n^2)$ $m \geq n$, r_i impaires	5.2
$P2\|AgreeG = (K, S; E)\|C_{max}$	NP-difficile	6.1
$P2\|AgreeG = (K, S; E), p_i \in \{1, 2\}$, $r_i \in \{0, r\}\|C_{max}$	NP-difficile	6.2
$P2\|AgreeG = (K, S; E), p_K = 1\|C_{max}$	Polynomial en $O(n^3)$	6.3

TABLE 7.8 – Résumé des principaux résultats

1. Etendre ce travail à d'autres types de graphes.
2. Etendre ce travail à d'autres critères d'optimalité comme la somme des dates de fin de traitement (pondérée ou non).
3. Considérer le cas des machines parallèles en général.
4. Résoudre le problème avec des méthodes exactes comme la méthode de Branch and Bound et la programmation dynamique.
5. Comparer les performances des algorithmes de liste avec la solution optimale pour des problèmes de petites tailles.
6. Développer des métaheuristiques comme la recherche taboue, récuit simulé et les algorithmes génétiques.
7. Améliorer les bornes inférieures .
8. etc.

Bibliographie

[1] N. Alon. A note on decomposition of graphs into isomorphic matchings. Acta Math Hungar 1983 ; 42 : 221-223.

[2] B.S. Baker,E.G. Coffman. Mutual Exclusion Scheduling. Theoretical Computer Science 1996 ; 162 : 225-243.

[3] M. Bendraouche and M. Boudhar. Scheduling on parallel processors subject to compatibility constraints ; 23rd Euro Conference on Operations Research 5-8 juillet 2009 ; Bonn (Germany).

[4] M. Bendraouche, M. Boudhar. Scheduling problem subject to compatibity constraints. In Proceedings du Colloque sur l'Optimisation et les Systèmes d'Information, 18-20 Avril, Ouargla, Algérie 2010 ; 203-214.

[5] M. Bendraouche and M. Boudhar. Scheduling with conflicts : bipartite and their complements agreement graphs, apparaitra dans les Proceedings de la cinquième conférence Internationale en Recherche Opérationnelle CIRO'10. Marrakech (Maroc) du 24 au 27 Mai 2010.

[6] M. Bendraouche, M. Boudhar. Scheduling jobs on identical machines with agreement graph. Computers and Operations Research 2012 ; 39 382–390.

[7] M. Boudhar and M. Bendraouche M. Scheduling jobs on identical machines with split agreement graph. A apparaitre in Proceedings of the third International Conference on Applied Operational Research. ICAOR 2011 August 24-26, 2011 Istanbul, Turkey.

[8] J. Blazewicz, K. Ecker , E. Pesch, G. Schmidt et J. Weglarz. Scheduling Computer and Manufacturing Processes. Springer-Verlag, Berlin, Allemagne. (deuxième édition) 2001.

[9] J. Blazewicz, J. Barcelo, W. Kubiak, H. Rock. Scheduling tasks on two processors with deadlines and additional resources. European Journal of Operational Research 1986 ; vol. 26, issue 3 : 364-370.

[10] J. Blazewicz, J.K Lenstra, A.H.G Rinnooy Kan. Scheduling subject to ressource constraints ; classification and complexity. Discrete Applied Mathematics 1983 ; 5 No 1 :11-24.

[11] J. Blazewicz. Complexity of computer scheduling aloithms under resourceconstraints. Proc. I Meeting AFCET-SMF on Applied Mathematics, Palaiseau (France) 1978 ; 169-178.

[12] J. Blazewicz, K. Ecker. A linear time algorithmfor restricted bin packing and scheduling problems. Operations Research Letters 1983 ; 2 : 80-83.

[13] H.L. Bodlaender, F.V. Fomin. Equitable colorings of bounded treewidth graphs. In Fiala, J., Koubek, V., Kratochvil, J. (Eds), Proceedings of MFCS 2003, the 29th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Springer, Berlin, Germany 2004 ; vol 3153 : 180-190.

[14] H.L. Bodlaender, K. Jansen. Restrictions of graph partition problems part I. Theoretical Computer Science 1995 ; 148 : 93-109.

[15] H.L. Bodlaender, K. Jansen, and G.J. Woeginger. Scheduling with incompatibile jobs. Discrete Applied Mathematics 1994, 55 : 219-232.

[16] M. Boudhar. Scheduling on a batch processing machine with split compatibility graphs. Journal of Mathematical Modelling and Algorithms 2005 ; Vol4,pp.391-407.

[17] M. Boudhar. Dynamic scheduling on a single batch processing machine with split compatibility graphs. Journal of Mathematical Modelling and Algorithms 2003 ; Vol2,pp.17-35.

[18] M. Boudhar, G. Finke. Scheduling on a batch machine with job compatibilities. Belgium Journal of Operations Research, Statistics and Computer Science (JORBEL) 2000 ; 40 :69-80.

[19] M. Boudhar M, A. Khelladi. Ordonnancement par batchs sous contraintes de compatibilité de tâches. Maghreb Mathematical Review 2004 ; Vol.13,pp.1-15.

[20] M. Boudhar. Scheduling on a batch processing machine with bipartite compatibility graphs. Mathematical Methods of Operations Research 2003 ; Vol.57,pp.513-527.

[21] J. Carlier and P. Chretienne. Problèmes d'ordonnancement : modélisation, complexité et algoithmes. Masson, Paris (France) ; 1984.

[22] J. Cheriyan, S.N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. SIAM Journal on Computing 1989 ; 18 : 1057-1086.

[23] E. Cohen, M. Tarsi. NP-completeness of graph decomposition problem. J Complexity 1991 ; 7 : 200-212.

[24] D.G. Corneil. The complexity of generalized clique packing. Discrete Applied Mathematics 1985 ; 12 : 233-239.

[25] E. Dahlhaus, M. Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. Discrete Appl Math 1998 ; 84 : 79-91.

[26] D. De Werra. Restricted coloring models for timetabling. Discrete Math 1997 ; 165/166 : 161-170.

[27] J. Edmonds. Paths, trees, and flowers. Canadian Journal of Mathematics 1965 ; 17, pp. 449-467.

[28] G. Even, M.M. Halldorson, L. Kaplan, D. Ron. Scheduling with conflicts : online and offline algorithms. Journal of scheduling 2009 ; 12 : 199-224.

[29] G. Finke, V. Jost, M. Queyranne and A. Sebo. Batch processing with interval graphs compatibilities between tasks. Discrete Applied Mathematics 2008 ; Vol. 156, No. pp.556-568.

[30] F. Gardi. Planification d'horaires de travail et théorie des graphes. In Actes du 5ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, pp. 99-100. Laboratoire d'Informatique d'Avignon (LIA, Université d'Avignon et des Pays de Vaucluse), Avignon, France.

[31] F. Gardi. Mutual exclusion scheduling with interval graphs or related classes Part I. Discrete Appl Math 2009 ; 157 : 19-35.

[32] M.R. Garey, D.S. Johnson. Computers and Intractability. New York. Freeman (1979).

[33] M.R. Garey. and R.L Graham. Bounds for multiprocessor scheduling with resource constraints. SIAM Journal on Computing 1975 ; 4 : 187–200.

[34] M.R. Garey , D.S. Johnson . "Strong" NP-completeness results : motivation, examples, and implications. J. Assoc. Comput. Mach 1978 ; 25(3) :499-508.

[35] M.R Garey, D.S. Johnson. Complexity results for multiprocessing scheduling under resource constraints. SIAM J. on Computing 1975 ; 4 : 397-411.

[36] M.C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Computer Science and Applied Mathematics Series, Academic Press, New York, NY 1980.

[37] R.L. Graham. Bounds for certain multiprocessing anomalies. Bell System Technical Journal 1966 ; 45 :1563-1581.

[38] R.E. Graham, E.L. Lawler, J.K. Lenstra,A.H.G. Rinnooy Kan. Optimisation and approximation in deterministic sequencing and scheduling : a survey, Ann. Discrete Math. 1979 ; 4, 287-326.

[39] K. Jansen. The mutual exclusion scheduling problem for permutation and comparability graphs. Inform and Comput 2003 ; 180 2 : 71-81.

[40] M.M. Halldorsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, J.A. Telle. Multicoloring trees. Information and Computation 2003 ;180(2) : 113-129.

[41] P. Hansen, A. Hertz, J. Kuplinski. Bounded vertex colorings of graphs. Discrete Math 1993 ; 111 : 305-312.

[42] J. Jarvis, B. Zhou. Bounded vertex coloring of trees. Discrete Math 2001 ; 232 : 145-151.

[43] J. Krarup, D. De Werra. Chromatic optimization : limitations, objectives, uses, references. European Journal of Operational Research 1982 ; 11, pp. 1-19.

[44] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker. Complexity of machine scheduling problems. Studies in integer programming (Proc. Workshop, Bonn, 1975). Ann. of Discrete Math 1977 ; 1, 343-362, North Holland, Amsterdam.

[45] Z. Lonc. On complexity of some chain and antichain partition problems. In G Schmidt, R Berghammer (Eds) Proceedings of WG'91, the 17th International Workshop on Graph - Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer Berlin Germany 1991 ; vol 570 : 97-104.

[46] L. Lovasz. Stable set and polynomials. Discrete Math 1994 ; 124 : 137-153.

[47] S. Sakai, M. Togasaki, K. Yamazaki. A note on greedy algorithms for maximum weighted independent set problem. Discrete Applied Mathematics 2003 ; 126 : 313-322.

[48] M. Sakarovitch. Optimisation combinatoire : théorie des graphes. Hermann, Paris, 1984.