N° d'ordre : 04 / 2004 - E / MT

الجمهورية الجزائرية الديمقراطية الشعبية
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

وزارة التعليم العالي والبحث العلمي

جامعة هواري بومدين للعلوم والتكنولوجيا

THÈSE

Présentée pour obtenir le diplôme de
DOCTEUR D'ÉTAT

En **Mathématiques**
Spécialité : **Recherche Opérationnelle**

par
BOUDHAR Mourad

Thème :

ORDONNANCEMENT SUR MACHINES À TRAITEMENT PAR BATCH SOUS CONTRAINTES DE COMPATIBILITÉ DE TÂCHES : COMPLEXITÉ ET APPROCHES ALGORITHMIQUES

Soutenue le samedi 26 juin 2004, devant le jury composé de :

M. Moncef ABBAS, Professeur, USTHB.	Président
M. Gerd FINKE, Professeur, UJF_Grenoble.	Directeur de thèse
M. Abdelkader KHELLADI, Professeur, USTHB.	Codirecteur de thèse
M. Méziane AIDER, Maître de Conférences, USTHB.	Examinateur
M. Hacène AIT HADDADENE, Maître de Conférences, USTHB.	Examinateur
M. Abdelhafid BERRACHEDI, Maître de Conférences, USTHB.	Examinateur
M. Mostafa BLIDIA, Maître de Conférences, USTB_Blida.	Examinateur
Mlle. Isma BOUCHEMAKH, Maître de Conférences, USTHB.	Examinatrice

Remerciements

Le travail présenté dans cette thèse a été initié au sien du laboratoire LEIBNIZ-IMAG de Grenoble dont je tiens à remercier le directeur Nicolas Balacheff et l'ancien directeur Philippe Jorand pour leur accueil. Sans oublier Messieurs les professeurs Jean-Marie Laborde et Charles Payan.

Je tiens à exprimer ma gratitude et ma reconnaissance à Monsieur Gerd Finke, directeur de thèse, responsable de la formation doctorale : Recherche Opérationnelle et Optimisation et responsable de l'équipe dans laquelle j'ai effectué une grande partie de mes travaux de recherche, pour m'avoir accueilli dans son équipe, proposé ce sujet, guidé dans mes recherches et dirigé cette thèse. Je le remercie également pour la confiance qu'il m'a témoignée, pour ses conseils fructueux et pour son aide précieuse.

Je remercie également Monsieur Abdelkader Khelladi, directeur de thèse, professeur à l'USTHB, pour l'attention qu'il m'a accordée. Ses remarques, commentaires et suggestions m'ont été d'une grande aide et m'ont permis d'améliorer ce travail.

Mes remerciements vont ensuite aux membres du jury :

à Monsieur Moncef Abbas, professeur à l'USTHB, qui m'a fait l'honneur de présider ce jury ;

aux professeurs Méziane Aider, Hacène Ait Haddadène, Abdelhafid Berrachedi, Mostafa Blidia et Isma Bouchemakh qui ont accepté de juger ce travail et d'être membres de ce jury.

Mes remerciements vont aussi à Monsieur Dominique De Werra, professeur à l'EPFL de Lausanne, pour l'intérêt qu'il a montré à mon travail.

Je tiens aussi à remercier Monsieur Abdellah Zemirline, de l'Université de Bretagne Occidentale à Brest, pour m'avoir donné la chance de découvrir et d'apprécier les problèmes de l'ordonnancement.

Je remercie ensuite mes collègues et amis de l'université qui m'ont encouragé et soutenu pour leurs gentillesses et leurs marques de sympathie. Sans oublier les personnes que j'ai côtoyées aux laboratoires LEIBNIZ et GILCO de Grenoble.

Je remercie particulièrement Larbi Ben Aissa pour avoir corrigé, au début de mes travaux de recherche, mes papiers rédigés en anglais.

Mes derniers mots seront pour ma famille : ma mère et mon père qui ont eu confiance en moi. Et bien sûr pour ma femme Abida et mon fils Mohamed El-Amine qui, en dépit des contraintes et des aléas liées à la thèse, m'ont soutenu avec patience tout le long de ces années. Sans oublier ma sœur karima et mes frères Lies, Toufik et Kamel. A tous, je dédie cette thèse.

Table des matières

Introduction

Dans la théorie classique de l'ordonnancement, on suppose que les machines ne peuvent traiter qu'une seule tâche à la fois. Or, en réalité il existe un autre type de machines qu'on peut appeler "machines à traitement par batch" et qui sont capables de traiter plusieurs tâches simultanément. Pour ces machines, le regroupement des tâches en batchs pour subir un certain traitement est nécessaire et les tâches d'un même batch sont traitées simultanément (chauffage simultané des pièces dans un four, peinture simultanée des pièces, laminage simultané des pièces, transport simultané des pièces par véhicule autoguidé, etc.). Dans ces cas, les tâches d'un même batch doivent être compatibles (les mêmes propriétés physiques ou chimiques, formes, couleurs, longueurs, etc.).

La motivation principale de l'ordonnancement par batch est l'ordonnancement des opérations de chauffage dans l'industrie des semi-conducteurs. L'étape finale dans la production des circuits intégrés est l'opération de chauffage, où les puces sont chargées sur une plaque, ensuite placées dans un four et exposées à de hautes températures. Le but de l'opération de chauffage est d'exposer les puces au stress thermique sur une grande période afin de déterminer d'éventuels défauts de fabrication.

Dans chacune des situations précédentes l'atelier de production est composé d'une cellule A contenant plusieurs types de machines ordinaires pouvant traiter plusieurs types de pièces. A la sortie de cette cellule, chaque pièce a une date de disponibilité qui est égale à la date de sa sortie de la cellule. Ces pièces seront ensuite transférées vers une autre cellule B, contenant un ensemble de machines à traitement par batch, pour subir d'autres traitements. Comme machines à traitement par batch, on peut penser à un four de chauffage ; chaque pièce doit être chauffée, pendant une certaine durée, à une certaine température T. Notons, qu'à chaque four est associée une capacité (c'est-à-dire qu'il ne peut contenir qu'un certain nombre de pièces simultanément). Notons, aussi, que si des pièces sont traitées simultanément dans un même four, alors la date de fin de traitement de chacune de ces pièces (qui forment un batch) est égale à la date de fin de traitement de la pièce qui nécessite le plus de temps de traitement (c.-à-d. ayant le plus long temps de traitement). Et compte tenu des propriétés physiques, chimiques ou autres, certaines pièces ne peuvent pas être chauffées simultanément dans un même four. La question, qui est posée, est : comment ordonnancer les pièces sur les machines de la cellule B en temps minimal?

Dans cette thèse, nous considérons le problème de l'ordonnancement de tâches indépendantes (sans constraintes d'antériorité) sur des machines à traitement par batch en minimisant le temps de fin de traitement de l'ensemble des tâches (makespan). Nous supposons qu'il existe une relation de compatibilité entre les tâches, où deux tâches sont dites compatibles si elles peuvent être traitées simultanément dans un même batch. Cette relation est représentée 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 dernières sont compatibles. La capacité de la machine à traitement par batch peut être finie (elle ne peut traiter qu'un nombre fini de tâches à la fois) ou infinie (elle peut traiter un nombre illimité de tâches à la fois). Chaque tâche a un temps de traitement et une date de disponibilité qui sont des entiers naturels. Le temps de traitement d'un batch est égal au maximum des temps de traitement des tâches appartenant à celui-ci. Toutes les tâches d'un même batch commencent leur exécution à une même date et terminent leur exécution à une même date. L'interruption des tâches n'est pas autorisée.

Une application concrète où la contrainte de compatibilité de tâches est représentée par un graphe a été décrite dans [32, 60]. Une tôle d'acier passe à travers une machine à k outils pour subir des opérations de poinçonnage. Dépendant de la position des trous sur la tôle, des trous constituant un batch sont poinçonnés simultanément dans chaque phase de la machine. Le temps de traitement d'une opération de poinçonnage est, évidemment, le même pour tous les trous, mais il peut être différent si nous considérons le cas où quelques trous (spéciaux) sont obtenus par plusieurs opérations de poinçonnage. Si les trous (ou les différentes opérations de poinçonnage qui sont les tâches) sont représentés par des sommets d'un graphe G et les arêtes indiquent la compatibilité entre les trous, alors ce problème devient un cas particulier du problème que nous considérons.

Cette nouvelle combinaison de l'ordonnancement par batch et de graphes de compatibilité définis entre les tâches apparaît comme un concept nouveau et intéressant utilisant, à la fois, la théorie de l'ordonnancement et la théorie des graphes.

Cette thèse est organisée comme suit :

Dans le chapitre 1, nous abordons les principaux termes et notions utilisés tout au long de cette thèse. Nous commençons par les graphes, ensuite l'ordonnancement, et nous terminons par la complexité des problèmes.

Le chapitre 2, est consacré aux principaux résultats qui existent dans le domaine de l'ordonnancement par batchs. Nous commençons par énumérer les principales motivations de l'ordonnancement par batchs. Ensuite, nous proposons quelques notations et définitions nécessaires. Par la suite, nous proposons une classification de ces problèmes, similaire à celle qui existe pour les problèmes de l'ordonnancement classique. Et nous terminons, ce chapitre, par l'état de l'art.

Le chapitre 3, est consacré à la définition du problème traité dans cette thèse ainsi que les différentes notations nécessaires à sa définition. Un exemple illustratif est aussi donné. Nous proposons ensuite une modélisation mathématique du problème sous forme d'un programme linéaire en nombres réels avec variables bivalentes. Dans le cas où les dates de disponibilité des tâches seraient toutes nulles, différentes formulations sont simplifiées et présentées. Un commentaire termine ce chapitre.

Nous montrons, dans les chapitres 4, 5, 6, et 7 que le problème est NP-difficile dans le cas, respectivement, d'un graphe quelconque, d'un graphe scindé, d'un graphe biparti et du complémentaire d'un graphe biparti. Il n'est donc pas possible de trouver un algorithme polynomial pour résoudre le problème général ou même certains de ces sous-problèmes, à moins que P=NP. Notons que les techniques utilisées sont spécifiques au problème étudié. Nous utilisons, pour certains types de problèmes, des preuves simples par restriction ou des techniques de remplacement local. Pour d'autres types de problèmes nous utilisons des techniques de construction plus complexes.

Nous proposons, dans les chapitres 4, 5, 6 et 7 un ensemble d'algorithmes polynomiaux exacts pour résoudre certains sous-problèmes polynomiaux dans le cas, respectivement, d'un graphe de compatibilité quelconque et d'une capacité de la machine à traitement par batch est égale à 2, d'un graphe scindé, d'un graphe biparti et du complémentaire d'un graphe biparti. Notons qu'un algorithme polynomial est spécifique au problème à résoudre. Nous faisons appel, pour certains types de problèmes, à des algorithmes classiques comme ceux du couplage maximum, du couplage de poids maximum, du flot maximum, etc. Pour d'autres types de problèmes, nous utilisons des algorithmes de listes, de programmation dynamique, etc.

Nous proposons au début du chapitre 8 des algorithmes polynomiaux exacts pour résoudre le problème dans le cas d'un graphe complet (c'est-à-dire toutes les tâches sont deux à deux compatibles). L'un aborde le cas où la capacité de la machine à traitement par batch est infinie, et les autres, le cas où la capacité de la machine à traitement par batch est finie avec des conditions sur les tâches. A la fin de ce chapitre nous étudions deux types de graphes spéciaux que nous construisons aux deux derniers paragraphes. Pour chacun de ces deux graphes nous présentons des méthodes de résolution polynomiales exactes.

Comme le problème général et quelques sous-problèmes sont NP-difficiles dont certains au sens fort, il est donc peu probable qu'il pourra exister un algorithme polynomial pour résoudre tous ces problèmes, à moins que P=NP. Néanmoins, Pour résoudre le problème général, nous présentons dans le chapitre 9, des méthodes exactes (par séparation et évaluation) dont le temps de calcul est, évidemment, exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles. Pour les problèmes de grandes tailles, les méthodes exactes connues ne sont plus envisageables, de par leur temps de calcul. Il est donc nécessaire dans ce cas, d'utiliser des méthodes approchées (heuristiques) qui donnent des solutions pas toujours optimales, mais obtenues rapidement. Ces solutions peuvent ensuite servir de solutions initiales pour les méthodes par séparation et évaluation ou des méthodes amélioratrices. Notons que les

méthodes approchées sont largement utilisées pour appréhender ce genre de problèmes. Les deux méthodes, exactes et heuristiques, sont basées sur le rangement des tâches.

Enfin, une conclusion termine cette thèse avec un résumé des principaux résultats présentés. Nous proposons aussi des perspectives à ce travail.

Chapitre 1

Graphes, ordonnancement et complexité

Nous abordons, dans ce chapitre, les principaux termes et notions utilisés tout au long de cette thèse, sans trop détailler et sans entrer dans le fond du problème. Le lecteur intéressé par de plus amples informations pourra consulter les différents ouvrages dédiés à cet effet, dont certains sont mentionnés dans la bibliographie. Nous commençons par les graphes, ensuite l'ordonnancement, et nous terminons par la complexité des problèmes.

1.1 Graphes et réseaux

Un graphe (simple et sans boucles) est défini par un couple de deux ensembles $G = (V, E)$: V est un ensemble de sommets, également appelés nœuds, et E est un ensemble de couples (distincts) de sommets (distincts) appelés arêtes. Les petits graphes se représentent graphiquement par un ensemble de points ou de petits cercles désignant les sommets, et par des lignes représentant les arêtes.

Nous donnons ci-dessous quelques définitions utilisées en théorie des graphes.

Deux graphes $G = (V, E)$ et $G' = (V, E')$ sont dit isomorphes s'il existe une fonction bijective $f : V \longrightarrow V$ telle que $(i, j) \in E$ si et seulement si $(f(i), f(j)) \in E'$.

Un graphe $G' = (V', E')$ est dit sous-graphe induit du graphe $G = (V, E)$ si $V' \subset V$ et E' est l'ensemble de toutes les arêtes de E qui relient les sommets de V' (c.-à-d. $\forall (i, j) \in V' \quad (i, j) \in E'$ si et seulement si $(i, j) \in E$).

Le graphe complémentaire de G noté $\overline{G}$ est le graphe $\overline{G} = (V, \overline{E})$ avec $\overline{E} = P(V) \setminus E$ où $P(V)$ est l'ensemble de tous les couples (distincts) de sommets (distincts) possibles.

Un graphe est complet si toute paire de sommets forme une arête. Une clique d'un graphe est un sous graphe complet.

Un stable est un sous-ensemble de sommets sans arêtes. En particulier si un graphe G est sans arêtes, on dit que le graphe est stable.

Un graphe $G = (V, E)$ est dit biparti s'il existe une partition $V = S_1 \cup S_2$ de ses sommets en deux stables S_1 et S_2.

Soit un graphe G=(V,E), désignons par :

- $\theta(G)$ le nombre minimum de cliques qui partitionnent V.
- $\gamma(G)$ le nombre minimum de stables qui partitionnent V.
- $\omega(G)$ le nombre maximum de sommets d'une clique.
- $\alpha(G)$ le nombre maximum de sommets d'un stable.

Un graphe $G = (V, E)$ est dit parfait si, pour tout sous graphe induit G' nous avons : $\omega(G') = \gamma(G')$ ou $\alpha(G') = \theta(G')$.

Parmi les graphes parfaits classiques, on distingue deux grandes classes [67] :

1. Graphes triangulés (à cordes) : Un graphe est dit à cordes si tout cycle (qui est une suite d'arêtes $(i_1, i_2), (i_2, i_3), \ldots, (i_{p-1}, i_p), (i_p, i_1)$) de longueur supérieure strictement à 3 possède une corde (qui est une arête joignant deux sommets non consécutifs du cycle).
2. Graphes de comparabilité : Un graphe est dit de comparabilité s'il admet une orientation transitive (notons que tout graphe biparti est un graphe de comparabilité).

et les cas particuliers [67] :

1. Graphes d'intervalles : Un graphe est dit 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. Notons qu'un graphe G est d'intervalles si et seulement si G est triangulé et $\overline{G}$ est de comparabilité.
2. Graphes scindés : Un graphe est dit scindé s'il existe une partition $V = S \cup K$ de ses sommets en un stable S et une clique K. Notons qu'un graphe G est scindé si et seulement si G et $\overline{G}$ sont des graphes triangulés.

D'autres graphes parfaits ont été étudiés dans la littérature, nous citons : les "line-graphs" de bipartis, les graphes i-triangulés, les graphes de parité, les graphes de Meyniel, les graphes faiblement triangulés, les graphes parfaitement ordonnables, les graphes alternés, les graphes fortement parfaits, les graphes Bip, les graphes de quasi-parité, etc.

Un réseau (de transport), noté $R = (X, U, C)$, est un graphe orienté (chaque arête a une orientation et est appelée arc) $G = (X, U)$ où chaque arc $u \in U$ est muni d'un nombre $C_u \geq 0$ appelé "la capacité de l'arc u". Lorsqu'on fait circuler un flot sur le graphe G, ce nombre indique la limite supérieure du flux admissible sur l'arc u.

1.2 Ordonnancement

Ordonnancer c'est programmer l'exécution d'un ensemble de tâches (travaux, tasks ou jobs en anglais) en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. Les problèmes d'ordonnancement apparaissent dans le suivi des projets, les ateliers de production, les emplois du temps, en informatique, etc. En effet, les tâches peuvent représenter des pièces mais également des projets, des programmes ou encore des activités etc. De même, les ressources peuvent représenter des machines industrielles mais également des processeurs ou des personnes etc. Les différentes données d'un problème d'ordonnancement sont donc les tâches, les contraintes, les ressources et les objectifs. Une ressource est un moyen matériel, financier ou humain à disposition pour la réalisation d'une tâche. Les contraintes représentent les limites imposées par l'environnement, tandis que l'objectif est le critère d'optimisation.

Les ordonnancements sont généralement représentés par des diagrammes dits de Gantt (voir l'exemple de la figure 1.1). Ceux-ci indiquent, selon une échelle temporelle donnée, l'occupation des machines par les différentes tâches, les temps morts (parties hachurées) dus essentiellement aux éventuelles indisponibilités des tâches ou contraintes d'antériorité et les éventuelles indisponibilités des machines dues aux changements de tâches, d'outils, etc.

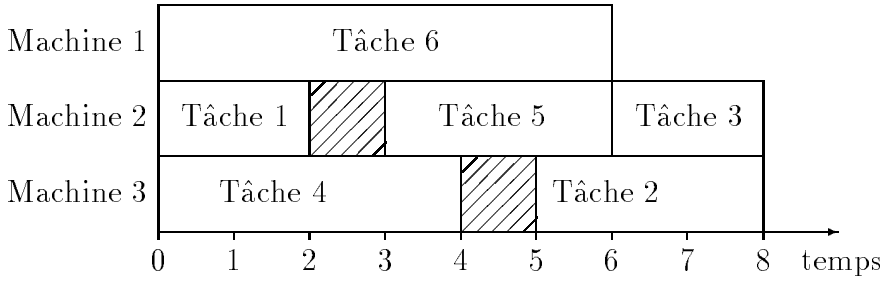


FIG. 1.1: *Exemple de diagramme de Gantt.*

1.3 Complexité

L'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre sur un ordinateur. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes : la classe P et la classe NP-complet. Nous exposons dans ce qui suit, les grands principes de la théorie de la complexité des problèmes.

1.3.1 Les classes P, NP et NP-complet

Pour pouvoir exposer la notion de classes de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est "oui" ou "non". Notons qu'il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil k correspondant à la fonction objectif f. Le problème de décision devient : "existe-t-il une solution réalisable (S) telle que $f(S) \leq$ (ou $\geq$) k?".

Il est alors possible de définir la classe P (Polynomial) qui regroupe les problèmes de décision résolus par des algorithmes polynomiaux. Un algorithme polynomial est défini comme un algorithme dont le temps d'exécution est en $O(p(x))$ où p est un polynôme et x est la longueur d'entrée (c.-à-d. le nombre de données) d'une instance du problème.

La classe NP (Non deterministic Polynomial) regroupe les problèmes 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 si une solution "potentielle" donnée satisfait la question posée.

Les algorithmes polynomiaux sont évidemment des cas particuliers des algorithmes non déterministes. Aussi tout problème de décision qui peut être résolu par un algorithme polynomial, et qui donc appartient à la classe P, appartient également à la classe NP. D'où P$\subseteq$ NP.

Parmi les problèmes de la classe NP, une large classe de problèmes, les problèmes NP-complets, 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. Afin de décrire cette classe d'équivalence, définissons tout d'abord la transformation (réduction) polynomiale entre deux problèmes. Soient D1 et D2, deux problèmes de décision. La transformation polynomiale de D1 vers D2 (noté D1 $\propto$ D2) peut être vue

comme une fonction f de l'ensemble des instances de D1 vers l'ensemble des instances de D2 qui satisfait aux deux conditions suivantes :

1. f est calculable par un algorithme polynomial.

2. Pour toute instance I de D1, I a pour réponse "oui" (pour D1) si et seulement si $f(I)$ a pour réponse "oui" (pour D2).

D'une manière équivalente, D1 se transforme (réduit) polynomialement à D2, s'il existe un algorithme de résolution de D1, qui fait appel à un algorithme de résolution de D2, et qui est polynomial lorsque la résolution de D2 est comptabilisée comme une opération élémentaire.

On dit alors d'un problème de décision qu'il est NP-complet si tout problème de la classe NP se transforme polynomialement à lui.

Ainsi, il est nécessaire de connaître des problèmes connus pour être NP-complets. Le premier problème qui a été prouvé comme étant NP-complet, par S.A. Cook en 1970, est le problème de satisfiabilité (SAT) qui peut être défini comme suit : étant donné une expression booléenne sous forme normale conjonctive (de la forme de conjonctions de disjonctions), existe-t-il une affectation en "vrai" et "faux" de ses variables de manière à ce que l'expression booléenne prenne la valeur "vrai" ? Dans le cas où la réponse est oui, l'expression sera dite satisfiable.

1.3.2 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 puisque, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème. Prouver qu'un problème de décision D est NP-complet consiste en les quatre étapes suivantes :

1. Montrer que D appartient à NP.

2. Choisir D', un problème NP-complet connu.

3. Construire une transformation f de D' vers D.

4. Prouver que f est une transformation polynomiale.

1.3.3 NP-complétude au sens fort

Un algorithme pseudo-polynomial est définit comme un algorithme dont le temps d'exécution est en $O(p(x))$ où p est un polynôme et x est la longueur des données d'une instance du problème. Par exemple, en ordonnancement, un algorithme est polynomial si son temps d'exécution peut s'exprimer comme un polynôme du nombre de tâches, par contre, s'il s'exprime comme un polynôme en temps de traitement maximal (par exemple), il sera dit pseudo-polynomial.

Un problème de décision D est dit NP-complet au sens fort, si :

1. D est un problème à nombres.

2. Il n'existe aucun polynôme q tel que $Max[I] \leq q(Nbre[I])$ pour toute instance I de D.

3. D est NP-complet.

4. Il n'existe aucun algorithme pseudo-polynomial pour le résoudre.

où (pour un codage raisonnable A) $Nbre[I]$ correspond au nombre de symboles utilisés pour décrire I et $Max[I]$ correspond à la magnitude du plus grand nombre de I.

Soient D1 et D2, deux problèmes de décision. La transformation pseudo-polynomiale de D1 vers D2 (noté D1 $\propto$ D2) peut être vue comme une fonction f de l'ensemble des instances de D1 vers l'ensemble des instances de D2 qui satisfait aux quatre conditions suivantes :

1. f est calculable par un algorithme polynomial en $Max1[I]$ et $Nbre1[I]$.

2. Pour toute instance I de D1, I a pour réponse "oui" (pour D1) si et seulement si $f(I)$ a pour réponse "oui" (pour D2).

3. Il existe un polynôme q_1 tel que, pour toute instance I de D1, $q_1(Nbre2[f(I)]) \geq Nbre1[I]$.

4. Il existe un polynôme (à deux variables) q_2 tel que, pour toute instance I de D1, $Max2[f(I)] \leq q_2(Max1[I], Nbre1[I])$.

Le problème **3-Partition** suivant est NP-complet au sens fort [59].

3-Partition :
Etant donnés un ensemble A à $3m$ éléments, une borne entière b et une taille a_i pour chaque élément $a \in A$, telle que $\frac{b}{4} < a_i < \frac{b}{2}$ et telle que $\sum_{a \in A} a_i = mb$. Existe-t-il m ensembles disjoints $S_1, \ldots, S_m$ tels que pour tout $1 \leq j \leq m$, $\sum_{a \in S_j} a_i = b$?

Pour montrer qu'un problème de décision D est NP-complet au sens fort, comme pour NP-complet (au sens faible), il suffit de considérer les quatre étapes suivantes :

1. Montrer que D appartient à NP.

2. Choisir D', un problème NP-complet au sens fort connu.

3. Construire une transformation f de D' vers D.

4. Prouver que f est une transformation pseudo-polynomiale.

1.3.4 Quelques problèmes NP-complets

Nous présentons ici quelques problèmes NP-complets dont la NP-complétude n'a pu être démontrée que grâce à l'existence du premier problème NP-complet. Ce sont les six problèmes de base exposés par Garey et Johnson [59], qui ont enrichi par la suite la littérature concernant la complexité des problèmes.

3SAT (3-satisfiabilité) :
Etant donné une expression booléenne sous forme normale conjonctive où chaque disjonction contient, exactement, 3 variables. Existe-t-il une affectation en "vrai" et "faux" de ses variables de manière à ce que l'expression booléenne prenne la valeur "vrai" ?

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

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

Clique :
Etant donnés un graphe $G = (V, E)$ et un entier k. Existe-t-il $X \subseteq V$ tel que $\mid X \mid \geq k$ 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 chacun des sommets de V?

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

Donnons ci-dessous deux problèmes classiques d'ordonnancement réputés pour être NP-complets (le second est NP-complet au sens fort).

Deux processeurs :
Etant donnés n tâches indépendantes et non morcelables, chaque tâche T_i a un temps 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?

Un processeur :
Etant donnés n tâches indépendantes et non morcelables, chaque tâche T_i a une date de disponibilité r_i, un temps de traitement p_i et une date échue d_i. Existe-t-il un ordonnancement de ces tâches sur un processeur qui respecte les dates de disponibilité et les dates échues?

La figure 1.2 ci-dessous représente les différentes transformations utilisées pour prouver la NP-complétude des six premiers problèmes présentés ci-dessus.

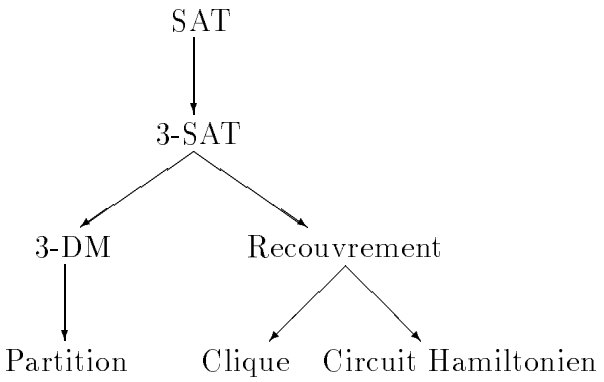


FIG. 1.2: *Enchaînement des preuves de la NP-complétude.*

1.3.5 Optimisation combinatoire et problèmes NP-difficiles

Un problème d'optimisation combinatoire est un problème mathématique où il faut déterminer un "meilleur" élément parmi un nombre fini, mais souvent très élevé, d'éléments. La difficulté de résolution d'un tel problème réside dans le fait que la solution doit être cherchée dans un ensemble de très grande cardinalité.

Un problème d'optimisation combinatoire est dit NP-difficile (resp. au sens fort) si le problème de décision associé est NP-complet (resp. au sens fort).

Le problème d'optimisation combinatoire suivant, associé à **Deux processeurs**, est NP-difficile.

"Etant donné n tâches indépendantes et non morcelables où chaque tâche T_i a un temps de traitement p_i. Déterminer un ordonnancement de ces tâches sur deux processeurs avec une durée de fin de traitement minimale."

1.4 Méthodes classiques de résolution

Nous savons qu'il existe des problèmes de complexité différente. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre. Ces algorithmes sont spécifiques au problème à résoudre. Nous citons ci-dessous quatre de ces problèmes.

1. 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. Un algorithme en $O(mn^{1/2})$ a été proposé par V.V. Vazirani en 1994 pour résoudre ce problème. ($n = |V|$ et $m = |E|$)

2. Couplage de poids maximum : Etant donné un graphe $G = (V, E)$ où chaque arête est munie d'un poids. Déterminer un sous-ensemble $E' \subseteq E$ de poids (la somme des poids de ses arêtes) maximum tel que deux arêtes quelconques de E' ne soient pas adjacentes. Pour résoudre ce problème, H.N. Gabow en 1990 a proposé un algorithme en $O(mn + n^2 logn)$.

3. Flot maximum : Etant donné un réseau $R = (X, U, C)$. Déterminer un flot maximum f dans R. Pour résoudre ce problème, un algorithme en $O(n^2\sqrt{m})$ a été proposé par J. Cheriyan et S.N. Maheshwari en 1989.

4. Ordonnancement avec des dates d'arrivées : Etant donnés n tâches indépendantes et non morcelables où chaque tâche T_i a un temps de traitement p_i et une date d'arrivée r_i. Déterminer un ordonnancement de ces tâches sur une machine avec une durée de fin de traitement minimale. Ce problème est résolu en $O(nlogn)$ en rangeant les tâches par ordre croissant de leurs dates d'arrivées.

Pour les problèmes appartenant à la classe NP-complet (ou NP-difficile), l'existence d'algorithmes polynomiaux semble peu réaliste. Ainsi, différentes méthodes de résolution (méthodes exactes ou heuristiques), sont largement utilisées pour appréhender ces problèmes. Nous citons ci-dessous les méthodes les plus connues. Pour de plus amples informations, le lecteur pourra se reporter aux différents ouvrages et articles traitant ces méthodes.

1.4.1 Méthodes exactes

Le temps de calcul de ces méthodes exactes est exponentiel ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles. Parmi ces méthodes, on trouve :

La programmation dynamique

Introduite par Bellman dans les années 50 [6], la programmation dynamique décompose un problème de dimension n en n sous-problèmes de dimensions 1. Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se fait à partir des lois d'évolution du système et d'une décision.

Le principe d'optimalité est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente.

La méthode par séparation et évaluation

La méthode par séparation et évaluation est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Pour cela, la séparation consiste à décomposer le problème initial en plusieurs sous-problèmes qui sont à leur tour décomposables. Ce processus peut se visualiser sous forme d'un arbre d'énumération. Pour chaque sous-problème (nœud de l'arbre), la procédure d'évaluation calcule (dans le cas d'un problème de minimisation) une borne inférieure de la solution obtenue à partir de ce sous-problème. Au préalable, une borne supérieure de la solution optimale a été 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 trouvée. Ainsi, l'exploration de certaines branches de l'arbre est coupée, ce qui permet de ne pas énumérer réellement toutes les solutions.

La modélisation

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également, parfois de le résoudre. L'idéal est d'obtenir un programme linéaire dont les variables sont réelles. Dans ce cas, il existe des solveurs efficaces pour le résoudre. Dès que le problème comporte des variables entières ou le modèle n'est pas linéaire, il devient plus difficile à résoudre.

Néanmoins, il est parfois surprenant de voir qu'un problème particulier de taille intéressante peut être résolu par la programmation mathématique. Il est donc justifié de commencer à étudier un problème en proposant une ou plusieurs modélisations analytiques. De plus, cette démarche a été simplifiée car il existe, actuellement, des langages de modélisation (comme MPL) permettant d'écrire les programmes linéaires de façon formelle, proche de l'écriture mathématique, et pouvant être couplés directement à un solveur comme CPLEX.

1.4.2 Heuristiques

Pour les problèmes de grandes tailles, les méthodes exactes ne sont pas envisageables de par leur temps de calcul. Il est dans ce cas possible d'utiliser des méthodes approchées qui donnent des solutions certes sous-optimales, mais obtenues rapidement. Ces solutions peuvent ensuite servir de solutions initiales pour les méthodes par séparation et évaluation ou des méthodes amélioratrices. Parmi ces méthodes, on trouve :

Les méthodes constructives

Les 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 dans un certain ordre sans jamais remettre en question un choix, une fois qu'il a été effectué. De principe très simple, ces algorithmes permettent de trouver une solution très rapidement.

Les méthodes de descente

Ce sont des heuristiques de recherche locale les plus simples. Elles consistent à rechercher dans le voisinage de la solution courante, une solution de coût plus faible. Elles procèdent ainsi jusqu'à arriver à un optimum local. Ces méthodes ont l'avantage d'être rapides, mais s'arrêtent dés qu'un optimum local est atteint, même si celui-ci n'est pas de bonne qualité.

Le recuit simulé

Cette méthode considère une solution initiale et recherche dans son voisinage une autre solution pouvant devenir solution courante. L'originalité de cette méthode est qu'il est possible de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle. Ceci permet d'échapper aux optimas locaux. C'est une méthode qui a été inspirée du recuit des métaux en métallurgie.

La recherche Tabou

Le principe de la méthode est le suivant : à chaque itération le voisinage (complet ou partiel) de la solution courante est examiné et la solution minimisant l'augmentation du coût est sélectionnée. Pour éviter le phénomène de cyclage, la méthode interdit de revisiter une solution déjà visitée. Pour cela, une liste taboue contenant les solutions visitées est utilisée.

Les algorithmes génétiques

Les algorithmes génétiques fonctionne sur une analogie avec la reproduction des êtres vivants. On part d'une population (ensemble de solutions) initiale sur laquelle des opérations de reproduction, de croisement ou de mutation vont être réalisées dans l'objectif d'exploiter au mieux les caractéristiques et propriétés de cette population. Ces opérations doivent mener à une amélioration (en terme de qualité des solutions) de l'ensemble de la population puisque les bonnes solutions sont encouragées à échanger, par croisement, leurs caractéristiques et à engendrer des solutions encore meilleures.

Chapitre 2

Ordonnancement par batchs et état de l'art

Ce chapitre est consacré aux principaux résultats qui existent dans le domaine de l'ordonnancement par batchs. Nous commençons par énumérer les principales motivations de l'ordonnancement par batchs. Ensuite, nous proposons quelques notations et définitions nécessaires. Par la suite, nous donnons une classification de ces problèmes, similaire à celle qui existe pour les problèmes de l'ordonnancement classique. Enfin, nous terminons le chapitre par l'état de l'art.

2.1 Motivation

Dans les problèmes de l'ordonnancement classique, deux hypothèses importantes sont communément considérées :

(a) à chaque instant, une machine ne peut traiter qu'une seule tâche à la fois et

(b) à chaque instant, une tâche ne peut être traitée que par, au plus, une machine.

En pratique, on rencontre de nombreuses situations où ces hypothèses sont à lever. Dans la suite de cette thèse nous allons nous intéresser aux problèmes de l'ordonnancement où la première hypothèse est levée, c'est-à-dire les problèmes de l'ordonnancement où une machine peut traiter plusieurs tâches simultanément. Nous appellerons ce type de machines "machines à traitement par batch". Citons quelques situations caractéristiques :

1. Chauffage simultané des pièces dans un four (dans l'industrie du verre, du métal, etc.), on attend d'avoir rempli un four avant de le mettre en marche, ceci afin de

minimiser les facteurs énergétiques. Il faut donc regrouper les pièces ayant des caractéristiques de cuisson identiques et correspondant à la capacité d'un four.

2. Laminage simultané des pièces (dans l'industrie métallurgique).
3. Peinture simultanée des pièces.
4. Transport simultané des pièces par bateau, camion, véhicule autoguidé, etc. où on regroupe les produits répondant à une certaine norme pour les transporter d'un endroit à un autre.

Dans toutes ces situations, le regroupement des tâches en batchs pour subir un certain traitement est nécessaire, et les tâches d'un même batch (groupe de tâches) pouvant s'exécuter simultanément doivent être compatibles (les mêmes propriétés physiques ou chimiques, formes, couleurs, longueurs, etc.).

La capacité de la machine peut être le nombre maximal de pièces qu'elle peut recevoir, ou une valeur physique telle que le poids, le volume ou la longueur (ou hauteur) des pièces présentes dans le batch.

La motivation principale de l'ordonnancement par batchs est l'ordonnancement des opérations de chauffage dans l'industrie des semi-conducteurs. L'étape finale dans la production des circuits intégrés est l'opération de chauffage, où les puces sont chargées sur une planche, ensuite placées dans un four et exposées à de hautes températures. Le but de l'opération de chauffage est d'exposer les puces au stress thermique sur une grande période afin de déterminer d'éventuels défauts de fabrication.

L'objectif, pour ces problèmes de l'ordonnancement par batchs, est de déterminer le nombre de batchs et la capacité de chacun de ces batchs afin de minimiser (ou maximiser) le critère d'optimisation considéré.

2.2 Notations et définitions

Pour une bonne compréhension des différents problèmes que nous allons étudier, nous commençons par proposer les notations suivantes :

n : le nombre de tâches.

p_i : le temps de traitement de la tâche numéro i.

r_i : la date de disponibilité de la tâche numéro i.

d_i : la date échue de la tâche numéro i.

D : une date échue commune pour toutes les tâches.

C_i : la date de fin de traitement de la tâche numéro i.

w_i : un poids associé à la tâche numéro i.

$L_i = C_i - d_i$: tardiveté (décalage temporel ou retard algébrique) de la tâche numéro i.

$T_i = max\{C_i - d_i, 0\}$: retard de la tâche numéro i.

$U_i = \begin{cases} 1 \; si \; C_i > d_i \\ 0 \; sinon \end{cases}$: indicateur de retard de la tâche numéro i.

$F_i = C_i - r_i$: le temps de flot (flow time) de la tâche numéro i.

NT : pas de tâches en retard (No tardy jobs)

Définition 2.1 *Soient $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ deux vecteurs de nombres entiers. On dit que x et y sont en accord (agreable en anglais), ce qu'on note par $x \uparrow y \uparrow$, lorsque : $\forall i, j \; (i \neq j)$ si $x_i < x_j$ alors $y_i \leq y_j$.*

D'une manière équivalente, x et y sont en accord s'il existe une matrice de permutations P telle que les éléments du vecteur Px soient rangés par ordre croissant et les éléments du vecteur Py soient rangés par ordre croissant.

Dans le cas de trois vecteurs, on utilise la notation $x \uparrow y \uparrow z \uparrow$ qui signifie que x, y et z sont en accord (ou encore que x est en accord avec y, y est en accord avec z et x est en accord avec z).

2.3 Classification

Etant donné la diversité des problèmes de l'ordonnancement, un formalisme permettant de distinguer les différents problèmes de l'ordonnancement entre eux et de les classifier est utilisé. Ce formalisme comporte trois champs $\alpha/\beta/\gamma$ permettant de décrire les différentes entités d'un problème d'ordonnancement.

Afin de faciliter la description et la classification de ces problèmes de l'ordonnancement par batchs en présence de machines à traitement par batch, nous allons adopter ce même formalisme et l'adapter au problème étudié.

2.3.1 Champ α : ressources

Ce champ α décrit l'environnement des ressources utilisées (machines) : type de machines, nombre de machines et type d'atelier. Il est composé de plusieurs sous-champs et désigné par $\alpha = \alpha_1\alpha_2(\ldots \longrightarrow \alpha_1\alpha_2)$.

- $\alpha_1 \in \{P, B\}$ décrit le type de machine.
  - $\alpha_1 = P$: machine discrète.
  - $\alpha_1 = B$: machine à traitement par batch.
- $\alpha_2 \in \{\emptyset, k\}$ dénote le nombre de machines.
  - $\alpha_2 = \emptyset$: nombre de machines variable.
  - $\alpha_2 = k$: nombre de machines égal à k ($k \in \mathrm{IN}^*$).
- $\longrightarrow$: indique que l'atelier est de type flow shop.

2.3.2 Champ β : contraintes

Ce champ β décrit les différentes contraintes et caractéristiques des machines et des tâches. Il est composé de 6 sous-champs et désigné par $\beta = \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6$.

- $\beta_1 \in \{\emptyset, CF, IF, G = (V, E), G = (S_1, S_2; E), G = (S, K; E), G = (K_1, K_2; E), G = INT, \ldots\}$.
  - $\beta_1 = \emptyset$: pas de contraintes d'incompatibilité entre les tâches (toutes les tâches sont deux à deux compatibles).
  - $\beta_1 = CF$: tâches divisées en familles compatibles.
  - $\beta_1 = IF$: tâches divisées en familles incompatibles
  - $\beta_1 =$ "$G = (V, E)$" : les contraintes de compatibilité entre les tâches sont représentées sous forme d'un graphe quelconque.
  - $\beta_1 =$ "$G = (S_1, S_2; E)$" : les contraintes de compatibilité entre les tâches sont représentées sous forme d'un graphe biparti.
  - $\beta_1 =$ "$G = (S, K; E)$" : les contraintes de compatibilité entre les tâches sont représentées sous forme d'un graphe scindé.
  - $\beta_1 =$ "$G = (K_1, K_2; E)$" : les contraintes de compatibilité entre les tâches sont représentées sous forme d'un graphe complémentaire d'un graphe biparti.
  - $\beta_1 =$ "$G = INT$" : les contraintes de compatibilité entre les tâches sont représentées sous forme d'un graphe d'intervalles.
  - *etc.*

- $\beta_2 \in \{b, b = k, b \geq n, ba, ba = a, ba = \infty\}$.
  - $\beta_2 = b$: la capacité de la machine à traitement par batch est quelconque.
  - $\beta_2 =$ "$b = k$" : la capacité de la machine à traitement par batch est une constante égale à k $(1 < k < n)$.
  - $\beta_2 =$ "$b \geq n$" : la capacité de la machine à traitement par batch est infinie.
    (pour ces trois premiers cas, b est un nombre de tâches et pour les trois cas suivants, ba est soit un poids, soit un volume, etc. et la capacité est donnée par le poids total, le volume total, etc.)
  - $\beta_2 = ba$: la capacité de la machine à traitement par batch est quelconque.
  - $\beta_2 =$ "$ba = a$" : la capacité de la machine à traitement par batch est une constante égale à a.
  - $\beta_2 =$ "$ba = \infty$" : la capacité de la machine à traitement par batch est infinie.

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

- $\beta_4 \in \{\emptyset, p_i = p, p_{mi}, p_{mi} = p\}$.
  - $\beta_4 = \emptyset$: les temps de traitement des tâches sont différents.
  - $\beta_4 =$ "$p_i = p$" : les temps de traitement des tâches sont tous égaux à p (constants).
  - $\beta_4 = p_{mi}$: le temps de traitement d'une tâche diffère d'une machine à l'autre et les temps de traitement des tâches sont différents.
  - $\beta_4 =$ "$p_{mi} = p$" : le temps de traitement d'une tâche diffère d'une machine à l'autre et les temps de traitement des tâches sont tous égaux sur une même machine.

- $\beta_5 \in \{\emptyset, d_i, \tilde{d}_i\}$.
  - $\beta_5 = \emptyset$: il n'y a pas des dates échues pour les tâches ou il existe des dates échues pour les tâches dans le cas où le critère d'optimisation serait en fonction de ces dernières.
  - $\beta_5 = d_i$: il existe des dates échues pour les tâches qui sont des dates de fin de traitement souhaitées.
  - $\beta_5 = \tilde{d}_i$: il existe des dates d'échéances impératives pour les tâches qui sont des dates de fin de traitement impératives.

- $\beta_6 \in \{\emptyset, a_i\}$.
  - $\beta_6 = \emptyset$: seules les temps de traitement, les dates de disponibilité ou les dates échues sont prises en considération.
  - $\beta_6 = a_i$: à chaque tâche est associé un poids, un volume, une hauteur, etc.

Remarque 2.1 *Dans le cas où les tâches seraient divisées en familles :*

1- les temps de traitement des tâches d'une même famille sont identiques.
2- Les tâches d'une même famille sont deux à deux compatibles.

2.3.3 Champ γ : critère d'optimisation

Le troisième champ γ décrit le critère d'optimisation. On cherche à minimiser γ avec : $\gamma \in \{C_{max}, \Sigma C_i, \Sigma w_i C_i, T_{max}, \Sigma T_i, \Sigma w_i T_i, L_{max}, \Sigma U_i, \Sigma w_i U_i, \Sigma|C_i - D|, ...\}$.

où $X_{max} = \max\limits_{1 \leq i \leq n} \{X_i\}$, $\Sigma X_i = \sum\limits_{i=1}^{n} X_i$ et $\Sigma w_i X_i = \sum\limits_{i=1}^{n} w_i X_i$

avec $X \in \{C, F, L, T, U\}$ et

- C_{max} : le makespan (date de fin de traitement de l'ensemble des tâches).
- ΣC_i : la somme des dates de fin de traitement.
- $\Sigma w_i C_i$: la somme pondérée des dates de fin de traitement.
- T_{max} : le plus grand retard.
- ΣT_i : la somme des retards.
- $\Sigma w_i T_i$: la somme pondérée des retards.
- L_{max} : le décalage temporel maximum.
- ΣU_i : la somme du nombre de tâches en retard.
- $\Sigma w_i U_i$: la somme pondérée du nombre de tâches en retard.

– $\Sigma|C_i - D|$: la somme des retards et avances des tâches.

– etc.

2.3.4 Exemples

Considérons les exemples suivants :

Exemple 2.1 *soit le problème* $B1/IF, b, r_i/L_{max}$. *Il indique qu'on veut traiter n tâches indépendantes sur une (seule) machine à traitement par batch de capacité variable. Chaque tâche est caractérisée par une date de disponibilité, un temps de traitement et une date d'échue. Les tâches sont divisées en familles incompatibles, c.-à-d. les tâches d'une même famille sont compatibles entre elles et les tâches de deux familles distinctes sont incompatibles entre elles. On veut minimiser le plus grand décalage temporel.*

Exemple 2.2 *Soit le problème* $B1/ba, a_i/C_{max}$. *Il indique qu'on veut traiter n tâches indépendantes sur une (seule) machine à traitement par batch de capacité variable. Chaque tâche est caractérisée par un poids et un temps de traitement. Les tâches sont toutes compatibles. On veut minimiser la date de fin de traitement de l'ensemble des tâches.*

Exemple 2.3 *Soit le problème* $B1 \longrightarrow P1/b \geq n, p_2 = p/\Sigma C_i$. *Il indique qu'on veut traiter n tâches indépendantes sur une (seule) machine à traitement par batch de capacité infinie ensuite sur une machine ordinaire (flow shop). Chaque tâche est caractérisée par un temps de traitement sur la première machine et toutes les tâches ont un même temps de traitement sur la deuxième machine. Les tâches sont toutes compatibles. On veut minimiser la somme des dates de fin de traitement.*

2.4 Etat de l'art

Ce paragraphe est consacré à l'état de l'art. On y trouve, différentes complexités de différents problèmes de l'ordonnancement par batchs rencontrés dans la littérature ainsi que les méthodes de résolution existantes. Tous ces différents problèmes de l'ordonnancement par batchs sont présentés dans les trois sous paragraphes suivants, et sont classés en trois catégories : problèmes polynomiaux, problèmes difficiles et problèmes ouverts (complexité inconnue).

D'autres problèmes plus spécifiques, le plus souvent industriels et difficilement classifiables, ont été étudiés dans la littérature, pour plus d'informations consulter la bibliographie.

2.4.1 Problèmes polynomiaux

Les tableaux 2.1 et 2.2 résument les principaux résultats de la complexité algorithmique concernant les problèmes polynomiaux de l'ordonnancement par batchs.

Type	Complexité algorithmique	Référence
$B1/b, r_i, p_i = p/C_{max}$	$O(n)$	[80]
$B1/b, r_i, p_i = p, \tilde{d}_i/C_{max}$ avec $r_i \uparrow d_i \uparrow$	$O(n^2)$	[80]
$B1/b/T_{max}$ avec $p_i \uparrow d_i \uparrow$	$O(bnlog(nmax\{p_i\}))$	[89]
$B1/b, p_i = p/T_{max}$	$O(nlogn)$	[89]
$B1/b, r_i, p_i = p/T_{max}$ avec $r_i \uparrow d_i \uparrow$	$O(bnlog((n-1)p))$	[89]
$B1/b, r_i, p_i = p/\Sigma U_i$ avec $r_i \uparrow d_i \uparrow$	$O(bn^2)$	[89]
$B1/b/\Sigma U_i$ avec $p_i \uparrow d_i \uparrow$	$O(bn^2)$	[89]
$B1/b, p_i = p/\Sigma U_i$	$O(nlogn)$	[89]
$B1/b, r_i/T_{max}$ avec $r_i \uparrow p_i \uparrow d_i \uparrow$	$O(bnlog\Sigma p_i)$	[90]
$B1/b, r_i/\Sigma U_i$ avec $r_i \uparrow p_i \uparrow d_i \uparrow$	$O(bn^2)$	[90]
$B1/b, r_i, p_i = p/\Sigma Ci$	$O(n^3)$	[123]
$B1/b, r_i, p_i = p/L_{max}$ avec $r_i \uparrow d_i \uparrow$	$O(n^3)$	[123]
$B1/b \geq n, r_i, p_i = p/\Sigma w_i C_i$	$O(n^3)$	[47]
$B1/b \geq n, r_i/\Sigma w_i C_i$ avec $r_i \in \{0, r\}$	$O(n^2 logn)$	[47]
$B1/b \geq n/C_{max}$	$O(n)$	[34]
$B1/b \geq n/\Sigma U_i$	$O(n^3)$	[34]
$B1/b \geq n/\Sigma C_i$	$O(nlogn)$	[34]
$B1/b \geq n/\Sigma w_i C_i$	$O(nlogn)$	[34]
$B1/b \geq n/L_{max}$	$O(n^2)$	[34]
$B1/b, \tilde{d}_i/C_{max}$	$\min\{O(nlogn), O(\frac{n^2}{b})\}$	[34]

TAB. 2.1: *Problèmes polynomiaux.*

Type	Complexité algorithmique	Référence
$B1/IF, b/\Sigma w_i C_i$	$O(nlogn)$	[121]
$B1/IF, b/C_{max}$	$O(n)$	[121]
$B1/IF, b/L_{max}$	$O(nlogn)$	[121]
$B1/IF, b, r_i/C_{max}$	$O(nlogn)$	[121]
$B1/CF, b = 2/\Sigma C_i$	$O(f^2)$ (f est le nombre de familles)	[74]
$B1 \longrightarrow P1/b, p1_i = p/C_{max}$	$O(nlogn)$	[3]
$P1 \longrightarrow B1/b, p2_i = p/C_{max}$	$O(nlogn)$	[3]
$B1 \longrightarrow B1/b_1, b_2, p1_i = p, p2_i = p/C_{max}$	$O(n^3)$	[3]
$P1 \longrightarrow B1/b, p2_i = p/\Sigma C_i$	$O(n^3)$	[3]
$B1 \longrightarrow B1/b_1, b_2, p1_i = p, p2_i = p/\Sigma C_i$	$O(n^3)$	[3]
$B1 \longrightarrow P1/IF, b/C_{max}$	$O(nlogn)$	[3]
$B1 \longrightarrow P1/b = k, p1_i = p/C_{max}$	$O(nlogn)$	[100]
$B1 \longrightarrow P1/b = k, p2_i = p/C_{max}$	$O(n^2)$	[100]
$B1 \longrightarrow P1/b \geq n/C_{max}$	$O(n^3)$	[100]
$B1 \longrightarrow B1/b \geq n/C_{max}$	$O(nlogn)$	[101]
$B1 \longrightarrow B1 \longrightarrow B1/b \geq n/C_{max}$	$O(n^{22})$	[101]
$B1/G = (K_1, K_2; E), b \geq n, p_i \in \{1, t\}/C_{max}$	$O(n^2)$	[46]
$B1/G = INT, b, p_i = 1/C_{max}$	$O(n^2)$	[55]
$B1/G = INT, b \geq n/C_{max}$	$O(n^3)$	[55]

TAB. 2.2: *Problèmes polynomiaux (suite).*

2.4.2 Problèmes difficiles

Les tableaux 2.3 et 2.4 résument les principaux résultats concernant les problèmes difficiles de l'ordonnancement par batchs. Un problème de décision (resp. d'optimisation combinatoire) est dit difficile s'il est NP-complet (resp. NP-difficile).

Type	Algorithmes proposés [1]	Référence
$B1/b, r_i/T_{max}$		[89]
$B1/b, r_i/\Sigma U_i$		[89]
$Bm/b, r_i/C_{max}$	H	[89]
$Bm/b, r_i/L_{max}$	H	[89]
$Bm/b, r_i/L_{max}$	H	[89]
avec $p_i \uparrow d_i \uparrow$		
$B1/b, r_i/NT$		[90]
avec $r_i \uparrow d_i \uparrow$		
$B1/b, r_i/T_{max}$		[90]
avec $r_i \uparrow d_i \uparrow$		
$B1/b, r_i/\Sigma U_i$		[90]
avec $r_i \uparrow d_i \uparrow$		
$B1/b \geq n/\Sigma w_i U_i$	$O(n^2 P)$	[34]
$B1/b \geq n/\Sigma w_i T_i$	$O(n^2 P)$	[34]
$B1/b = 2/\Sigma U_i$		[34]
$B1/b = 2/L_{max}$		[34]
$B1/b = 2/\Sigma w_i U_i$		[34]
$B1/b = 2/\Sigma w_i T_i$		[34]
$B1/b = 2/\Sigma T_i$		[34]
	$P = \sum_{i=1}^{n} p_i$	
$B1/b \geq n/\Sigma T_i$		[92]
$B1/b \geq n, r_i/\Sigma f_i$	$O(n^4 p^3)$	[92]
$B1/b \geq n, r_i/f_{max}$	$O(n^4 p^3)$	[92]
	f est une fonction régulière	
	$p = \max_{1 \leq i \leq n}\{r_i\} + \sum_{i=1}^{n} p_i$	
$B1/b \geq n, r_i/\Sigma w_i C_i$		[47]
$B1/b \geq n, r_i/\Sigma w_i C_i$	$O(2^k n^k n log n)$	[47]
avec $r_i \in \{r_1, \ldots, r_k\}$		
$B1/b \geq n, r_i/\Sigma w_i C_i$		[47]
avec $p_i \in \{p_1, \ldots, p_k\}$		

TAB. 2.3: *Problèmes difficiles.*

Type	Algorithmes proposés [1]	Référence
$B1/b, r_i/C_{max}$	B&B, H	[114]
$B1/b = 2, r_i/C_{max}$ avec $r_i \in \{0, r\}$		[91]
$B1/r_i/C_{max}$	H	[91]
$B1/r_i/C_{max}$	$O(nc^k(\sum_{i=1}^{n} p_i)^{k-1})$ k est le nombre de dates de disponibilité distinctes et c une constante	[91]
$B1/ba, a_i/C_{max}$	H	[52]
$B1/ba, a_i/C_{max}$	B&B, H	[120]
$B1/ba, a_i/\Sigma C_i$	B&B, H	[120]
$B1/IF, ba, a_i/\Sigma w_i C_i$	H	[49]
$B1/IF, b/\Sigma T_i$	PD, H	[95]
$Bm/IF, b/\Sigma w_i C_i$	H	[121]
$Bm/IF, b/C_{max}$	H	[121]
$Bm/IF, b, /L_{max}$	H	[121]
$B1/IF, b, r_i/L_{max}$	H	[121]
$B1/IF, b/\Sigma\|C_i - D\|$	PD	[86]
$B1 \longrightarrow P1/b, p1_i = p/\Sigma C_i$	H	[3]
$P1 \longrightarrow B1 \longrightarrow P1/b, p2_i = p/C_{max}$	H	[3]
$B1 \longrightarrow P1/b, p1_i = p/\Sigma C_i$	H	[78]
$B1 \longrightarrow \ldots \longrightarrow B1/b_1, \ldots, b_m$, $p1_i = p, \ldots, pm_i = p/C_{max}$	H	[115]
$B1 \longrightarrow \ldots \longrightarrow B1/b_1, \ldots, b_m$, $p1_i = p, \ldots, pm_i = p/\Sigma C_i$	H	[115]
$B1 \longrightarrow P1/b = k/C_{max}$	H	[100]
$B1 \longrightarrow \ldots \longrightarrow B1/b \geq n/C_{max}$	$O(n^{m(r-2)+1+\lfloor m/r \rfloor})$ si r=nombre de batchs est fixé	[101]
$B1/G = (V, E), b \geq n/C_{max}$ où G est le complémentaire d'un graphe : - biparti - scindé - line-graph L(H) d'un multigraphe biparti régulier avec $\Delta(H) = 3$ - biparti cubique - biparti k-régulier		[46]

TAB. 2.4: *Problèmes difficiles (suite).*

2.4.3 Problèmes ouverts (complexité inconnue)

Le tableau 2.5 résume les principaux résultats pour les problèmes ouverts (de complexité inconnue) de l'ordonnancement par batchs.

Type	Algorithmes proposés [1]	Référence
$B1/b \geq n/\Sigma T_i$		[34]
$B1/b = 2/\Sigma w_i C_i$		[34]
$B1/b/\Sigma C_i$	$O(n^{b(b-1)})$	[34]
$B1/CF, b/\Sigma C_i$	PD	[34]
$B1/b/\Sigma C_i$	B&B	[51]
$B1/b/\Sigma C_i$	H	[51]
$B1/CF, b/\Sigma \lvert C_i - D \rvert$	PD	[86]
$B1/CF, b/\Sigma C_i$	PD	[42]
$B1/CF, b/\Sigma C_i$	OPT, H	[74]
$B1/b/\Sigma w_i C_i$	B&B, H	[122]
$B1/b/\Sigma C_i$	B&B, H	[42]
$Bm/b/\Sigma C_i$	H	[42]

TAB. 2.5: *Problèmes ouverts.*

1.

B&B: Séparation et évaluation (Branch and Bound).

PD: Programmation dynamique.

H: Heuristique.

$\emptyset$: aucun algorithme n'a été proposé.

Chapitre 3

Notations, position du problème et modélisation mathématique

Le début de ce chapitre est consacré à la définition du problème traité dans cette thèse ainsi que les différentes notations nécessaires à sa définition. Un exemple illustratif est aussi donné.

Nous proposons ensuite une modélisation mathématique du problème sous forme d'un programme linéaire en nombres réels avec variables bivalentes. Dans le cas où les dates de disponibilité des tâches seraient toutes nulles, différentes formulations sont simplifiées et présentées.

Un commentaire termine ce chapitre.

3.1 Notations et position du problème

Nous considérons dans cette thèse le problème de l'ordonnancement de n tâches indépendantes $T_1, \ldots, T_n$ sur une machine à traitement par batch en minimisant le temps de fin de traitement (le makespan) où :

1. Nous supposons qu'il existe une relation de compatibilité entre les tâches, deux tâches sont dites compatibles si elles peuvent être traitées simultanément dans un même batch. Cette relation est représentée 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 dernières sont compatibles.

2. La capacité de la machine à traitement par batch peut être finie (elle peut traiter au plus b tâches à la fois) ou infinie (elle peut traiter un nombre illimité de tâches à la fois).

3. Chaque tâche T_i a un temps de traitement p_i et une date de disponibilité r_i.

4. Le temps de traitement d'un batch est égal au maximum des temps de traitement des tâches appartenant à celui-ci.

5. Toutes les tâches d'un même batch commencent leur exécution à une même date et terminent leur exécution à une même date.

6. L'interruption des tâches n'est pas autorisée.

Pour un ordonnancement quelconque des tâches sur la machine à traitement par batch, notons :

- $pb_j = \max_{T_i \in B_j}\{p_i\}$: le temps de traitement du batch B_j.
- $rb_j = \max_{T_i \in B_j}\{r_i\}$: la date de disponibilité du batch B_j.
- sb_j : la date de début de traitement du batch B_j.
- Cb_j : la date de fin de traitement du batch B_j.
- D_i : la date de début de traitement de la tâche T_i.
- C_i : la date de fin de traitement de la tâche T_i (elle est égale à Cb_j si $T_i \in B_j$).

Notons que nous cherchons une partition $B_1, \ldots, B_q$ ($|B_j| \leq b$ pour $j = 1, \ldots, q$) des tâches $T_1, \ldots, T_n$ qui minimise la date de fin de traitement de l'ensemble des batchs. Bien que les tâches d'un même batch forme une clique dans le graphe de compatibilité G, cette partition n'est pas nécessairement la partition minimum en cliques, chacune de taille inférieure ou égale à b, du graphe G.

Remarque 3.1 *Le nombre de batchs n'est pas fixé au départ, mais il est à déterminer.*

3.2 Exemple illustratif

Pour bien illustrer le problème, défini dans le paragraphe précédent, considérons l'exemple 3.1 ci-dessous.

Exemple 3.1 *Nous disposons de 5 tâches indépendantes T_1, T_2, T_3, T_4 et T_5 à traiter sur une machine à traitement par batch. Les temps de traitement des tâches sont donnés dans le tableau 3.1.*

T_i	T_1	T_2	T_3	T_4	T_5
p_i	1	1	3	4	1

TAB. 3.1: *Temps de traitement des tâches de l'exemple 3.1.*

Premier cas : Problème $B1/b = 2/C_{max}$.

Ici les tâches sont deux à deux compatibles, c'est-à-dire que le graphe de compatibilité est complet. La capacité de la machine à traitement par batch est égale à 2. Les tâches sont toutes disponibles à l'instant 0.

La solution optimale est :

- Le nombre de batchs est égal à 3 ;

$$\begin{array}{lll} B_1 = \{T_1, T_2\} & ; & pb_1 = max\{1,1\} = 1 \quad ; \\ B_2 = \{T_3, T_4\} & ; & pb_2 = max\{3,4\} = 4 \quad ; \\ B_3 = \{T_5\} & ; & pb_3 = 1 \quad ; \end{array}$$

- $C_{max} = 6$;

- L'ordonnancement optimal est représenté par la figure 3.1.

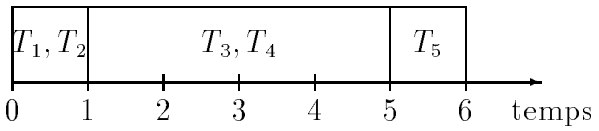


FIG. 3.1: *Ordonnancement optimal du premier cas.*

Second cas : Problème $B1/G = (V, E), b = 2/C_{max}$.

La capacité de la machine à traitement par batch est égale à 2. Les tâches sont toutes disponibles à l'instant 0. Le graphe de compatibilité est représenté par la figure 3.2 ci-dessous ($V = \{1, 2, 3, 4, 5\}$ et $E = \{(1, 2), (1, 3), (2, 3), (2, 4), (4, 5)\}$). Ici, par exemple, les tâches T_1 et T_4 sont incompatibles, on ne doit pas les traiter dans un même batch. Par contre, les tâches T_1 et T_3 sont compatibles, on peut les traiter dans un même batch.

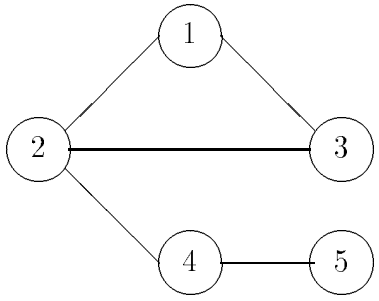


FIG. 3.2: *Graphe de compatibilité du deuxième cas.*

La solution optimale est :

- Le nombre de batchs est égal à 3 ;

$$\begin{array}{lll} B_1 = \{T_1, T_3\} & ; & pb_1 = max\{1, 3\} = 3 \quad ; \\ B_2 = \{T_2, T_4\} & ; & pb_2 = max\{1, 4\} = 4 \quad ; \\ B_3 = \{T_5\} & ; & pb_3 = 1 \quad ; \end{array}$$

- $C_{max} = 8$;
- L'ordonnancement optimal est représenté par la figure 3.3.

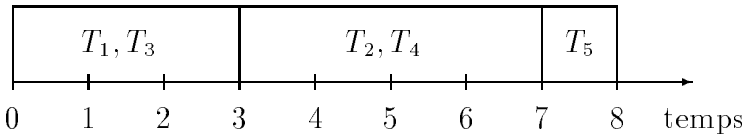


FIG. 3.3: *Ordonnancement optimal du deuxième cas.*

Remarquons que cette solution ne reste par optimale si on ajoute une arête entre les tâches T_3 et T_4.

Troisième cas : Problème $B1/G = (V, E), b = 2, r_i/C_{max}$.

Le graphe de compatibilité est le même que celui représenté par la figure 3.2 du second cas. La capacité de la machine à traitement par batch est toujours égale à 2. Les dates de disponibilité sont données dans le tableau 3.2. Comme les tâches d'un même batch commencent leur exécution à une même date alors chaque batch n'est disponible que lorsque toutes ses tâches sont disponibles.

T_i	T_1	T_2	T_3	T_4	T_5
r_i	0	0	1	3	5

TAB. 3.2: *Dates de disponibilité des tâches de l'exemple 3.1.*

La solution optimale est la suivante :

- Le nombre de batchs est égal à 3 ;

$$\begin{array}{lllll} B_1 = \{T_1\} & ; & pb_1 = 1 & ; & rb_1 = 0 \quad ; \\ B_2 = \{T_2, T_3\} & ; & pb_2 = max\{1,3\} = 3 & ; & rb_2 = max\{0,1\} = 1 \quad ; \\ B_3 = \{T_4, T_5\} & ; & pb_3 = max\{4,1\} = 4 & ; & rb_3 = max\{3,5\} = 5 \quad ; \end{array}$$

- $C_{max} = 9$;

- L'ordonnancement optimal est représenté par la figure 3.4.

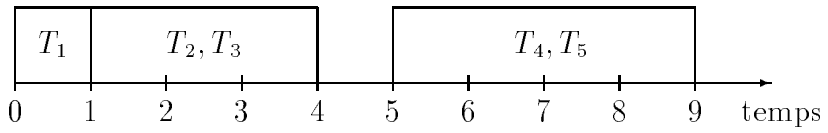


FIG. 3.4: *Ordonnancement optimal du troisième cas.*

Remarque 3.2 *Si on augmente la capacité d'une machine à traitement par batch, la durée totale de l'ordonnancement peut ne pas diminuer. Dans le pire des cas, cette durée reste inchangée (car on peut garder la même solution).*

3.3 Modélisation mathématique

Considérons les variables bivalents suivantes :

$$x_{ij} = \begin{cases} 1 & si\ la\ tâche\ T_i\ appartient\ au\ batch\ B_j \\ 0 & sinon \end{cases}$$

pour $i = 1, \ldots, n$ et $j = 1, \ldots, n$ (le nombre maximal possible de batches est égal à n)

et soient

$$a_{ij} = \begin{cases} 1 & si\ les\ tâches\ T_i\ et\ T_j\ sont\ compatibles \\ 0 & sinon \end{cases}$$

pour $i = 1, \ldots, n$ et $j = 1, \ldots, n$ avec $i \neq j$.

3.3.1 Problème $B1/G = (V, E), r_i, b/C_{max}$

Ce problème se traduit par le programme linéaire (MG), en variables bivalentes et réelles, présenté ci-dessous.

La fonction objectif (3.1) vise la minimisation de la date de fin de traitement du dernier batch. La contrainte (3.2) assure qu'une tâche appartient à un et un seul batch. La contrainte (3.3) interdit au nombre de tâches d'un batch de dépasser la capacité de la machine à traitement par batch. La contrainte (3.4) interdit le chevauchement de deux batchs. La contrainte (3.5) assure le respect de la date de disponibilité de chaque tâche. La contrainte (3.6) interdit à deux tâches incompatibles d'appartenir à un même batch.

$$min\ Cb_n \qquad j = 1, \ldots, n \tag{3.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} \leq b \qquad j = 1, \ldots, n \tag{3.3}$$

$$Cb_j - x_{ij}p_i \geq Cb_{j-1} \qquad i = 1, \ldots, n\ et\ j = 2, \ldots, n \tag{3.4}$$

$$\sum_{j=1}^{n} x_{ij}Cb_j - p_i \geq r_i \qquad i = 1, \ldots, n \tag{3.5}$$

$$x_{ij} + x_{i'j} - a_{ii'} \leq 1 \qquad i = 1, \ldots, n,\ i' = 1, \ldots, n,\ i \neq i'\ et\ j = 1, \ldots, n \tag{3.6}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n\ et\ j = 1, \ldots, n \tag{3.7}$$

$$Cb_j \in IR \qquad j = 1, \ldots, n \tag{3.8}$$

Remarque 3.3 *Si, pour une solution optimale quelconque, on a* $Cb_j = Cb_{j-1}$ *ou* $\sum_{i=1}^{n} x_{ij} = 0$ *pour une certaine valeur* j*, alors le batch* B_j *est vide. Le nombre de batches non vides est, donc, égal à* $\sum_{j=1}^{n} min\{1, \sum_{i=1}^{n} x_{ij}\}$.

3.3.2 Problème $B1/G = (V, E), b \geq n, p_i = p/C_{max}$

Nous proposons dans ce paragraphe une modélisation mathématique (SM1) du problème, sous forme d'un programme linéaire en variables bivalentes et réelles, dans le cas où les dates de disponibilité seraient toutes nulles, les temps de traitement identiques et la capacité de la machine à traitement par batch infinie.

$$min \sum_{j=1}^{n} pb_j \tag{3.9}$$

$$x_{ij} + x_{i'j} - a_{ii'} \leq 1 \qquad i = 1, \ldots, n,\ i' = 1, \ldots, n,\ i \neq i' \text{ et } j = 1, \ldots, n \tag{3.10}$$

$$x_{ij} p \leq pb_j \qquad i = 1, \ldots, n \text{ et } j = 1, \ldots, n \tag{3.11}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \text{ et } j = 1, \ldots, n \tag{3.12}$$

$$pb_j \in IR \qquad j = 1, \ldots, n \tag{3.13}$$

Dans toute solution optimale, nous avons $pb_j \in \{0, p\}$ pour $j = 1, \ldots, n$

3.3.3 Problème $B1/G = (V, E), b \geq n/C_{max}$

Dans le cas où les dates de disponibilité seraient toutes nulles et la capacité de la machine à traitement par batch infinie, nous proposons dans ce paragraphe une modélisation mathématique (SM2) du problème, sous forme d'un programme linéaire en variables bivalentes et réelles.

$$min \sum_{j=1}^{n} pb_j \tag{3.14}$$

$$x_{ij} + x_{i'j} - a_{ii'} \leq 1 \qquad i = 1, \ldots, n,\ i' = 1, \ldots, n,\ i \neq i' \text{ et } j = 1, \ldots, n \tag{3.15}$$

$$x_{ij} p_i \leq pb_j \qquad i = 1, \ldots, n \text{ et } j = 1, \ldots, n \tag{3.16}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \text{ et } j = 1, \ldots, n \tag{3.17}$$

$$pb_j \in IR \qquad j = 1, \ldots, n \tag{3.18}$$

pb_j est soit nul si aucune tâche n'est affectée au batch B_j, soit égal au maximum des temps de traitement des tâches appartenant au batch B_j.

3.3.4 Problème $B1/G = (V, E), b, p_i = p/C_{max}$

Dans ce paragraphe est présentée, sous forme d'un programme linéaire en variables bivalentes et réelles, une modélisation mathématique (SM3) du problème dans le cas où les dates de disponibilité seraient toutes nulles, les temps de traitement identique et la capacité de la machine à traitement par batch finie.

$$min \sum_{j=1}^{n} pb_j \tag{3.19}$$

$$x_{ij} + x_{i'j} - a_{ii'} \leq 1 \qquad i = 1, \ldots, n,\ i' = 1, \ldots, n,\ i \neq i' \ et \ j = 1, \ldots, n \tag{3.20}$$

$$x_{ij} p \leq pb_j \qquad i = 1, \ldots, n \ et \ j = 1, \ldots, n \tag{3.21}$$

$$\sum_{i=1}^{n} x_{ij} \leq b \qquad j = 1, \ldots, n \tag{3.22}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \ et \ j = 1, \ldots, n \tag{3.23}$$

$$pb_j \in IR \qquad j = 1, \ldots, n \tag{3.24}$$

Dans toute solution optimale, nous avons $pb_j \in \{0, p\}$ pour $j = 1, \ldots, n$.

3.3.5 Problème $B1/G = (V, E), b/C_{max}$

Nous proposons dans ce paragraphe une modélisation mathématique (SM4) du problème, sous forme d'un programme linéaire en variables bivalentes et réelles, dans le cas où les dates de disponibilité seraient toutes nulles et la capacité de la machine à traitement par batch finie.

$$min \sum_{j=1}^{n} pb_j \tag{3.25}$$

$$x_{ij} + x_{i'j} - a_{ii'} \leq 1 \qquad i = 1, \ldots, n,\ i' = 1, \ldots, n,\ i \neq i' \ et \ j = 1, \ldots, n \tag{3.26}$$

$$x_{ij} p_i \leq pb_j \qquad i = 1, \ldots, n \ et \ j = 1, \ldots, n \tag{3.27}$$

$$\sum_{i=1}^{n} x_{ij} \leq b \qquad j = 1, \ldots, n \tag{3.28}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \ et \ j = 1, \ldots, n \tag{3.29}$$

$$pb_j \in IR \qquad j = 1, \ldots, n \tag{3.30}$$

pb_j est soit nul si aucune tâche n'est affectée au batch B_j, soit égal au maximum des temps de traitement des tâches appartenant au batch B_j.

3.3.6 Dimensions de chaque modèle

Le nombre de variables et le nombre de contraintes d'un modèle mathématique linéaire sont deux indices par lesquels on peut mesurer la dimension et l'efficacité de la modélisation donnée. Le nombre de variables de chaque modèle est de n^2 variables bivalentes et n variables réelles. Le nombre de contraintes de chaque modèle est donné dans le tableau 3.3.

Modèle	Nombre de contraintes
MG	$n^3 + 2n$
SM1	n^3
SM2	n^3
SM3	$n^3 + n$
SM4	$n^3 + n$

TAB. 3.3: *Nombres de contraintes des différents modèles.*

Le but de l'expérience est d'avoir une idée sur la capacité de résoudre les différents sous-problèmes avec ces modèles. Le jugement final sur la possibilité d'application de ces modèles dépend du logiciel et des équipements informatiques disponibles.

3.4 Quelques remarques

Le problème $Bm/b = 1/C_{max}$ est équivalent au problème d'ordonnancement sur machines parallèles $Pm//C_{max}$. En conséquence, ces problèmes d'ordonnancement par batchs sont NP-difficiles pour $m \geq 2$. Il reste la classe des problèmes d'ordonnancement sur une machine à traitement par batch avec $b \geq 2$. Notons que le cas $b = 1$ coïncide avec le problème d'ordonnancement classique sur une seule machine.

Le problème $B1/G = (V, E), b \geq n, p_i = 1/C_{max}$ peut être résolu en temps polynomial si le problème de la partition du graphe G en nombre minimum de cliques est polynomial. De ce fait, si s est le nombre minimum de cliques, alors $C_{max} = s$. Le problème est donc polynomial si le graphe de compatibilité est un graphe parfait.

Nous donnons ci-dessous une liste non exhaustive de graphes où ce problème est résolu en temps polynomial.

- graphes arc-circulaires [62].
- graphes triangulés [61].
- graphes de comparabilité [66].

et les cas particuliers [67] :

- graphes scindés.
- graphes de permutations.
- graphes d'intervalles.

Le problème où G est un graphe d'intervalle a des applications intéressantes. Dans [32] un problème de l'industrie des feuilles de métal est donné.

Demange et al. [46] se sont intéressés aux problèmes d'ordonnancement de tâches sur une machine à traitement par batch dans le cas où les tâches d'un même batch formeraient un stable (c'est-à-dire deux à deux incompatibles). Ils ont considéré le graphe d'incompatibilité qui est le complémentaire du graphe de compatibilité et ils ont traité le cas où la machine à traitement par batch est de capacité infinie (elle peut traiter une infinité de tâches à la fois). Une partie de leur papier a été inspirée des travaux de Boudhar et Finke [25]. Les types de problèmes et les types de graphes étudiés sont résumés dans le chapitre précédent réservé à l'état de l'art.

Chapitre 4

Graphe quelconque

Nous considérons dans ce chapitre le cas d'un graphe quelconque et nous montrons que plusieurs sous-problèmes du problème général sont difficiles en utilisant des techniques spécifiques à chaque sous-problème étudié. Nous utilisons, pour certains types de sous-problèmes, des preuves simples par restriction ou des techniques de remplacement local. Pour d'autres types de sous-problèmes nous utilisons des techniques de construction plus complexes. Il n'est donc pas possible de trouver un algorithme polynomial pour résoudre le problème général ou même certains de ces sous-problèmes, à moins que P=NP.

A la fin de ce chapitre nous montrons que le problème est polynomial si la capacité de la machine à traitement par batch est égale à 2 et les dates de disponibilités des tâches sont toutes nulles.

4.1 Problèmes difficiles

Nous montrons dans ce paragraphe que le problème est difficile lorsque les temps de traitement des tâches sont tous égaux à 1 pour des cas particuliers de machines (capacité fixée à une constante ou capacité infinie) ou de graphes de compatibilité (Clique particulière ou Partition minimum en cliques connue).

4.1.1 capacité finie

Nous considérons dans ce paragraphe le cas où la machine à traitement par batch a une capacité finie. c'est-à-dire qu'elle ne peut traiter qu'un ensemble fini de tâches simultanément. Nous montrons que, lorsque la capacité de la machine à traitement par batch est supérieure ou égale à 3 et toutes les tâches sont disponibles à l'instant zéro,

le problème est difficile. Aussi, nous montrons que, lorsque la capacité de la machine à traitement par batch est égale 2, le problème est difficile en présence de dates de disponibilité.

Ordonnancement statique[1]

Nous considérons dans ce paragraphe le cas où les dates de disponibilité des tâches sont toutes nulles.

Nous montrons que le problème est difficile si les temps de traitement des tâches sont tous identiques à 1 et si la capacité de la machine à traitement par batch est supérieure ou égale à 3.

Théorème 4.1 *Le problème* $B1/G = (V, E), b = k, p_i = 1/C_{max}$ *avec* $k \geq 3$ *est NP-difficile.*

Preuve. Soit PIIS (Partition Into Isomorphic Sub-graphs) le problème de décision suivant : Etant donnés deux graphes $H = (V, E)$ et $H' = (V', E')$ avec $\mid V \mid = q \mid V' \mid$ pour un certain entier $q \neq 0$. Question : Peut-on partitionner les sommets de H en q sous-ensembles disjoints $V_1, \ldots, V_q$ tels que, pour tout i $(1 \leq i \leq q)$, le graphe induit par V_i est isomorphe à H'? PIIS est NP-Complet même si H' est un graphe fixé et complet avec $\mid V' \mid \geq 3$ [59].

Considérons le cas où H' est un graphe complet de taille $\mid V' \mid$ avec $3 \leq \mid V' \mid \leq \frac{n}{2}$ et montrons que PIIS se réduit polynomialement au problème de décision BSCG suivant :
$n = \mid V \mid$,
$T = \{T_1, \ldots, T_n\}$ (à chaque sommet i de V, nous associons une tâche T_i),
$p_i = 1$ pour $i = 1, \ldots, n$,
$b = \mid V' \mid$ avec $3 \leq b \leq \frac{n}{2}$ et
$G = H$.
Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = q$?

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que PIIS a une solution (c'est-à-dire que la réponse à la question posée est 'oui') si et seulement si BSCG a une solution.

Si le problème PIIS a une solution, alors le graphe H peut être partitionné en q cliques $V_1, \ldots, V_q$ où la taille de chaque clique est égale à $\mid V' \mid$. Nous construisons une solution pour BSCG, comme suit : Dans chaque batch B_j $(1 \leq j \leq q)$ nous mettons les tâches qui correspondent à la clique V_j. Le temps de traitement de chaque batch B_j $(1 \leq j \leq q)$ est donc égal à p avec $Cb_j = j$ pour tout j $(1 \leq j \leq q)$ et $\max\limits_{1 \leq j \leq n} \{C_j\} = q$.

1. Les résultats de ce paragraphe sont publiés dans [27]

Si le problème BSCG a une solution avec un makespan inférieur ou égal à q, alors il y a exactement q batchs et dans chaque batch il y a exactement b tâches (car $n = bq$). Comme à chaque batch correspond un graphe complet, alors il existe une partition des sommets de G en q sous-ensembles disjoints $V_1, \ldots, V_q$ tel que pour tout i $(1 \leq i \leq q)$, le graphe induit par V_i est complet et de taille $\mid V' \mid$. □

Dans le théorème 5.1 précédent, la capacité de la machine à traitement par batch est inférieure ou égale à $\frac{n}{2}$.

Ordonnancement dynamique [2]

Dans ce paragraphe nous considérons le cas où les dates de disponibilité des tâches ne sont pas toutes nulles. c'est-à-dire que différentes dates de disponibilité sont allouées aux tâches. Nous montrons que le problème reste difficile même si la capacité de la machine à traitement par batch est égale à 2.

Théorème 4.2 *Le problème* $B1/G = (V, E), b = 2, r_i, p_i = 1/C_{max}$ *est NP-difficile au sens fort.*

Preuve. Soit OCR le problème de décision correspondant au problème d'ordonnancement sous contraintes de ressources $P2/res.11, r_i, p_i = 1/C_{max}$, défini comme suit : Etant donnés deux processeurs P_1 et P_2, n tâches $T_1, \ldots, T_n$ avec des temps de traitement identiques tous égaux à 1, des dates de disponibilité $r_1, \ldots, r_n$, et m ressources $R_1, \ldots, R_m$ (chaque tâche T_i nécessite pour son exécution au plus une unité de chaque ressource $R_1, \ldots, R_m$). Question : Peut-on ordonnancer les tâches $T_1, \ldots, T_n$ sur les processeurs P_1 et P_2 avec un makespan inférieur ou égal à x ? OCR est NP-Complet au sens fort [11].

Soient $R_j(i) = \begin{cases} 1 & \text{si la tâche } T_i \text{ nécessite la ressource } R_j \\ 0 & \text{sinon} \end{cases}$

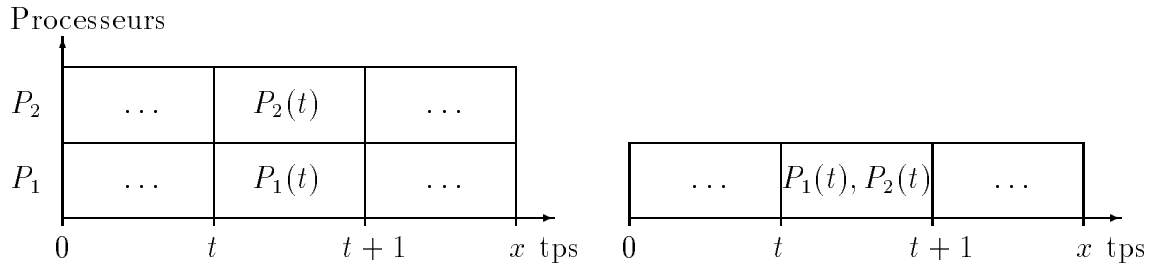
Montrons que OCR se réduit polynomialement au problème de décision BSCG suivant :
Les tâches sont $T_1, \ldots, T_n$ (les mêmes),
$p_i = 1$ pour $i = 1, \ldots, n$,
r_i pour $i = 1, \ldots, n$ (les mêmes dates de disponibilité) et
$G = (V, E)$ où $V = \{1, \ldots, n\}$ et $(i, j) \in E$ si et seulement si $R_k(i) + R_k(j) \leq 1$ pour $k = 1, \ldots, m$ (c.-à-d. les deux tâches T_i et T_j ne nécessite pas une même ressource).
Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = x$?

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que OCR a une solution si et seulement si BSCG a une solution.

2. Les résultats de ce paragraphe sont publiés dans [25]

Si OCR a une solution avec un makespan inférieur ou égal à x, alors il existe une affectation des tâches $T_1, \ldots, T_n$ aux processeurs P_1 et P_2 telle que : $C_{max} \leq x, D_i \geq r_i$ pour $i = 1, \ldots, n$ et $R_k(P_1(t)) + R_k(P_2(t)) \leq 1$ pour $t = 0, \ldots, x-1$ et $k = 1, \ldots, m$ (où $P_j(t)$ indique la tâche affectée au processeur P_j à l'instant t, si $P_j(t) = \emptyset$ alors $R_k(P_j(t)) = 0$ pour $k = 1, \ldots, m$).

Nous construisons une solution pour BSCG comme suit (voir la figure 4.1) : si une tâche est ordonnancée à l'instant t alors nous formons le batch $\{P_1(t), P_2(t)\}$ (les tâches $P_1(t)$ et $P_2(t)$ sont compatibles car $R_k(P_1(t)) + R_k(P_2(t)) \leq 1$ pour $k = 1, \ldots, m$).



a/ Une solution de OCR b/ La solution correspondante de BSCG

FIG. 4.1: *Correspondance entre une solution de BSCG et une solution de OCR.*

Donc BSCG a une solution avec un makespan inférieur ou égal à x.

Si BSCG a une solution avec un makespan inférieur ou égal à x, nous construisons une solution pour OCR comme suit : Les tâches appartenant au batch B_j et ordonnancées à l'instant t, sont dispatchées sur les deux processeurs P_1 et P_2 à l'instant t (voir la figure 4.1). Donc les contraintes de dates de disponibilité sont respectées et, aussi, les contraintes de ressources sont respectées car les tâches sont compatibles. Nous obtenons donc, une solution pour OCR avec un makespan inférieur ou égal à x. □

4.1.2 Capacité infinie [3]

Nous considérons dans ce paragraphe le cas où la machine à traitement par batch a une capacité infinie. c'est-à-dire qu'elle peut traiter toutes les tâches du problème simultanément. Nous montrons que le problème est difficile même si les dates de disponibilité des tâches sont toutes nulles.

Théorème 4.3 *Le problème* $B1/G = (V, E), b = n, p_i = 1/C_{max}$ *est NP-difficile.*

3. Les résultats de ce paragraphe sont publiés dans [28]

Preuve. Soit PIC (Partition Into Cliques) le problème de décision suivant : Etant donnés un graphe $H = (V, E)$ et un entier non nul $q \leq \mid V \mid$. Question : Peut-on partitionner les sommets de H en $k \leq q$ sous-ensembles disjoints $V_1, \ldots, V_k$ tels que pour tout i $(1 \leq i \leq k)$, le sous-graphe induit par V_i est complet ? PIC est NP-Complet pour $q \geq 3$ [59].

Montrons que PIC se réduit polynomialement au problème de décision BSCG suivant :
$n = \mid V \mid$,
$T = \{T_1, \ldots, T_n\}$ (A chaque sommet i de H on fait correspondre une tâche T_i),
$p_i = 1$ pour $i = 1, \ldots, n$,
$b = n$ et
$G = H$.
Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = q$?

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que PIC a une solution si et seulement si BSCG a une solution.

Si le problème PIC admet une solution alors le graphe H peut être partitionné en k cliques $V_1, \ldots, V_k$ où $\mid V_i \mid \leq n$ pour $i = 1, \ldots, k$. On construit une solution pour BSCG de la manière suivante : On forme k batchs $B_1, \ldots, B_k$ où chaque batch B_j contient les tâches correspondant à la clique V_j, ainsi $Cb_j = j$ pour $j = 1, \ldots, k$ avec $\mid B_j \mid \leq n$ pour $j = 1, \ldots, k$. On a donc $\max_{1 \leq i \leq n} \{C_i\} = k$ et le problème BSCG admet une solution de longueur inférieure ou égale à y.

Si le problème BSCG admet une solution de durée inférieure ou égale à q, alors il existe k $(k \leq q)$ batchs $B_1, \ldots, B_k$ avec $Cb_j = j$ pour $j = 1, \ldots, k$ (on ne considère que les ordonnancements sans temps mort puisque les dates de disponibilité sont toutes nulles) et $\mid B_j \mid \leq n$ pour $j = 1, \ldots, k$. Comme à chaque batch correspond un graphe complet, il existe une partition des sommets de G et donc de H en k sous-ensembles disjoints $V_1, \ldots, V_k$ tels que pour tout i $(1 \leq i \leq k)$ le sous-graphe induit par V_i est une clique. Donc PIC admet une solution. □

Le problème est donc difficile même si la capacité de la machine à traitement par batch est infinie. Car une machine à traitement par batch ne peut contenir plus de n tâches.

4.1.3 Clique particulière [4]

Considérons le cas où le graphe de compatibilité contient une clique K de taille $n_2 \geq \frac{n}{4}$ et ayant chaque sommet relié à tous les autres sommets du graphe (notons un tel graphe par $H_{n_1} + K_{n_2}$). Les tâches de la clique K ont toutes une date de disponibilité égale à 1 (notons ceci par $r_K = 1$) et les tâches restantes ont une date de disponibilité nulle

4. Les résultats de ce paragraphe sont publiés dans [28]

(notons ceci par $r_H = 0$). Montrons que ce problème est difficile si la capacité b est variable, donc une donnée du problème.

Théorème 4.4 *Le problème* $B1/H_{n_1} + K_{n_2}, b, r_H = 0, r_K = 1, p_i = 1/C_{max}$ *avec* $n_2 \geq \frac{n_1}{3}$ *et* $n_1 \geq 3$ *(le sous-graphe* H_{n_1} *contient au moins 3 sommets) est NP-difficile.*

Preuve. Soit CLIQUE le problème de décision suivant : Etant donnés un graphe $H = (V, E)$ (H contient au moins 3 sommets) et un entier positif $k \neq 0$. Question : Est-ce que H contient une clique de taille k (un sous-graphe complet à k sommets) ? Ce problème est NP-Complet [109].

Montrons que CLIQUE se réduit polynomialement au problème de décision BSCG suivant :
$n = n_1 + n_2$ avec $n_1 = \mid V \mid$ et $n_2 = (k-1)(n_1 - k)$,
$T = \{T_1, \ldots, T_{n_1}\} \cup \{T_{n_1+1}, \ldots, T_n\}$ (A chaque sommet i du graphe H on fait correspondre une tâche T_i et auxquelles on ajoute $(k-1)(\mid V \mid -k)$ tâches),
$p_i = 1$ pour $i = 1, \ldots, n$,
$r_i = 0$ pour $i = 1, \ldots, n_1$,
$r_i = 1$ pour $i = n_1 + 1, \ldots, n$,
$b = k$, (b est une donnée du problème, $1 < b < n_1$) et
$G = H_{n_1} + K_{n_2} = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3)$ (voir la figure 4.2) où
$V_1 = \{1, \ldots, n_1\}$,
$V_2 = \{n_1 + 1, \ldots, n\}$,
$(i,j) \in E_1$ si et seulement si $(i,j) \in E$,
$(i,j) \in E_2$ si et seulement si $i \in V_2$, $j \in V_2$ et $i \neq j$
(le sous-graphe $K_{n_2} = (V_2, E_2)$ est une clique de taille n_2)
$(i,j) \in E_3$ si et seulement si $i \in V_1$ et $j \in V_2$.
Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = \mid V \mid -k + 1$?

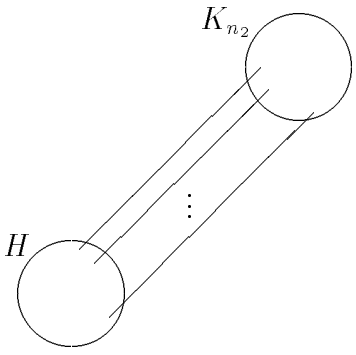


FIG. 4.2: *Construction du graphe de compatibilité G.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que CLIQUE a une solution si et seulement si BSCG a une solution.

Soit $f(b) = n_1 - 3n_2 = 3b^2 - 3(n_1+1)b + 4n_1$. Nous avons $\Delta = 9n_1^2 - 30n_1 + 9 \geq 0$ car $n_1 \geq 3$. Donc $f(b) \leq 0$ pour $b \in [b_1, b_2]$ où b_1 et b_2 sont les racines de l'équation $f(b) = 0$. Comme $1 < b < n_1$, $f(2) = 2(3 - n_1) \leq 0$ et $f(n_1 - 1) = 2(3 - n_1) \leq 0$ alors $f(b) \leq 0$. Donc $n_2 \geq \frac{n_1}{3}$ ou encore $n_2 \geq \frac{n}{4}$.

Supposons que CLIQUE admet une solution, donc il existe une clique de taille k dans le graphe H. On construit une solution pour BSCG comme suit. Supposons, sans perte de généralité, que les sommets du graphe H sont ordonnés de la manière suivante: $V = \{1, \ldots, k\} \cup \{k+1, \ldots, \mid V \mid\}$ où les sommets $1, \ldots, k$ forment une clique. Les batchs sont (voir la figure 4.3):

$B_1 = \{T_1, \ldots, T_k\}$ et
$B_j = \{T_{k+(j-1)}\} \cup \{T_{n_1+(j-2)(k-1)+1}, \ldots, T_{n_1+(j-1)(k-1)}\}$ pour $j = 2, \ldots, n_1 - k + 1$

avec $Cb_j = j$ pour $j = 1, \ldots, n_1 - k + 1$.

Les batchs sont ordonnancés suivant l'ordre croissant de leurs indices.

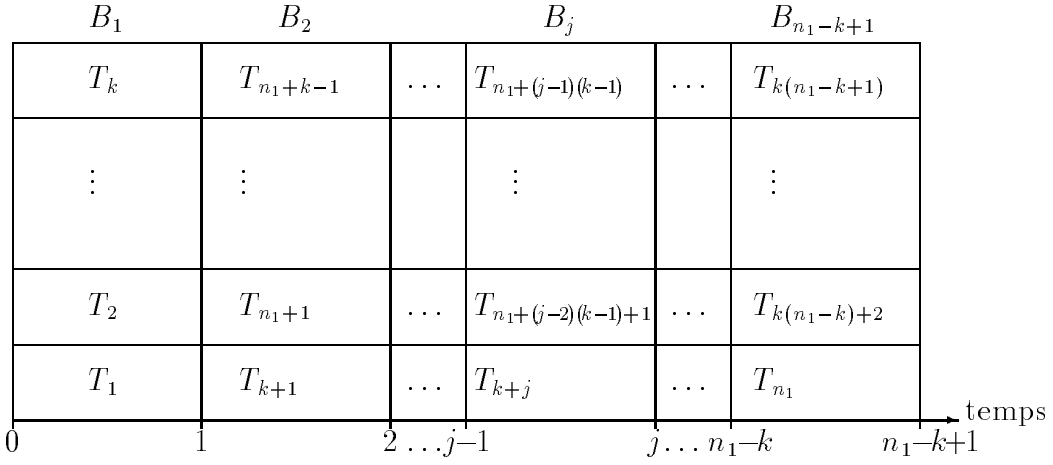


FIG. 4.3: *Construction d'une solution pour BSCG.*

$C_{max} = \max_{1 \leq i \leq n} \{C_i\} = n_1 - k + 1 = \mid V \mid -k + 1.$

Supposons que BSCG admet une solution de durée inférieure ou égale à $\mid V \mid -k + 1$. Comme le nombre de tâches est égal à $n_1 + (k-1)(n_1 - k) = k(n_1 - k + 1)$ et le temps de traitement de chaque tâche est égal à 1, dans chaque batch il y a exactement k tâches, car toutes les tâches ont été ordonnancées avec une durée égale à $n_1 - k + 1$. Comme les tâches T_i $(n_1 + 1 \leq i \leq n)$ ont une date de disponibilité égale à 1, elles seront ordonnancées dans l'intervalle de temps $[1, n_1 - k + 1]$. Le nombre de ces tâches est égal à $(n_1 - k)(k - 1)$. Dans l'intervalle de temps $[1, n_1 - k + 1]$ reste à ordonnancer, $n_1 - k$ tâches parmi les tâches restantes. Finalement, les tâches ordonnancées dans l'intervalle

de temps $[0,1]$ sont au nombre de k et elles appartiennent à $\{T_1,\ldots,T_{n_1}\}$ et forment nécessairement une clique. Donc CLIQUE admet une solution. □

On constate que le problème général est, aussi, difficile si les dates de disponibilité des tâches appartiennent à {0,1} avec, au moins, deux dates de disponibilité distinctes.

Remarque 4.1 *Dans le théorème précédent la capacité de la machine à traitement par batch est inférieure ou égale à* $\frac{n}{2}$.

Preuve. Nous avons $n = n_1 + (k-1)(n_1-k)$, donc $n = -k^2 + (n_1+1)k$ et $\frac{n}{k} = n_1 + 1 - k \geq 2$ (car $k \leq n_1 - 1$). Ainsi, $k \leq \frac{n}{2}$. □

4.1.4 Partition minimum en cliques connue [5]

Supposons qu'on connaît une partition minimum en cliques du graphe de compatibilité G. Soit b la taille maximale des batchs et soit $B_1,\ldots,B_s$ une partition minimum réalisable, c.-à-d. $\mid B_k \mid \leq b$ et s est minimum. Notons ce graphe de compatibilité par $G(K_b^1,\ldots,K_b^s)$. Même dans ce cas, le problème est difficile en présence de dates de disponibilité.

Théorème 4.5 *Le problème* $B1/G(K_b^1,\ldots,K_b^s),b,r_i,p_i=1/C_{max}$ *est NP-difficile.*

Preuve. Soit CLIQUE le problème de décision suivant : Etant donnés un graphe $H = (V,E)$ et un entier $k > 0$. Question : Est-ce que H contient une clique de taille k (un sous-graphe complet à k sommets)? Ce problème est NP-complet [59].

Montrons que CLIQUE se réduit polynomialement au problème de décision BSCG suivant :
$n = n_1 + n_2$ où $n_1 = \mid V \mid$ et $n_2 = (k-1) \mid V \mid$,
$T = \{T_1,\ldots,T_{n_1}\} \cup \{T_{n_1+1},\ldots,T_n\}$ (à chaque sommet i de V est associé une tâche T_i auxquelles nous ajoutons $(k-1) \mid V \mid$ tâches),
$p_i = 1$ pour $i = 1,\ldots,n$,
$r_i = 0$ pour $i = 1,\ldots,n_1$, $r_i = 1$ pour $i = n_1+1,\ldots,n$,
$b = k$ et
$G = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3)$ (voir la figure 4.4) où,
$V_1 = \{1,\ldots,n_1\}$, $V_2 = \{n_1+1,\ldots,n\}$,
$(i,j) \in E_1$ si et seulement si $(i,j) \in E$ $(E_1 = E)$,
$(i,j) \in E_2$ si et seulement si $i \in V_2$, $j \in V_2$ et $i \neq j$
(le sous-graphe $K_{n_2} = (V_2, E_2)$ est une clique de taille n_2) et

5. Les résultats de ce paragraphe sont publiés dans [25]

$(i,j) \in E_3$ si et seulement si $i \in V_1$ et $j \in V_2$.
Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y =\mid V \mid$?

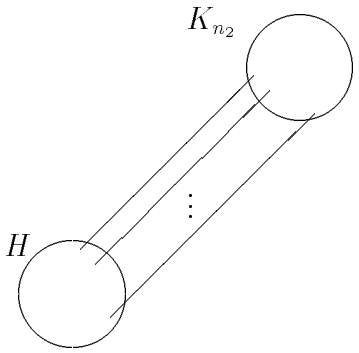


FIG. 4.4: *Construction de G.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que CLIQUE a une solution si et seulement si BSCG a une solution.

Comme $n_1 =\mid V \mid$, $n_2 =\mid V \mid (k-1)$ et K_{n_2} est un graphe complet, le graphe G a une partition en $\mid V \mid$ cliques de taille k (chaque clique est composée d'un sommet de H et $k-1$ sommets de K_{n_2}). Cette partition est optimale, car le nombre de sommets du graphe est égal à $k \mid V \mid$.

Si CLIQUE a une solution, alors il existe une clique de taille k dans le graphe H. On construit une solution pour BSCG comme suit. Supposons, sans perte de généralité, que les sommets de H sont rangés comme suit $V = \{1,\ldots,k\} \cup \{k+1,\ldots,\mid V \mid\}$ où les sommets $1,\ldots,k$ forment une clique. Les batchs sont (voir la figure 4.5) :

$B_1 = \{T_1,\ldots,T_k\}$,
$B_j = \{T_{k+(j-1)}\} \cup \{T_{n_1+(j-2)(k-1)+1},\ldots,T_{n_1+(j-1)(k-1)}\}$ pour $j = 2,\ldots,n_1-k+1$
$B_j = \{T_{w+(j-n_1+k-2)k+1},\ldots,T_{w+(j-n_1+k-1)k}\}$ pour $j = n_1-k+2,\ldots,n_1$
où $w = n_1+(n_1-k)(k-1)$

avec :

$Cb_1 = 1$ et $Cb_j = j$ pour $j = 2,\ldots,n_1$.

Les batchs sont ordonnancés dans l'ordre croissant de leurs indices.

$C_{max} = \max\limits_{1\le i\le n}\{C_i\} = n_1 =\mid V \mid$.

Si BSCG a une solution avec un makespan inférieur ou égal à $n_1 =\mid V \mid$, alors dans chaque batch il y a exactement k tâches, car le nombre de tâches est égal à $n_1 +$

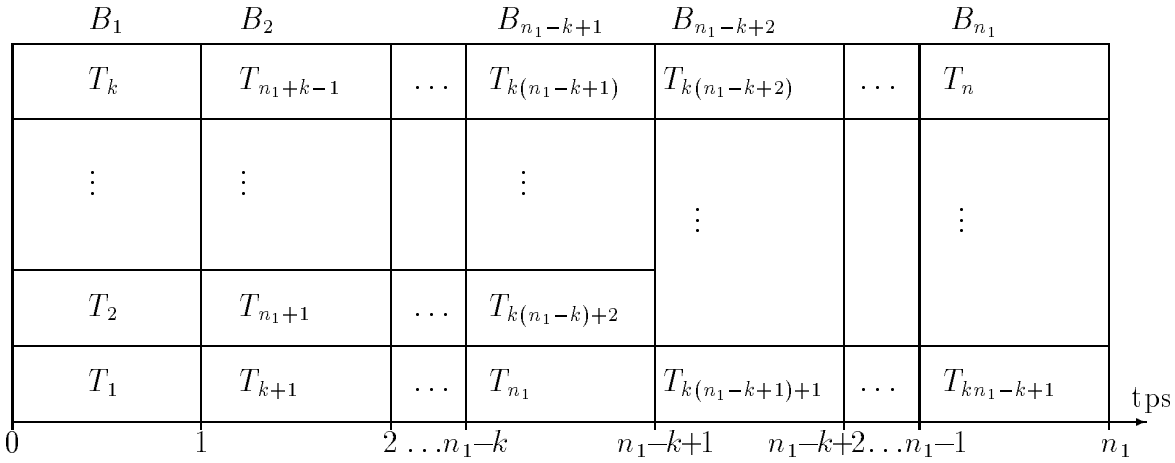


FIG. 4.5: *Construction d'une solution de OL.*

$(k-1)n_1 = kn_1$ et le temps de traitement de toute tâche est égale à 1. Comme les tâches T_i $(n_1+1 \leq i \leq n)$ ont une date de disponibilité égale à 1, alors elles seront ordonnancées dans l'intervalle de temps $[1, n_1]$. Le nombre de ces tâches est égal à $n_1(k-1)$. Dans l'intervalle de temps $[1, n_1]$, sont ordonnancées les $n_1 - k$ tâches restantes parmi les tâches T_i $(1 \leq i \leq n_1)$, car $(n_1-1)k = n_1(k-1) + (n_1-k)$. Finalement, les tâches ordonnancées entre l'instant 0 et l'instant 1 sont au nombre de k et forment nécessairement une clique. Donc CLIQUE a une solution. □

4.2 Problèmes polynomiaux[6]

Nous montrons dans ce paragraphe que, dans le cas où les dates de disponibilité seraient toutes nulles et la capacité de la machine à traitement par batch égale à 2, le problème est résolu par un algorithme polynomial basé sur l'algorithme du couplage de poids maximum.

Théorème 4.6 *Le problème* $B1/G = (V,E), b = 2/C_{max}$ *se réduit au problème du couplage de poids maximum.*

Preuve. Soit A la matrice d'incidence sommet-arête du graphe de compatibilité $G = (V,E)$ où

$$a_{ij} = \begin{cases} 1 & \text{si l'arête } j \text{ est incidente au sommet } i \\ 0 & \text{sinon} \end{cases}$$

pour $i = 1 \ldots n$ et $j = 1 \ldots q$ (q est le nombre d'arêtes).

6. Les résultats de ce paragraphe sont publiés dans [25]

Soit $c_j = max\{p_{s_j}, p_{t_j}\}$ le poids de l'arête j, si l'arête j est incidente aux sommets s_j et t_j.

Le programme linéaire, correspondant à ce problème, s'écrit comme suit :

$$\min C_{max} = (\sum_{j=1}^{q} c_j x_j) + \sum_{i=1}^{n} (1 - \sum_{j=1}^{q} a_{ij} x_j) p_i$$

$$\text{s.c.} \begin{cases} \sum_{j=1}^{q} a_{ij} x_j \leq 1 \text{ for } i = 1, \ldots, n \\ x_j \in \{0, 1\} \text{ for } j = 1, \ldots, q \end{cases}$$

où

- $x_j = \begin{cases} 1 \text{ si les deux tâches reliées par l'arête j sont dans un même batch} \\ 0 \text{ sinon} \end{cases}$
- $\sum_{j=1}^{q} c_j x_j$: la somme des temps de traitement des batchs contenant deux tâches.
- $\sum_{i=1}^{n} (1 - \sum_{j=1}^{q} a_{ij} x_j) p_i$: la somme des temps de traitement des batchs contenant une seule tâche.
- $\sum_{j=1}^{q} a_{ij} x_j \leq 1$ indique que chaque tâche doit appartenir à, au plus, un batch contenant deux tâches.

Nous avons :

$$\begin{aligned} (\sum_{j=1}^{q} c_j x_j) + \sum_{i=1}^{n} (1 - \sum_{j=1}^{q} a_{ij} x_j) p_i &= (\sum_{i=1}^{n} p_i) - (\sum_{i=1}^{n} \sum_{j=1}^{q} a_{ij} x_j p_i - \sum_{j=1}^{q} c_j x_j) \\ &= (\sum_{i=1}^{n} p_i) - (\sum_{j=1}^{q} \sum_{i=1}^{n} a_{ij} p_i x_j - \sum_{j=1}^{q} c_j x_j) \\ &= (\sum_{i=1}^{n} p_i) - \sum_{j=1}^{q} (\sum_{i=1}^{n} a_{ij} p_i - c_j) x_j \\ &= (\sum_{i=1}^{n} p_i) - \sum_{j=1}^{q} (p_{s_j} + p_{t_j} - \max\{p_{s_j}, p_{t_j}\}) x_j \\ &= (\sum_{i=1}^{n} p_i) - \sum_{j=1}^{q} \min\{p_{s_j}, p_{t_j}\} x_j. \end{aligned}$$

Puisque $\sum_{i=1}^{n} p_i$ est une constante supérieure à $\sum_{j=1}^{q} \min\{p_{s_j}, p_{t_j}\} x_j$. Alors, minimiser C_{max} est équivalent à maximiser $\sum_{j=1}^{q} l_j x_j$ où $l_j = \min\{p_{s_j}, p_{t_j}\}$ si l'arête j est incidente aux sommets s_j et t_j. Ainsi, le problème se réduit au problème du couplage de poids maximum. □

L'algorithme MWMA suivant résout le problème $B1/G = (V, E), b = 2/C_{max}$.

Algorithme MWMA ;
début

1- A partir du graphe $G = (V, E)$, construire un nouveau graphe valué $H = (V, E)$ où chaque arête de E est valuée par $min\{p_i, p_j\}$ si l'arête est incidente aux sommets i et j.

2- Déterminer un couplage M de poids maximum dans le graphe H.

3- Former les batchs de la manière suivante :

- Pour chaque couple de M, mettre les deux tâches correspondantes dans un même batch.
- Mettre chacune des tâches restantes dans un batch.

4- Ordonnancer les batchs dans un ordre arbitraire et sans temps mort.

5- La date de fin de traitement est égale à la somme de tous les temps de traitement à laquelle on soustrait la valeur du couplage de poids maximum.

fin.

Corollaire 4.7 *L'algorithme MWMA résout le problème* $B1/G = (V, E), b = 2/C_{max}$ *en* $O(n^3)$.

Preuve. Le nombre maximum d'arêtes possibles dans le graphe G est $\frac{n(n-1)}{2}$. Par le théorème 6.1 et par le fait que le meilleur algorithme connu pour le problème du couplage de poids maximum est en $O(n^3)$. Alors, l'algorithme MWMA résout le problème en $O(n^3)$. □

Exemple 4.1 *Nous disposons de 5 tâches* $T_1, \ldots, T_5$ *dont les temps de traitement sont donnés dans le tableau 4.1. Le graphe de compatibilité est représenté par la figure 4.6.*

T_i	T_1	T_2	T_3	T_4	T_5
p_i	1	3	2	1	4

TAB. 4.1: *Temps de traitement des tâches de l'exemple 4.1.*

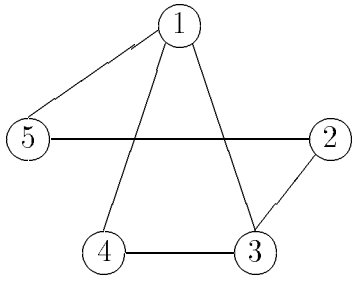


FIG. 4.6: *Graphe de compatibilité de l'exemple 4.1.*

Le déroulement de l'algorithme est le suivant :

1- Construction du graphe $H = (V, E)$ où chaque arête (i, j) est valuée par $min\{p_i, p_j\}$ (voir la figure 4.7).

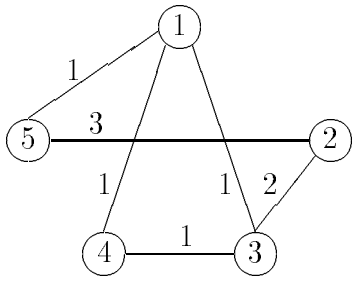


FIG. 4.7: *Le graphe valué $H = (V, E)$.*

2- Un couplage de poids maximum est M={(2,5),(3,4)}.

3- La solution optimale est :

$$B_1 = \{T_2, T_5\} \quad ; \quad pb_1 = 4 \quad ;$$
$$B_2 = \{T_3, T_4\} \quad ; \quad pb_2 = 2 \quad ;$$
$$B_3 = \{T_1\} \quad ; \quad pb_3 = 1 \quad ;$$

4- La date de fin de traitement est égale à 7.

L'ordonnancement optimal est représenté par la figure 4.8.

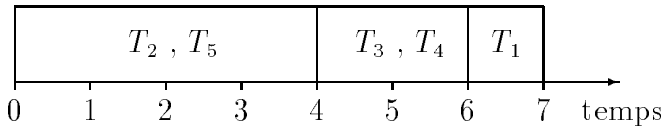


FIG. 4.8: *Ordonnancement optimal de l'exemple 4.1.*

Corollaire 4.8 *Le problème $B1/G = (V, E), b = 2, p_i = 1/C_{max}$ se réduit au problème de couplage maximum.*

Preuve. Dans la preuve du théorème 6.1, nous avons $\min\{p_{s_j}, p_{t_j}\} = 1$. Donc, minimiser C_{max} revient à maximiser $\sum_{j=1}^{q} x_j$. Ainsi, ce problème est équivalent au problème du couplage maximum. □

L'algorithme MCMA suivant résout le problème $B1/G = (V, E), b = 2, p_i = 1/C_{max}$.

Algorithme MCMA ;
début

1- Déterminer un couplage maximum M dans le graphe G.

2- Former les batchs de la manière suivante :

- Pour chaque couple de M, mettre les deux tâches correspondantes dans un même batch.
- Mettre chacune des tâches restantes dans un batch.

3- Ordonnancer les batchs dans un ordre arbitraire et sans temps mort.

4- La date de fin de traitement est égale au nombre de tâches auquel on soustrait la cardinalité du couplage.

fin.

Corollaire 4.9 *L'algorithme MCMA résout le problème* $B1/G = (V, E), b = 2, p_i = 1/C_{max}$ *en* $O(n^{2.5})$.

Preuve. Le nombre maximum d'arêtes possibles dans le graphe G est $\frac{n(n-1)}{2}$. Par le corollaire 6.3 et par le fait que le meilleur algorithme connu pour le problème du couplage maximum est en $O(n^{2.5})$. Alors, l'algorithme MCMA résout le problème en $O(n^{2.5})$. □

Exemple 4.2 *Considérons 5 tâches* $T_1, \ldots, T_5$ *dont les temps de traitement sont tous égaux à 2. Le graphe de compatibilité est représenté par la figure 4.9.*

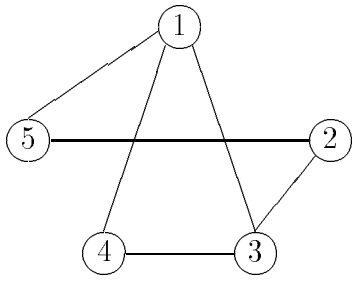


FIG. 4.9: *Graphe de compatibilité de l'exemple 4.2.*

Le déroulement de l'algorithme est le suivant :

1- Un couplage maximum est {(1,3),(2,5)}.

2- La solution optimale est :

$$B_1 = \{T_1, T_3\} \quad ; \quad pb_1 = 2 \quad ;$$
$$B_2 = \{T_2, T_5\} \quad ; \quad pb_2 = 2 \quad ;$$
$$B_3 = \{T_4\} \quad ; \quad pb_3 = 2 \quad ;$$

3- La date de fin de traitement est égale à 6.

L'ordonnancement optimal est représenté par la figure 4.10.

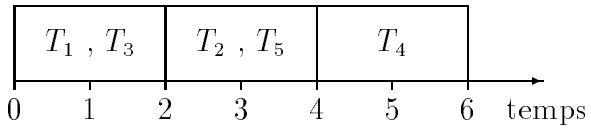


FIG. 4.10: *Ordonnancement optimal de l'exemple 4.2.*

Soit G=(V,E) un graphe ne contenant pas un sous-graphe complet à 3 sommets. Pour ce graphe, les sous-ensembles de toute partition contiennent au plus 2 sommets. Donc, le problème $B1/G = (V, E), b/C_{max}$ est résolu en temps polynomial par application de l'algorithme MWMA.

4.3 Inexistence d'un algorithme absolu[7]

Un algorithme absolu (ou fournissant une approximation absolue) est une heuristique fournissant, pour toute instance du problème, une solution approchée dont l'écart par rapport à la solution optimale est borné par une constante k. Nous montrons, ci-dessous, qu'il n'est pas possible de trouver un algorithme absolu polynomial, à moins que P=NP.

Théorème 4.10 *S'il existe un algorithme polynomial fournissant une approximation absolue pour le problème $B1/G = (V, E), b/C_{max}$, alors il existe également un algorithme polynomial exact.*

Preuve. Supposons qu'il existe un algorithme polynomial APP fournissant une approximation absolue d'ordre k, c.-à-d. $APP(I) - OPT(I) \leq k$ pour toute instance I (OPT est un algorithme exact). On aura alors pour une instance $I1$, du problème $B1/G = (V, E), b/C_{max}$, $APP(I1) - OPT(I1) \leq k$. Pour l'instance $I2$ obtenue à partir de $I1$ en recopiant $k+1$ fois les données de $I1$, on aura aussi : $APP(I2) - OPT(I2) \leq k$.

7. Les résultats de ce paragraphe sont publiés dans [28]

Comme les tâches de l'instance $I2$ forment $k+1$ familles de tâches $F_1, \ldots, F_{k+1}$ où le graphe de compatibilité de chaque famille est isomorphe au graphe de compatibilité G et où deux tâches quelconques appartenant à deux familles différentes sont incompatibles entre elles, alors $OPT(I2) = (k+1)OPT(I1)$.

Construisons, à partir de APP, un algorithme polynomial $APPX$ qui, en présence de plusieurs copies d'une même instance, ne fait que répéter pour chaque copie le meilleur ordonnancement fourni par l'algorithme APP sur les différentes copies. On a alors $APPX(I2) = (k+1)APP(I1)$, $APP(I2) \geq APPX(I2)$ et $APP(I2) - (k+1)OPT(I1) \leq k$. Donc $(k+1)APP(I1) - (k+1)OPT(I1) \leq k$. En divisant par $k+1$, on obtient $APP(I1) - OPT(I1) \leq \frac{k}{k+1} < 1$. D'où, $APP(I1) = OPT(I1)$. $\square$

Chapitre 5

Graphe scindé

Nous considérons dans ce chapitre le problème où le graphe de compatibilité est scindé (un tel graphe est noté $G = (S, K; E)$). Nous établissons dans le premier paragraphe la difficulté du problème dans le cas où la capacité de la machine à traitement par batch serait supérieure ou égale à 3 dans le cas statique et supérieure ou égale à 2 dans le cas dynamique. Dans le deuxième paragraphe, nous présentons plusieurs algorithmes exacts polynomiaux pour résoudre plusieurs sous-problèmes polynomiaux (graphe scindé complet en présence de dates de disponibilité pour les tâches, temps de traitement tous identiques à 1 sans dates de disponibilité pour les tâches, etc.).

5.1 Problèmes difficiles

Nous montrons dans ce paragraphe, que le problème d'ordonnancement est NP-difficile pour $b = k$ avec $k \geq 3$ dans le cas statique et $k \geq 2$ dans le cas dynamique.

5.1.1 Ordonnancement statique [1]

Nous montrons, dans ce sous-paragraphe, que le problème est NP-difficile lorsque les dates de disponibilité des tâches sont toutes nulles.

Théorème 5.1 *Le problème* $B1/G = (S, K; E), b = 3/C_{max}$ *est NP-difficile.*

Preuve. Considérons le problème de décision BSCG associé à ce problème.

1. Les résultats de ce paragraphe sont publiés dans [15] et soumis

Etant donné une instance arbitraire du problème 3DM (définit dans la page 18), nous construisons une instance pour le problème BSCG de la manière suivante :

Arrangeons l'ensemble M de sorte que $M = M_1 \cup \ldots \cup M_q$, où chaque sous-ensemble M_i contient tous les triplets contenant l'élément x_i.

Les tâches du problème d'ordonnancement sont partitionnées en 4 groupes :

- Les tâches de type A, notées par A_i $(i = 1, \ldots, |M|)$ et ayant un temps de traitement égal à 2.

- Les tâches de type B, notées par B_i $(i = 1, \ldots, q)$ et ayant un temps de traitement égal à 2.

- Les tâches de type C, notées par C_i $(i = 1, \ldots, q)$ et ayant un temps de traitement égal à 2.

- Les tâches de type D, notées par D_i $(i = 1, \ldots, 2(|M| - q))$ et ayant un temps de traitement égal à 3.

Les tâches de type A sont incompatibles entre elles (elles forment le stable S de taille $|M|$ dans le graphe de compatibilité G) et les tâches de type B, C et D sont toutes compatibles (elles forment la clique de taille $2|M|$ dans le graphe de compatibilité G). Le nombre total de tâches est égal à $3|M|$.

- Chaque tâche de type A est associée à un élément de M.

- Chaque tâche de type B est associée à un élément de Y.

- Chaque tâche de type C est associée à un élément de Z.

On construit les autres arêtes du graphe de compatibilité G comme suit (voir l'exemple de la figure 5.1) :

- Chaque tâche A_i (associée au triplet $(x_{f(i)}, y_{g(i)}, z_{h(i)})$) est compatible avec les deux tâches $B_{g(i)}$ et $C_{h(i)}$.

- Toutes les tâches de type A, qui correspondent aux éléments du même sous-ensemble M_i, sont compatibles avec $2(|M_i| - 1)$ tâches de type D.

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $3|M| - q$?

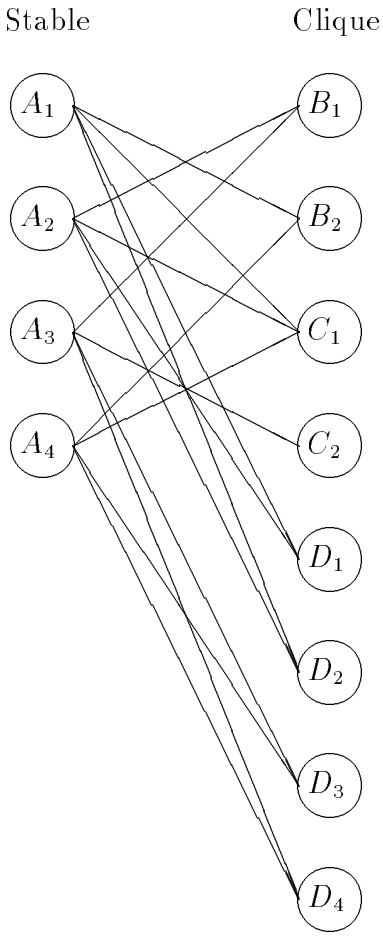


FIG. 5.1: *Graphe de compatibilité correspondant à l'instance* $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $Z = \{z_1, z_2\}$ *et* $M = \{(x_1, y_2, z_1), (x_1, y_1, z_1), (x_2, y_1, z_2), (x_2, y_2, z_1)\} = \{(x_1, y_2, z_1), (x_1, y_1, z_1)\} \cup \{(x_2, y_1, z_2), (x_2, y_2, z_1)\}$.

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que 3DM a une solution si et seulement si BSCG a une solution.

Premièrement, supposons que le problème 3DM a une solution, c.-à-d. qu'il existe un sous-ensemble $M' \subseteq M$ tel que $|M'| = q$ et M' ne contient pas deux éléments identiques de $X \cup Y \cup Z$. On construit une solution pour le problème BSCG comme suit (voir la figure 5.2) :

- Les $|M| - q$ premiers batchs sont composés de deux tâches de type D et d'une tâche de type A, ils correspondent aux triplets qui ne sont pas dans le sous-ensemble M'. Le temps de traitement, de chacun, de ces batchs est égal à 3.

- Les q batchs restantes sont composés d'une tâche de type B, d'une tâche de type C et d'une tâche de type A, ils correspondent aux triplets du sous-ensemble M'. Le temps de traitement, de chacun, de ces batchs est égal à 2.

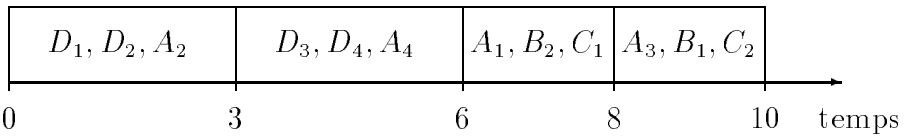


FIG. 5.2: *Une solution du problème BSCG correspondant à une solution du problème 3DM de la figure 5.1 ($M' = \{(x_1, y_2, z_1), (x_2, y_1, z_2)\}$).*

Le temps de fin de traitement de cet ordonnancement est égal à $3(|M| - q) + 2q = 3|M| - q$.

Inversement, supposons que le problème BSCG a une solution avec un temps de fin de traitement inférieur ou égal à $3|M| - q$.

Pour commencer, montrons qu'il n'existe pas de batchs contenant, chacun, 3 tâches de type D.

Supposons qu'il existe x batchs contenant, chacun, 3 tâches de type D. Alors, le temps de fin de traitement de l'ordonnancement (noté par Y) est supérieur ou égal à la somme des temps de traitement de ces batchs + la somme des temps de traitement des batchs contenant un ou deux tâches de type D avec, au plus, une tâche de type A ou une tâche de type B ou une tâche de type C + la somme des temps de traitement des batchs ne contenant pas une tâche de type D. Nous avons $Y \geq 3x + 3d + 2(|M| - d)$ où $d = \left\lceil \frac{2(|M|-q)-3x}{2} \right\rceil$ est le nombre minimum de batchs qui peuvent contenir une ou deux tâches de type D, et $|M| - d$ est le nombre minimum de batchs ne contenant par une tâche de type D (car les tâches de type A sont incompatibles entre elles et leur nombre est égal à $|M|$). Donc,

$$Y \geq \begin{cases} 3x + 3\frac{2(|M|-q)-3x}{2} + 2(|M| - \frac{2(|M|-q)-3x}{2}) & si\ 2(|M| - q) - 3x\ est\ pair \\ 3x + 3\frac{2(|M|-q)-3x+1}{2} + 2(|M| - \frac{2(|M|-q)-3x+1}{2}) & sinon \end{cases}$$

Ainsi,

$$Y \geq \begin{cases} 3|M| - q + \frac{3x}{2} & si\ x\ est\ pair \\ 3|M| - q + \frac{3x+1}{2} & sinon \end{cases}$$

Ce qui donne $Y > 3|M| - q$.

Finalement, dans toute solution de temps de fin de traitement inférieur ou égal à $3|M| - q$, il ne peut exister un batch contenant 3 tâches de type D.

Montrons, aussi, qu'il ne peut exister de batchs contenant, chacun, une seule tâche de type D.

Supposons qu'il existe x batchs contenant, chacun, une tâche de type D. Alors, le temps de fin de traitement de l'ordonnancement (noté par Y) est supérieur ou égal à la somme des temps de traitement de ces batchs + la somme des temps de traitement des batchs contenant deux tâches de type D avec une tâche de type A ou une tâche de type B ou une tâche de type C + la somme des temps de traitement des batchs ne contenant pas une tâche de type D. Nous avons $Y \geq 3x + 3(\frac{2(|M|-q)-x}{2}) + 2(|M| - (\frac{2(|M|-q)-x}{2}) - x)$ avec x pair, où $\frac{2(|M|-q)-x}{2}$ est le nombre minimum de batchs qui peuvent contenir deux tâches de type D, et $|M| - (\frac{2(|M|-q)-x}{2}) - x$ le nombre minimum de batchs ne contenant pas une tâche de type D (car les tâches de type A sont incompatibles entre eux et leur nombre est égal à $|M|$). Donc, $Y \geq 3|M| - q + \frac{x}{2}$. Ce qui donne, $Y > 3|M| - q$. Finalement, dans toute solution, de temps de fin de traitement inférieur ou égal à $3|M| - q$, il n'existe pas de batchs contenant une seule tâche de type D.

Le nombre de batchs contenant deux tâches de type D est égal à $|M| - q$. Donc, La somme des temps de traitement de ces batchs est égale à $3(|M| - q)$. Comme le nombre de tâches de type A est égal à $|M|$, Alors le temps de fin de traitement de l'ordonnancement, dans toute solution réalisable, est supérieur ou égal à $3(|M| - q) + 2(|M| - (|M| - q)) = 3(|M| - q) + 2q = 3|M| - q$. Donc, le temps de fin de traitement de cette solution est égal à $3|M| - q$. Ainsi, les batchs contenant deux tâches de type D contiennent, nécessairement, une tâche de type A. Les q autres batchs contiennent une tâche de type A, une tâche de type B et une tâche de type C.

Les tâches de type A associées à tous les triplets d'un sous-ensemble M_i (le sous-ensemble M_i contient tous les triplets contenant x_i) sont connectées à $2(|M_i| - 1)$ tâches de type D. Donc, une de ces tâches est, nécessairement, ordonnancée avec une tâche de type B et une tâche de type C (selon le graphe de compatibilité G). Donc, dans les q triplets correspondant aux q batchs contenant une tâche de type A, une tâche de type B et une tâche de type C, il y a exactement $x_1, \ldots, x_q$, $y_1, \ldots, y_q$ et $z_1, \ldots, z_q$. Il s'en suit que le problème 3DM a une solution. □

Théorème 5.2 *Le problème* $B1/G = (S, K; E), b = k/C_{max}$ *est NP-difficile pour* $k \geq 4$.

Preuve. Considérons le problème de décision BSCG associé à ce problème.

Etant donné une instance arbitraire du problème 3DM, nous construisons une instance du problème BSCG comme suit :

Les tâches du problème d'ordonnancement sont partitionnées en 4 groupes :

- Les tâches de type A, notées par A_i $(i = 1, \ldots, q)$ et ayant un temps de traitement égal à $|M| + 1$.

– Les tâches de type B, notées par B_i $(i = 1, \ldots, q)$ et ayant un temps de traitement égal à $|M| + 1$.

– Les tâches de type C, notées par C_i $(i = 1, \ldots, q)$ et ayant un temps de traitement égal à $|M| + 1$.

– Les tâches de type D, notées par D_i $(i = 1, \ldots, |M|)$ et ayant un temps de traitement égal à $|M|$.

Le nombre total de tâches est égal à $|M| + 3q$ et $b = k$ avec $k \geq 4$.

- Chaque tâche de type A est associée à un élément de X.

- Chaque tâche de type B est associée à un élément de Y.

- Chaque tâche de type C est associée à un élément de Z.

- Chaque tâche de type D est associée à un élément de M.

On construit les arêtes du graphe scindé G comme suit (voir l'exemple de la figure 5.3) :

- Les tâches des types A, B et C sont toutes compatibles entre elles (elles forment la clique K de taille $3q$).

- Chaque tâche D_i (associée au triplet $(x_{f(i)}, y_{g(i)}, z_{h(i)})$) est compatible avec les trois tâches $A_{f(i)}$, $B_{g(i)}$ et $C_{h(i)}$.

- Les tâches de type D sont incompatibles entre elles (elles forment le stable S de taille $|M|$).

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = |M|^2 + q$?

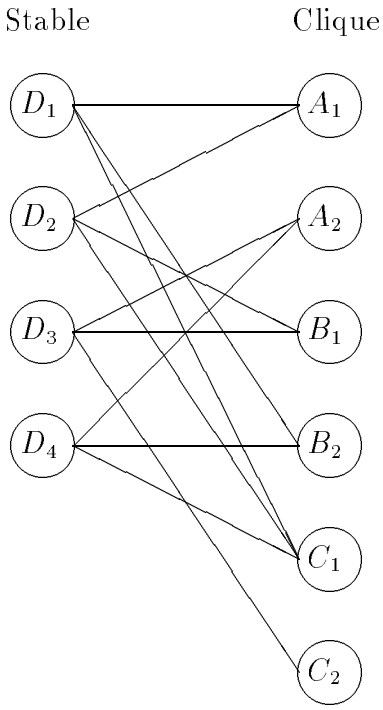


FIG. 5.3: *Graphe de compatibilité scindé correspondant à l'instance* $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $Z = \{z_1, z_2\}$ *et* $M = \{(x_1, y_2, z_1)$, (x_1, y_1, z_1) , (x_2, y_1, z_2) , $(x_2, y_2, z_1)\}$ *de 3DM.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que 3DM a une solution si et seulement si BSCG a une solution.

Premièrement, supposons que le problème 3DM a une solution, c.-à-d. qu'il existe un sous-ensemble $M' \subseteq M$ tel que $|M'| = q$ et M' ne contient pas deux éléments identiques de $X \cup Y \cup Z$. On construit une solution pour le problème SP comme suit (voir la figure 5.4) :

- Les $|M| - q$ premiers batchs sont composés d'une tâche de type D. Le temps de traitement de, chacun de, ces batchs est égal à $|M|$.

- Les q batchs restantes sont composés d'une tâche de type D et d'une tâche de chaque type A, B et C dont le triplet correspondant est dans le sous-ensemble M'. Le temps de traitement de, chacun de, ces batchs est égal à $|M| + 1$.

La date de fin de traitement de cet ordonnancement est égale à $(|M| - q)|M| + q(|M| + 1) = |M|^2 + q$.

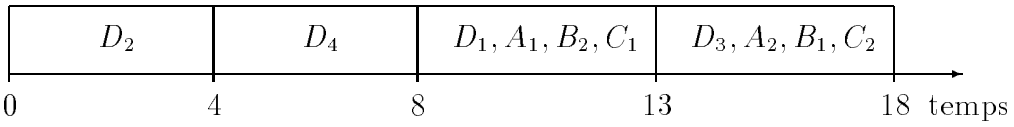


FIG. 5.4: *Solution du problème BSCG correspondant à la solution du problème 3DM de la figure 5.3 ($M' = \{(x_1, y_2, z_1), (x_2, y_1, z_2)\}$).*

Inversement, supposons que le problème BSCG a une solution avec une date de fin de traitement inférieure ou égale à $|M|^2 + q$.

Il n'est pas difficile de voir qu'il est impossible d'avoir une date de fin de traitement inférieure à $|M|^2 + q$, car le nombre de tâches du stable S est $|M|$ et chacune de ces tâches est compatible avec, exactement, trois tâches (au nombre de $3q$) de la clique $|K|$ ayant un temps de traitement égal à $|M| + 1$. Donc, il y a q batchs ayant, chacun, un temps de traitement égal à $|M| + 1$ (car $|M|^2 + q \ mod \ |M| = q < |M| < |M| + 1$). dans chacun de ces batchs, il y a une tâche de type A, une tâche de type B et une tâche de type C (selon le graphe de compatibilité scindé G). Ainsi, dans les q triplets correspondant aux q batchs contenant, chacun, une tâche de type A, une tâche de type B et une tâche de type C, il y a exactement $x_1, \ldots, x_q$, $y_1, \ldots, y_q$ et $z_1, \ldots, z_q$. Il s'en suit que le problème 3DM a une solution. □

Par le théorème précédent, il vient que le problème $B1/G = (S, K; E), b \geq n/C_{max}$ est NP-difficile même si les tâches du stable S ont toutes un même temps de traitement égal à p et les tâches de la clique K ont toutes un même temps de traitement égal à $p + 1$.

Comme un graphe scindé est un graphe à cordes [67] alors le problème est, aussi, NP-difficile pour les graphes triangulés.

5.1.2 Ordonnancement dynamique[2]

Nous montrons, dans ce sous-paragraphe, que le problème est NP-difficile lorsque les temps de traitement des tâches sont tous identiques et les dates de disponibilité des tâches ne sont pas toutes nulles.

Théorème 5.3 *Le problème $B1/G = (S, K; E), b = 2, r_i, p_i = 1/C_{max}$ est NP-difficile au sens fort.*

Preuve. Considérons le problème de décision BSCG associé à ce problème.

2. Les résultats de ce paragraphe sont publiés dans [17]

Etant donné une instance arbitraire du problème TRIPARTITE ONE-IN-THREE 3-3 SAT, nous construisons une instance du problème BSCG comme suit :

- A chaque variable booléenne x_i est associée huit tâches : les trois premières tâches, notées par T_i, $T_i^{'}$ et $T_i^{''}$, représentent le choix d'avoir $x_i = vrai$; les trois autres, notées par F_i, $F_i^{'}$ et $F_i^{''}$, représentent l'alternative $x_i = faux$, et les deux tâches restantes, notées par G_i et E_i, sont pour la prise de décisions. Le nombre total de tâches est $24m$.

- Toutes les tâches ont un même temps de traitement égal à 1.

- Les dates de disponibilité sont données à la table 5.1.

- Toutes les tâches T_i, F_i, E_i et G_i pour $i = 1, \ldots, 3m$ sont compatibles entre elles, elles forment la clique K. Toutes les tâches $T_i^{'}$, $T_i^{''}$, $F_i^{'}$ et $F_i^{''}$ pour $i = 1, \ldots, 3m$ sont incompatibles entre elles, elles forment le stable S. Les autres contraintes de compatibilité sont définies entre les tâches ayant un même indice i, elles sont données à la figure 5.5.

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $12m$?

Tâche	Date de disponibilité
T_i	$7m - 2t(i)$
$T_i^{'}$	$9m - s(i)$
$T_i^{''}$	$3m - u(i)$
F_i	$7m - t(i)$
$F_i^{'}$	$8m - 2s(i)$
$F_i^{''}$	$7m - 2u(i)$
E_i	0
G_i	$9m$

TAB. 5.1: *Dates de disponibilité des tâches indicées i.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que TRIPARTITE ONE-IN-THREE 3-3 SAT a une solution si et seulement si BSCG a une solution.

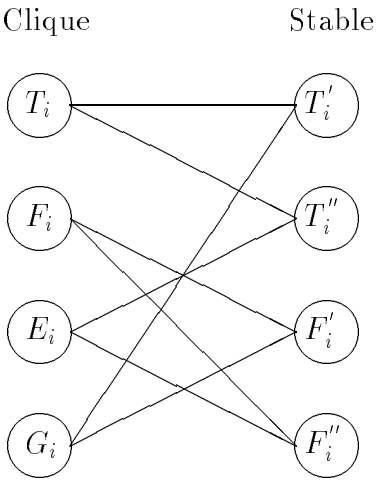


FIG. 5.5: *Graphe de compatibilité scindé des tâches indicées i.*

Supposons que TRIPARTITE ONE-IN-THREE 3-3 SAT a une solution (il existe une affectation de la valeur de vérité "vrai" aux variables booléennes de X telle que chaque clause de C a exactement une variable booléenne ayant la valeur de vérité "vrai"). Nous construisons une solution pour BSCG comme suit : (voir les figures 5.6 et 5.7)

- Pour chaque variable booléenne x_i ayant une affectation "vrai", ordonnancer les tâches suivantes dans un même batch.
  - E_i et T_i'' à l'instant $3m - u(i)$,
  - F_i et F_i'' à l'instant $7m - t(i)$,
  - T_i et T_i' à l'instant $9m - s(i)$ et
  - G_i et F_i' à l'instant $12m - i$.

- Pour chaque variable booléenne x_i ayant une affectation "faux", ordonnancer les tâches G_i et T_i' dans, un même batch, à l'instant $12m - i$.

- Pour chaque paire x_i et x_j de variables booléennes ayant une affectation "faux" et
  - pour chaque clause C_u ($u = 2m+1, \ldots, 3m$), ordonnancer à l'instant $7m-2u$ et $7m-2u+1$, respectivement, les deux tâches E_i et F_i'' dans un même batch et les deux tâches E_j et F_j'' dans un même batch, telles que $u(i) = u(j) = u$ (les dates de disponibilité des tâches F_i'' et F_j'' sont égales à $7m - 2u$).
  - pour chaque clause C_t ($t = m+1, \ldots, 2m$), ordonnancer à l'instant $7m - 2t$ et $7m-2t+1$, respectivement, les deux tâches T_i et T_i'' dans un même batch

et les deux tâches T_j et T_j'' dans un même batch, telles que $t(i) = t(j) = t$ (les dates de disponibilité des tâches T_i et T_j sont égales à $7m - 2t$).

- pour chaque clause C_s $(s = 1, \ldots, m)$, ordonnancer à l'instant $8m - 2s$ et $8m - 2s + 1$, respectivement, les deux tâches F_i et F_i' dans un même batch et les deux tâches F_j et F_j' dans un même batch, telles que $s(i) = s(j) = s$ (les dates de disponibilité des tâches F_i' et F_j' sont égales à $8m - 2s$).

L'ordonnancement obtenu est représenté par la figure 5.6.

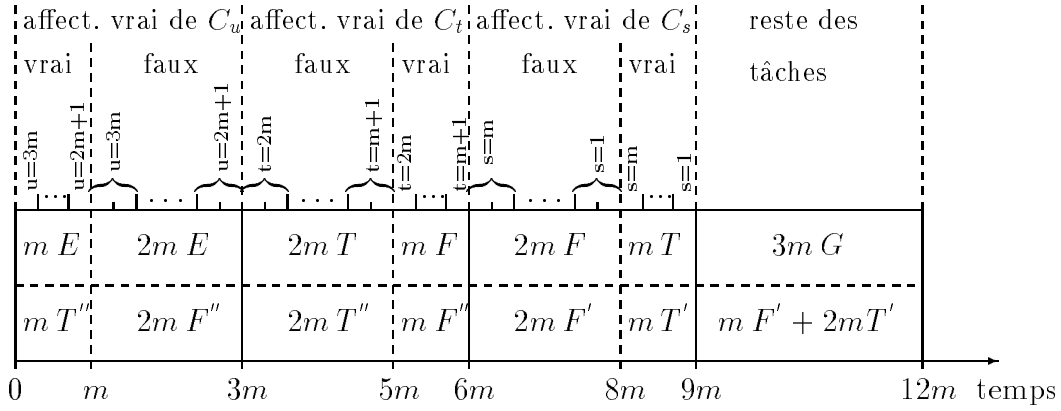


FIG. 5.6: *Ordonnancement des tâches du problème BSCG.*

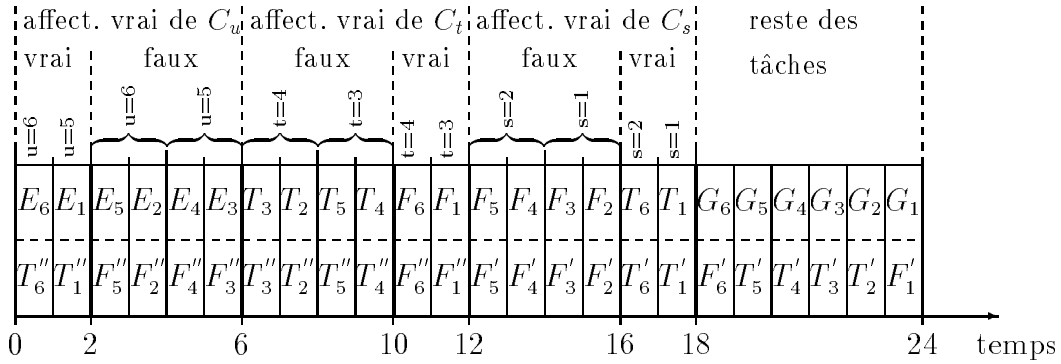


FIG. 5.7: *Solution du problème BSCG correspondant à la solution ($x_1 = vrai$, $x_6 = vrai$ et $x_i = faux$ pour $i = 2, 3, 4, 5$) du problème TRIPARTITE-ONE-THREE 3-3 SAT : $X = \{x_1, \ldots, x_6\}$, $m = 2$ et $F = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_6) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_5 \vee x_6)$.*

Inversement, supposons qu'il existe un ordonnancement avec une date de fin de traitement $C_{max} \leq 12m$. Comme le nombre total des tâches du stable S est égal à $12m$,

alors $C_{max} = 12m$ et deux tâches de la clique K ne peuvent pas appartenir à un même batch. Ainsi, dans chaque batch, nous avons une tâche de la clique K (T_i, F_i, E_i ou G_i) et une tâche du stable S ($T_i^{'}$, $T_i^{''}$, $F_i^{'}$ ou $F_i^{''}$). De la structure du graphe scindé de compatibilité, nous concluons que pour chaque i, les huit tâches T_i, F_i, E_i, G_i, $T_i^{'}$, $T_i^{''}$, $F_i^{'}$ et $F_i^{''}$ ne peuvent s'ordonnancer que selon les deux alternatives de configuration de batchs suivantes :

(a) T_i avec $T_i^{'}$, G_i avec $F_i^{'}$, F_i avec $F_i^{''}$ et E_i avec $T_i^{''}$, ou

(b) T_i avec $T_i^{''}$, G_i avec $T_i^{'}$, F_i avec $F_i^{'}$ et E_i avec $F_i^{''}$.

Les dates de disponibilité impliquent que toutes les $3m$ G_i tâches doivent être ordonnancées dans l'intervalle de temps $[9m, 12m]$, et toutes les $3m$ E_i tâches doivent être ordonnancées dans l'intervalle de temps $[0, 3m]$ (car les tâches T_i et F_i sont disponibles après l'instant $3m$).

Des considérations précédentes, nous avons :

- m des $3m$ $T_i^{''}$ tâches doivent être ordonnancées, avec m des $3m$ E_i tâches, entre l'instant 0 et l'instant m (car les tâches $F_i^{''}$ sont disponibles après l'instant m).

- Aussi, $2m$ des $3m$ T_i tâches doivent être ordonnancées, avec les $2m$ $T_i^{''}$ tâches restantes, dans l'intervalle de temps $[3m, 5m]$ (car les tâches F_i sont disponibles après l'instant $5m$ et les tâches $T_i^{'}$ sont disponibles après l'instant $8m$).

- Ainsi, $2m$ des $3m$ $F_i^{''}$ tâches doivent être ordonnancées, avec les $2m$ E_i tâches restantes, dans l'intervalle de temps $[m, 3m]$, et les m $F^{''}$ tâches restantes doivent être ordonnancées dans l'intervalle de temps $[5m, 6m]$ (car toutes les tâches $T_i^{'}$ et $F_i^{'}$ sont disponibles après l'instant $6m$) avec m des $3m$ F_i tâches.

- Aussi, $2m$ des $3m$ $F_i^{'}$ tâches doivent être ordonnancées dans l'intervalle de temps $[6m, 8m]$ (car les tâches $T_i^{'}$ sont disponibles après l'instant $8m$) avec les $2m$ F_i tâches restantes.

- Donc, les m T_i tâches restantes doivent être ordonnancées dans l'intervalle de temps $[8m, 9m]$ avec m des $3m$ $T_i^{'}$ tâches.

- Finalement, les m $F_i^{'}$ et $2m$ $T_i^{'}$ tâches restantes doivent être ordonnancées, entre l'instant $9m$ et l'instant $12m$, avec les $3m$ G_i tâches.

Nous concluons que nous avons m sous-ensembles de tâches (T_i, F_i, E_i, G_i, $T_i^{'}$, $T_i^{''}$, $F_i^{'}$ et $F_i^{''}$) qui apparaissent dans la configuration (a) et les $2m$ autres dans la configuration (b). En procédant par récurrence sur s ($s = 1, \ldots, m$), il vient des dates de disponibilité des tâches $T^{'}$ que, pour chaque s, les tâches T_i et $T_i^{'}$, ordonnancées dans un même batch à l'instant $9m - s$, doivent avoir l'indice i tel que $s(i) = s$. Il s'en suit aussi que, à partir des dates de disponibilité des tâches $F^{'}$, que pour chaque s, les tâches F_j et $F_j^{'}$ ordonnancées dans un même batch à l'instant $8m - 2s$ et les tâches F_k et $F_k^{'}$ ordonnancées dans un même bacth à l'instant $8m - 2s + 1$, doivent avoir les indices j et k tels que $s(j) = s(k) = s$. Par des récurrences analogues sur t ($t = m+1, \ldots, 2m$) et u ($u = 2m+1, \ldots, 3m$), respectivement, on montre que les tâches F_i et $F_i^{''}$, ordonnancées dans un même batch à l'instant $7m - t$, doivent avoir l'indice i tel que $t(i) = t$ et les taches E_i et $T_i^{''}$, ordonnancées dans un même batch à l'instant $3m - u$, doivent avoir l'indice i tel que $u(i) = u$. Il vient que les m variables booléennes pour lesquelles les sous-ensembles de tâches correspondants apparaissent dans la configuration (a) auront l'affectation "vrai" et les $2m$ autres variables booléennes auront l'affectation "faux", ainsi nous avons une affectation de valeur de vérité "vrai" satisfaisant la fonction booléenne F du problème TRIPARTITE ONE-IN-THREE 3-3 SAT. □

Théorème 5.4 *Le problème* $B1/G = (S, K; E), b = 3, r_i, p_i = 1/C_{max}$ *est NP-difficile.*

Preuve. Considérons le problème de décision BSCG associé à ce problème.

Etant donné une instance arbitraire du problème 3DM, nous construisons une instance du problème BSCG comme suit :

Arrangeons l'ensemble M de sorte que $M = M_1 \cup \ldots \cup M_q$ où chaque sous-ensemble M_i contient tous les triplets contenant l'élément x_i.

Les tâches du problème d'ordonnancement sont partitionnées en 4 groupes :

- Les tâches de type A, notées par A_{ij} ($i = 1, \ldots, q$ et $j = 1, \ldots, |M_i|$) et ayant une date de disponibilité égale à $\sum_{k=1}^{i-1} |M_i|$ (pour $i = 1$ nous avons $A_{1j} = 0$).

- Les tâches de type B, notées par B_i ($i = 1, \ldots, q$) et ayant une date de disponibilité nulle.

- Les tâches de type C, notées par C_i ($i = 1, \ldots, q$) et ayant une date de disponibilité nulle.

- Les tâches de type D, notées par D_{ij} ($i = 1, \ldots, q$ et $j = 1, \ldots, 2(|M_i| - 1)$) et ayant une date de disponibilité égale à $r_{A_{ij}} + \left[\frac{j+1}{2}\right]$ ($r_{A_{ij}}$ est la date de disponibilité de la tâche $r_{A_{ij}}$).

Toutes les tâches ont un temps de traitement égal à 1.

Les tâches de type A sont incompatibles entre elles (elles forment le stable S de taille $|M|$ dans le graphe de compatibilité G) et les tâches de types B, C et D sont toutes compatibles (elles forment la clique de taille $2|M|$ dans le graphe de compatibilité G). Le nombre total de tâches est égal à $3|M|$.

- Chaque tâche de type A est associée à un élément de M.

- Chaque tâche de type B est associée à un élément de Y.

- Chaque tâche de type C est associée à un élément de Z.

Nous construisons les autres arêtes du graphe de compatibilité G comme suit (voir l'exemple de la figure 5.8) :

- Chaque tâche A_{ij} (associées au triplet $(x_{f(i,j)}, y_{g(i,j)}, z_{h(i,j)})$) est compatible avec les deux tâches $B_{g(i,j)}$ et $C_{h(i,j)}$.

- Chaque tâche A_{ij} est compatible avec les tâches D_{ik} pour $k = 1, \ldots, 2(|M_i| - 1)$.

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = |M|$?

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que 3DM a une solution si et seulement si BSCG a une solution.

Comme le nombre de tâches est égal à $3|M|$, alors, il y a $|M|$ batchs et, dans chacun, il y a 3 tâches. Notons ces batchs par B_{ij} ($i = 1, \ldots, q$ et $j = 1, \ldots, |M_i|$).

Premièrement, supposons que le problème 3DM a une solution, c.-à-d. qu'il existe un sous-ensemble $M' \subseteq M$ telle que $|M'| = q$ et M' ne contient pas deux éléments identiques de $X \cup Y \cup Z$. On construit une solution pour le problème BSCG comme suit (voir la figure 5.9) :

- Chaque batch B_{i1} (pour $i = 1, \ldots, q$) est composé d'une tâche de type B, d'une tâche de C et d'une tâche de type A_{ij}, correspondant au triplet du sous-ensemble M'.

- Le reste des $|M| - q$ batchs B_{ij} (pour $j \neq 1$) sont composés de deux tâches de type D (les tâches $D_{i,2(j-1)-1}$ et $D_{i,2(j-1)}$) et une tâche de type A dont le triplet correspondant n'est pas dans le sous ensemble M'.

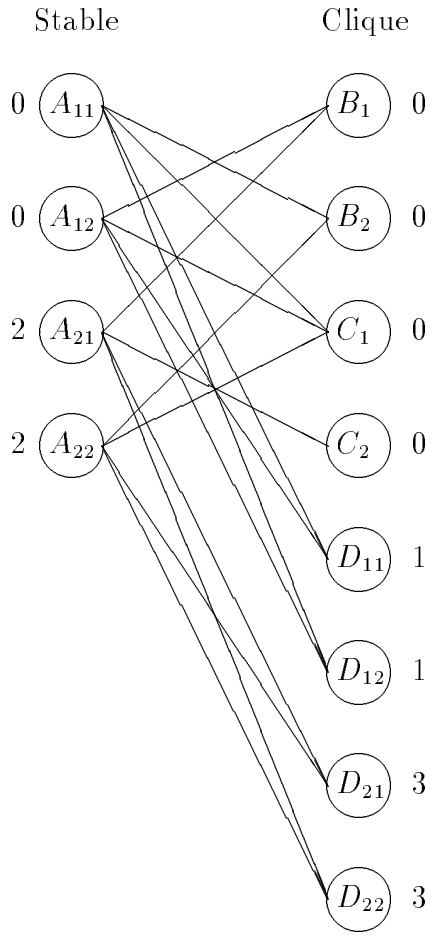


FIG. 5.8: *Graphe de compatibilité et dates de disponibilité correspondant à l'instance* $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $Z = \{z_1, z_2\}$ *et* $M = \{(x_1, y_2, z_1), (x_1, y_1, z_1), (x_2, y_1, z_2), (x_2, y_2, z_1)\} = \{(x_1, y_2, z_1), (x_1, y_1, z_1)\} \cup \{(x_2, y_1, z_2), (x_2, y_2, z_1)\}$.

Les dates de disponibilité de ces tâches sont respectées et la date de fin de traitement de cet ordonnancement est égal à $|M|$.

Réciproquement, supposons que le problème BSCG a une solution avec une date de fin de traitement inférieure ou égale à $|M|$.

Considérons l'instant $|M|$ et les $|M_q|$ dernières tâches $A_{q1}, \ldots, A_{q|M_q|}$. Leurs dates de disponibilité sont égales à $\sum_{i=1}^{q-1} |M_i| = |M| - |M_q|$ et elles sont incompatibles entre elles. Donc, elles seront ordonnancées séparément dans les $|M_q|$ batchs $B_{q1}, \ldots, B_{q|M_q|}$. Comme les dates de disponibilité des tâches D_{qj} $(j = 1, \ldots, 2(|M_q| - 1))$ sont égales à $|M| - |M_q| + \left\lceil \frac{j+1}{2} \right\rceil$, alors, pour $j = 1, \ldots, |M_q| - 1$, les deux tâches $D_{q,2j-1}$ et $D_{q,2j}$ seront ordonnancées dans le même batch $B_{q,j+1}$. Finalement, dans le batch B_{q1} seront ordonnancées avec une tâche de type A, une tâche de type B et une tâche de type C

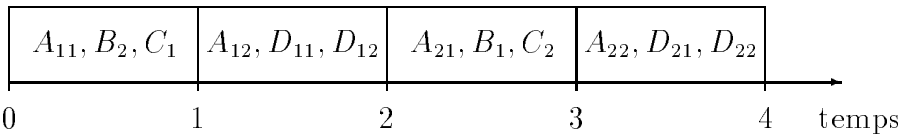


FIG. 5.9: *La solution du problème BSCG correspondant à la solution du problème 3DM de la figure 5.8 ($M' = \{(x_1, y_2, z_1), (x_2, y_1, z_2)\}$).*

(car ce sont les seules tâches restantes compatibles avec la tâche de type A).

En considérons les instants $|M|-|M_q|, |M|-(|M_q|+|M_{q-1}|), \ldots, |M|-\sum_{i=q}^{2}|M_i| = |M_1|$, nous obtenons, récursivement et de la même manière, que les batchs $B_{q-11}, \ldots, B_{11}$ contiennent une tâche de type A, une tâche de type B et une tâche de type C. Donc, dans les q triplets, correspondant aux q batchs $B_{q1}, \ldots, B_{11}$, il y a exactement $x_1, \ldots, x_q$, $y_1, \ldots, y_q$ et $z_1, \ldots, z_q$. Il s'en suit que le problème 3DM a une solution. □

Théorème 5.5 *Le problème $B1/G = (S, K; E), b = k, r_i, p_i = 1/C_{max}$ est NP-difficile pour $k \geq 4$.*

Preuve. Considérons le problème de décision BSCG associé à ce problème.

Etant une instance arbitraire du problème 3DM, nous construisons une instance du problème SP comme suit :

Les tâches du problème d'ordonnancement sont partitionées en 4 groupes :

- Les tâches de type A, notées par A_i $(i = 1, \ldots, q)$ et ayant une date de disponibilité égale à $|M| - q$.

- Les tâches de type B, notées par B_i $(i = 1, \ldots, q)$ et ayant une date de disponibilité égale à $|M| - q$.

- Les tâches de type C, notées par C_i $(i = 1, \ldots, q)$ et ayant une date de disponibilité égale à $|M| - q$.

- Les tâches de type D, notées par D_i $(i = 1, \ldots, |M|)$ et ayant une date de disponibilité égale à 0.

Toutes les tâches ont un temps de traitement égal à 1.

Le nombre total de tâches est égal à $|M| + 3q$ et $b = k$ avec $k \geq 4$.

- Chaque tâche de type A est associée à un élément de X.
- Chaque tâche de type B est associée à un élément de Y.
- Chaque tâche de type C est associée à un élément de Z.
- Chaque tâche de type D est associée à un élément de M.

Nous construisons les arêtes du graphe de compatibilité scindé G comme suit (voir l'exemple de la figure 5.10) :

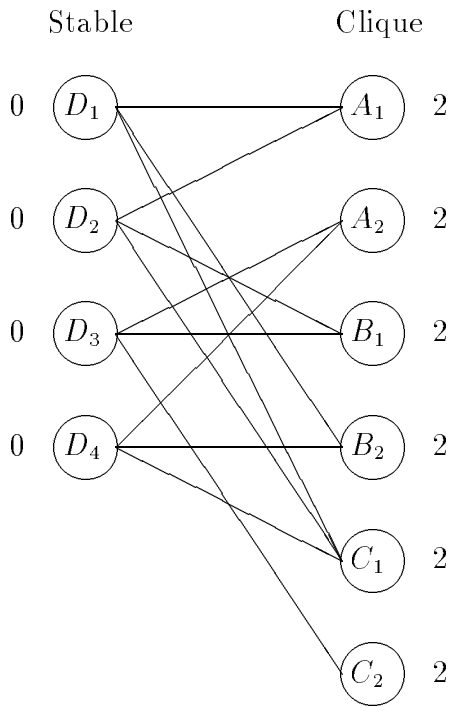


FIG. 5.10: *Dates de disponibilité et graphe de compatibilité scindé correspondant à l'instance* $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, $Z = \{z_1, z_2\}$ *et* $M = \{(x_1, y_2, z_1), (x_1, y_1, z_1), (x_2, y_1, z_2), (x_2, y_2, z_1)\}$ *de 3DM.*

- Les tâches de types A, B et C sont toutes compatibles entre elles (elles forment la clique K de taille $3q$).

- Les tâches de type D sont incompatibles entre elles (elles forment le stable S de taille $|M|$).

- Chaque tâche D_i (associée au triplet $(x_{f(i)}, y_{g(i)}, z_{h(i)})$) est compatible avec les trois tâches $A_{f(i)}$, $B_{g(i)}$ et $C_{h(i)}$.

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $y = |M|$?

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que 3DM a une solution si et seulement si BSCG a une solution.

Premièrement, supposons que le problème 3DM a une solution, c.-à-d. qu'il existe un sous-ensemble $M' \subseteq M$ telle que $|M'| = q$ et M' ne contient pas deux éléments identiques de $X \cup Y \cup Z$. On construit une solution pour le problème BSCG comme suit (voir la figure 5.11) :

- Les $|M| - q$ premiers batchs sont composés chacun d'une seule tâche de type D et sont ordonnancées dans l'intervalle de temps $[0, |M| - q]$.

- Les q batchs restants sont composés d'une tâche de type D et une tâche de chaque type A, B et C dont le triplet correspondant est dans le sous-ensemble M'. Ces batchs sont ordonnancés dans l'intervalle de temps $[|M| - q, |M|]$.

La date de fin de traitement de cet ordonnancement est égal à $|M|$.

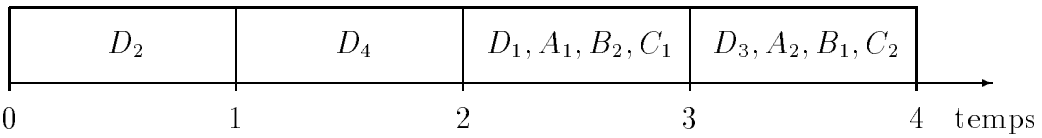


FIG. 5.11: *Solution du problème BSCG correspondant à la solution du problème 3DM de la figure 5.10 ($M' = \{(x_1, y_2, z_1), (x_2, y_1, z_2)\}$).*

Inversement, supposons que le problème BSCG a une solution avec une date de fin de traitement inférieure ou égale à $|M|$.

Dans le stable S il y a $|M|$ tâches incompatibles entre elles, donc le nombre total de batchs est égal à $|M|$ ayant, chacun, une tâche de type D. Comme les tâches de type A, B et C ont une date de disponibilité égale à $|M| - q$, alors dans chacun de ces batchs, ordonnancés après l'instant $|M| - q$, il y a une tâche de type A, une tâche de type B

et une tâche de type C (selon le graphe de compatibilité scindé G). Ainsi, dans les q triplets correspondant aux q derniers batchs contenant, chacun, une tâche de type A, une tâche de type B et une tâche de type C, il y a exactement $x_1, \ldots, x_q$, $y_1, \ldots, y_q$ et $z_1, \ldots, z_q$. Il s'en suit que le problème 3DM a une solution. □

Du théorème précédent, il en découle que le problème $B1/G = (S, K; E), b \geq n/C_{max}$ est NP-difficile même si les tâches du stable S ont toutes une date de disponibilité nulle et les tâches de la clique K ont toutes une même date de disponibilité.

Comme un graphe scindé est un graphe triangulé [67] alors le problème est, aussi, NP-difficile pour les graphes triangulés.

5.2 Problèmes polynomiaux

Nous proposons, dans ce paragraphe, quelques algorithmes polynomiaux exacts pour résoudre le problème dans le cas où le graphe de compatibilité serait un graphe scindé.

5.2.1 Ordonnancement statique[3]

Nous considérons, dans ce sous-paragraphe, le cas où les dates de disponibilité des tâches sont toutes nulles.

capacité finie de la machine

Nous considérons, le cas où les temps de traitement des tâches sont tous identiques et où la capacité de la machine à traitement par batch est quelconque. Un algorithme polynomial, basé sur l'algorithme du flot maximum dans un réseau, est donné ci-dessous.

Algorithme MFA ;
début

1- Construire un réseau $R = (V, E', C)$ comme suit (voir l'exemple de la figure 5.12) :

- Connecter la source s à chaque sommet de la clique K par un arc de capacité égale à 1 ;
- Connecter chaque sommet du stable S au puits p par un arc de capacité égale à $b - 1$;

3. Les résultats de ce paragraphe sont publiés dans [15] et soumis

– Un sommet de la clique K et un sommet du stable S sont connectés par un arc de capacité égale à 1 si et seulement si ces deux sommets sont connectés dans le graphe de compatibilité G ;

2- Déterminer un flot maximum f dans le réseau R ;

3- La solution optimale est :

– Chaque tâche du stable S sera ordonnancée, dans un même batch, avec les tâches de la clique K correspondant aux arcs ayant un flux entrant égal à 1 ;

– Le reste des tâches de la clique K seront ordonnancées dans $\left\lceil \frac{|K|-f}{b} \right\rceil$ batchs ;

4- $C_{max} = |S| + \left\lceil \frac{|K|-f}{b} \right\rceil$;

fin.

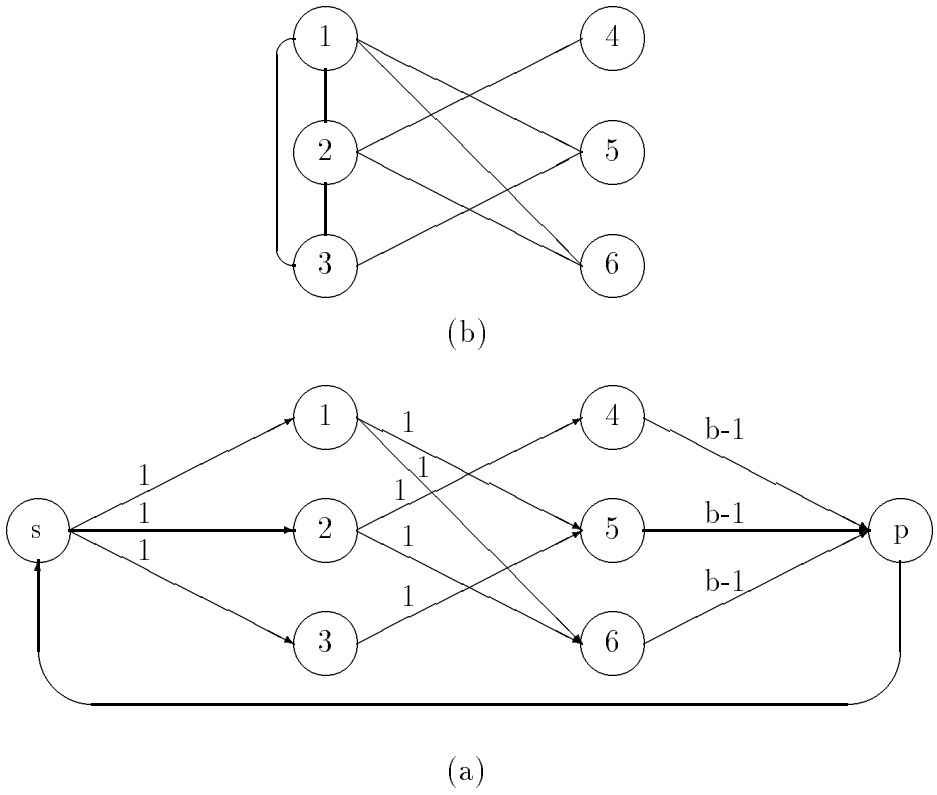


FIG. 5.12: *(a) Un graphe scindé G=(S,K;E) et (b) le réseau correspondant* $R = (V, E', C)$.

Théorème 5.6 *L'algorithme MFA résout le problème* $B1/G = (S, K; E), b, p_i = 1/C_{max}$ *en* $O(n^3)$.

Preuve. La date de fin de traitement du problème avec un graphe de compatibilité complet est égale à $\left\lceil \frac{n}{b} \right\rceil$. Donc, comme les tâches du stable S sont incompatibles entre elles, la date de fin de traitement de l'ordonnancement, de toute solution réalisable, s'écrit $|S| + \left\lceil \frac{|K|-f}{b} \right\rceil$ où f est le nombre de tâches, de la clique K, ordonnancées avec les tâches du stable S. Minimiser $|S| + \left\lceil \frac{|K|-f}{b} \right\rceil$, revient à maximiser f. Donc, maximiser le nombre de tâches, de la clique K, ordonnancées avec les tâches du stable S. Par conséquent, maximiser le flot du réseau R construit par l'algorithme MFA. La construction du réseau est de l'ordre de $O(|S||K|)$. Comme le réseau est sans cycles et le flot maximum est $|K|$, alors, la complexité du problème pour trouver un flot maximum est $O(|S||K|^2)$, par la méthode "minimum cost augmentation"[118]. Donc, la complexité de l'algorithme MFA est $O(n^3)$. □

Exemple 5.1 *Considérons l'ordonnancement de 6 tâches $T_1, \ldots, T_6$ sur une machine à traitement par batch de capacité égale à 3. Les temps de traitement des tâches sont tous égaux à 1. Le graphe de compatibilité est donné à la figure 5.12a.*

L'exécution de l'algorithme donne :

1- Le réseau est donné à la figure 5.12b avec $b = 3$;

2- $f = 3$;

3- Les batchs sont :
$B_1 = \{T_2, T_4\}$;
$B_2 = \{T_1, T_3, T_5\}$;
$B_3 = \{T_6\}$;

4- $C_{max} = 3 - \frac{3-3}{3} = 3$;

L'ordonnancement optimal est représenté par la figure 5.13.

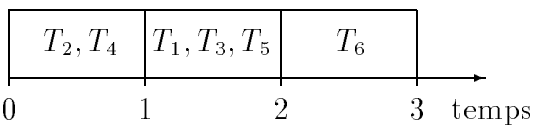


FIG. 5.13: *L'ordonnancement optimal de l'exemple 5.1.*

Capacité infinie de la machine

Dans le cas où les temps de traitement des tâches seraient tous identiques nous donnons, ci-dessous, un algorithme linéaire pour le résoudre.

Algorithme LA3 ;
début
1- Affecter chaque tâche du stable S à un batch (notons ces batchs par $B_1, \ldots, B_{|S|}$) ;
2- **Pour** $i := 1$ **à** $|S|$ **faire**
affecter au batch B_i toutes les tâches de la clique K compatibles avec ses tâches
fait ;
3- S'il existe une tâche de la clique K qui n'est pas compatible avec, au moins, une tâche du stable S, alors affecter le reste des tâches au même $B_{|S|+1}$;
fin.

Théorème 5.7 *L'algorithme LA3 résout le problème* $B1/G = (S, K; E), b \geq n, p_i = 1/C_{max}$ *en* $O(n)$.

Preuve. Ce problème est équivalent au problème de la partition minimum en cliques du graphe à cordes G, qui est a été résolu et discuté dans la littérature. Chaque tâche de la clique K est connectée à, au moins, une tâche du stable S ou non. Dans le premier cas $C_{max} = |S|$ et dans le second $C_{max} = |S| + 1$. Cette propriété est vérifiée en $O(n)$. □

Considérons, maintenant, l'algorithme MWMAA donné ci-dessous, qui résout le problème lorsque les tâches du stable S ont toutes un même temps de traitement égal à p et chaque tâche du stable S est compatible avec, au plus, deux tâches de la clique K ayant un temps de traitement supérieur à p.

Algorithme MWMAA ;
début
1- $Z0 := |S|p + \max\limits_{T_i \in K}\{p_i\}$;
2- Affecter toutes les tâches du stable S aux différents batchs $B_1, \ldots, B_{|S|}$;
3- S'il existe, une tâche de la clique K, ayant un temps de traitement supérieur à p et non compatible avec, au moins, une tâche du stable S, alors, affecter toutes les tâches de la clique K au batch $B_{|S|+1}$ et stop ; cette solution est optimale ;
4- Affectation des tâches de la clique K ayant un temps de traitement inférieur ou égal à p ;
   - Affecter toutes les tâches non compatibles avec, au moins, une tâche du stable S au batch $B_{|S|+1}$;
   - Affecter toutes les tâches compatibles avec, au moins, une tâche du stable S au batchs appropriés de $B_1, \ldots, B_{|S|}$;
5- Construction du graphe valué $G = (V, E)$;
   - Soit V l'ensemble des indices des tâches restantes de la clique K (ayant un temps de traitement supérieur à p et compatible avec, au moins, une tâche du stable S) ;

– (i, j) est dans E si et seulement si les taches T_i et T_j sont compatibles avec une même tâche du stable S ;

– Chaque arête $(i, j) \in E$ est valuée par $\min\{p_i, p_j\} - p$;

6- Déterminer un couplage M de poids maximum dans le graphe $G = (V, E)$;

7- $Z = \sum_{(T_i,T_j)\in M} \max\{p_i, p_j\} + \sum_{T_i \notin M} p_i + (|S| + |M| - |V|)p + pb_{|S|+1}$;
($pb_{|S|+1} = 0$ si $B_{|S|+1}$ est vide)

8- **Si** $Z < Z0$ et toutes les tâches restantes de la clique K sont ordonnancées avec les tâches du stable S
alors La solution optimale est :

– Pour chaque couple de M, ordonnancer ses deux tâches dans un même batch avec la tâche correspondante du stable S.

– Les autres tâches sont ordonnancés chacune avec une tâche du stable S.

sinon La solution optimale est :

– Les tâches du stable S sont ordonnancées seules.

– Les tâches de la clique K sont ordonnancées dans le batch $B_{|S|+1}$.

fsi ;

fin.

Théorème 5.8 *L'algorithme MWMAA résout le problème* $B1/G = (S, K; E), b \geq n/C_{max}$ *en temps polynomial sous les conditions suivantes :*

a) *les temps de traitement des tâches du stable* S *sont tous identiques à* p *et*

b) *chaque tâche du stable* S *est compatible avec, au plus, deux tâches de la clique* K *ayant un temps de traitement supérieur à* p.

Notons cette condition par : $p_S = p$ et $|\{T_j \in K \ / \ (i, j) \in E \text{ et } p_j > p\}_{T_i \in S}| \leq 2$.

Preuve. Il est claire que, dans une solution optimale, les tâches du stable S sont ordonnancées dans des batchs différents $B_1, \ldots, B_{|S|}$ et le nombre de batchs est soit égal à $|S|$, soit égal à $|S| + 1$. Aussi, il existe une solution optimale où toutes les tâches de la clique K ayant un temps de traitement inférieur ou égal à p sont ordonnancées dans les batchs $B_1, \ldots, B_{|S|}$.

Considérons le cas où toutes les tâches restantes de la clique K, ayant un temps de traitement supérieur à p, sont ordonnancées dans le même batch $B_{|S|+1}$. S'il existe une meilleure solution (c.-à-d. $C_{max} < |S|p + \max_{T_i \in K}\{p_i\}$), alors, toutes les tâches de la clique K, ayant un temps de traitement supérieur à p, doivent être ordonnancées dans les batchs $B_1, \ldots, B_{|S|}$. Car la tâche ayant le plus long temps de traitement doit être dans l'un des batchs $B_1, \ldots, B_{|S|}$ et C_{max} doit être supérieur ou égal à $(|S| - 1)p + \max_{T_i \in K}\{p_i\} +$

$pb_{|S|+1} = |S|p + \max_{T_i \in K}\{p_i\} + (pb_{|S|+1} - p) > |S|p + \max_{T_i \in K}\{p_i\}$ si le batch $B_{|S|+1}$ contient une tâche de la clique K ayant un temps de traitement supérieur à p, ce qui absurde.

Donc, s'il existe une tâche de la clique K, ayant un temps de traitement supérieur à p et non compatible avec, au moins, une tâche du stable S, on construit une solution optimale simplement en affectant toutes les tâches de la clique K au batch $B_{|S|+1}$.

Maintenant, affectons toutes les tâches de la clique K non compatibles avec, au moins, une tâche du stable S et ayant un temps de traitement inférieur ou égal à p au batch $B_{|S|+1}$. Aussi, affectons chaque tâche de la clique K compatible avec, au moins, une tâche du stable S et ayant un temps de traitement inférieur ou égal à p au batch approprié de $B_1, \ldots, B_{|S|}$.

Supposons que toutes les tâches restantes de la clique K (soit H l'ensemble de ces tâches) doivent être ordonnancées avec les tâches du stable S (chacune de ces tâches sera ordonnancée avec une tâche du stable S). Comme chaque tâche du stable S est compatible avec, au plus, deux tâches de H, alors le temps de traitement du batch contenant une tâche du stable S est égal soit à p si aucune tâche de H n'est ordonnancée dans ce batch, soit à p_i si une tâche (T_i) de H est ordonnancée dans ce batch, soit à $max\{p_i, p_j\}$ si deux tâches (T_i et T_j) de H sont ordonnancées dans ce batch. La date de fin de traitement est, donc, égal à $pb_{|S|+1} + |S|p + \sum_{\substack{B_i \text{ contenant deux} \\ \text{tâches } (T_{s_i}, T_{t_i}) \text{ de } H}} (max\{p_{s_i}, p_{t_i}\} - p) + \sum_{\substack{B_i \text{ contenant une} \\ \text{tâche } (T_{s_i}) \text{ de } H}} (p_{s_i} - p)$ ($pb_{|S|+1} = 0$ si $B_{|S|+1}$ est vide). Comme $pb_{|S|+1} + |S|p$ est une constante, alors, il suffit de minimiser $\sum_{\substack{B_i \text{ contenant deux} \\ \text{tâches } (T_{s_i}, T_{t_i}) \text{ de } H}} (max\{p_{s_i}, p_{t_i}\} - p) + \sum_{\substack{B_i \text{ contenant une} \\ \text{tâche } (T_{s_i}) \text{ de } H}} (p_{s_i} - p)$.

Considérons le sous-ensemble $F = \{$ tâche du stable S compatible avec deux tâches de $H\}$.

Soient $a_{ij} = \begin{cases} 1 & \text{si la tâche } T_i \in F \text{ et la tâche } T_j \in H \text{ sont compatibles} \\ 0 & \text{sinon} \end{cases}$

et $c_i = max\{p_{s_i}, p_{t_i}\}$ (si les deux tâches T_{s_i} et T_{t_i} sont compatibles avec la tâche $T_i \in F$).

Le modèle linéaire correspondant à ce problème est :

$$\min C_{max} = (\sum_{i=1}^{|F|}(c_i - p)x_i) + \sum_{j=1}^{|H|}(1 - \sum_{i=1}^{|F|} a_{ij}x_i)(p_j - p)$$

$$\text{s.c.} \begin{cases} \sum_{i=1}^{|F|} a_{ij}x_i \leq 1 \ \text{ pour } j = 1, \ldots, |H| \\ x_i \in \{0,1\} \ \text{ pour } i = 1, \ldots, |F| \end{cases}$$

où

- $x_i = \begin{cases} 1 \ \text{si les tâches } T_{s_i} \text{ et } T_{t_i} \text{ sont ordonnancées avec la même tâche } T_i \in F \\ 0 \ \text{sinon} \end{cases}$
- $\sum_{i=1}^{|F|}(c_i - p)x_i$: la somme des augmentations des temps de traitement des batchs contenant deux tâches de H.
- $\sum_{j=1}^{|H|}(1 - \sum_{i=1}^{|F|} a_{ij}x_i)(p_j - p)$: la somme des augmentations des temps de traitement des batchs contenant une tâche de H.
- $\sum_{i=1}^{|F|} a_{ij}x_i \leq 1$ indique que chaque tâche de H doit appartenir à, au plus, un batch contenant deux tâches.

Nous avons

$$\begin{aligned} C_{max} &= \sum_{i=1}^{|F|}(c_i - p)x_i + \sum_{j=1}^{|H|}(1 - \sum_{i=1}^{|F|} a_{ij}x_i)(p_j - p) \\ &= \sum_{i=1}^{|F|}(c_i - p)x_i + \sum_{j=1}^{|H|}(p_j - p) - \sum_{j=1}^{|H|}\sum_{i=1}^{|F|} a_{ij}x_i(p_j - p) \\ &= \sum_{i=1}^{|F|}(c_i - p)x_i + \sum_{j=1}^{|H|}(p_j - p) - \sum_{i=1}^{|F|}\sum_{j=1}^{|H|} a_{ij}x_i(p_j - p) \\ &= \sum_{j=1}^{|H|}(p_j - p) + \sum_{i=1}^{|F|}(c_i - p - \sum_{j=1}^{|H|} a_{ij}(p_j - p))x_i \\ &= \sum_{j=1}^{|H|}(p_j - p) + \sum_{i=1}^{|F|}(\max\{p_{s_i}, p_{t_i}\} - p - (p_{s_i} - p) - (p_{t_i} - p))x_i \\ &= \sum_{j=1}^{|H|}(p_j - p) - \sum_{i=1}^{|F|}(\min\{p_{s_i}, p_{t_i}\} - p)x_i. \end{aligned}$$

$\sum_{j=1}^{|H|}(p_j - p)$ est une constante et est supérieure à $\sum_{i=1}^{|F|}(\min\{p_{s_i}, p_{t_i}\} - p)x_i$. Donc, minimiser C_{max} est équivalent à maximiser $\sum_{i=1}^{|F|} l_i x_i$ où $l_i = \min\{p_{s_i}, p_{t_i}\} - p$ si la tâche T_i est compatible avec les deux tâches T_{s_i} et T_{t_i}. Ainsi le modèle linéaire se réduit au problème du couplage de poids maximum dans le graphe valué $G = (V, E)$ où V est l'ensemble des indices des tâches de H, $(i, j) \in E$ si et seulement si les deux tâches T_i et T_j sont

compatibles avec une tâche du stable S et chaque arête (i,j) est valuée par $\min\{p_i,p_j\}-p$.

Finalement,

$$C_{max} = pb_{|S|+1} + |S|p + \sum_{\substack{B_i \text{ contenant deux} \\ \text{tâches } (T_{s_i},T_{t_i}) \text{ de } H}} (max\{p_{s_i},p_{t_i}\}-p) + \sum_{\substack{B_i \text{ contenant une} \\ \text{tâche } (T_{s_i}) \text{ de } H}} (p_{s_i}-p)$$

$$= pb_{|S|+1} + |S|p + \sum_{(T_i,T_j)\in M} \max\{p_i,p_j\} - |M|p + \sum_{T_i\notin M} p_i - (|H|-2|M|)p$$

$$= \sum_{(T_i,T_j)\in M} \max\{p_i,p_j\} + \sum_{T_i\notin M} p_i + (|S|+|M|-|V|)p + pb_{|S|+1}$$

où M est le couplage. □

Le nombre maximum d'arêtes possibles dans le graphe G est $|S|$ et il existe un algorithme pour le couplage de poids maximum en $O(mn+n^2logn)$ proposé par H.N. Gabow [113]. Donc, la complexité de l'algorithme MWMAA est $O(|S||K|+|K|^2log|K|)$. Par conséquent, l'algorithme s'exécute en $O(n^2logn)$.

Exemple 5.2 *Nous disposons de 12 tâches $T_1,\ldots,T_{12}$ à ordonnancer sur une machine à traitement par batch d'une capacité infinie. Les temps de traitement des tâches sont donnés dans le tableau 5.2. Le graphe de compatibilité scindé est représenté par la figure 5.14.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
p_i	6	8	7	6	9	3	5	5	5	5	5	5

TAB. 5.2: *Temps de traitement des tâches de l'exemple 5.2.*

L'exécution de l'algorithme donne :

1- $Z0 = 5\times 6 + max\{6,8,7,6,9,3\} = 39$;

2- Les 6 premiers batchs sont :
$B_1=\{T_7\}$; $pb_1=5$;
$B_2=\{T_8\}$; $pb_2=5$;
$B_3=\{T_9\}$; $pb_3=5$;
$B_4=\{T_{10}\}$; $pb_4=5$;
$B_5=\{T_{11}\}$; $pb_5=5$;
$B_6=\{T_{12}\}$; $pb_6=5$;

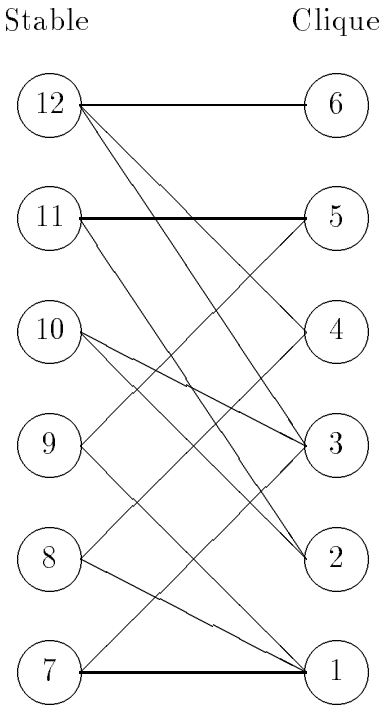


FIG. 5.14: *Graphe de compatibilité scindé de l'exemple 5.2.*

3- Non ;

4- $B_6 = \{T_{12}, T_6\}$; $pb_6 = 5$;

5- Le graphe valué est représenté par la figure 5.15 ;

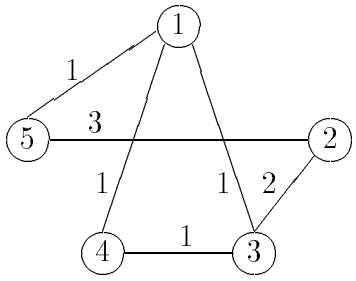


FIG. 5.15: *Le graphe valué* $G = (V, E)$.

6- $M = \{(2,5), (3,4)\}$;

7- $Z = 9 + 7 + 6 + (6 + 2 - 5) \times 5 = 37 < 39$;

8- La solution optimale est :

$B_1 = \{T_7, T_1\}$; $pb_1 = 6$;
$B_2 = \{T_8\}$; $pb_2 = 5$;
$B_3 = \{T_9\}$; $pb_3 = 5$;
$B_4 = \{T_{10}\}$; $pb_4 = 5$;
$B_5 = \{T_{11}, T_2, T_5\}$; $pb_5 = 9$;
$B_6 = \{T_{12}, T_6, T_3, T_4\}$; $pb_6 = 7$;
$C_{max} = 37$;

L'ordonnancement optimal est représenté par la figure 5.16.

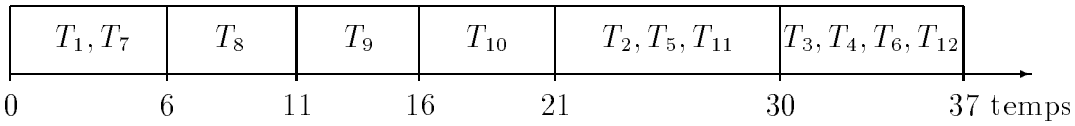


FIG. 5.16: *L'ordonnancement optimal de l'exemple 5.2.*

5.2.2 Ordonnancement dynamique[4]

Nous considérons dans ce sous-paragraphe le cas où le graphe scindé est complet ensuite le cas où les tâches de la clique K sont toutes disponibles à l'instant 0.

Si chaque tâche du stable S est compatible avec chaque tâche de la clique K, alors on peut penser à construire pour le stable S l'ordonnancement optimal pour le problème $1/r_i, p_i = p/C_{max}$, ensuite construire un ordonnancement réalisable avec une date de fin de traitement supérieure ou égale à la date de fin de traitement de cet sous-ordonnancement, qui est bien sûr une borne inférieure. Soit l'algorithme DPA1 suivant :

Algorithme DPA1 ;
début

1- Ranger les tâches du stable S suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_1, \ldots, T_{|S|}$)
2- $sS_1 := r_1$;
3- **Pour** $i := 2$ **à** $|S|$
faire $sS_i := max\{r_i, sS_{i-1} + 1\}$
fait ;
4- Ordonnancer chaque tâche T_i du stable S à l'instant sS_i ;
5- Ranger les tâches de la clique K suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_{|S|+1}, \ldots, T_{|S|+|K|}$)

4. Les résultats de ce paragraphe sont publiés dans [17]

6- **Pour** $j := |S| + 1$ **à** $|S| + |K|$
faire dK_j := la plus petite date à laquelle on peut ordonnancer la tâche T_j (la dates de disponibilité, les contraintes de compatibilité et la capacité du batch sont respectés)
fait ;
7- Ordonnancer chaque tâche T_j de la clique K à l'instant dK_j ;
fin.

Théorème 5.9 *L'algorithme DPA1 résout le problème* $B1/G = (S, K; S \times K), b, r_i, p_i = 1/C_{max}$ *en* $O(nlogn)$ *(le cas où le graphe scindé est complet).*

Preuve. Dans ce cas, chaque tâche du stable S est compatible avec toutes les tâches de la clique K.

Montrons qu'il existe une solution optimale où chaque tâche T_i du stable S est ordonnancée exactement à l'instant sS_i donné par l'algorithme DPA1. Supposons que, dans une solution optimale, il existe une tâche T_i du stable S ordonnancée après l'instant sS_i (la tâche T_i est ordonnancée seule ou avec, au moins, une tâche de la clique K). 3 cas peuvent apparaître :

- aucun batch n'est ordonnancé à cet instant sS_i. Dans ce cas, la tâche T_i est simplement translatée et ordonnancée à l'instant sS_i.
- un batch contenant, seulement, des tâches de la clique K est ordonnancé à l'instant sS_i. Dans ce cas, permuter la tâche T_i avec une tâche de ce batch.
- un batch contenant une tâche du stable S et, éventuellement, des tâches de la clique K est ordonnancé à l'instant sS_i. Dans ce cas, permuter les deux tâches du stable S.

En appliquant ce raisonnement au plus $|S|$ fois, on obtient une solution optimale où chaque tâche T_i du stable S est ordonnancée avant ou à la même date associée sS_i. La tâche T_1 doit être ordonnancée à l'instant sS_1, car elle est disponible à cet instant ($sS_1 = r_1$). La tâche T_2 doit être ordonnancée à l'instant sS_2, car $sS_2 = max\{r_2, sS_1 + 1\}$. En considérant les tâches $T_3, \ldots, t_{|S|}$, on obtient, récursivement et de la même manière, que les tâches $T_1, \ldots, T_{|S|}$ doivent être ordonnancées, respectivement, aux instants $sS_1, \ldots, sS_{|S|}$.

Supposons qu'il existe une solution $B_1, \ldots, B_i, B_{i+1}, \ldots, B_m$ avec $T_k \in B_i$ et $T_{k'} \in B_{i+1}$ et $r_k > r_{k'}$ (c.-à-d. dans l'ordonnancement, les tâches de la clique K ne sont pas rangées suivant l'ordre croissant de leurs dates de disponibilité après avoir rangé les tâches de chaque batch suivant l'ordre croissant de leurs dates de disponibilité). Notons par y la date de fin de traitement de cet ordonnancement. Formons, à partir de cette solution, une nouvelle solution $B_1, \ldots, B'_i = B_i \setminus \{T_k\} \cup \{T_{k'}\}, B'_{i+1} = B_{i+1} \setminus \{T_{k'}\} \cup \{T_k\}, \ldots, B_m$

et notons par y' la date de fin de traitement de cet ordonnancement. Nous avons $Cb'_i \leq Cb_i$ et $Cb'_j \leq Cb_j$ donc $y' \leq y$. Ainsi, il existe une solution optimale où les tâches de la clique K sont rangées suivant l'ordre croissant de leurs dates de disponibilité.

Finalement, pour déterminer une solution optimale, il suffit, de ranger les tâches de la clique K suivant l'ordre croissant de leurs dates de disponibilité et ordonnancer chacune d'elles, dans l'ordre, à la plus petite date possible (en respectant la date de disponibilité et la capacité du batch). □

Si les tâches de la clique K ont une date de disponibilité nulle, alors il suffit de résoudre le problème $1/r_i, p_i = 1/C_{max}$ pour les tâches du stable S, construire un réseau de transport et déterminer un flot maximum dans ce réseau. Pour avoir l'ordonnancement optimal, il suffit de traiter les tâches de la clique K ayant un flux égal à 1 avec une tâche du stable S dans un même batch et le reste des tâches de la clique K dans les temps morts. Considérons l'algorithme DPMFA suivant :

Algorithme DPMFA ;
début
1- Ranger les tâches du stable S suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_1, \ldots, T_{|S|}$)
2- $sS_1 := r_1$;
3- **Pour** $i := 2$ **à** $|S|$
faire $sS_i := max\{r_i, sS_{i-1} + 1\}$
fait ;
4- Ordonnancer chaque tâche T_i du stable S à l'instant sS_i ;
5- Soit $X = \{0, \ldots, x\}$ avec $x \geq sS_{|S|}$ et où chaque $[i, i+1]$ $(i \in X)$ est un intervalle de temps, et soit $L = \{x+1, \ldots, x+|K|\}$ où chaque (sommet) $x+i$ correspond à la tâche $T_{|S|+i}$ de la clique K.
Définissons un réseau $R_x = (X \cup L, E, C)$ comme suit :

- Connecter la source s à chaque sommet $i \in L$ par un arc de capacité égale à 1 ;
- Connecter chaque sommet $j \in X$ au puits t par un arc de capacité égale à $b-1$ si $j \in \{sS_1, \ldots, sS_{|S|}\}$ ou égale à b si $j \notin \{sS_1, \ldots, sS_{|S|}\}$;
- Connecter chaque sommet $i \in L$ à tout sommet $j \in X \setminus \{sS_1, \ldots, sS_{|S|}\}$, par un arc de capacité égale à 1 ;
- Connecter un sommet $i \in L$ à un sommet $j \in \{sS_1, \ldots, sS_{|S|}\}$ par un arc de capacité égale à 1 si la tâche $T_{|S|+i-x}$ et la tâche T_t ayant $sS_t = j$ sont compatibles ;

6- Soit q le plus petit entier tel que l'ensemble $F = \{0, \ldots, q\}$ contient toutes les valeurs $sS_1, \ldots, sS_{|S|}$ et $\frac{|K|}{b}$ autres valeurs ;
si $q = sS_{|S|}$
alors - Ordonnancer toutes les tâches de la clique K dans les $\frac{|K|}{b}$ intervalles de temps libres ;
- $C_{max} = sS_{|S|} + 1$;

sinon - Déterminer, par dichotomie, le plus petit entier x ($sS_{|S|} \leq x \leq q$) tel que le flot maximum, dans le réseau R_x, soit égal à $|K|$ (toutes les tâches de la K sont affectées) ;

- Ordonnancer la tâche $T_{|S|+i}$ de la clique K à l'instant $j \in X$ si le flux entre les sommets $x+i$ et j est égal à 1 ;

- $C_{max} = x + 1$;

finsi ;
fin.

Théorème 5.10 *L'algorithme DPMFA résout le problème* $B1/G = (S, K; E), b, r_S, r_K = 0, p_i = 1/C_{max}$ *avec une complexité* $O(n^3 log\frac{n}{b})$ *(le cas où les tâches de la clique* K *sont toutes disponibles à l'instant* 0*).*

Preuve. Dans ce cas, chaque tâche de la clique K est disponible avant, ou à la même date que, les tâches du stable S.

Montrons qu'il existe une solution optimale où chaque tâche T_i du stable S est ordonnancée exactement à l'instant sS_i donné par l'algorithme DPMFA. Supposons que, dans une solution optimale, il existe une tâche T_i du stable S ordonnancée après l'instant sS_i (la tâche T_i est ordonnancée seule ou avec, au moins, une tâche de la clique K). 3 cas peuvent apparaître :

- aucun batch n'est ordonnancé à cet instant sS_i. Dans ce cas, le batch contenant la tâche T_i est simplement translaté et ordonnancé à l'instant sS_i.

- un batch contenant, seulement, des tâches de la clique K est ordonnancé à l'instant sS_i. Dans ce cas, permuter les deux batchs.

- un batch contenant une tâche du stable S et, éventuellement, des tâches de la clique K est ordonnancé à l'instant sS_i. Dans ce cas, permuter les deux batchs.

En appliquant ce raisonnement au plus $|S|$ fois, on obtient une solution optimale où chaque tâche T_i du stable S est ordonnancée avant ou à la même date associée sS_i. La tâche T_1 doit être ordonnancée à l'instant sS_1, car elle est disponible à cet instant ($sS_1 = r_1$). La tâche T_2 doit être ordonnancée à l'instant sS_2, car $sS_2 = max\{r_2, sS_1 + 1\}$. En considérant les tâches $T_3, \ldots, t_{|S|}$, on obtient, récursivement et de la même manière, que les tâches $T_1, \ldots, T_{|S|}$ doivent être ordonnancées, respectivement, aux instants $sS_1, \ldots, sS_{|S|}$.

Pour ordonnancer les tâches de la clique K, qui sont deux à deux compatibles et disponibles à la même date 0, il suffit de les affecter en utilisant la technique du flot maximum définie dans l'algorithme DPMFA.

La construction du réseau est de l'ordre de $(|S| + \frac{|K|}{b})|K|$ (on ne considère que $|S| + \frac{|K|}{b}$ sommets de X). Comme le réseau est sans cycles et le flux maximum borné par $|K|$, alors, la complexité du problème pour déterminer un flot maximum est $O(|S||K|^2 + \frac{|K|^3}{b})$ en utilisant la méthode "minimum cost augmentation" [118]. Ainsi, la complexité de cet algorithme est $O(n^3 log\frac{n}{b})$ □

Exemple 5.3 *Soient à ordonnancer 6 tâches T_1, T_2, T_3, T_4, T_5 et T_6 sur une machine à traitement par batch de capacité égale à 2. Les temps de traitement des tâches sont tous égaux à 1. Les dates de disponibilité des tâches sont données dans le tableau 5.3. Le graphe de compatibilité scindé est représenté par la figure 5.17.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6
r_i	1	2	0	0	0	0

TAB. 5.3: *Dates de disponibilité des tâches de l'exemple 5.3.*

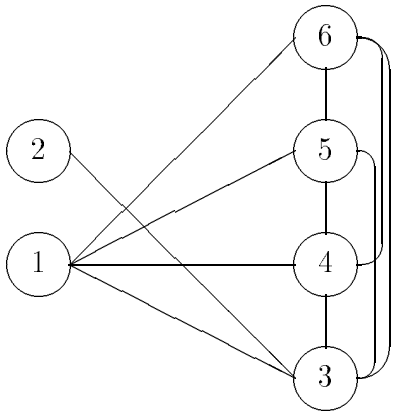


FIG. 5.17: *graphe de compatibilité scindé $G = (S, K; E)$ de l'exemple 5.3.*

L'exécution de l'algorithme donne :

- $sS_1 = 1$; $sS_2 = 2$;
  la tâche T_1 est ordonnancée à l'instant 1 ;
  la tâche T_2 est ordonnancée à l'instant 2 ;

- $q = 3$ et $F = \{0, 1, 2, 3\}$;

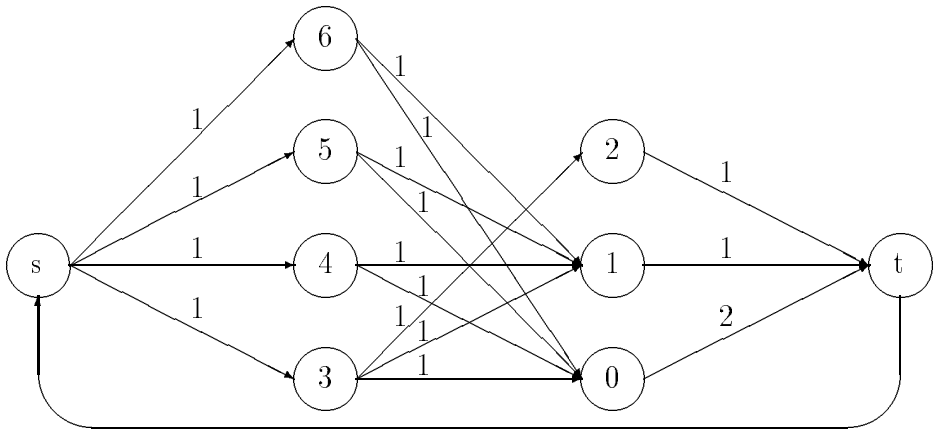


FIG. 5.18: *Réseau R_2 de l'exemple 5.3.*

- $x = 2$;
  Le réseau correspondant R_2 est représenté par la figure 5.18 ;
  Les tâches T_5 et T_6 sont ordonnancées à l'instant 0 ;
  La tâche T_4 est ordonnancée à l'instant 1 ;
  La tâche T_3 est ordonnancée à l'instant 2 ;
  $C_{max} = 3$;

L'ordonnancement est représenté par la figure 5.19.

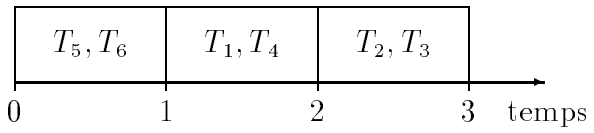


FIG. 5.19: *Ordonnancement de l'exemple 5.3.*

Notons que cet algorithme résout, aussi, les problèmes où les dates de disponibilité des tâches du stable S_2 ne sont pas toutes égales à 0.

Corollaire 5.11 *L'algorithme DPMFA résout le problème $B1/G = (S, K; E), b = 2, r_i, p_i = 1/C_{max}$ où la condition "si les tâches T_i ($\in S$) et T_j ($\in K$) sont compatibles alors $r_i \geq r_j$" est satisfaite (la tâche T_j est disponible avant, ou à la même date que, la date de disponibilité de la tâche T_i).*

Preuve. Remarquons que, si un batch B_k contient une tâche $T_i \in S$ avec d'autres tâches de la clique K alors la date de disponibilité de ce batch est égale à la date de disponibilité de la tâche T_i c.-à-d. r_i. Ainsi, la permutation de deux batchs dans le preuve précédente est, aussi, possible. □

Aussi cet algorithme résout le problème $B1/G = (S, K; E), b = 2, r_i, p_i = p/C_{max}$ avec les conditions : $r_i = \alpha_i p$ pour $i = 1, \ldots, n$ et "si les tâches T_i $(\in S)$ et T_j $(\in K)$ sont compatibles alors $r_i \geq r_j$". Il suffit, de résoudre le problème avec α_i comme dates de disponibilité et, dans la solution optimale, multiplier toutes les dates de début d'exécution par p.

Chapitre 6

Graphe biparti[1]

Nous considérons dans ce chapitre le cas d'un graphe biparti noté $G = (S_1, S_2; E)$. Notons que la taille de toute clique est inférieure ou égale à 2, donc toutes les contraintes sur la taille du batch sont équivalentes à $b = 2$. Dans le premier paragraphe, nous montrons que le problème est difficile. Nous présentons, dans le second paragraphe, quelques sous-problèmes polynomiaux. Enfin, dans le dernier paragraphe nous présentons une heuristique avec un rapport de performance dans le pire des cas ègal à 2.

6.1 Problème difficile

Nous montrons, dans ce paragraphe, que le problème $B1/G = (S_1, S_2; E), b = 2, r_i, p_i = 1/C_{max}$ est NP-difficile au sens fort.

Pour la preuve suivante, nous utilisons le problème de décision ci-dessous, qui est connu pour être NP-Complet au sens fort [10], et utilisé dans la même référence pour le problème d'ordonnancement à contraintes de ressources. Nous développons de nouvelles idées, pour le problème de l'ordonnancement par batchs avec un graphe de compatibilité biparti, en s'inspirant des idées développées dans cette référence.

TRIPARTITE ONE-IN-THREE 3-3 SAT : Etant donnés un ensemble $X = \{x_i \ / \ 1 \leq i \leq 3m\}$ de variables booléennes et une collection C de $3m$ clauses sur X (c.-à-d. une fonction booléenne $F = C_1 \wedge \ldots \wedge C_{3m}$ où chaque clause C_i est une disjonction), tels que :

- aucune clause $C_i \in C$ ne contient une négation de variable booléenne.

1. Les résultats de ce chapitre sont publiés dans [18]

- chaque clause $C_i \in C$ a $|C_i| = 3$.

- chaque variable $x_i \in X$ apparaît dans exactement trois clauses.

- l'ensemble des clauses est partitionable en trois sous-ensembles tels que chaque variable booléenne apparaît, exactement, une seule fois dans, exactement, une seule clause de chaque sous ensemble.

Existe-t-il une affectation de la valeur de vérité "vrai" aux variables booléennes de X telle que chaque clause de C a, exactement, une seule variable booléenne ayant la valeur de vérité "vrai" ?

Les trois sous-ensembles de C sont indicés C_s $(s = 1, \ldots, m)$ pour le premier sous-ensemble, C_t $(t = m+1, \ldots, 2m)$ pour le second et C_u $(u = 2m+1, \ldots, 3m)$ pour le troisième. Pour chaque variable $x_i \in X$ $(i = 1, \ldots, 3m)$, soient $s(i)$, $t(i)$ et $u(i)$ les indices des clauses où x_i apparaît, c.-à-d. $x_i \in C_{s(i)}$, $C_{t(i)}$ et $C_{u(i)}$.

Théorème 6.1 *Le problème* $B1/G = (S_1, S_2; E), b = 2, r_i, p_i = 1/C_{max}$ *est NP-difficile au sens fort.*

Preuve. Considérons le problème de décision BSCG associé à ce problème.

Etant donné une instance arbitraire du problème TRIPARTITE ONE-IN-THREE 3-3 SAT, nous construisons une instance du problème BSCG comme suit :

- A chaque variable booléenne x_i est associée huit tâches : les trois premières tâches, notées par T_i, $T_i^{'}$ et $T_i^{''}$, représentent le choix d'avoir $x_i = vrai$; les trois autres, notées par F_i, $F_i^{'}$ et $F_i^{''}$, représentent l'alternative $x_i = faux$, et les deux tâches restantes, notées par G_i et E_i, sont pour la prise de décisions. Le nombre total de tâches est $24m$.

- Toutes les tâches ont un même temps de traitement égal à 1.

- Les dates de disponibilité sont données à la table 6.1.

- Toutes les tâches T_i, F_i, E_i et G_i pour $i = 1, \ldots, 3m$ sont incompatibles entre elles, elles forment le stable S_1. Toutes les tâches $T_i^{'}$, $T_i^{''}$, $F_i^{'}$ et $F_i^{''}$ pour $i = 1, \ldots, 3m$ sont incompatibles entre elles, elles forment le stable S_2. Les autres contraintes de compatibilité sont définies entre les tâches ayant un même indice i, elles sont données à la figure 6.1.

Existe-t-il un ordonnancement avec un makespan inférieur ou égal à $12m$?

Tâche	Date de disponibilité
T_i	$7m - 2t(i)$
T_i'	$9m - s(i)$
T_i''	$3m - u(i)$
F_i	$7m - t(i)$
F_i'	$8m - 2s(i)$
F_i''	$7m - 2u(i)$
E_i	0
G_i	$9m$

TAB. 6.1: *Dates de disponibilité des tâches indicées i.*

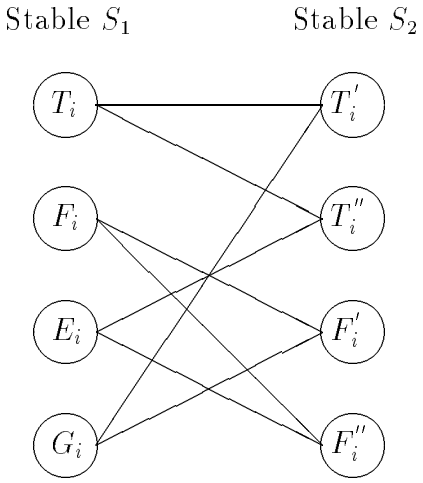


FIG. 6.1: *Graphe de compatibilité biparti des tâches indicées i.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que TRIPARTITE ONE-IN-THREE 3-3 SAT a une solution si et seulement si BSCG a une solution.

Supposons que TRIPARTITE ONE-IN-THREE 3-3 SAT a une solution (il existe une affectation de la valeur de vérité "vrai" aux variables booléennes de X telle que chaque clause de C a exactement une variable booléenne ayant la valeur de vérité "vrai"). Nous construisons une solution pour BSCG comme suit : (voir les figures 6.2 et 6.3)

- Pour chaque variable booléenne x_i ayant une affectation "vrai", ordonnancer les tâches suivantes dans un même batch.
  - E_i et T_i'' à l'instant $3m - u(i)$,

- F_i et F_i'' à l'instant $7m - t(i)$,
- T_i et T_i' à l'instant $9m - s(i)$ et
- G_i et F_i' à l'instant $12m - i$.

- Pour chaque variable booléenne x_i ayant une affectation "faux", ordonnancer les tâches G_i et T_i' dans, un même batch, à l'instant $12m - i$.

- Pour chaque paire x_i et x_j de variables booléennes ayant une affectation "faux" et
  - pour chaque clause C_u ($u = 2m+1, \ldots, 3m$), ordonnancer à l'instant $7m-2u$ et $7m-2u+1$, respectivement, les deux tâches E_i et F_i'' dans un même batch et les deux tâches E_j et F_j'' dans un même batch, telles que $u(i) = u(j) = u$ (les dates de disponibilité des tâches F_i'' et F_j'' sont égales à $7m - 2u$).
  - pour chaque clause C_t ($t = m+1, \ldots, 2m$), ordonnancer à l'instant $7m - 2t$ et $7m-2t+1$, respectivement, les deux tâches T_i et T_i'' dans un même batch et les deux tâches T_j et T_j'' dans un même batch, telles que $t(i) = t(j) = t$ (les dates de disponibilité des tâches T_i et T_j sont égales à $7m - 2t$).
  - pour chaque clause C_s ($s = 1, \ldots, m$), ordonnancer à l'instant $8m - 2s$ et $8m - 2s + 1$, respectivement, les deux tâches F_i et F_i' dans un même batch et les deux tâches F_j et F_j' dans un même batch, telles que $s(i) = s(j) = s$ (les dates de disponibilité des tâches F_i' et F_j' sont égales à $8m - 2s$).

L'ordonnancement obtenu est représenté par la figure 6.2.

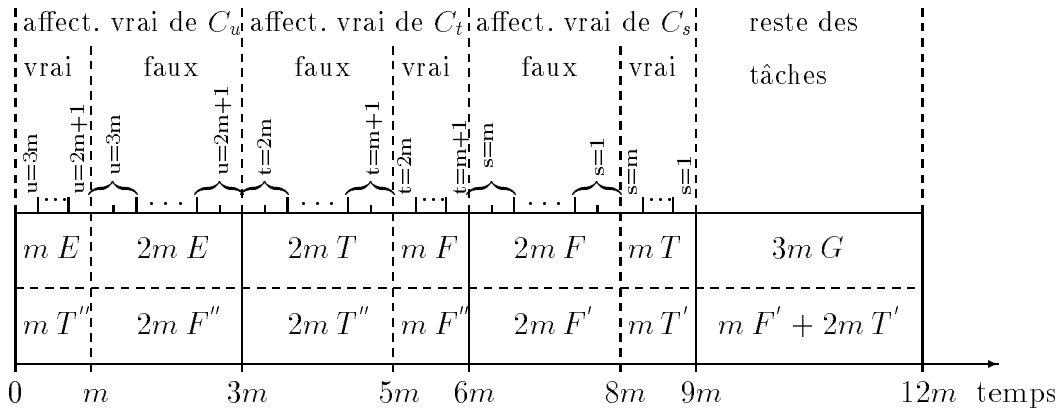


FIG. 6.2: *Ordonnancement des tâches du problème BSCG.*

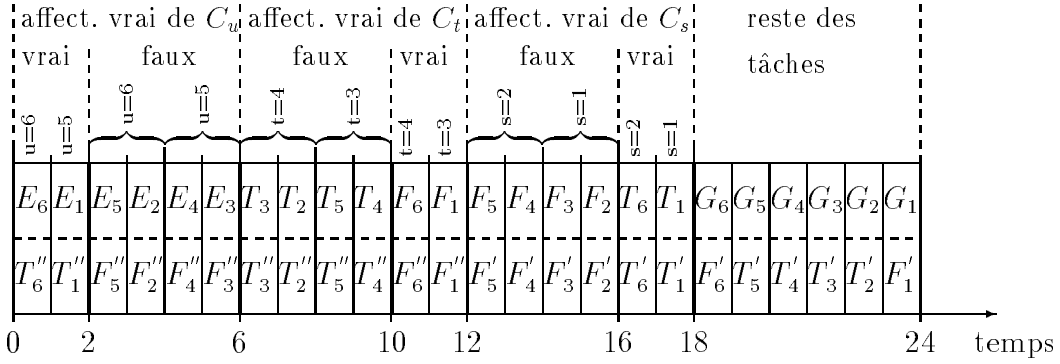


FIG. 6.3: *Solution du problème BSCG correspondant à la solution ($x_1 = vrai$, $x_6 = vrai$ et $x_i = faux$ pour $i = 2, 3, 4, 5$) du problème TRIPARTITE-ONE-THREE 3-3 SAT : $X = \{x_1, \ldots, x_6\}$, $m = 2$ et $F = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_6) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_5 \vee x_6)$.*

Inversement, supposons qu'il existe un ordonnancement avec une date de fin de traitement $C_{max} \leq 12m$. Comme le nombre total des tâches dans chaque stable, S_1 et S_2, est égal à $12m$, alors, $C_{max} = 12m$ et dans chaque batch, nous avons une tâche du stable S_1 (T_i, F_i, E_i ou G_i) et une tâche du stable S_2 (T'_i, T''_i, F'_i ou F''_i). De la structure du graphe biparti de compatibilité, nous concluons que pour chaque i, les huit tâches T_i, F_i, E_i, G_i, T'_i, T''_i, F'_i et F''_i ne peuvent s'ordonnancer que selon les deux alternatives de configuration de batchs suivantes :

(a) T_i avec T'_i, G_i avec F'_i, F_i avec F''_i et E_i avec T''_i, ou

(b) T_i avec T''_i, G_i avec T'_i, F_i avec F'_i et E_i avec F''_i.

Les dates de disponibilité impliquent que toutes les $3m$ G_i tâches doivent être ordonnancées dans l'intervalle de temps $[9m, 12m]$, et toutes les $3m$ E_i tâches doivent être ordonnancées dans l'intervalle de temps $[0, 3m]$ (car les tâches T_i et F_i sont disponibles après l'instant $3m$).

Des considérations précédentes, nous avons :

- m des $3m$ T''_i tâches doivent être ordonnancées, avec m des $3m$ E_i tâches, entre l'instant 0 et l'instant m (car les tâches F''_i sont disponibles après l'instant m).
- Aussi, $2m$ des $3m$ T_i tâches doivent être ordonnancées, avec les $2m$ T''_i tâches restantes, dans l'intervalle de temps $[3m, 5m]$ (car les tâches F_i sont disponibles après l'instant $5m$ et les tâches T'_i sont disponibles après l'instant $8m$).

- Ainsi, $2m$ des $3m$ $F_i^{''}$ tâches doivent être ordonnancées, avec les $2m$ E_i tâches restantes, dans l'intervalle de temps $[m, 3m]$, et les m $F^{''}$ tâches restantes doivent être ordonnancées dans l'intervalle de temps $[5m, 6m]$ (car toutes les tâches $T_i^{'}$ et $F_i^{'}$ sont disponibles après l'instant $6m$) avec m des $3m$ F_i tâches.
- Aussi, $2m$ des $3m$ $F_i^{'}$ tâches doivent être ordonnancées dans l'intervalle de temps $[6m, 8m]$ (car les tâches $T_i^{'}$ sont disponibles après l'instant $8m$) avec les $2m$ F_i tâches restantes.
- Donc, les m T_i tâches restantes doivent être ordonnancées dans l'intervalle de temps $[8m, 9m]$ avec m des $3m$ $T_i^{'}$ tâches.
- Finalement, les m $F_i^{'}$ et $2m$ $T_i^{'}$ tâches restantes doivent être ordonnancées, entre l'instant $9m$ et l'instant $12m$, avec les $3m$ G_i tâches.

Nous concluons que nous avons m sous-ensembles de tâches (T_i, F_i, E_i, G_i, $T_i^{'}$, $T_i^{''}$, $F_i^{'}$ et $F_i^{''}$) qui apparaît dans la configuration (a) et les $2m$ autres dans la configuration (b). En procédant par récurrence sur s ($s = 1, \ldots, m$), il vient des dates de disponibilité des tâches $T^{'}$ que, pour chaque s, les tâches T_i et $T_i^{'}$, ordonnancées dans un même batch à l'instant $9m - s$, doivent avoir l'indice i tel que $s(i) = s$. Il s'en suit aussi que, à partir des dates de disponibilité des tâches $F^{'}$, que pour chaque s, les tâches F_j et $F_j^{'}$ ordonnancées dans un même batch à l'instant $8m - 2s$ et les tâches F_k et $F_k^{'}$ ordonnancées dans un même bacth à l'instant $8m - 2s + 1$, doivent avoir les indices j et k tels que $s(j) = s(k) = s$. Par des récurrences analogues sur t ($t = m+1, \ldots, 2m$) et u ($u = 2m+1, \ldots, 3m$), respectivement, on montre que les tâches F_i et $F_i^{''}$, ordonnancées dans un même batch à l'instant $7m - t$, doivent avoir l'indice i tel que $t(i) = t$ et les taches E_i et $T_i^{''}$, ordonnancées dans un même batch à l'instant $3m - u$, doivent avoir l'indice i tel que $u(i) = u$. Il vient que les m variables booléennes pour lesquelles les sous-ensembles de tâches correspondants apparaissent dans la configuration (a) auront l'affectation "vrai" et les $2m$ autres variables booléennes auront l'affectation "faux", ainsi nous avons une affectation de valeur de vérité "vrai" satisfaisant la fonction booléenne F du problème TRIPARTITE ONE-IN-THREE 3-3 SAT. □

6.2 Problèmes polynomiaux

Nous considérons dans ce paragraphe le cas où le graphe biparti est complet, ensuite le cas où les tâches du stable S_2 sont toutes disponibles à l'instant 0.

Si chaque tâche du stable S_1 est compatible avec chaque tâche du stable S_2, alors on peut penser à construire pour chaque stable S_1 et S_2 l'ordonnancement optimal pour le problème $1/r_i, p_i = 1/C_{max}$, ensuite fusionner ces deux sous-ordonnancements, pour avoir un ordonnancement réalisable avec une date de fin de traitement supérieure, ou égale, au maximum des dates de fin de traitement de ces deux sous-ordonnancements, qui est évidemment une borne inférieure. Soit l'algorithme suivant :

Algorithme DPA2 ;
début

1- Ranger les tâches du stable S_1 suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_1, \ldots, T_{|S_1|}$)

2- $sS1_1 := r_1$;

3- **Pour** $i := 2$ **à** $|S_1|$
faire $sS1_i := max\{r_i, sS1_{i-1} + 1\}$
fait ;

4- Ordonnancer chaque tâche T_i du stable S_1 à l'instant $sS1_i$;

5- Ranger les tâches du stable S_2 suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_{|S_1|+1}, \ldots, T_{|S_1|+|S_2|}$)

6- **Pour** $j := |S_1| + 1$ **à** $|S_1| + |S_2|$
faire $sS2_j$:= la plus petite date à laquelle on peut ordonnancer la tâche T_j (la date de disponibilité, les contraintes de compatibilité et la capacité du batch sont respectés)
fait ;

7- Ordonnancer chaque tâche T_j du stable S_2 à l'instant $sS2_j$;

fin.

Théorème 6.2 *L'algorithme DPA2 résout le problème* $B1/G = (S_1, S_2; S_1 \times S_2), b = 2, r_i, p_i = 1/C_{max}$ *en* $O(nlogn)$ *(le cas où le graphe biparti est complet).*

Preuve. Dans ce cas, chaque tâche du stable S_1 est compatible avec toutes les tâches du stable S_2.

Montrons qu'il existe une solution optimale où chaque tâche T_i du stable S_1 est ordonnancée exactement à l'instant $sS1_i$ donné par l'algorithme DPA2. Supposons que, dans une solution optimale, il existe une tâche T_i du stable S_1 ordonnancée après l'instant $sS1_i$ (la tâche T_i est ordonnancée seule ou avec, au moins, une tâche du stable S_2). 3 cas peuvent apparaître :

- aucun batch n'est ordonnancé à cet instant $sS1_i$. Dans ce cas, la tâche T_i est simplement translatée et ordonnancée à l'instant $sS1_i$.
- un batch contenant, seulement, des tâches du stable S_2 est ordonnancé à l'instant $sS1_i$. Dans ce cas, la tâche T_i est simplement translatée et ordonnancée à l'instant $sS1_i$.
- un batch contenant une tâche du stable S_1 et, éventuellement, des tâches du stable S_2 est ordonnancé à l'instant $sS1_i$. Dans ce cas, permuter les deux tâches du stable S_1.

En appliquant ce raisonnement au plus $|S_1|$ fois, on obtient une solution optimale où chaque tâche T_i du stable S_1 est ordonnancée avant ou à la même date associée $sS1_i$. La tâche T_1 doit être ordonnancée à l'instant $sS1_1$, car elle est disponible à cet instant ($sS1_1 = r_1$). La tâche T_2 doit être ordonnancée à l'instant $sS1_2$, car $sS1_2 = max\{r_2, sS1_1 + 1\}$. En considérant les tâches $T_3, \ldots, t_{|S_1|}$, on obtient, récursivement et de la même manière, que les tâches $T_1, \ldots, T_{|S_1|}$ doivent être ordonnancées, respectivement, aux instants $sS1_1, \ldots, sS1_{|S_1|}$.

Supposons qu'il existe une solution $B_1, \ldots, B_i, B_{i+1}, \ldots, B_m$ avec $T_k \in B_i$ et $T_{k'} \in B_{i+1}$ et $r_k > r_{k'}$ (c.-à-d. dans l'ordonnancement, les tâches du stable S_2 ne sont pas rangées suivant l'ordre croissant de leurs dates de disponibilité après avoir rangé les tâches de chaque batch suivant l'ordre croissant de leurs dates de disponibilité). Notons par y la date de fin de traitement de cet ordonnancement. Formons, à partir de cette solution, une nouvelle solution $B_1, \ldots, B'_i = B_i \setminus \{T_k\} \cup \{T_{k'}\}, B'_{i+1} = B_{i+1} \setminus \{T_{k'}\} \cup \{T_k\}, \ldots, B_m$ et notons par y' la date de fin de traitement de cet ordonnancement. Nous avons $Cb'_i \leq Cb_i$ et $Cb'_j \leq Cb_j$ donc $y' \leq y$. Ainsi, il existe une solution optimale où les tâches du stable S_2 sont rangées suivant l'ordre croissant de leurs dates de disponibilité.

Finalement, pour déterminer une solution optimale, il suffit, de ranger les tâches du stable S_2 suivant l'ordre croissant de leurs dates de disponibilité et ordonnancer chacune d'elles, dans l'ordre, à la plus petite date possible (en respectant la date de disponibilité et la capacité du batch). □

Si toutes les tâches du stable S_2 ont une date de disponibilité nulle, alors on peut penser à résoudre le problème $1/r_i, p_i = 1/C_{max}$ de la manière suivante : construire un couplage maximum, ensuite ordonnancer chaque paire de tâches du couplage dans un même batch et le reste des tâches (du stable S_2) dans les temps morts. Considérons l'algorithme DPMCMA suivant :

Algorithme DPMCMA ;
début

1- Ranger les tâches du stable S_1 suivant l'ordre croissant de leurs dates de disponibilité ; (soit $T_1, \ldots, T_{|S_1|}$)

2- $sS1_1 := r_1$;

3- **Pour** $i := 2$ **à** $|S_1|$
faire $sS1_i := max\{r_i, sS1_{i-1} + 1\}$
fait ;

4- Ordonnancer chaque tâche T_i du stable S_1 à l'instant $d1_i$;

5- Soit $X = \{0, \ldots, x\}$ avec $x \geq sS1_{|S_1|}$ et où chaque $[i, i+1]$ $(i \in X)$ est un intervalle de temps, et soit $L = \{x+1, \ldots, x+|S_2|\}$ où chaque (sommet) $x+i$ correspond à la tâche $T_{|S_1|+i}$ du stable S_2.
Définir un nouveau graphe biparti $H_x = (X, L; E)$ comme suit :

- Connecter chaque sommet $i \in L$ à tous les sommets $j \in X \setminus \{sS1_1, \ldots, sS1_{|S_1|}\}$ par une arête ;

– Connecter le sommet $i \in L$ à un sommet $j \in \{sS1_1, \ldots, sS1_{|S_1|}\}$ par une arête si la tâche $T_{|S_1|+i-x}$ et la tâche T_t ayant $sS1_t = j$ sont compatibles ;

6- Soit q le plus petit entier tel que l'ensemble $F = \{0, \ldots, q\}$ contient toutes les valeurs $sS1_1, \ldots, sS1_{|S_1|}$ et $|S_2|$ autres valeurs ;
Si $q = sS1_{|S_1|}$
alors - Ordonnancer toutes les tâches du stable S_2 dans les $|S_2|$ intervalles de temps libres ;
- $C_{max} = sS1_{|S_1|} + 1$;
sinon - Déterminer, par dichotomie, le plus petit entier x $(sS1_{|S_1|} \leq x \leq q)$ tel que la cardinalité du couplage maximum, dans le graphe biparti H_x, soit égale à $|S_2|$ (toutes les tâches du stable S_2 sont affectées) ;
- Ordonnancer chaque tâche $T_{|S_1|+i}$ du stable S_2 à l'instant $j \in X$ si les sommets $x + i$ et j sont couplées dans le couplage ;
- $C_{max} = x + 1$;
finsi ;
fin.

Théorème 6.3 *L'algorithme DPMCMA résout le problème* $B1/G = (S_1, S_2; E), b = 2, r_{S_1}, r_{S_2} = 0, p_i = 1/C_{max}$ *avec une complexité de* $O(n^{2.5}logn)$ *(le cas où les tâches du stable* S_2 *sont toutes disponibles à l'instant* 0*).*

Preuve. Dans ce cas, chaque tâche du stable S_2 est disponible avant, ou à la même date que, les tâches du stable S_1.

Montrons qu'il existe une solution optimale où chaque tâche T_i du stable S_1 est ordonnancée exactement à l'instant $sS1_i$ donné par l'algorithme DPMCMA. Supposons que, dans une solution optimale, il existe une tâche T_i du stable S_1 ordonnancée après l'instant $sS1_i$ (la tâche T_i est ordonnancée seule ou avec, au moins, une tâche du stable S_2). 3 cas peuvent apparaître :

– aucun batch n'est ordonnancé à cet instant $sS1_i$. Dans ce cas, le batch contenant la tâche T_i est simplement translatée et ordonnancé à l'instant $sS1_i$.

– un batch contenant, seulement, des tâches du stable S_2 est ordonnancé à l'instant $sS1_i$. Dans ce cas, permuter les deux batchs.

– un batch contenant une tâche du stable S_1 et, éventuellement, des tâches du stable S_2 est ordonnancé à l'instant $sS1_i$. Dans ce cas, permuter les deux batchs.

En appliquant ce raisonnement au plus $|S_1|$ fois, on obtient une solution optimale où chaque tâche T_i du stable S_1 est ordonnancée avant ou à la même date associé $sS1_i$.

La tâche T_1 doit être ordonnancée à l'instant $sS1_1$, car elle est disponible à cet instant ($sS1_1 = r_1$). La tâche T_2 doit être ordonnancée à l'instant $sS1_2$, car $sS1_2 = max\{r_2, sS1_1 + 1\}$. En considérant les tâches $T_3, \ldots, t_{|S_1|}$, on obtient, récursivement et de la même manière, que les tâches $T_1, \ldots, T_{|S_1|}$ doivent être ordonnancées, respectivement, aux instants $sS1_1, \ldots, sS1_{|S_1|}$.

Pour ordonnancer les tâches du stable S_2, qui sont deux à deux compatibles disponibles à la même date 0, il suffit, de les affecter en utilisant la technique du couplage maximum définie dans l'algorithme DPMCMA.

Le nombre maximum de sommets dans le graphe H est de l'ordre de $|S_1| + |S_2|$ (on ne considère que $|S_1| + |S_2|$ sommets de X). Donc, la complexité du problème pour déterminer un couplage maximum $O((|S_1| + |S_2|)^{2.5})$. Ainsi, la complexité de cet algorithme est $O(n^{2.5}logn)$ □

Exemple 6.1 *Nous disposons de 6 tâches T_1, T_2, T_3, T_4, t_5 et T_6 à ordonnancer sur une machine à traitement par batch de capacité égale à 2. Les temps de traitement des tâches sont tous égaux à 1. Les dates de disponibilité des tâches sont données dans le tableau 6.2. Le graphe de compatibilité biparti est représenté par la figure 6.4.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6
r_i	1	2	0	0	0	0

TAB. 6.2: *Dates de disponibilité des tâches de l'exemple 6.1.*

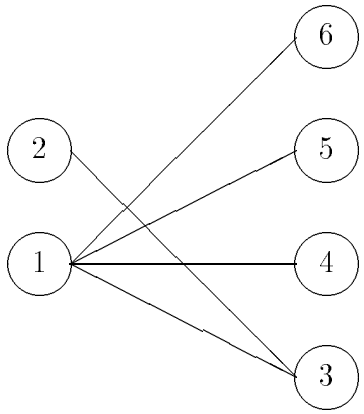


FIG. 6.4: *Graphe de compatibilité biparti $G = (S_1, S_2; E)$ de l'exemple 6.1.*

L'exécution de cet algorithme donne :

- $sS1_1 = 1$; $sS1_2 = 2$;
  la tâche T_1 est ordonnancée à l'instant 1 ;
  la tâche T_2 est ordonnancée à l'instant 2 ;

- $q = 5$ et $F = \{0, 1, 2, 3, 4, 5\}$;
- $x = 3$;
  Le graphe biparti correspondant H_3 est donné à la figure 6.5.
  la tâche T_3 est ordonnancée à l'instant 2 ;
  la tâche T_4 est ordonnancée à l'instant 0 ;
  la tâche T_5 est ordonnancée à l'instant 3 ;
  la tâche T_6 est ordonnancée à l'instant 1 ;
  $C_{max} = 4$;

L'ordonnancement est représenté par la figure 6.6.

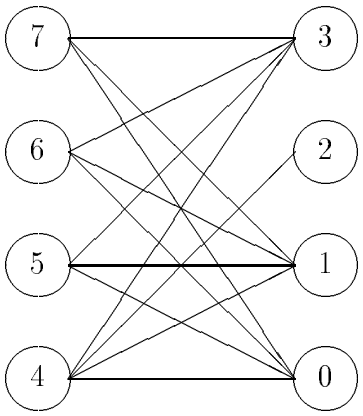


FIG. 6.5: *Graphe biparti H_3 de l'exemple 6.1.*

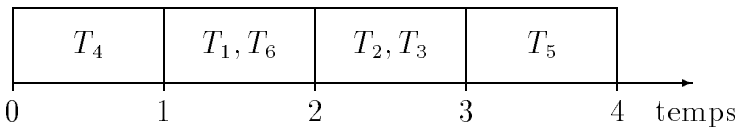


FIG. 6.6: *Ordonnancement de l'exemple 6.1.*

Notons que cet algorithme résout, aussi, les problèmes où les dates de disponibilité des tâches du stable S_2 ne sont pas toutes égales à 0.

Corollaire 6.4 *L'algorithme DPMCMA résout le problème $B1/G = (S_1, S_2; E), b = 2, r_i, p_i = 1/C_{max}$ où la condition "si les tâches T_i ($\in S_1$) et T_j ($\in S_2$) sont compatibles alors $r_i \geq r_j$" est satisfaite (la tâche T_j est disponible avant, ou à la même date que, la tâche T_i).*

Preuve. Remarquons que, si un batch B_k contient deux tâches T_i et T_j (bien sûr $T_i \in S_1$ et $T_j \in S_2$) alors la date de disponibilité de ce batch est égale à la date de

disponibilité de la tâche T_i c.-à-d. r_i. Ainsi, la permutation de deux batchs dans la preuve précédente est, aussi, possible. □

Aussi, cet algorithme résout le problème $B1/G = (S_1, S_2; E), b = 2, r_i, p_i = p/C_{max}$ avec les deux conditions : $r_i = \alpha_i p$ pour $i = 1, \ldots, n$ et "si les tâches T_i $(\in S_1)$ et T_j $(\in S_2)$ sont compatibles alors $r_i \geq r_j$". Il suffit, de résoudre le problème avec α_i comme dates de disponibilité et, dans la solution optimale, multiplier toutes dates de début d'exécution par p.

6.3 Heuristique

Nous présentons dans ce paragraphe, pour le problème $B1/G = (S1, S2; E), b = 2, r_i, p_i = p/C_{max}$, une algorithme approché polynomial garantissant un ration de performance dans le mauvais cas borné par une constante fixée.

Notons par $APP(I)$ la valeur de la solution obtenue par une heuristique pour une instance I de ce problème et soit $OPT(I)$ la valeur optimale. Une heuristique a une performance garantie ρ $(\rho \geq 1)$, si pour toute instance I de ce problème, nous avons $R(I) = \frac{APP(I)}{OPT(I)} \leq \rho$.

Comme heuristique pour ce problème, on peut penser à construire pour les deux stables S_1 et S_2 les ordonnancements optimaux pour le problème $1/r_i, p_i = p/C_{max}$, ensuite mélanger ces deux sous-ordonnancements optimaux pour avoir un ordonnancement réalisable avec une date de fin de traitement supérieure, ou égale, au maximum des dates de fin de traitement de ces deux sous-ordonnancements, qui est bien sûr une borne inférieure. Nous modifions l'algorithme $DPA2$ du paragraphe précédent pour obtenir l'heuristique DPH qui résout le problème en temps polynomial.

Considérons la procédure suivante qui résout le problème $1/r_i, p_i = p/C_{max}$:

Procédure date$(T_1, T_2, \ldots, T_k)$;
début
1- Ranger les tâches par ordre croissant de leurs dates de disponibilité ; (soit $T_1, \ldots, T_k$)
2- $sS1_1 := r_1$;
3- **Pour** i:=2 **à** k
faire $sS1_i := max\{r_i, sS1_{i-1} + p\}$
fait ;
fin ;

Notons par $sS1$ la date de fin de traitement de l'ordonnancement obtenu par la procédure date appliquée aux tâches du stable S_1 et notons par $sS2$ la date de fin

de traitement de l'ordonnancement obtenu par la procédure date appliquée aux tâches du stable S_2. Il est clair que $\max\{sS1, sS2\}$ est une borne inférieure.

Supposons sans perte de généralité que $sS1 \geq sS2$. L'heuristique DPH s'écrit alors comme suit :

Heuristique DPH ;
début
1- Considérons $sS1_i$ la date de la tâche T_i obtenue par la procédure date appliquée aux tâches du stable S_1 ;
2- Ordonnancer chaque tâche T_i du stable S_1 à l'instant $sS1_i$;
3- Ranger les tâches du stable S_2 par ordre croissant de leurs dates de disponibilité ; (soit $T_{|S_1|+1}, \ldots, T_{|S_1|+|S_2|}$)
4- **Pour** $j := |S_1| + 1$ **à** $|S_1| + |S_2|$
faire $sS2_j :=$ la plus petite date à laquelle on peut ordonnancer la tâche T_j (la date de disponibilité, la contrainte de compatibilité et la capacité du batch sont respectés)
fait ;
5- Ordonnancer chaque tâche T_j du stable S_2 à l'instant $sS2_j$;
fin.

Théorème 6.5 *L'heuristique DPH fournie un ordonnancement avec une performance garantie* $\rho = 1 + \frac{\min\{|S_1|,|S_2|\}}{\max\{|S_1|,|S_2|\}}$ *et cette borne est atteinte.*

Preuve. Soit S_{DPH} l'ordonnancement obtenu par l'heuristique DPH et notons par $C_{max}(S_{DPH})$ la date de fin de traitement de S_{DPH}.

Comme $sS1 \geq sS2$ nous avons : $C_{max}(S_{DPH}) = \max\{sS1_{|S_1|}, sS2_{|S_1|+|S_2|}\} + p = sS1 + xp + p$ où x est le nombre de tâches du stable S_2 ordonnancées seules, après la date $sS1$.

Soit S^* solution optimale du problème et $C_{max}(S^*)$ sa date de fin de traitement.

Alors, $C_{max}(S^*) \geq sS1 + p$.

Donc, nous avons $R(I) \leq 1 + \frac{xp}{sS1+p}$.

Si $|S_1| > |S_2|$, alors $x \leq |S_2|$ et $sS1 + p \geq |S_1|p$, donc $R(I) \leq 1 + \frac{|S_2|}{|S_1|}$.

Si $|S_1| \leq |S_2|$, alors $x \leq |S_1|$ et $sS1 + p \geq |S_2|p$, donc $R(I) \leq 1 + \frac{|S_1|}{|S_2|}$.

Pourquoi $x \leq |S_1|$? parce qu'en arrangeant, séparément, les tâches $T_1, \ldots, T_{|S_1|}$ du stable S_1 et les tâches $T_{|S_1|+1}, \ldots, T_{|S_1|+|S_2|}$ du stable S_2 avec la procédure date et en les décalant à droite jusqu'à la date $sS1 + p$, comme on le voit sur la figure 6.7 :

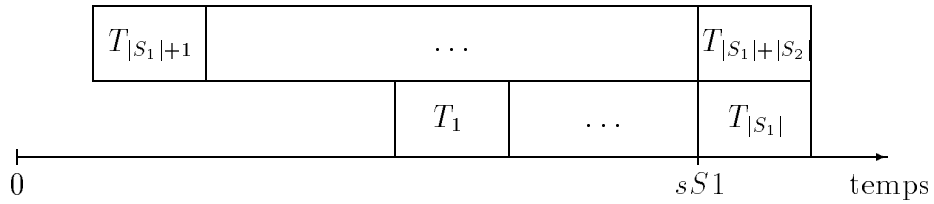


FIG. 6.7: *Décalage des tâches*

Il vient, que le nombre maximum de tâches du stable S_2 qu'on peut ordonnancer seules, après l'instant $sS1$, est $|S_1|$.

Nous présentons, maintenant, une instance pour laquelle cette borne est atteinte.

Nous avons 2n-1 tâches $T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2n-1}$. Chaque tâche T_i $(1 \leq i \leq n)$ appartient au stable S_1 et a une date de disponibilité nulle. Chaque tâche T_i $(n+1 \leq i \leq 2n-1)$ appartient au stable S_2 et a une date de disponibilité égale à $(i-n)p$. Le graphe biparti de compatibilité est construit comme suit : Chaque tâche T_i $(1 \leq i \leq n-1)$ du stable S_1 est compatible, seulement, avec la tâche T_{n+i} du stable S_2.

Le makespan $C_{max}(S_{DPH})$ de cet ordonnancement S_{DPH}, donné par l'heuristique DPH, est égal à $(2n-1)p$ (voir la figure 6.8). La valeur optimale du makespan pour l'instance I est donné par $C_{max}(S^*) = np$ (voir la figure 6.9).

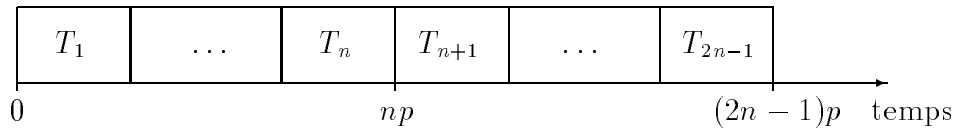


FIG. 6.8: *Ordonnancement S_{DPH}*

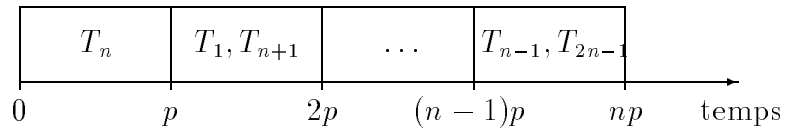


FIG. 6.9: *Ordonnancement S^**

Ainsi $\frac{C_{max}(S_{DPH})}{C_{max}(S^*)} = \frac{(2n-1)p}{np} = 1 + \frac{\min\{|S_1|,|S_2|\}}{\max\{|S_1|,|S_2|\}} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n}$ □

Corollaire 6.6 *L'heuristique DPH fournie un ordonnancement avec une performance garantie $\rho \leq 2$.*

Preuve. La preuve est immédiate, il suffit de considérer $|S_1| = |S_2|$. □

Remarquons que la solution optimale de l'exemple du plus mauvais cas précédent peut être obtenue en temps polynomial par l'algorithme DPMCMA du chapitre précédent.

Chapitre 7

Complémentaire d'un graphe biparti [1]

Nous considérons dans ce paragraphe, le graphe complémentaire d'un graphe biparti (c.-à-d. un graphe formé de deux cliques K_1 et K_2 noté $G = (K_1, K_2; E)$) dans le cas où la machine à traitement par batch aurait une capacité infinie. Nous montrons, dans le premier paragraphe, que le problème dynamique avec temps de traitement identiques (égaux à 1) pour les tâches est difficile. Dans le deuxième paragraphe, nous présentons un sous-problème polynomial. Pour établir les résultats de ce chapitre, nous nous sommes inspirés des idées développées dans la référence [46]

7.1 Problème difficile

Théorème 7.1 *Le problème* $B1/G = (K_1, K_2; E), b \geq n, r_i, p_i = 1/C_{max}$ *est NP-difficile au sens fort.*

Preuve. Soit 1-PrExt le problème de décision suivant : Etant donnés un graphe biparti $\overline{G} = (S_1, S_2; \overline{E})$ avec $|S_1 \cup S_2| \geq 3$ et trois sommets v_1, v_2, v_3. Existe-t-il une 3-coloration (C_1, C_2, C_3) des sommets de $\overline{G}$ de sorte que chaque sommet v_i a la couleur C_i pour $i = 1, 2, 3$? Le problème 1-PrExt est NP-complet [14] pour $v_1, v_2, v_3 \in S_1$.

Etant donné une instance arbitraire du problème 1-PrExt, nous construisons une instance du problème de décision BSCG (associé au problème $B1/G = (K_1, K_2; E), b \geq n, r_i, p_i = 1/C_{max}$) comme suit :

- Nous considérons le graphe complémentaire $G = (K_1, K_2; E)$ de $\overline{G} = (S_1, S_2; \overline{E})$ (voir les figures 7.1 et 7.2) auquel on ajoute 3 sommets u_1, u_2, u_3 à la clique K_2

1. Les résultats de ce chapitre sont publiés dans [29]

et les arêtes $(v_1, u_1), (v_2, u_2), (v_3, u_3)$ et (x, u_i) pour $x \in K_1 \setminus \{v_1, v_2, v_3\}$ (chaque sommet u_i est connecté à chaque sommet de $K_1 \cup K_2 \setminus \{v_1, v_2, v_3\}$).

- $r_{u_i} = r_{v_i} = i - 1$ pour $i = 1, 2, 3$ et $r_i = 0$ pour les autres tâches.
- Les temps de traitement des tâches sont tous identiques à 1.
- La machine à traitement par batch a une capacité infinie.

Existe-t-il un ordonnancement avec une date de fin de traitement inférieure ou égale à 3 ?

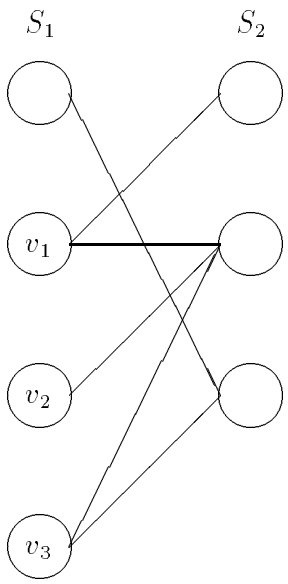


FIG. 7.1: *Une instance du problème 1-PrExt.*

Il est clair que BSCG est dans NP et que cette construction est polynomiale. Nous allons montrer que 1-PrExt a une solution si et seulement si BSCG a une solution.

Si le problème 1-PrExt a une solution, alors le graphe $\overline{G}$ admet une 3-coloration. Nous construisons une solution pour BSCG comme suit : Nous ordonnançons toutes les tâches du graphe G correspondant aux sommets du graphe $\overline{G}$ ayant la même couleur C_i (elles sont deux à deux compatibles) et la tâche u_i (qui est compatible avec toutes les tâches associées à la clique $K_1 \setminus \{v_j, j \neq i\}$) dans un même batch B_i à la date $i - 1$. La date de fin de traitement de cet ordonnancement est égale à 3.

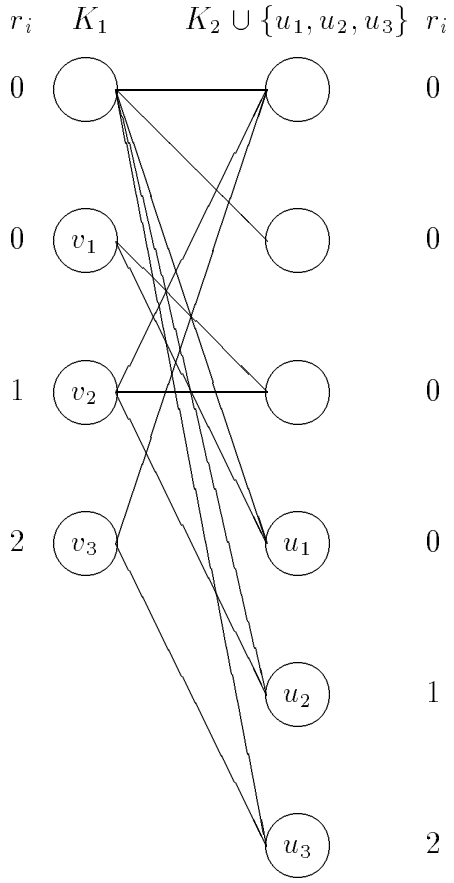


FIG. 7.2: *L'instance associée au problème 1-PrExt de la figure 5.20.*

Inversement, si le problème BSCG a une solution, alors il existe 3 batchs B_1, B_2, B_3 tels que :

- les tâches associées aux sommets u_3 et v_3 sont traitées dans le batch B_3 à l'instant 2,
- les tâches associées aux sommets u_2 et v_2 sont traitées dans le batch B_2 à l'instant 1, et
- les tâches associées aux sommets u_1 et v_1 sont traitées dans le batch B_1 à l'instant 0.

Ainsi, les sommets de $B_i \setminus \{u_i\}$ forment nécessairement une clique dans le graphe G et un stable dans le graphe $\overline{G}$. Par conséquent 1-PrExt a une solution. □

7.2 Problème polynomial

Nous considérons dans ce paragraphe le cas où il y a deux dates de disponibilité distinctes (0 et r) pour les tâches et les temps de traitement tous identiques à p.

Notons :

- $V(0)$ l'ensemble des tâches ayant une date de disponibilité nulle.
- $V(r)$ l'ensemble des tâches ayant une date de disponibilité égale à r.

Supposons que $|K_1 \cap V(r)| \geq |K_2 \cap V(r)|$ et considérons l'algorithme TA suivant :

Algorithme TA;
début si les tâches de $V(r)$ ne forment pas une clique
alors
- Les tâches de la clique K_1 sont traitées dans un même batch à l'instant r.
- Les tâches de la clique K_2 sont traitées dans un même batch à l'instant $r + p$.
- $C_{max} = r + 2p$.

sinon (les tâches de $V(r)$ forment une clique)

si $K_2 \cap V(r) = \emptyset$
alors
- Les tâches de la clique K_2 sont traitées dans un même batch à l'instant 0.
- Les tâches de la clique K_1 sont traitées dans un même batch à l'instant $\max\{p, r\}$.
- $C_{max} = \max\{p, r\} + p$.

sinon $(K_2 \cap V(r) \neq \emptyset)$

si les tâches de $V(0)$ forment une clique
alors
- Les tâches de la clique $V(0)$ sont traitées dans un même batch à l'instant 0.
- Les tâches de la clique $V(r)$ sont traitées dans un même batch à l'instant $\max\{p, r\}$.
- $C_{max} = \max\{p, r\} + p$.

sinon (les tâches de $V(0)$ ne forment pas une clique)

si $r \geq p$

alors
- Les tâches de la clique $K_2 \setminus V(r)$ sont traitées dans un même batch à l'instant 0.
- Les tâches de la clique $K_1 \setminus V(r)$ sont traitées dans un même batch à l'instant p.
- Les tâches de la clique $V(r)$ sont traitées dans un même batch à l'instant $\max\{2p, r\}$.
- $C_{max} = \max\{2p, r\} + p$.

sinon
- Les tâches de la clique K_1 sont traitées dans un même batch à l'instant r.
- Les tâches de la clique K_2 sont traitées dans un même batch à l'instant $r + p$.
- $C_{max} = r + 2p$.

fsi

fsi

fsi

fsi

fin.

Théorème 7.2 *Le problème* $B1/G = (K_1, K_2; E), b \geq n, r_i \in \{0, r\}, p_i = p/C_{max}$ *est résolu en* $O(n^2)$ *par l'algorithme TA.*

Preuve. Si les tâches de l'ensemble $V(r)$ ne forment pas une clique, alors il existe deux tâches incompatibles T_i et T_j ($T_i, T_j \in V(r)$) avec $r_i = r_j = r$. Donc, $C_{max} \geq r + 2p$. Dans ce cas, il suffit de traiter les tâches de la clique K_1 dans un même batch à l'instant r et les tâches de la clique K_2 dans un même batch à l'instant $r + p$ pour avoir un makespan égal à $r + 2p$.

Sinon (c'est-à-dire que les tâches de l'ensemble $V(r)$ forment une clique), si $K_2 \cap V(r) = \emptyset$, alors il existe deux tâches incompatibles T_i et T_j ($T_i \in K_1$ et $T_j \in K_2$) avec $r_i = r$ et $r_j = 0$. Donc, $C_{max} \geq \max\{p, r\} + p$. Dans ce cas, il suffit de traiter les tâches de la clique K_1 dans un même batch à l'instant 0 et les tâches de la clique K_2 dans un même batch à l'instant $\max\{p, r\}$ pour avoir un makespan égal à $\max\{p, r\} + p$.

Sinon (c'est-à-dire que les tâches de l'ensemble $V(r)$ forment une clique et $K_2 \cap V(r) \neq \emptyset$), si les tâches de $V(0)$ forment une clique, alors il existe deux tâches incompatibles T_i et T_j ($T_i \in V(0)$ et $T_j \in V(r)$) avec $r_i = 0$ et $r_j = r$. Donc, $C_{max} \geq \max\{p, r\} + p$. Dans ce cas, il suffit de traiter les tâches de la clique $V(0)$ dans un même batch à l'instant 0 et les tâches de la clique $V(r)$ dans un même batch à l'instant $\max\{p, r\}$ pour avoir un makespan égal à $\max\{p, r\} + p$.

Sinon (c'est à dire que les tâches de l'ensemble $V(r)$ forment une clique, $K_2 \cap V(r) \neq \emptyset$ et les tâches de $V(0)$ ne forment pas une clique) il existe deux tâches incompatibles T_i et T_j ($T_i, T_j \in V(0)$) avec $r_i = r_j = 0$. Nous considérons ici deux sous-cas :

Si $r \geq p$ alors $C_{max} \geq \max\{2p, r\} + p$. Dans ce cas, il suffit de traiter les tâches de la clique $K_2 \setminus V(r)$ dans un même batch à l'instant 0, les tâches de la clique $K_1 \setminus V(r)$ dans un même batch à l'instant p et les tâches de la clique $V(r)$ dans un même batch à l'instant $\max\{2p, r\}$ pour avoir un makespan égal à $\max\{2p, r\} + p$.

Sinon ($r < p$) alors $r + 2p < \max\{2p, r\} + p$ et il faut donc traiter les tâches de la clique K_1 à l'instant r et les tâches de la clique K_2 à l'instant $r + p$ pour avoir un makespan égal à $r + 2p$.

Vérifier que les tâches de $V(0)$ ou $V(r)$ forment une clique a une complexité $O(|K_1||K_2|)$. Par conséquent, l'algorithme s'exécute en $O(n^2)$. □

Exemple 7.1 *Considérons l'ordonnancement de 6 tâches $T_1, \ldots, T_6$ sur une machine à traitement par batch avec une capacité infinie. Les temps de traitement sont tous égaux à 2 et les dates de disponibilité sont données dans le tableau 7.1. Le graphe de compatibilité est représenté par la figure 7.3.*

T_i	T_1	T_2	T_3	T_4	T_5	T_6
r_i	3	3	0	0	3	0

TAB. 7.1: *Dates de disponibilité des tâches de l'exemple 7.1.*

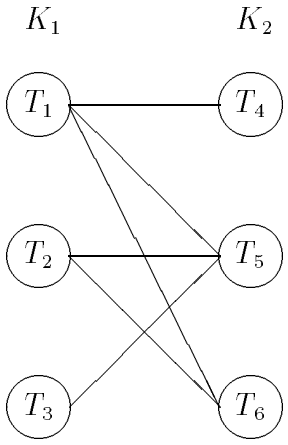


FIG. 7.3: *Graphe de compatibilité de l'exemple 7.1.*

L'exécution de l'algorithme donne :

- $V(0) = \{T_3, T_4, T_6\}$;
- $V(3) = \{T_1, T_2, T_5\}$;
- les tâches de $V(3)$ forment une clique ;
- $K_2 \cap V(3) = \{T_5\} \neq \emptyset$;
- les tâches de $V(0)$ ne forment pas une clique et $r \geq p$, donc :

$$\begin{array}{llllll} B_1 = K_2 \setminus V(3) = \{T_4, T_6\} & ; & sb_1 = 0 & ; & pb_1 = 2 & ; \\ B_2 = K_1 \setminus V(3) = \{T_3\} & ; & sb_2 = 2 & ; & pb_2 = 2 & ; \\ B_3 = V(3) = \{T_1, T_2, T_5\} & ; & sb_3 = 4 & ; & pb_3 = 2 & ; \end{array}$$

- $C_{max} = 6$;

L'ordonnancement optimal est représenté par la figure 7.4.

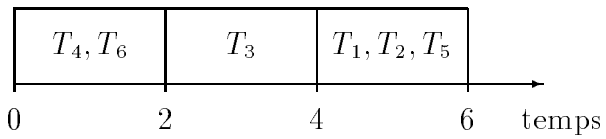


FIG. 7.4: *L'ordonnancement optimal de l'exemple 7.1.*

Exemple 7.2 *Considérons les données de l'exemple précédent avec* $r = 1$ *(les tâches* T_1*,* T_2 *et* T_5 *sont disponibles à l'instant 1 et les autres à l'instant 0).*

Comme pour l'exemple précédent, l'exécution de l'algorithme donne :

- $V(0) = \{T_3, T_4, T_6\}$;
- $V(1) = \{T_1, T_2, T_5\}$;
- les tâches de $V(1)$ forment une clique ;
- $K_2 \cap V(1) = \{T_5\} \neq \emptyset$;

– les tâches de $V(0)$ ne forment pas une clique et $r < p$, donc :

$$B_1 = K_1 = \{T_1, T_2, T_3\} \quad ; \quad sb_1 = 1 \quad ; \quad pb_1 = 2 \quad ;$$
$$B_2 = K_2 = \{T_4, T_5, T_6\} \quad ; \quad sb_2 = 3 \quad ; \quad pb_2 = 2 \quad ;$$

– $C_{max} = 5$;

L'ordonnancement optimal est représenté par la figure 7.5.

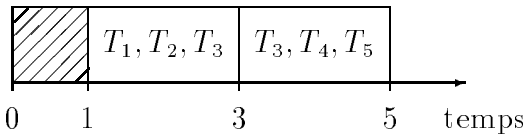


FIG. 7.5: *L'ordonnancement optimal de l'exemple 7.2.*

Chapitre 8

Graphe complet et quelques graphes spéciaux

Nous proposons au début de ce chapitre des algorithmes polynomiaux exacts pour résoudre le problème dans le cas d'un graphe complet (c'est-à-dire que toutes les tâches sont deux à deux compatibles). L'un aborde le cas où la capacité de la machine à traitement par batch est infinie (elle peut traiter un nombre illimité de tâches à la fois), et les autres, le cas où la capacité de la machine à traitement par batch est finie (elle peut traiter un nombre limité b de tâches à la fois) avec certaines conditions sur les tâches.

A la fin de ce chapitre nous étudions quelques types de graphes spéciaux que nous construisons aux paragraphes 3 et 4. Pour chacun de ces graphes spéciaux nous présentons des algorithmes de résolution polynomiaux exacts en se basant sur les algorithmes des deux premiers paragraphes.

8.1 Graphe complet et capacité infinie[1]

Nous proposons dans ce paragraphe, un algorithme polynomial exact, basé sur le rangement des tâches et la programmation dynamique, pour résoudre le problème lorsque toutes les tâches sont deux à deux compatibles et la capacité de la machine à traitement par batch est infinie, c'est-à-dire qu'elle peut traiter un nombre illimité de tâche à la fois.

Soit l'algorithme DPA3 suivant :

1. Les résultats de ce paragraphe sont publiés dans [28]

Algorithme DPA3 ;
début
1- Ranger les tâches suivant l'ordre décroissant de leurs temps de traitement ;
2- $Z_0 := -\infty$;
3- **Pour** $j := 1$ **à** n **faire**
$Z_j := \min\limits_{0 \leq k < j} \{\max\{Z_k, \max\limits_{k+1 \leq i \leq j}\{r_i\}\} + p_{k+1}\}$
fait ;
4- Déterminer les batchs par retour arrière ;
5- La date de fin de traitement est égale à Z_n ;
fin.

Théorème 8.1 *L'algorithme DPA3 résout le problème* $B1/b \geq n, r_i/C_{max}$ *en* $O(n^2)$.

Preuve. Ici toutes les tâches sont compatibles entre elles. La preuve est faite en trois étapes.

- Supposons qu'il existe une solution $B_1, \ldots, B_i, \ldots, B_j, \ldots, B_m$ avec $i < j$, $T_k \in B_i$, $T_{k'} \in B_j$ et $p_k < p_{k'}$ (c.-à-d. que l'ensemble des tâches n'est pas ordonné par ordre décroissant des temps de traitement des tâches), et notons par y la durée de cet ordonnancement. Formons, à partir de cette solution, une nouvelle solution $B_1, \ldots, B'_i = B_i \setminus \{T_k\}, \ldots, B'_j = B_j \cup \{T_k\}, \ldots, B_m$, et notons par y' la durée de cet ordonnancement. Nous avons $pb'_i \leq pb_i$ et $pb'_j = pb_j$ (car $p_k < p'_k$), donc $Cb'_i \leq Cb_i$ et $Cb'_j \leq Cb_j$; ce qui signifie que y' est inférieure ou égale à y. D'où, il existe une solution optimale où l'ensemble des tâches est ordonné par ordre décroissant des temps de traitement des tâches.

- Soit Z_j la valeur minimale de la durée de l'ordonnancement des tâches $T_1, \ldots, T_j$. Si $\{T_{k+1}, \ldots, T_j\}$ forme un batch à la fin de l'ordonnancement des tâches $T_1, \ldots, T_j$ alors la durée de l'ordonnancement des tâches $T_1, \ldots, T_j$ est égale au maximum entre la durée de l'ordonnancement des taches $T_1, \ldots, T_k$ augmentée de p_{k+1} et $\max\limits_{k+1 \leq i \leq j}\{r_i\} + p_{k+1}$, donc la durée de l'ordonnancement des tâches $T_1, \ldots, T_j$ est égale au maximum entre la durée de l'ordonnancement des tâches $T_1, \ldots, T_k$ et $\max\limits_{k+1 \leq i \leq j}\{r_i\}$ augmentée de p_{k+1}. D'où la relation de récurrence :

$$\begin{cases} Z_0 := -\infty \\ Z_j := \min\limits_{0 \leq k < j}\{\max\{Z_k, \max\limits_{k+1 \leq i \leq j}\{r_i\}\} + p_{k+1}\} \qquad pour \;\; j = 1, \ldots, n \end{cases}$$

  de laquelle est déduit l'algorithme.

- Comme le nombre d'itérations à l'étape j est égal à j, alors la complexité de l'algorithme est $O(n^2)$. □

Exemple 8.1 *Nous disposons de 4 tâches $T_1, \ldots, T_4$ dont les dates de disponibilité et les temps de traitement sont donnés dans le tableau 8.1.*

T_i	T_1	T_2	T_3	T_4
r_i	0	4	1	2
p_i	3	2	1	1

TAB. 8.1: *Temps de traitement de l'exemple 8.1.*

Le déroulement de l'algorithme est le suivant :

1- Les tâches sont déjà ordonnées suivant l'ordre décroissant de leurs temps de traitement ;

2- $Z_0 = -\infty$;

3- $Z_1 = \min\{\max\{-\infty, \max\{\underline{0}_{(*)}\}\} + 3\} = 3$;
$Z_2 = \min\{\max\{-\infty, \max\{0,4\}\} + 3, \max\{3, \max\{\underline{4}\}\} + 2\} = 6$;
$Z_3 = \min\{\max\{-\infty, \max\{0,4,1\}\} + 3, \max\{3, \max\{\underline{4},1\}\} + 2,$
$\max\{6, \max\{1\}\} + 1\} = 6$;
$Z_4 = \min\{\max\{-\infty, \max\{0,4,1,2\}\} + 3, \max\{3, \max\{\underline{4}_{(**)},1,2\}\} + 2,$
$\max\{6, \max\{1,2\}\} + 1, \max\{6, \max\{2\}\} + 1\} = 6$;

4- Les batchs sont déterminés, par retour arrière, comme suit :

- le premier batch est composé des tâches T_2,T_3 et T_4 (voir (**)), et
- le second batch est composé de la tâche T_1 (voir(*))

On obtient donc :

$B_1 = \{T_1\}$; $rb_1 = 0$; $pb_1 = 3$;
$B_2 = \{T_2, T_3, T_4\}$; $rb_2 = 4$; $pb_2 = 2$;

5- La durée de l'ordonnancement est $Z_4 = 6$;

L'ordonnancement optimal est représenté par la figure 8.1.

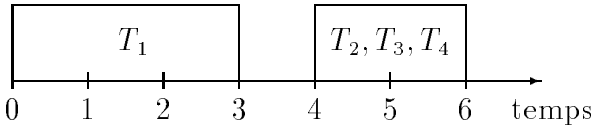


FIG. 8.1: *Ordonnancement optimal de l'exemple 8.1.*

8.2 Graphe complet et capacité finie[2]

Nous proposons, dans ce sous-paragraphe, un algorithme polynomial exact, basé sur la programmation dynamique, pour résoudre le problème lorsque toutes les tâches sont compatibles entre elles et la capacité de la machine à traitement par batch finie avec une certaine condition sur les tâches. Notons que le problème $B1/b = 2, r_i/C_{max}$ est NP-difficile au sens fort [34] et qu'il a été étudié dans [88].

Lemme 8.2 *Il existe une solution optimale où les dates de disponibilité rb_i des batchs sont rangées dans l'ordre croissant.*

Preuve. Supposons que, dans une solution optimale, il existe deux batchs B_i et B_j où $rb_i < rb_j$ et B_i ordonnancé après B_j. Alors, en ordonnançant le batch B_i à la date sb_j, la nouvelle solution est aussi réalisable (les dates de disponibilité sont respectées) et reste optimale, car la valeur du C_{max} ne change pas (la date de fin de traitement du batch $B_k(j \leq k < i)$ est incrémentée au plus de pb_i et la date de fin de traitement du batch $B_k(k > i)$ ne change pas). En appliquant ce raisonnement, un nombre fini de fois, on obtient le résultat souhaité. □

Soit l'algorithme DPA4 suivant :

Algorithme DPA4 ;
début

1- Ranger les tâches par ordre croissant de leurs dates de disponibilité et par ordre décroissant de leurs temps de traitement en cas d'égalité des dates de disponibilité ;

2- $Z_0 := -\infty$;

3- **Pour** $j := 1$ **à** n
faire $Z_j := \min_{\max\{0,j-b\} \leq k < j} \{\max\{Z_k, r_j\} + p_{k+1}\}$
fait ;

4- Déterminer les batchs par retour arrière ;

5- La date de fin de traitement est égale à Z_n ;

fin.

Remarque 8.1 *La notation "$r_i \uparrow p_i \downarrow$" indique que : $\forall ij$ $(i \neq j)$ si $r_i < r_j$ alors $p_i \geq p_j$ (ou d'une manière équivalente : il existe une séquence de tâches où celles-ci sont rangées dans l'ordre croissant de leurs dates de disponibilité et dans l'ordre décroissant de leurs temps de traitement).*

2. Les résultats de ce paragraphe sont publiés dans [28] ou bien [29]

Théorème 8.3 *L'algorithme DPA4 résout le problème* $B1/b, r_i/C_{max}$ *avec* $r_i \uparrow p_i \downarrow$ *et en* $\max\{O(nlogn), O(nb)\}$.

Preuve. Ici toutes les tâches sont compatibles entre elles. La preuve est faite en trois étapes.

- Supposons qu'il existe une solution $B_1, \ldots, B_i, B_{i+1}, \ldots, B_m$ (sans perte de généralité, supposons que les tâches d'un quelconque batch sont rangées dans l'ordre croissant de leurs dates de disponibilité et dans l'ordre décroissant de leurs temps de traitement, et que les batchs sont rangés dans l'ordre croissant de leurs dates de disponibilité (lemme 8.2)) avec T_a la première tâche de B_i, T_b la dernière tâche de B_i, T_c la première tâche de B_{i+1}, T_d la dernière tâche de B_{i+1} et $r_b > r_c$ (c.-à-d. que les tâches ne sont pas rangées dans l'ordre croissant de leurs dates de disponibilité). Sachant que $pb_i = p_a$, $rb_i = r_b$, $pb_{i+1} = p_c$ et $rb_{i+1} = r_d$. Notons par y la longueur de cet ordonnancement.

  Rangeons les tâches des deux batchs B_i et B_{i+1} dans l'ordre croissant de leurs dates de disponibilité et dans l'ordre décroissant de leurs temps de traitement, et formons, à partir de cette solution, une nouvelle solution $B_1, \ldots, B'_i, B'_{i+1}, \ldots, B_m$ telle que dans B'_i nous mettons la plus longue première séquence de tâches ne contenant pas T_b et respectant la taille du batch, et dans B'_{i+1} les tâches restantes. Notons par y' la longueur de cette nouvelle solution. Trois cas peuvent se présenter :
  Soit $r_a < r_c$ ($p_a \geq p_c$), donc $rb'_i \leq rb_i$ (car $T_b \in B'_{i+1}$), $pb'_i = pb_i$ (car $T_a \in B'_i$ et $p_a \geq p_c$), $rb'_{i+1} = rb_{i+1}$ (car $T_d \in B'_{i+1}$) et $pb'_{i+1} \leq pb_{i+1}$ (car $T_c \in B'_i$ du fait que $r_b > r_c$). Donc $Cb'_{i+1} \leq Cb_{i+1}$.

  Soit $r_a > r_c$ ($p_c \geq p_a$), donc $rb'_i \leq rb_i$ (car $T_b \in B'_{i+1}$), $pb'_i = pb_{i+1}$ (car $T_c \in B'_i$ et $p_c \geq p_a$), $rb'_{i+1} = rb_{i+1}$ (car $T_d \in B'_{i+1}$) et $pb'_{i+1} \leq pb_i$ (T_a est soit dans B'_i, soit dans la première position de B'_{i+1} si toutes les tâches de B_{i+1} sont dans B'_i avec $|B'_i| = b$). Donc $Cb'_{i+1} \leq Cb_{i+1}$.

  Soit $r_a = r_c$, donc deux sous-cas peuvent se présenter :

  Soit $p_a \geq p_c$, en appliquant le même raisonnement que celui de $r_a < r_c$, on obtient $Cb'_{i+1} \leq Cb_{i+1}$.

  Soit $p_a < p_c$, en appliquant le même raisonnement que celui de $r_a > r_c$, on obtient $Cb'_{i+1} \leq Cb_{i+1}$.

  Par conséquent y' est inférieure ou égale à y. Donc il existe une solution optimale où les tâches sont rangées par ordre croissant de leurs dates de disponibilité et par ordre décroissant de leurs temps de traitement.

- Soit Z_j la valeur minimale de la longueur d'ordonnancement des tâches $T_1, \ldots, T_j$. Si $\{T_{k+1}, \ldots, T_j\}$ forment un batch à la fin de l'ordonnancement des tâches $T_1, \ldots, T_j$ alors la longueur de l'ordonnancement des tâches $T_1, \ldots, T_j$ est égale au maximum entre la longueur de l'ordonnancement des tâches $T_1, \ldots, T_k$ augmentée de p_{k+1} et $r_j + p_{k+1}$. Par conséquent, la longueur de l'ordonnancement des

tâches $T_1, \ldots, T_j$ est égale au maximum entre la longueur de l'ordonnancement des tâches $T_1, \ldots, T_k$ et r_j augmentée de p_{k+1}. D'où, la relation de récurrence :

$$\begin{cases} Z_0 := -\infty \\ Z_j := \min\limits_{\max\{0,j-b\} \leq k < j} \{\max\{Z_k, r_j\} + p_{k+1}\} \qquad pour\ j = 1, \ldots, n \end{cases}$$

de laquelle est déduit l'algorithme.

– Comme le nombre d'itérations à l'itération j est égal à b, alors la complexité de l'étape 3 de l'algorithme est en $O(nb)$. Donc, la complexité de l'algorithme est $\max\{O(nlogn), O(nb)\}$. □

Exemple 8.2 *Soient à ordonnancer 4 tâches T_1, T_2, T_3 et T_4 sur une machine à traitement par batch avec une capacité égale à 2. Les dates de disponibilité et les temps de traitement sont donnés dans le tableau 8.2.*

T_i	T_1	T_2	T_3	T_4
r_i	0	1	2	4
p_i	3	1	1	1

TAB. 8.2: *Temps de traitement des tâches de l'exemple 8.2.*

L'exécution de l'algorithme donne :

1- Les tâches sont déjà ordonnées suivant l'ordre croissant de leurs dates de disponibilité et suivant l'ordre décroissant de leurs temps de traitement ;

2- $Z_0 = -\infty$;

3- $Z_1 = \min\{\underbrace{\max\{-\infty, 0\} + 3}_{(*)}\} = 3$;
$Z_2 = \min\{\max\{-\infty, 1\} + 3, \max\{3, 1\} + 1\} = 4$;
$Z_3 = \min\{\underbrace{\max\{3, 2\} + 1}_{(**)}, \max\{4, 2) + 1\} = 4$;
$Z_4 = \min\{\max\{4, 4\} + 1, \underbrace{\max\{4, 4\} + 1}_{(***)}\} = 5$;

4- les batchs sont déterminés, par retour arrière, comme suit :

– le premier batch est composé de la tâche T_4 (voir $(***)$) ;

– le second batch est composé des tâches T_2 et T_3 (voir $(**)$) ;

– le troisième batch est composé de la tâche T_1 (voir (*)) ;

On obtient donc :

$$\begin{array}{lllll} B_1 = \{T_1\} & ; & rb_1 = 0 & ; & pb_1 = 3 \quad ; \\ B_2 = \{T_2, T_3\} & ; & rb_1 = 2 & ; & pb_1 = 1 \quad ; \\ B_3 = \{T_4\} & ; & rb_1 = 4 & ; & pb_1 = 1 \quad ; \end{array}$$

5- La durée de l'ordonnancement est $Z_4 = 5$;

L'ordonnancement optimal est représenté par la figure 8.2.

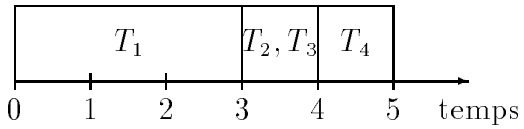


FIG. 8.2: *Ordonnancement optimal de l'exemple 8.2.*

Remarque 8.2 *La notation "$r_i \uparrow p_i \uparrow$" indique que : $\forall ij$ $(i \neq j)$ si $r_i < r_j$ alors $p_i \leq p_j$ (ou d'une manière équivalente : il existe une séquence de tâches où celles-ci sont rangées dans l'ordre croissant de leurs dates de disponibilité et dans l'ordre croissant de leurs temps de traitement).*

Comme les conditions "$r_i \uparrow p_i \uparrow$", "$r_i \uparrow p_i \downarrow$", "$r_i \downarrow p_i \uparrow$" et "$r_i \downarrow p_i \downarrow$" sont équivalentes (il suffit de remarquer par exemple que $p_i \leq p_j$ si et seulement si $M - p_i \geq M - p_j$ où M est un plus grand élément supérieur à $\max_{1 \leq i \leq n} \{p_i\}$), alors le problème avec l'une de ces conditions est aussi résolu en temps polynomial.

La complexité du problème a été, par la suite, améliorée. Nous donnons, ci-dessous, un Nouvel algorithme avec une meilleure complexité.

Notons :

– "$reste(a, b)$" le reste de la division de a par b.

– "$\lceil x \rceil$" le plus petit entier supérieur à x.

Soit LA1 l'algorithme suivant basé sur le rangement des tâches :

Algorithme LA1 ;
début

1- Ranger les tâches par ordre croissant de leurs dates de disponibilité et par ordre croissant de leurs temps de traitement en cas d'égalité des dates de disponibilité ;

2- $k := reste(n, b)$;
si $k = 0$
alors $k := b$
fsi ;

3- $B_1 := \{T_1, \ldots, T_k\}$;
$sb_1 := r_k$;
$pb_1 := p_k$;

4- **pour** $i := 2$ **à** $\lceil n/b \rceil$
faire - $B_i := \{T_{k+(i-2)b+1}, \ldots, T_{k+(i-1)b}\}$;
- $sb_i := \max\{r_{k+(i-1)b}, sb_{i-1} + pb_{i-1}\}$;
- $pb_i := p_{k+(i-1)b}$;
fait ;

5- La date de fin de traitement est égale à $sb_{\lceil n/b \rceil} + p_n$;

fin.

Théorème 8.4 *L'algorithme LA1 résout le problème* $B1/b, r_i/C_{max}$ *avec* $r_i \uparrow p_i \uparrow$ *en* $O(nlogn)$.

Preuve. Ici toutes les tâches sont compatibles entre elles. La preuve est faite en trois étapes.

– Supposons qu'il existe une solution $B_1, \ldots, B_i, B_{i+1}, \ldots, B_m$ (sans perte de généralité, nous supposons que les tâches de chaque batch sont rangées dans l'ordre croissant de leurs dates de disponibilité et dans l'ordre croissant de leurs temps de traitement, et les batchs sont rangés dans l'ordre croissant de leurs dates de disponibilité (lemme 8.2)) avec T_k la dernière tâche du batch B_i, $T_{k'}$ la première tâche du batch B_{i+1}, $r_k > r_{k'}$ (c.-à-d. que les tâches ne sont pas rangées dans l'ordre croissant de leurs dates de disponibilité) et $T_{k''}$ la dernière tâche du batch B_{i+1}. Notons par y la date de fin de traitement de cet ordonnancement.

Rangeons les tâches des deux batchs B_i et B_{i+1} par ordre croissant de leurs dates de disponibilité et par ordre croissant de leurs temps de traitement, et formons, à partir de cette solution, une nouvelle solution $B_1, \ldots, B'_i, B'_{i+1}, \ldots, B_m$ telle que B'_i contient la première plus longue séquence de tâches ne contenant pas les tâches T_k et $T_{k''}$ et qui respecte la taille du batch, et B'_{i+1} contient le reste des tâches. Notons par y' la date de fin de traitement de ordonnancement.

Si $r_k < r_{k''}$, nous avons $rb'_i \leq rb_i$ et $pb'_i \leq pb_i$ (car $T_k \in B'_{i+1}$), et $rb'_{i+1} = rb_{i+1}$ et $pb'_{i+1} = pb_{i+1}$ (car $T_{k''} \in B'_{i+1}$). Donc $Cb'_{i+1} \leq Cb_{i+1}$.

Si $r_k = r_{k''}$, nous avons $rb'_i \leq rb_i$ et $pb'_i \leq min\{pb_i, pb_{i+1}\}$, et $rb'_{i+1} \leq rb_i$ et $pb'_{i+1} \leq max\{pb_i, pb_{i+1}\}$ (car $T_k \in B'_{i+1}$ et $T_{k''} \in B'_{i+1}$). Donc $Cb'_{i+1} \leq Cb_{i+1}$.

Ainsi, y' est inférieure ou égale à y. Par conséquent, il existe une solution optimale où les tâches sont rangées par ordre croissant de leurs dates de disponibilité et par ordre croissant de leurs temps de traitement.

– Considérons une solution donnée par l'algorithme LA2 et soit m le nombre de batchs de cette solution.

Si $sb_m = r_n$ alors cette solution est optimale.

Si $sb_m > r_n$ alors il existe une séquence de batchs $B_q, \ldots, B_m$ ordonnancés sans interruption (c.-à-d. $Cb_j = sb_{j+1}$ pour $j = q, \ldots, m-1$) avec le batch B_q ordonnancé exactement à la date $r_{n-(m-q)b}$.

$B_q = \{T_{n-(m-q+1)b+1}, \ldots, T_{n-(m-q)b}\}$;
$rb_q = r_{n-(m-q)b}$;
$pb_q = p_{n-(m-q)b}$;

$B_{q+1} = \{T_{n-(m-q)b+1}, \ldots, T_{n-(m-q-1)b}\}$;
$rb_{q+1} = r_{n-(m-q-1)b}$;
$pb_{q+1} = p_{n-(m-q-1)b}$;

$\vdots$

$B_m = \{T_{n-b+1}, \ldots, T_n\}$;
$rb_m = r_n$;
$pb_m = p_n$;

Comme $sb_q = r_{n-(m-q)b}$ alors l'algorithme minimise la date de fin de traitement de la tâche $T_{n-(m-q)b}$. Aussi, il minimise la date de fin de traitement de la tâche $T_{n-(m-q-1)b}$ (car dans le batch B_q il y a exactement b tâches et le temps de traitement du batch B_q est égal à $p_{n-(m-q)b}$). En procédant récursivement, on montre que l'algorithme minimise la date de fin de traitement de la tâche T_n.

– Il est facile de voir que la complexité de l'algorithme est $O(nlogn)$. □

Soit LA2 l'algorithme suivant basé sur le rangement des tâches :

Algorithme LA2 ;
début
1- Ranger les tâches par ordre croissant de leurs temps de traitement ;
2- $k := reste(n, b)$;
si $k = 0$
alors $k := b$
fsi ;
3- $B_1 := \{T_1, \ldots, T_k\}$;
$pb_1 := p_k$;
4- **Pour** $i := 2$ **to** $\lceil n/b \rceil$
faire - $B_i := \{T_{k+(i-2)b+1}, \ldots, T_{k+(i-1)b}\}$;
- $pb_i := p_{k+(i-1)b}$;
fait
5- Ordonnancer les batchs dans un ordre arbitraire et sans interruption ;
6- La date de fin de traitement est $\sum_{i=1}^{\lceil n/b \rceil} pb_i$;
fin.

Corollaire 8.5 *L'algorithme LA2 résout le problème* $B1/b/C_{max}$ *et en* $O(nlogn)$.

Preuve. Il est facile de voir que ce problème est un cas particulier du problème défini dans le théorème 8.4 où les dates de disponibilité sont toutes nulles. Cet algorithme est déduit de l'algorithme LA1. □

Pour ce problème $B1/b/C_{max}$, un algorithme similaire, basé sur l'ordre décroissant des temps de traitement, est donné dans [88].

8.3 Premier type de graphes spéciaux[3]

Proposition 8.6 *Une solution optimale* $B_1, \ldots, B_m$ *pour un problème d'ordonnancement par batch P où toutes les tâches sont compatibles entre elles ; est aussi optimale pour le problème d'ordonnancement par batch P' identique à P contenant une graphe de compatibilité* $G = (V, E)$ *vérifiant : si* $(T_i, T_j) \in B_k$ *(sont ordonnancées dans un même batch) alors* $(i, j) \in E$.

Preuve. Supposons que cette solution n'est pas optimale pour le problème d'ordonnancement par batch P'. Par conséquent, il existe, pour le problème P', une solution $B'_1, \ldots, B'_{m'}$ avec une durée d'ordonnancement inférieure strictement à Cb_m. Comme chaque paire de tâches dans chaque batch B'_i $(1 \leq i \leq m')$ sont compatibles, alors cette

3. Les résultats de ce paragraphe sont publiés dans [28]

nouvelle solution est, aussi, réalisable pour le problème P et elle est meilleure. Ceci contredit le fait que la solution $B_1, \ldots, B_m$ est optimale. Donc la solution $B_1, \ldots, B_m$ est optimale pour le problème P'. $\square$

Notons par $GK(n,b)$ le graphe $G = (V, E)$ où $V = \{1, \ldots, n\}$ et $\forall i \in V$ le sous-graphe induit par le sous-ensemble de sommets $V' = \{i, \ldots, \min\{i + b - 1, n\}\}$ est une clique (voir les exemples de la figure 8.3).

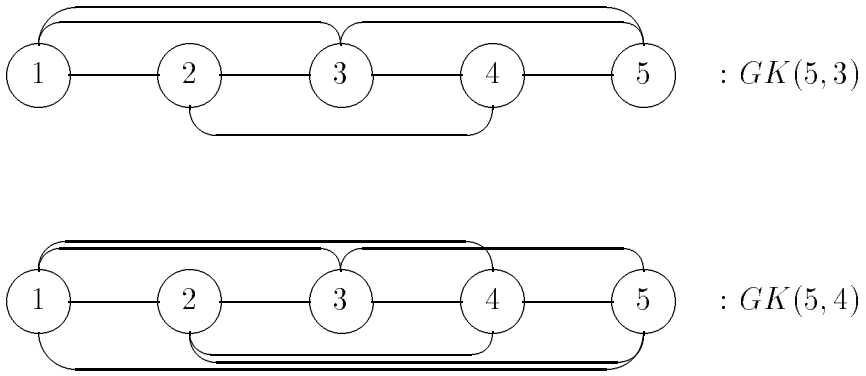


FIG. 8.3: *Exemples de graphes de type* $GK(n,b)$.

Soit l'algorithme suivant :

Algorithme DPA5 ;
début

1- Renuméroter les tâches de sorte que celles-ci soient rangées par ordre croissant de leurs dates de disponibilité et par ordre décroissant de leurs temps de traitement en cas d'égalité des dates de disponibilité et le graphe ainsi obtenu soit de type GK(n,b) (Soit $T_1, \ldots, T_n$ cette séquence) ;

2- $Z_0 := -\infty$;

3- **Pour** $j := 1$ **à** n
faire $Z_j := \min\limits_{\max\{0, j-b\} \leq k < j} \{\max\{Z_k, r_j\} + p_{k+1}\}$
fait ;

4- Déterminer les batchs par retour arrière ;

5- La date de fin de traitement est Z_n ;

fin.

Remarque 8.3 *La notation "*$GK(n,b) \wedge r_i \uparrow p_i \downarrow$*" indique qu'il existe une séquence ou une numérotation* $(T_1, \ldots, T_n)$ *des tâches telle que* $r_i \leq r_{i+1}$ *et* $p_i \geq p_{i+1}$ *pour* $i = 1, \ldots, n-1$ *et le graphe de compatibilité ainsi obtenu est de type* $GK(n,b)$.

Corollaire 8.7 *L'algorithme DPA5 résout le problème* $B1/GK(n,b), b, r_i/C_{max}$ *avec* $GK(n,b) \wedge r_i \uparrow p_i \downarrow$ *en* $\max\{O(nb), O(nlogn)\}$.

Preuve. Comme les tâches de la solution optimale du problème défini dans le théorème 8.3 sont rangées par ordre croissant de leurs dates de disponibilité et par ordre décroissant de leurs temps de traitement, la capacité de la machine à traitement par batch est égale à b et les tâches de n'importe quel batch sont toutes compatibles entre elles. Alors, par la propositions 8.6, la solution obtenue par l'algorithme DPA5 est optimale pour ce problème. Le nombre d'itérations à l'étape i est égal à b, donc l'algorithme est en $\max\{O(nb), O(nlogn)\}$. □

Exemple 8.3 *Considérons l'ordonnancement de 4 tâche* T_1, T_2, T_3 *et* T_4 *sur une machine à traitement par batch avec une capacité égale à 2. Les temps de traitement et les dates de disponibilité sont données dans le tableau 8.3. Le graphe de compatibilité est représenté par la figure 8.4.*

T_i	T_1	T_2	T_3	T_4
r_i	0	1	2	4
p_i	3	1	1	1

TAB. 8.3: *Les données de l'exemple 8.3.*

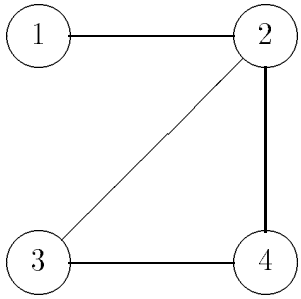


FIG. 8.4: *Le graphe de compatibilité de l'exemple 8.3.*

L'exécution de l'algorithme donne :

1- Les tâches sont déjà rangées par ordre croissant de leurs dates de disponibilité et par ordre décroissant de leurs temps de traitement et le graphe est de type GK(4,2) ;

2- $Z_0 = -\infty$;

3- $Z_1 = \min\{\underline{\max\{-\infty, 0\} + 3}_{(*)}\}\} = 3$;
$Z_2 = \min\{\max\{-\infty, 1\} + 3, \max\{3, 1\} + 1\}\} = 4$;
$Z_3 = \min\{\underline{\max\{3, 2\} + 1}_{(**)}, \max\{4, 2) + 1\} = 4$;
$Z_4 = \min\{\max\{4, 4\} + 1, \underline{\max\{4, 4\} + 1}_{(***)}\} = 5$;

4- Les batchs sont déterminés, par retour arrière, comme suit :

- le premier batch est composé de la tâche T_4 (voir $(***)$) ;
- le second batch est composé des tâches T_2 et T_3 (voir $(**)$) ;
- le troisième batch est composé de la tâche T_1 (voir $(*)$) ;

On obtient donc :
$B_1 = \{T_1\}$,
$B_2 = \{T_2, T_3\}$ et
$B_3 = \{T_4\}$;

5- La date de fin de traitement de l'ordonnancement est $Z_4 = 5$;

L'ordonnancement optimal est représenté par la figure 8.5.

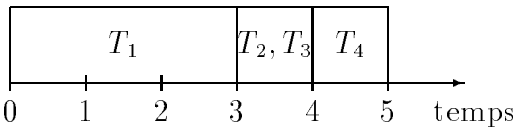


FIG. 8.5: *Ordonnancement optimal de l'exemple 8.3.*

8.4 Second type de graphes spéciaux[4]

Notons par $GM(n, b)$ (voir les exemples de la figure 8.6) le graphe $G = (V, E)$ où :

- $V = \{1, \ldots, n\}$,
- le sous-graphe induit par le sous-ensemble de sommets $\{1, \ldots, k\}$ ($k = reste(n, b)$) est une clique et
- $\forall i \in \{k + 1, k + 1 + b, \ldots, k + 1 + (reste(n, b) - 1)b\}$ le sous-graphe induit par les sommets $V' = \{i, i + 1, \ldots, i + b - 1\}$ est une clique.

4. Les résultats de ce paragraphe sont publiés dans [30]

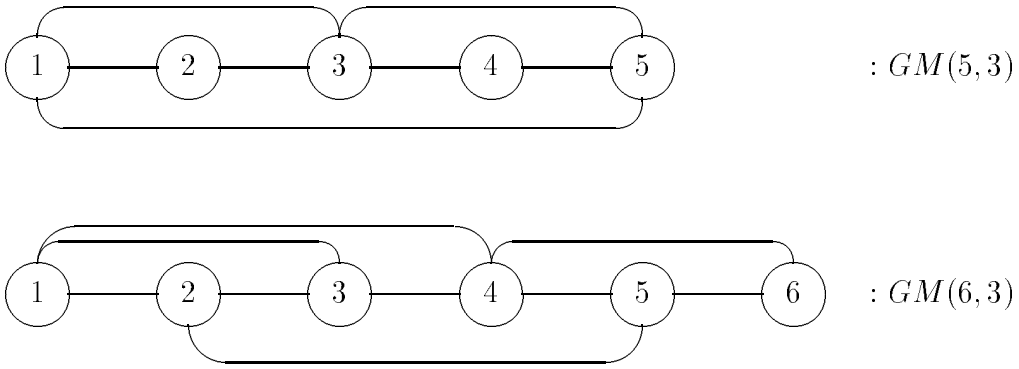


FIG. 8.6: *Exemples de graphes de type* $GM(n,b)$.

Soit l'algorithme suivant :

Algorithme LA4 ;
début

1- Renuméroter les tâches de sorte que celles-ci soient rangées par ordre croissant de leurs dates de disponibilité et par ordre croissant de leurs temps de traitement en cas d'égalité des dates de disponibilité et le graphe ainsi obtenu soit de type GM(n,b) (Soit $T_1, \ldots, T_n$ cette séquence) ;

2- $k := reste(n,b)$;
si $k = 0$
alors $k := b$
fsi ;

3- $B_1 := \{T_1, \ldots, T_k\}$;
$sb_1 := r_k$;
$pb_1 := p_k$;

4- **Pour** $i := 2$ **à** $\lceil n/b \rceil$
faire - $B_i := \{T_{k+(i-2)b+1}, \ldots, T_{k+(i-1)b}\}$;
- $sb_i := \max\{r_{k+(i-1)b}, sb_{i-1} + pb_{i-1}\}$;
- $pb_i := p_{k+(i-1)b}$;
fait ;

5- La date de fin de traitement est $sb_{\lceil n/b \rceil} + p_n$;

fin.

Remarque 8.4 *La notation "*$GM(n,b) \wedge r_i \uparrow p_i \uparrow$*" indique qu'il existe une séquence ou une numérotation* $(T_1, \ldots, T_n)$ *des tâches telle que* $r_i \leq r_{i+1}$ *et* $p_i \leq p_{i+1}$ *pour* $i = 1, \ldots, n-1$ *et le graphe de compatibilité ainsi obtenu est de type* $GM(n,b)$.

Corollaire 8.8 *L'algorithme LA4 résout le problème* $B1/GM(n,b), b, r_i/C_{max}$ *avec* $GM(n,b) \wedge r_i \uparrow p_i \uparrow$ *en* $O(nlogn)$.

Preuve. Comme les tâches de la solution optimale du problème définie dans le théorème 8.4 sont rangées par ordre croissant de leur dates de disponibilité et par ordre croissant de leurs temps de traitement, la capacité de la machine à traitement par batch est égale à b et les tâches de chaque batch sont deux à deux compatibles. Par la proposition 8.6, la solution obtenue par l'algorithme LA4 est optimale pour ce problème. Le rangement des tâches s'exécute en $O(nlogn)$. □

Exemple 8.4 *Nous disposons de 5 tâches T_1, T_2, T_3, T_4 et T_5 à ordonnancer sur une machine à traitement par batch avec une capacité égale à 2. Les temps de traitement et les dates de disponibilité des tâches sont données dans le tableau 8.4. Le graphe de compatibilité est représenté par la figure 8.7.*

T_i	T_1	T_2	T_3	T_4	T_5
r_i	0	3	4	5	6
p_i	2	2	3	3	4

TAB. 8.4: *Les donnés de l'exemple 8.4.*

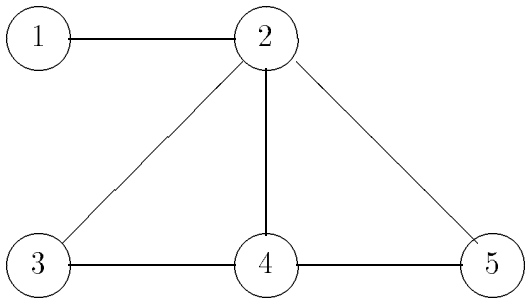


FIG. 8.7: *Le graphe de compatibilité de l'exemple 8.4.*

L'exécution de l'algorithme donne :

1- Les tâches sont déjà rangées par ordre croissant de leurs dates de disponibilité et par ordre croissant de leurs temps de traitement et le graphe est de type GM(5,2) ;

2- $k = 1$;

3- $B_1 = \{T_1\}$; $sb_1 = 0$; $pb_1 = 2$;

4- $B_2 = \{T_2, T_3\}$; $sb_2 = 4$; $pb_2 = 3$;
$B_3 = \{T_4, T_5\}$; $sb_3 = 7$; $pb_3 = 4$;

5- La date de fin de traitement est égale à 11 ;

L'ordonnancement optimal est représenté par la figure 8.8.

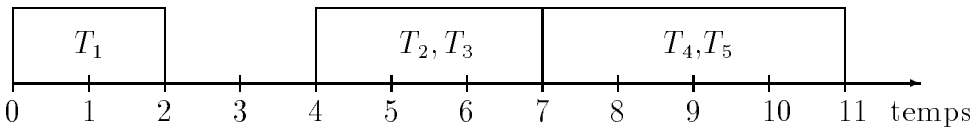


FIG. 8.8: *Ordonnancement optimal de l'exemple 8.4.*

Soit l'algorithme suivant :

Algorithme LA5 ;
Début

1- Renuméroter les tâches de sorte que celles-ci soient rangées par ordre croissant de leurs temps de traitement et le graphe ainsi obtenu soit de type GM(n,b) (Soit $T_1, \ldots, T_n$ cette séquence) ;

2- $k := reste(n, b)$;
Si $k = 0$
alors $k := b$
fsi ;

3- $B_1 := \{T_1, \ldots, T_k\}$;
$pb_1 := p_k$;

4- **Pour** $i := 2$ **à** $\lceil n/b \rceil$
faire - $B_i := \{T_{k+(i-2)b+1}, \ldots, T_{k+(i-1)b}\}$;
- $pb_i := p_{k+(i-1)b}$;
fait ;

5- Ordonnancer les batchs dans un ordre arbitraire ;

6- La date de fin de traitement est égale à $sb_{\lceil n/b \rceil} + p_n$;

fin.

Remarque 8.5 *La notation "$GM(n,b) \wedge p_i \uparrow$" indique qu'il existe une séquence ou une numérotation $(T_1, \ldots, T_n)$ des tâches telle que $p_i \leq p_{i+1}$ pour $i = 1, \ldots, n-1$ et le graphe de compatibilité ainsi obtenu est de type $GM(n,b)$.*

Corollaire 8.9 *L'algorithme LA5 résout le problème $B1/GM(n,b), b/C_{max}$ avec la condition $GM(n,b) \wedge p_i \uparrow$ en $O(nlogn)$.*

Preuve. Comme les tâches de la solution optimale du problème $B1/b/C_{max}$ sont rangées par ordre croissant de leurs dates de disponibilité, la capacité de la machine à traitement par batch est égale à b et les tâches de chaque batch sont deux à deux compatibles. Par la proposition 8.6, la solution obtenue par l'algorithme LA5 est optimale pour ce problème. Le rangement des tâches s'exécute en $O(nlogn)$. □

Exemple 8.5 *Nous avons 5 tâches T_1, T_2, T_3, T_4 et T_5 à ordonnancer sur une machine à traitement par batch avec une capacité égale à 2. Les temps de traitement des tâches sont données dans le tableau 8.5. Le graphe compatibilité est représenté par la figure 8.9.*

T_i	T_1	T_2	T_3	T_4	T_5
p_i	2	2	3	3	4

TAB. 8.5: *Temps de traitement des tâches de l'exemple 8.5.*

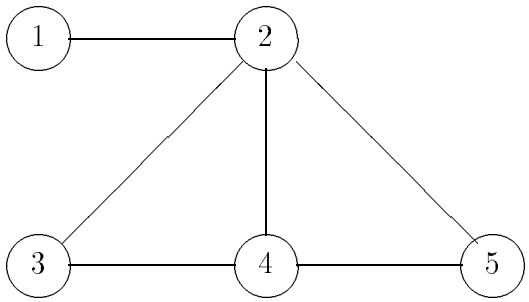


FIG. 8.9: *Le graphe de compatibilité de l'exemple 8.5.*

L'exécution de l'algorithme donne :

1- Les tâches sont déjà rangées par ordre croissant de leurs temps de traitement et le graphe est de type GM(5,2) ;

2- $k = 1$;

3- $B_1 = \{T_1\}$; $pb_1 = 2$;

4- $B_2 = \{T_2, T_3\}$; $pb_2 = 3$;
$B_3 = \{T_4, T_5\}$; $pb_3 = 4$;

5- la date de fin de traitement de l'ordonnancement est égale à 9 ;

L'ordonnancement optimal est représenté par la figure 8.10.

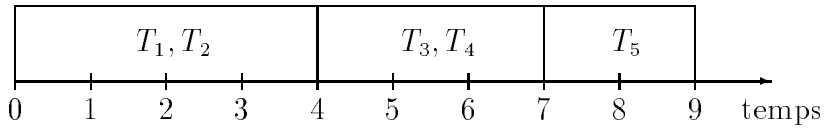


FIG. 8.10: *Ordonnancement optimal de l'exemple 8.5.*

Chapitre 9

Résolution et expérimentations numériques

Nous avons montré dans les chapitres précédents que le problème général et nombreuses sous-problèmes particuliers sont NP-difficiles dont certains au sens fort, l'existence d'algorithmes polynomiaux semble, donc, peu réaliste et ceci nous pousse à considérer d'autres méthodes de résolution.

Pour résoudre ces problèmes, nous présentons, dans ce chapitre, des méthodes exactes (par séparation et évaluation) dont le temps de calcul est, évidemment, exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petites tailles. Pour les problèmes de grandes tailles, les méthodes exactes ne sont plus envisageables, de par leur temps de calcul. Il est donc nécessaire dans ce cas, d'utiliser des méthodes approchées (heuristiques) qui donnent des solutions, certes, pas toujours optimales, mais obtenues rapidement. Ces solutions peuvent ensuite servir de solutions initiales pour les méthodes par séparation et évaluation ou des méthodes amélioratrices. Notons que les méthodes approchées sont largement utilisées pour appréhender ce genre de problèmes. Les deux méthodes, exactes et heuristiques, sont basées sur le rangement des tâches.

Enfin, les méthodes proposées dans ce chapitre sont testées et comparées sur des instances générées aléatoirement. Les résultats numériques obtenus sont donnés sous forme de tableaux.

9.1 Problème $B1/G = (V, E), b \geq n, p_i = p/C_{max}$ [1]

Dans ce paragraphe, nous proposons des algorithmes exacts et des heuristiques pour résoudre ce problème.

Soient :

- C : la cardinalité d'un couplage maximum dans le graphe de compatibilité $G = (V, E)$.
- $\delta(G)$: le degré minimum du graphe G.
- $\Delta(G)$: le degré maximum du graphe G.

Proposition 9.1 $C_{max} \leq (n - max\{\delta(G), C\}).p$

Preuve. Ce problème est équivalent au problème de la partition minimum en cliques du graphe de compatibilité G. Or, un graphe G admet une partition en k cliques si et seulement si le graphe complémentaire de G admet une partition en k stables et un graphe G admet une partition en k stables si et seulement si il admet une k coloration. Donc, le nombre de batchs est inférieur ou égal à $\gamma(\overline{G})$ où $\gamma(\overline{G})$ est le nombre chromatique du graphe complémentaire de G. Or, $\gamma(\overline{G}) \leq \Delta(\overline{G}) + 1$ [8] et $\Delta(\overline{G}) + \delta(G) = n - 1$. Donc, le nombre de batchs est inférieur ou égale à $n - \delta(G)$. Comme le nombre de batchs obtenu par le couplage maximum est $C + (n - 2C) = n - C$, on obtient : $C_{max} \leq (n - max\{\delta(G), C\}).p$ □

Remarque 9.1 *L'inégalité précédente est serrée.*

Preuve. Considérons le graphe de comptabilité $G = (V, E)$ où $|V| = 2n$ et $E = \{(i, i + n)\ pour\ i = 1 \ldots n\}$. Il est clair que :

- La date de fin de traitement est égale à np et
- $(2n - max\{\delta(G), C\})p = (2n - max\{1, n\})p = np.$ □

1. Les résultats de ce paragraphe sont publiés dans [27]

9.1.1 Heuristiques

Considérons des heuristiques, basées sur le rangement des tâches.

Soit l'heuristique H11 suivante :

Algorithme H11 ;
début
1- Ranger les tâches (soit $T_1, \ldots, T_n$) ;
2- $h := 1$; $B_1 := \{T_1\}$;
3- **Pour** $i := 2$ **à** n **faire**
si il existe un batch (soit B_j) tel que toutes les tâches de ce batch soient compatibles avec la tâche T_i
alors $B_j := B_j \cup \{T_i\}$
sinon $h := h + 1$; $B_h := \{T_i\}$
finsi
fait
fin.

Définition 9.1 *On appelle nombre de compatibilité d'une tâche T_i, le nombre de tâches compatibles avec elle.*

L'heuristique ci-dessus a été testée avec plusieurs types de procédures de tri, citons :

1. Ordre croissant des nombres de compatibilité des tâches.

2. Ordre décroissant des nombres de compatibilité des tâches.

3. Ordre aléatoire.

4. T_1 est la tâche ayant le plus petit nombre de compatibilité.
   T_k $(k = 2, \ldots, n)$ est la tâche ayant le plus petit nombre de compatibilité dans l'ensemble des tâches $T \setminus \{T_1, \ldots, T_{k-1}\}$.

5. T_1 est la tâche ayant le plus grand nombre de compatibilité.
   T_k $(k = 2, \ldots, n)$ est la tâche ayant le plus grand nombre de compatibilité dans l'ensemble des tâches $T \setminus \{T_1, \ldots, T_{k-1}\}$.

6. T_1 est la tâche ayant le plus petit nombre de compatibilité.
   T_k $(k = 2, \ldots, n)$ est la tâche ayant le plus petit nombre de compatibilité dans l'ensemble des tâches $\{T_1, \ldots, T_{k-1}\}$.

7. T_1 est la tâche ayant le plus grand nombre de compatibilité.
   T_k $(k = 2, \ldots, n)$ est la tâche ayant le plus grand nombre de compatibilité dans l'ensemble des tâches $\{T_1, \ldots, T_{k-1}\}$.

Nous avons remarqué que le tri numéro 6 donne de très bons résultats.

Algorithme Tri6 ;
début
1- O_1 est la tâche qui a moins de tâches compatibles ;
2- $T := T \setminus \{O_1\}$;
3- $L := \{O_1\}$; (L est un ensemble ordonné, i est l'ordre de la tâche O_i)
4- **Pour** $j := 2$ **à** $n - 1$ **faire**

- Pour chaque tâche T_i de T, calculer le nombre de tâches de L compatibles avec T_i (soit Co_i) ;
- Soit O_j la tâche de T telle que Co_j est minimal ;
- $T := T \setminus \{O_j\}$;
- $L := L \cup \{O_j\}$;

fait ;
5- $L := L \cup T$;
fin.

Proposition 9.2 *L'algorithme Tri6 est en* $O(n^2)$.

Preuve. Pour déterminer O_1 , il faut n tests (en supposant que le nombre de compatibilité de chaque tâche est déjà calculé). Pour implémenter cet algorithme, prenons comme structure de données une table $T[i, j]$ où i est l'indice de la tâche T_i. $T[i, 1]$ est un compteur indiquant le nombre de tâches compatibles avec la tâche T_i et $T[i, 2]$ est un pointeur qui pointe sur une liste chaînée des tâches compatibles avec la tâche T_i. A chaque itération j de la boucle, il faut au maximum n opérations pour déterminer la tâche O_j et au maximum n opérations pour la mise à jour de la table T (mettre 0 à la case $T[1, j]$ et décrémenter de 1 toutes les cases $T[i, 2]$ des tâches de L compatibles avec une tâche T_i de T. Le nombre d'opérations est donc borné par $O(n^2)$. □

Proposition 9.3 *L'algorithme H11 est en* $O(n^2)$.

Preuve. L'algorithme tri6 est en $O(n^2)$. A chaque itération i de la boucle, le nombre de tâches déjà affectées étant $i-1$; donc, pour déterminer si la tâche T_i peut appartenir à un batch, il faut faire (dans le pire des cas) $i - 1$ tests. Ainsi, le nombre total de test est borne par $\sum_{i=2}^{n} i - 1 = \frac{n(n-1)}{2}$. □

Nous avons remarqué aussi que les résultats obtenus seront meilleurs si nous considérons le tri6 en première priorité et le tri1 en seconde priorité ; c.-à-d. en cas d'égalité dans le premier tri, on considérera le second.

Remarque 9.2 *Si on utilise le tri numéro 1, qui est en O(nlogn), l'heuristique H11 sera en* $O(n^2)$.

Cette heuristique a été améliorée en faisant des permutations aléatoires entre les tâches et pour chaque permutation, calculer une valeur approchée ; et après n permutations prendre le minimum.

Algorithme H12 ;
début
1- min:=∞ ;
2- **Pour** $i := 1$ **à** n **faire**

- générer aléatoirement deux nombres k et l $(1 \leq k, l \leq n)$;
- permuter les deux tâches T_k et T_l ;
- calculer une valeur approchée (val) par l'heuristique H11 ;
- **si** $val < min$
  alors min:=val ; sauvegarder la solution
  finsi

fait
fin.

Remarque 9.3 *L'heuristique H12 est en* $O(n^3)$.

Preuve. Evident, car l'heuristique H11 est utilisée n fois. □

9.1.2 Méthodes exactes

Soient les variables

$$x_{ir} = \begin{cases} 1 & si\ la\ tâche\ T_i\ appartient\ au\ batch\ B_r \\ 0 & sinon \end{cases}$$

pour $i = 1, ..., n$ et $r = 1, ..., k$.

et soient

$$a_{ij} = \begin{cases} 1 & si\ les\ tâches\ T_i\ et\ T_j\ sont\ compatibles \\ 0 & sinon \end{cases}$$

pour $i = 1, ..., n$ et $j = 1, ..., n$ avec $i \neq j$

Le problème se formule de la manière suivante :

$$min\ k\ p \tag{9.1}$$

$$x_{ir} + x_{jr} - a_{ij} \leq 1 \qquad i = 1..n,\ j = 1..n,\ i \neq j\ et\ r = 1..k \tag{9.2}$$

$$\sum_{r=1}^{k} x_{ir} = 1 \qquad i = 1..n \tag{9.3}$$

$$x_{ir} \in \{0, 1\} \qquad i = 1..n\ et\ r = 1..k \tag{9.4}$$

La première contrainte indique que deux tâches incompatibles ne peuvent pas appartenir au même batch. La seconde contrainte indique que chaque tâche doit appartenir à un et un seul batch. La fonction objectif montre qu'il suffit de minimiser le nombre de batchs (c.-à-d. k).

Une solution réalisable de ce programme est en fait une partition du graphe de compatibilité G en cliques, qui est aussi une partition d'un ensemble à n éléments. Ainsi, une partition d'un ensemble à n éléments, dont les éléments représentent les sommets du graphe de compatibilité G où chaque partie forme une clique, est une solution réalisable de ce problème.

Partant de cette idée, nous construisons un algorithme énumératif de type branch and bound dont le principe est le suivant : En démarrant avec une solution initiale de s batchs (soit obtenue par heuristique, soit estimée par la proposition 9.1) l'algorithme prend les tâches une à une (dans un certain ordre) et les affectent aux batchs en construisant toutes les solutions réalisables possibles, et de ce fait il parcourt toute l'arborescence des solutions réalisables. Si a un sommet donné de l'arborescence, toutes les tâches sont affectées (ou on est sur un nœud terminal) alors si le nombre de batchs de cette solution est inférieur strictement à s, la meilleure solution trouvée sera remplacée par cette nouvelle solution. Et si sur un nœud quelconque de l'arborescence, le nombre de batchs obtenu est supérieur ou égal à s alors ce nœud est stérilisé, car on ne peut obtenir une solution dont le nombre de batchs est inférieur strictement à s à partir de ce nœud. L'algorithme construit une arborescence en la parcourant en profondeur d'abord. Si le graphe est complet et si on ne considère pas la stérilisation alors les nœuds terminaux de l'arborescence sont les différentes partitions possibles d'un ensemble à n éléments. Le nombre de nœuds terminaux est donc très inférieur au nombre de partitions d'un ensemble à n éléments qui est égal au nombre de Bell B_n.

Algorithme OPT ;

Procédure initialisation ;
début
- Ranger les tâches (soit $T_1, \ldots, T_n$) ;
- s:=solution de l'heuristique H12 (ou la borne de la proposition 9.1) ;
- k:=0 ;
fin ;

Procédure optimale($B_1, \ldots, B_k, i$) ;
début si $i > n$ (toutes les tâches sont affectées)
alors si $k < s$
alors $s := k$ et sauvegarder la solution courante
finsi
sinon si $k < s$
alors - pour $j := 1$ **à** k **faire**
si la tâche T_i peut être traiter dans le batch B_j
alors - $B_j := B_j \cup \{T_i\}$;
- optimale($B_1, \ldots, B_k, i+1$) ;
- $B_j := B_j \setminus \{T_i\}$;
fsi
fait ;
- $B_{k+1} := T_i$; (créer un nouveau batch B_{k+1})
- optimale($B_1, \ldots, B_{k+1}, i+1$) ;
finsi
finsi
fin ;

début
- initialisation ;
- optimale(pas de batchs(k=0),1) ;
fin.

Nous avons remarqué que si nous considérons l'algorithme tri6 pour ranger les tâches, nous obtenons de meilleurs temps de réponses, cela vient du fait que la stérilisation de sommets n'est pas très profonde ; on a pu résoudre des problèmes à 100 tâches en quelques minutes sur un terminal d'une station de travail UNIX.

Dans le tableau 9.2 sont indiqués les différents tests réalisés sur les instances du tableau 9.1. On remarquera que l'heuristique H12 améliore la solution donnée par l'heuristique H11 dans presque 47% des cas où la solution n'est pas optimale. Pour les problèmes de petite taille les solutions données par l'heuristique H12 sont presque toujours optimales. Pour les problèmes de grande taille les solutions données par l'heuristique H12 ne sont pas très éloignées de la solution optimale.

Instance	Nombre de tâches	Nombre d'arêtes	Densité (%)
1	5	5	50,00
2	10	30	66,67
3	15	68	64,76
4	20	134	70,53
5	25	214	71,33
6	30	288	66,21
7	35	363	61,01
8	40	544	69,74
9	45	598	60,40
10	50	717	58,53
11	55	917	61,75
12	60	1166	65,88
13	65	1247	59,95
14	70	1328	54,99
15	75	1632	58,81
16	80	2103	66,55
17	85	2085	58,40
18	90	2493	62,25
19	95	2711	60,72
20	100	2863	57,84

TAB. 9.1: *Les 20 différentes instances des différents tests. Les graphes sont générés aléatoirement, créant les arêtes uniformément entre les paires de tâches. Les temps de traitement sont tous égaux à 1 ou variables. Dans ce dernier cas, les différentes valeurs utilisées sont générées aléatoirement et uniformément entre 1 et 10.*

Instance	H11+tri6	H12	Solution optimale	Nombre de nœuds	Durée (sec.)
1	2	2^{a}	2	3	<1
2	3	3^{a}	3	4	<1
3	4	4^{a}	4	5	<1
4	4	4^{a}	4	13	<1
5	5	5^{a}	5	154	<1
6	7	$6^{a,b}$	6	240	<1
7	8	8	7	132	<1
8	7	7	6	126	<1
9	9	$8^{a,b}$	8	11	<1
10	10	10	9	1356	<1
11	11	10^{b}	9	1494	<1
12	11	$9^{a,b}$	9	2422	<1
13	12	12	11	489	<1
14	13	13	11	135577	5
15	14	13^{b}	12	309861	14
16	12	12	10	66086	2
17	15	14^{b}	12	686311	31
18	13	13	12	3217	<1
19	16	15^{b}	13	303441	15
20	16	16	15	4582470	273

TAB. 9.2: *Tests du problème* $B1/G = (V, E), b \geq n, p_i = 1/C_{max}$.

[a] : La solution obtenue est optimale.

[b] : La solution a été améliorée.

9.2 Problème $B1/G = (V, E), b, p_i = p/C_{max}$ [2]

Proposition 9.4 *Nous avons* $\left\lceil \frac{n}{b} \right\rceil p \leq C_{max} \leq (n - C)p$

Preuve. Le nombre de batchs obtenu par le couplage maximum est $C + (n - 2C) = n - C$, donc $C_{max} \leq (n - C)p$. Comme le nombre de batchs est supérieur ou égal à $\left\lceil \frac{n}{b} \right\rceil$ alors $C_{max} \geq \left\lceil \frac{n}{b} \right\rceil p$. □

Remarque 9.4 *L'inégalité précédente est serrée.*

Preuve. Considérons le problème d'ordonnancement suivant :

- Le nombre de tâches est égale à $2n$.
- Le graphe de comptabilité $G = (V, E)$ est tel que $E = \{(i, i + n)$ *pour* $i = 1, \ldots, n\}$.
- $b = n$.

Il est clair que :

- La date de fin de traitement est égale à np,
- $(2n - C)p = (2n - n)p = np$ et
- $\left\lceil \frac{2n}{n} \right\rceil p = np$. □

Les algorithmes utilisés sont les mêmes que ceux du paragraphe précédent, auxquels il faut ajouter un test pour que la taille d'un batch ne dépasse pas b (Appelons H21 et H22 ces deux heuristiques).

Le tableau 9.3 résume les résultats des différents tests réalisés sur les instances du tableau 9.1. On pourra remarquer que l'heuristique H22 améliore la solution donnée par l'heuristique H21 dans presque 50% des cas où la solution n'est pas optimale. Pour les problèmes de petite taille les solutions données par l'heuristique H22 sont presque toujours optimales. Pour les problèmes de grande taille les solutions données par l'heuristique H22 ne sont pas très éloignées de la solution optimale.

Aussi nous avons remarqué que si nous considérons l'algorithme tri6 pour ranger les tâches, nous obtenons de meilleurs temps de réponses, cela vient du fait que la stérilisation des sommets n'est pas très profonde ; on a pu résoudre des problèmes à 100 tâches en quelques minutes sur un terminal d'une station de travail UNIX.

2. Les résultats de ce paragraphe sont publiés dans [27]

Instance	b	H21+tri6	H22	Solution optimale	Nombre de nœuds	Durée (sec.)
1	2	3	3^{a}	3	8	<1
2	3	4	4^{a}	4	23	<1
3	3	5	5^{a}	5	56	<1
4	4	5	5^{a}	5	116	<1
5	4	7	7^{a}	7	53956	1
6	4	8	8^{a}	8	114086	2
7	4	10	$9^{a,b}$	9	163311	4
8	6	8	$7^{a,b}$	7	9795	<1
9	6	9	$8^{a,b}$	8	11	<1
10	6	10	10	9	44970	1
11	7	11	10^{b}	9	128064	4
12	7	11	$9^{a,b}$	9	2408	<1
13	7	12	12	11	481	<1
14	8	13	13	12	13074271	601
15	8	14	14	12	149697	7
16	8	12	12	11	11944883	481
17	9	15	14^{b}	12	4462441	238
18	9	13	13	12	9224	<1
19	9	16	15^{b}	13	293800	16
20	10	16	16	15	3998189	260

TAB. 9.3: *Tests du problème* $B1/G = (V, E), b, p_i = p/C_{max}$.

^{a} : La solution obtenue est optimale.

^{b} : La solution a été améliorée.

9.3 Problème $B1/G = (V, E), b \geq n/C_{max}$ [3]

Proposition 9.5 $C_{max} \leq \sum_{i=1}^{n-max\{\delta(G),C\}} p_i$ *avec* $p_1 \geq p_2 \geq \ldots \geq p_n$.

Preuve. Comme la date de fin de traitement de l'ordonnancement optimal est inférieure ou égale à la date de fin de traitement de toute solution réalisable, elle est donc inférieure ou égale à la solution obtenue par la partition minimum en cliques des sommets (tâches) du graphe $G = (V, E)$ qui, elle-même, est inférieure ou égale à $\sum_{i=1}^{k} p_i$ (en supposant que le graphe $G = (V, E)$ admet une partition en k cliques). Or, d'après la proposition 9.1, $k \leq n - max\{\delta(G), C\}$. □

Remarque 9.5 *L'inégalité précédente est serrée.*

Preuve. Considérons le problème d'ordonnancement défini comme suit :

- Le nombre de tâches est égal à $2n$.
- Le graphe de comptabilité $G = (V, E)$ est tel que $E = \{(i, i + n)$ *pour* $i = 1, \ldots, n\}$.
- $p_i = 1$ pour $i = 1, \ldots, 2n$ (tous les temps de traitement des tâches sont identiques et égaux à 1).

Il est clair que :

- La date de fin de traitement de l'ensemble des tâches est égale à n (c.-à-d. $C_{max} = n$), et
- $\sum_{i=1}^{2n-max\{\delta(G),C\}} p_i = \sum_{i=1}^{n} 1 = n$ □

Les algorithmes (heuristiques et exacts) utilisés pour le problème de ce paragraphe sont les mêmes que ceux utilisés pour le problème défini au paragraphe 9.1, sauf qu'il faut remplacer, dans la procédure optimale, le test $k < s$ par $d < s$ où d est la durée d'une solution partielle et s la durée de la meilleure solution trouvée ; et calculer avant le premier appel de la procédure optimale, dans la boucle **pour**, la nouvelle valeur de d comme suit :

3. Les résultats de ce paragraphe sont publiés dans [27]

si $p_i > pb_j$
alors - $d := d + p_i - pb_j$;
- $pb_j := p_i$;
finsi

où pb_j est le temps de traitement du batch B_j, c.-à-d. le plus long temps de traitement des tâches affectées à B_j.

et avant le second appel, ajouter :

$pb_{k+1} := p_i$; $d := d + p_i$.

Les nouveaux paramètres de la procédure optimale sont donc :

$B_1, \ldots, B_k$, $pb_1, \ldots, pb_k$, d et i.

Le premier appel de la procédure optimale, dans la procédure optimale, est :

Optimale$(B_1, \ldots, B_k, pb_1, \ldots, pb_k, d, i + 1)$

et le second appel est :

Optimale$(B_1, \ldots, B_{k+1}, pb_1, \ldots, pb_{k+1}, d, i + 1)$.

Après plusieurs tests, nous avons remarqué que le tri qui donne de meilleurs résultats pour les deux mèthodes proposées (l'heuristique et l'algorithme optimal) est : ranger les tâches par ordre décroissant de leurs temps de traitement et en cas d'égalité choisir la tâche qui a moins de tâches compatibles avec les tâches déjà rangées (Appelons H31 et H32 ces deux heuristiques).

Les résultats des différents tests réalisés sur les différentes instances du tableau 9.1 sont résumées et indiqués dans le tableau 9.4 ci-dessous. On remarquera que l'heuristique H32 améliore la solution donnée par l'heuristique H31 dans presque 61% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. Pour les problèmes de petite taille les solutions données par l'heuristique H32 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H32 n'est pas très éloignée de la solution obtenue par la procédure optimale. Au-delà de 50 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale ; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en l'arrêtant après une certaine durée d'exécution qui dépasse une heure dans tous les cas.

Instance	H31	H32	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	16	16[a]	16	3	<1
2	19	19[a]	19	52	<1
3	33	29[b]	26	75	<1
4	31	28[b]	26	148	<1
5	48	38[b]	37	8450	<1
6	46	46	43	33475	<1
7	61	61	53	1759563	51
8	46	44[b]	40	363333	11
9	60	59[b]	55	307963419	9686
10	74	74	66	109544350	3557
11	90	85[b]	83[c]	-	-
12	79	79	72[c]	-	-
13	90	90	90[c]	-	-
14	103	102[b]	96[c]	-	-
15	101	101	97[c]	-	-
16	97	94[b]	94[c]	-	-
17	121	117[b]	115[c]	-	-
18	115	114[b]	114[c]	-	-
19	107	105[b]	105[c]	-	-
20	139	139	139[c]	-	-

TAB. 9.4: *Tests du problème* $B1/G = (V, E), b \geq n/C_{max}$.

[a] : La solution obtenue est optimale.

[b] : La solution a été améliorée.

[c] : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

9.4 Problème $B1/G = (V, E), b/C_{max}$ [4]

Proposition 9.6 $\sum_{i=1}^{\lceil \frac{n}{b} \rceil} p_{1+(i-1)b} \leq C_{max} \leq \sum_{i=1}^{n-C} p_i$. *avec* $p_1 \geq p_2 \geq \ldots \geq p_n$.

Preuve. Comme la durée totale de l'ordonnancement optimale est inférieure ou égale à la durée totale de toute solution réalisable, elle est donc inférieure ou égale à la durée totale de la solution obtenue par le couplage de poids maximum qui, elle-même, est inférieure ou égale à $\sum_{i=1}^{n-C} p_i$. D'où, $C_{max} \leq \sum_{i=1}^{n-C} p_i$. Aussi, la durée totale de l'ordonnancement est supérieure ou égale à la durée totale de la solution du problème $B1/b/C_{max}$ qui est égale à $\sum_{i=1}^{\lceil \frac{n}{b} \rceil} p_{1+(i-1)b}$. □

Remarque 9.6 *L'inégalité précédente est serrée.*

Preuve. Considérons le problème d'ordonnancement suivant : le nombre de tâches est égale à $2n$, le graphe de comptabilité $G = (V, E)$ est tel que $E = \{(i, i+n)$ *pour* $i = 1, \ldots, n\}$, $p_i = 1$ pour $i = 1, \ldots, 2n$ et $b = 2$.

Il est clair que : $C_{max} = n$, $\sum_{i=1}^{2n-n} p_i = \sum_{i=1}^{n} 1 = n$ et $\sum_{i=1}^{\lceil \frac{2n}{n} \rceil} p_{1+(i-1)b} = n$ □

Les algorithmes (heuristiques et exacts) utilisés pour le problème de ce paragraphe sont les mêmes, que ceux utilisés pour le problème du paragraphe précédent, auxquels il faut ajouter un test pour que la taille d'un batch ne dépasse pas b (Appelons H41 et H42 ces deux heuristiques).

Dans le tableau 9.5 sont résumés les résultats des différents tests réalisés sur les différentes instances du tableau 9.1. On pourra remarquer que l'heuristique H42 améliore la solution donnée par l'heuristique H41 dans presque 72% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. Pour les problèmes de petite taille les solutions données par l'heuristique H42 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H42 n'est pas très éloignée de la solution obtenue par la procédure optimale. Au-delà de 50 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale ; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en arrêtant son exécution après une certaine durée.

4. Les résultats de ce paragraphe sont publiés dans [27]

Instance	b	H41	H42	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	2	20	20^{a}	20	8	<1
2	3	29	$27^{a,b}$	27	37	<1
3	3	37	$34^{a,b}$	34	198	<1
4	4	31	$30^{a,b}$	30	205	<1
5	4	46	$42^{a,b}$	42	1407	<1
6	4	58	57^{b}	56	79545	2
7	4	64	64^{a}	64	197053	7
8	6	46	45^{b}	44	337676	12
9	6	65	$61^{a,b}$	61	78749730	3011
10	6	74	74	73	68564099	2934
11	7	90	89^{b}	83^{c}	-	-
12	7	80	80	73^{c}	-	-
13	7	90	90	90^{c}	-	-
14	8	103	102^{b}	96^{c}	-	-
15	8	101	101	97^{c}	-	-
16	8	98	96^{b}	96^{c}	-	-
17	9	121	117^{b}	115^{c}	-	-
18	9	115	114^{b}	114^{c}	-	-
19	9	107	105^{b}	105^{c}	-	-
20	10	139	139	139^{c}	-	-

TAB. 9.5: *Test du problème* $B1/G = (V, E), b/C_{max}$.

[a] : La solution obtenue est optimale.

[b] : La solution a été améliorée.

[c] : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

9.5 Problème $B1/G = (V, E), b \geq n, r_i, p_i = p/C_{max}$

En utilisant le même raisonnement que celui du lemme 8.2, on montre qu'il existe une solution optimale où les dates de disponibilité rb_i des batchs sont rangées suivant l'ordre croissant. Il suffit donc de ranger les tâches par ordre croissant de leurs dates de disponibilité et en cas d'égalité, on prendra la tâche qui a moins de tâches compatibles avec les tâches déjà rangées.

Les algorithmes utilisés sont les mêmes que ceux du paragraphe 9.1, sauf qu'il faut remplacer, dans la procédure optimale, le test $k < s$ par $d < s$ où d est la durée d'une solution partielle et s la durée de la meilleure solution trouvée. Ensuite, calculer avant chaque appel de la procédure optimale la nouvelle valeur de d comme suit :

- $d := rb_1 + p$;
- **pour** $i := 2$ **à** k
  faire si $d < rb_i$
  alors $d := rb_i + p$
  sinon $d := d + p$
  finsi
  fait

(Appelons H51 et H52 ces deux heuristiques).

Les nouveaux paramètres de la procédure optimale sont donc :

$B_1, \ldots, B_k$, $rb_1, \ldots, rb_k$, d et i.

Le premier appel de la procédure optimale, dans la procédure optimale, est :

optimale$(B_1, \ldots, B_k, rb_1, \ldots, rb_k, d, i + 1)$

et le second appel est :

optimale$(B1, \ldots, B_{k+1}, rb_1, \ldots, rb_{k+1}, d, i + 1)$.

Dans le tableau 9.6, sont donnés les différents tests réalisés sur les instances du tableau 9.1. On remarquera que l'heuristique H52 améliore la solution donnée par l'heuristique H51 dans presque 47% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. pour les problèmes de petite taille, les solutions données par l'heuristique H52 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H52 n'est pas très éloignée de la

solution obtenue par la procédure optimale. Au-delà de 80 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en arrêtant celle-ci après une certaine durée d'exécution.

9.6 Problème $B1/G = (V, E), b, r_i, p_i = p/C_{max}$

Les algorithmes utilisés pour ce problème sont les mêmes que ceux présentés au paragraphe précédent 9.5, auxquels il faut ajouter, bien sûr, un test pour que la taille d'un batch ne dépasse pas la valeur b qui est la capacité de la machine à traitement par batch (Appelons H61 et H62 ces deux heuristiques).

Les différents tests réalisés sur les différentes instances du tableau 9.1 sont résumés dans le tableau 9.7 de la page suivante. On pourra remarquer que l'heuristique H62 améliore la solution obtenue par l'heuristique H61 dans presque 62% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. Pour les problèmes de petite taille les solutions obtenues par l'heuristique H62 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H62 n'est pas très éloignée de la solution obtenue par la procédure optimale. Au-delà de 80 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en l'arrêtant après une certaine durée d'exécution qui peut dépassée une heure.

Notons que dans le cas d'un graphe biparti, une heuristique avec un rapport de performance inférieur ou égal à $1 + \frac{min\{|S_1|,|S_2|\}}{max\{|S_1|,|S_2|\}}$ est donnée au paragraphe 6.3. De plus cette borne de performance est atteinte pour certains exemples et est dans le mauvais cas égale à 2. Cette heiristique peut se généraliser au cas où les temps de traitement des tâches ne sont pas tous identiques.

Remarquons que, pour un graphe biparti, la taille maximale d'un batch est égale à 2, c'est-à-dire que toutes les contraintes sur la capacité de la machine à traitement par batch se ramènent au cas $b = 2$. Cette heuristique peut donc remplacer les deux heuristiques H61 et H62. Seulement, aucun test de comparaison n'a été réalisé entre cette heuristique et les heuristiques H61 et H62, évidemment dans le cas d'un graphe biparti.

D'autres heuristiques ou algorithmes exacts peuvent aussi être proposés pour un graphe biparti ou pour d'autres types de graphes et feront certainement l'objet d'autres publications ou d'autres thèses. Les tests de comparaison seront dans ce cas plus que nécessaire.

Instance	H51	H52	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	22	22^a	22	2	<1
2	30	30^a	30	2	<1
3	23	23^a	23	2	<1
4	22	22^a	22	2	<1
5	21	21^a	21	2	<1
6	26	26	25	79	<1
7	42	42^a	42	2	<1
8	46	46	46	42	<1
9	46	45^b	45	3	<1
10	42	42	41	174	<1
11	46	46	45	904	<1
12	46	46	45	181	<1
13	56	56	48	1441436	78
14	57	54^b	51	14774	<1
15	49	49	48	28560857	1893
16	51	49^b	44	1037768	56
17	57	54^b	52^c	-	-
18	56	56	53^c	-	-
19	60	59^b	55^c	-	-
20	70	63^b	63^c	-	-

TAB. 9.6: *Test du problème* $B1/G = (V, E), b \geq n, r_i, p_i = p/C_{max}$.

^{a} : La solution obtenue est optimale.

^{b} : La solution a été améliorée.

^{c} : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

Instance	b	H61	H62	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	2	22	22[a]	22	2	<1
2	3	30	30[a]	30	2	<1
3	3	23	23[a]	23	2	<1
4	4	22	22[a]	22	2	<1
5	4	26	24[a,b]	24	348	<1
6	4	30	30	29	9042	<1
7	4	43	43[a]	43	380	<1
8	6	46	46[a]	46	42	<1
9	6	46	45[a,b]	45	3	<1
10	6	42	42	41	52243	3
11	7	49	49	45	21571	<1
12	7	46	46	45	176	<1
13	7	56	56	48	359437	22
14	8	57	54[b]	51	12687	<1
15	8	51	50[b]	48	1873960	161
16	8	52	49[b]	44	912744	56
17	9	57	54[b]	52[c]	-	-
18	9	56	56	53[c]	-	-
19	9	60	59[b]	54[c]	-	-
20	10	70	63[b]	63[c]	-	-

TAB. 9.7: *Test du problème* $B1/G = (V, E), b, r_i, p_i = p/C_{max}$.

[a] : La solution obtenue est optimale.

[b] : La solution a été améliorée.

[c] : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

9.7 Problème $B1/G = (V, E), b \geq n, r_i/C_{max}$

Comme il existe une solution optimale où les dates de disponibilité rb_i des batchs sont rangées suivant l'ordre croissant (lemme 8.2). Alors il suffit de ranger les tâches par ordre croissant de leurs dates de disponibilité et en cas d'égalité, on prendra la tâche qui a moins de tâches compatibles avec les tâches déjà rangées.

Les algorithmes utilisés sont les mêmes que ceux du paragraphe 9.3, sauf que la valeur de d se calcule, avant chaque appel de la procédure optimale, de la manière suivante :

- $d := rb_1 + pb_1$;
- **pour** $i := 2$ **à** k
  faire si $d < rb_i$
  alors $d := rb_i + pb_i$
  sinon $d := d + pb_i$
  finsi
  fait

(Appelons H71 et H72 ces deux heuristiques).

Les nouveaux paramètres de la procédure optimale sont donc :

$B_1, \ldots, B_k$, $rb_1, \ldots, rb_k$, $pb_1, \ldots, pb_k$, d et i.

Le premier appel de la procédure optimale, dans la procédure optimale, est :

optimale$(B_1, \ldots, B_k, rb_1, \ldots, rb_k, pb_1, \ldots, pb_k, d, i + 1)$

et le second appel est :

optimale$(B1, \ldots, B_{k+1}, rb_1, \ldots, rb_{k+1}, pb_1, \ldots, pb_{k+1}, d, i + 1)$.

Dans le tableau 9.8 sont indiqués les différents tests réalisés sur les instances du tableau 9.1. On remarquera que l'heuristique H72 améliore la solution donnée par l'heuristique H71 dans presque 45% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. Pour les problèmes de petite taille les solutions données par l'heuristique H72 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H72 n'est pas très éloignée de la solution obtenue par la procédure optimale. Au-delà de 45 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale ; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en arrêtant son exécution après une certaine durée.

Instance	H71	H72	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	32	32[a]	32	5	<1
2	35	35[a]	35	28	<1
3	40	40	37	2405	<1
4	37	37	33	285	<1
5	46	46	43	159507	5
6	54	54	50	3414104	129
7	81	81	67	926611	33
8	70	68[b]	66	4739859	157
9	82	82	72	5600226	260
10	96	92[b]	80[c]	-	-
11	107	107	99[c]	-	-
12	95	95	90[c]	-	-
13	128	128	128[c]	-	-
14	135	129[b]	124[c]	-	-
15	126	123[b]	121[c]	-	-
16	118	115[b]	114[c]	-	-
17	158	142[b]	142[c]	-	-
18	153	153	147[c]	-	-
19	152	136[b]	136[c]	-	-
20	170	162[b]	162[c]	-	-

TAB. 9.8: *Test du problème* $B1/G = (V, E), b \geq n, r_i/C_{max}$.

[a] : La solution obtenue est optimale.

[b] : La solution a été améliorée.

[c] : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

9.8 Problème $B1/G = (V, E), b, r_i/C_{max}$

Les algorithmes utilisés pour ce problème sont les mêmes que ceux du paragraphe précédent, auxquels il faut ajouter un test pour que la taille d'un batch ne dépasse pas b (Appelons H81 et H82 ces deux heuristiques).

Les différents tests réalisés sur les différentes instance du tableau 9.1 sont donnés dans le tableau 9.9. On pourra remarquer que l'heuristique H82 améliore la solution donnée par l'heuristique H81 dans presque 73% des cas où la solution n'est pas optimale ou bien la solution optimale n'est pas encore connue. Pour les problèmes de petite taille les solutions données par l'heuristique H82 sont presque toujours optimales. Pour les problèmes de grande taille la solution obtenue par l'heuristique H82 n'est pas très éloignée de la solution obtenue par la procédure optimale. Au-delà de 45 tâches, le temps d'exécution (dépassant une heure sur un terminal d'une station de travail UNIX) de la procédure optimale ne nous a pas permis de calculer la solution optimale ; cela ne nous a pas empêché d'utiliser cette procédure pour avoir une bonne solution approchée en arrêtant celle-ci après une certaine durée d'exécution.

Instance	b	H81	H82	Procédure optimale	Nombre de nœuds	Durée (sec.)
1	2	32	32^{a}	32	5	<1
2	3	40	40	39	41	<1
3	3	47	47	45	792	<1
4	4	43	$38^{a,b}$	38	1137	<1
5	4	62	$60^{a,b}$	60	303885	13
6	4	76	76	72	2246221	109
7	4	87	$81^{a,b}$	81	267252	14
8	6	70	68^{b}	67	1861624	77
9	6	91	91	79	21049790	1169
10	6	98	95^{b}	90^{c}	-	-
11	7	116	114^{b}	106^{c}	-	-
12	7	111	109^{b}	95^{c}	-	-
13	7	128	128	122^{c}	-	-
14	8	135	129^{b}	124^{c}	-	-
15	8	131	130^{b}	129^{c}	-	-
16	8	122	120^{b}	117^{c}	-	-
17	9	158	142^{b}	142^{c}	-	-
18	9	153	153	147^{c}	-	-
19	9	155	138^{b}	138^{c}	-	-
20	10	170	163^{b}	162^{c}	-	-

TAB. 9.9: *Test du problème* $B1/G = (V, E), b, r_i/C_{max}$.

^{a} : La solution obtenue est optimale.

^{b} : La solution a été améliorée.

^{c} : Cette solution a été obtenue par arrêt de la procédure après une heure d'exécution (elle n'est pas forcément optimale).

Conclusion

L'ordonnancement par batchs est une extension récente de l'ordonnancement classique où une machine ne peut traiter qu'une seule tâche à la fois. Il s'intéresse à l'ordonnancement d'un groupe (ou lot) de tâches qui doivent être traitées simultanément. Dans ce cas, les tâches d'un même batch doivent être compatibles. Cet aspect de compatibilité de tâches a été longuement ignoré, ce qui nous a amené à introduire cette notion formelle de "graphe de compatibilité" à l'ordonnancement par batchs. La compatibilité entre les tâches est donc représentée par un graphe (appelé graphe de compatibilité) où les sommets représentent les tâches et deux sommets sont reliés, par une arête, si elles (les tâches correspondantes) sont compatibles. Chaque batch contient uniquement les tâches deux à deux compatibles. Cette liaison entre la théorie de l'ordonnancement et la théorie des graphes apparaît comme un aspect nouveau très intéressant.

Nous avons considéré, dans cette thèse, le problème de l'ordonnancement d'un ensemble de tâches indépendantes sur une machine à traitement par batch en minimisant le temps de fin de traitement, de l'ensemble des tâches, en présence d'un graphe de compatibilité, de temps de traitement et de dates de disponibilité pour les tâches. Des applications concrètes, directes ou indirectes, de ce problème existent et sont actuellement à l'étude.

Le cas d'un graphe quelconque a été étudié et analysé. Nous avons montré que dans le cas où la capacité de la machine à traitement par batch serait égale à 2, le problème est polynomial sans la présence de dates de disponibilité. Ce problème devient difficile dès que la capacité devient supérieure ou égale à 3 (même avec la capacité illimitée).

Aussi, nous avons étudié et analysé plusieurs types de graphes comme les graphes scindés, les graphes bipartis, les complémentaires de graphes bipartis, les graphes complets, etc. Pour ces types particuliers de graphes, nous avons établi la difficulté du problème dans le cas où la capacité de la machine à traitement par batch serait égale à 2 dans le cas dynamique, à 3 dans le cas statique ou à l'infinie. Néanmoins, nous avons montrer qu'il est possible de trouver des algorithmes polynomiaux exacts pour résoudre quelques problèmes particuliers dont certains paraissent, a première vue, difficiles. Ainsi, nous avons présenté plusieurs algorithmes polynomiaux exacts pour résoudre de nombreux sous-problèmes polynomiaux.

Pour résoudre le problème dans le cas général (c'est-à-dire lorsque le graphe de compatibilité est quelconque), nous avons présenté des méthodes exactes (par séparation

et évaluation) dont le temps de calcul est, évidemment, exponentiel, ce qui explique qu'elles sont utilisables que sur des problèmes de petites tailles. Pour les problèmes de grandes tailles, nous avons proposé des méthodes approchées (heuristiques) qui donnent des solutions, certes, pas toujours optimales, mais obtenues rapidement. Les deux méthodes, exactes et heuristiques, sont basées sur le rangement des tâches. Ces différentes méthodes ont été testées et comparées sur des instances générées aléatoirement.

Dans les tableaux C.1 et C.2 des pages suivantes sont donnés les principaux types de problèmes étudiés et leurs complexités.

Etant donné la difficulté et la complexité du problème général, nous pensons que les algorithmes proposés (heuristiques et exacts) sont très efficaces. Les différents algorithmes ont été présentés d'une manière simple et facilement compréhensible afin qu'ils soient faciles à mettre en œuvre.

Les durées d'exécution des algorithmes exacts non polynomiaux, dans le cas général, ne sont pas absolues (temps CPU), car les programmes ont été exécutés sur un ordinateur multitâches où plusieurs processus s'exécutent simultanément. Nous avons remarqué que, dans la majorité des cas, lors de l'exécution de la procédure optimale, la solution de départ s'améliore au début et change que rarement par la suite. De ce fait, les solutions obtenues par arrêt de la procédure optimale après une certaine durée d'exécution, ont une grande chance d'être optimales, ce qui fournit une très bonne méthode approchée.

Beaucoup de problèmes ont été étudiés dans cette thèse, certains sont difficiles, d'autre sont faciles (au sens complexité) et beaucoup d'algorithmes ont été présentés, néanmoins il reste beaucoup de choses à faire : améliorer les algorithmes existants, étudier d'autres sous-problèmes, proposer d'autres approches de résolution, etc.

Les résultats obtenus ouvrent la voie à de nouvelles pistes de recherches très intéressantes :

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- Développer des métaheuristiques comme la recherche taboue, les algorithmes génétiques ou le recuit simulé.
4- Considérer, pour les tâches, des dates échues et utiliser des critères d'optimalité comme le retard maximum, somme des retards, nombre de tâches en retard,...
5- etc.

Problème	Algorithmes proposés	Paragraphe
$B1/G=(V,E), b=k, p_i=1/C_{max}$ $(k \geq 3)$		4.1.1.1
$B1/G=(V,E), b=2, r_i, p_i=1/C_{max}$		4.1.1.2
$B1/G=(V,E), b \geq n, p_i=1/C_{max}$		4.1.2
$B1/H_{n_1}+K_{n_2}, b, r_H=0, r_K=1, p_i=1/C_{max}$ avec $n_2 \geq \frac{n_1}{3}$ et $n_1 \geq 3$		4.1.3
$B1/G(K_b^1,\ldots,K_b^s), b, r_i, p_i=1/C_{max}$ (le graphe G admet une partition minimum en cliques)		4.1.4
$B1/G=(S,K;E), b=k/C_{max}$ $(k \geq 3)$		5.1.1
$B1/G=(S,K;E), b=k, r_i, p_i=1/C_{max}$ $(k \geq 2)$		5.1.2
$B1/G=(S_1,S_2;E), b=2, r_i, p_i=1/C_{max}$		6.1
$B1/G=(S_1,S_2;E), b=2, r_i, p_i=p/C_{max}$	H	6.3
$B1/G=(K_1,K_2;E), b \geq n, r_i, p_i=1/C_{max}$		7.1
$B1/G=(V,E), b \geq n, p_i=p/C_{max}$	B & B , H	9.1
$B1/G=(V,E), b, p_i=p/C_{max}$	B & B , H	9.2
$B1/G=(V,E), b \geq n/C_{max}$	B & B , H	9.3
$B1/G=(V,E), b/C_{max}$	B & B , H	9.4
$B1/G=(V,E), b \geq n, r_i, p_i=p/C_{max}$	B & B , H	9.5
$B1/G=(V,E), b, r_i, p_i=p/C_{max}$	B & B , H	9.6
$B1/G=(V,E), b \geq n, r_i/C_{max}$	B & B , H	9.7
$B1/G=(V,E), b, r_i/C_{max}$	B & B , H	9.8

TAB C.1: Résumé des principaux problèmes difficiles.

B&B : Séparation et évaluation (Branch and Bound).

H : Heuristique.

$\emptyset$: aucun algorithme n'a été proposé.

Problème	Complexité algorithmique	Paragraphe
$B1/G=(V,E), b=2/C_{max}$	$O(n^3)$	4.2
$B1/G=(V,E), b=2, p_i=1/C_{max}$	$O(n^{2.5})$	4.2
$B1/G=(V,E), b/C_{max}$ où G est sans K_3	$O(n^3)$	4.2
$B1/G=(V,E), b, p_i=1/C_{max}$ où G est sans K_3	$O(n^{2.5})$	4.2
$B1/G=(V,E), b \geq n, p_i=1/C_{max}$		
pour des graphes arc-circulaires	$O(n^3)$	3.4
pour des graphes triangulés	$O(n^2)$	3.4
pour des graphes de comparabilité	$O(n^3)$	3.4
$B1/G=(S,K;E), b, p_i=1/C_{max}$	$O(n^3)$	5.2.1.1
$B1/G=(S,K;E), b \geq n, p_i=1/C_{max}$	$O(n)$	5.2.1.2
$B1/G=(S,K;E), b \geq n/C_{max}$ avec $p_S=p$ et $\lvert\{T_j \in K \ / \ (i,j) \in E \ et \ p_j > p\}_{T_i \in S}\rvert \leq 2$	$O(n^2 logn)$	5.2.1.2
$B1/G=(S,K;E), b, r_S, r_K=0, p_i=1/C_{max}$	$O(n^3 log\frac{n}{b})$	5.2.2
$B1/G=(S,K;S\times K), b, r_i, p_i=1/C_{max}$	$O(nlogn)$	5.2.2
$B1/G=(S_1,S_2;E), b=2, r_{S_1}, r_{S_2}=0, p_i=1/C_{max}$	$O(n^{2.5} logn)$	6.2
$B1/G=(S_1,S_2;S_1\times S_2), b=2, r_i, p_i=1/C_{max}$	$O(nlogn)$	6.2
$B1/G=(K_1,K_2;E), b \geq n, r_i \in \{0,r\}, p_i=p/C_{max}$	$O(n^2)$	7.2
$B1/b \geq n, r_i/C_{max}$	$O(n^2)$	8.1
$B1/b, r_i/C_{max}$ avec $r_i \uparrow p_i \downarrow$	$\max\{O(nlogn), O(nb)\}$	8.2
$B1/b, r_i/C_{max}$ avec $r_i \uparrow p_i \uparrow$	$O(nlogn)$	8.2
$B1/b/C_{max}$	$O(nlogn)$	8.2
$B1/GK(n,b), b, r_i/C_{max}$ avec $GK(n,b) \wedge r_i \uparrow p_i \downarrow$	$\max\{O(nlogn), O(nb)\}$	8.3
$B1/GM(n,b), b/C_{max}$ avec $GM(n,b) \wedge p_i \uparrow$	$O(nlogn)$	8.4
$B1/GM(n,b), b, r_i/C_{max}$ avec $GM(n,b) \wedge r_i \uparrow p_i \uparrow$	$O(nlogn)$	8.4

TAB C.2: Résumé des principaux problèmes polynomiaux.

Bibliographie

[1] J. Acher and J. Gardelle. *Programmation linéaire.* Dunod, Paris, 1978.

[2] S. Achmanov. *Programmation linéaire.* Mir, Moscou, 1984.

[3] J.H. Ahmadi, R.H. Ahmadi, S. Dasu, and C.S. Tang. Batching and scheduling jobs on batch and discrete processors. *Operations Research*, 40:750–763, 1992.

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

[5] H. Belkouche, F. Louiz, and A. Mahdoum. Fewer colors: a new technique for solving the graphs coloring problem. *Working paper, laboratoire de Microélectronique, CDTA (Alger)*, 2000.

[6] R. Bellman. *Dynamic programming.* Princeton University Press, Princeton, USA, 1972.

[7] C. Berge. *Théorie des graphes et applications.* Dunod, Paris, 1967.

[8] C. Berge. *Graphes et hypergraphes.* Dunod, Paris, 1983.

[9] J. Blazewicz. Selected topics in scheduling theory. *Annals of discrete mathematics*, 31:1–60, 1987.

[10] J. Blazewicz, J. Barcelo, W. Kubiak, and H. Rőck. Scheduling tasks on two processors with deadlines and additional resources. *European Journal of Operational Research*, 26:364–370, 1986.

[11] J. Blazewicz, W. Cellary, R. Slowinsky, and J. Weglarz. Scheduling under resource constraints - deterministic models. *Annals of Operations Research*, 7, 1986.

[12] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes.* Springer-Verlag, Berlin, 1996.

[13] J. Blazewicz, J. Lazewicz, K.H. Ecker, G. Schmidt, and J. Weglarz. *Scheduling in computer and manufacturing systems.* Springer-Lehrbuch, Berlin, 1994.

[14] H.L. Bodlaender, K. Jansen, and G.J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55:219–232, 1994.

[15] M. Boudhar. Static scheduling on a single batch processing machine with split compatibility graphs. *Cahiers du Laboratoire Leibniz-IMAGrenoble*, 28, 2001.

[16] M. Boudhar. Scheduling on a single batch processing machine with bipartite compatibility graphs. *Cahiers du Laboratoire Leibniz-IMAGrenoble*, 45, 2002.

[17] M. Boudhar. Dynamic scheduling on a single batch processing machine with split compatibility graphs. *Journal of Mathematical Modelling and Algorithms*, 2:17–35, 2003.

[18] M. Boudhar. Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, 57:513–527, 2003.

[19] M. Boudhar. Scheduling on a single batch processing machine with split compatibility graphs. *soumis*, 2003.

[20] M. Boudhar and G. Finke. Batch processing with compatibility graph. In *INTAS Workshop*, Troyes (french), 4-5 September 1998.

[21] M. Boudhar and G. Finke. Ordonnancement par batch: motivations et résultats. *Prépublication de l'Institut de Mathématiques (USTHB, Alger)*, 262, 1998.

[22] M. Boudhar and G. Finke. Etude de quelques problèmes d'ordonnancement sur des machines à traitement par batchs avec des contraintes d'incompatibilité entre les tâches. (problèmes difficiles). *Prépublication de l'Institut de Mathématiques (USTHB, Alger)*, 003, 1999.

[23] M. Boudhar and G. Finke. Etude de quelques problèmes d'ordonnancement sur des machines à traitement par batchs avec des contraintes d'incompatibilité entre les tâches. (problèmes polynomiaux). *Prépublication de l'Institut de Mathématiques (USTHB, Alger)*, 004, 1999.

[24] M. Boudhar and G. Finke. Problème d'ordonnancement de tâches sur une machine à traitement par batch. In *Rencontre des Mathématiciens Algériens (RMA'2000)*, Alger, 21-24 Mai 2000.

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

[26] M. Boudhar and G. Finke. Scheduling on a batch processing machine with capacity equal to two. In *Fourteenth Conference on Quantitative Methods for Decision Making (ORBEL'14)*, Mons (Belgium), 20-21 January 2000.

[27] M. Boudhar and G. Finke. Scheduling on batch processing machines with constraints of compatibility between jobs. *Proceeding of Second IFAC/ IFIP/ IEEE Conference on Management and Control of Production and Logistics (MCPL'2000)*, 2:703–708, 2000.

[28] M. Boudhar and G. Finke. Problème d'ordonnancement de tâches sur une machine à traitement par batch. *Maghreb Mathematical Review (MMR)*, 10:161–178, 2001.

[29] M. Boudhar and A. Khelladi. Ordonnancement par batchs sous contraintes de compatibilité de tâches. *soumis*, 2003.

[30] M. Boudhar and A. Khelladi. Ordonnancement sur une machine à traitement par batch avec graphes de compatibilité. *Proceedings du Colloque International sur l'Optimisation et les Systèmes d'Information, COSI'2004 (Tizi-ouzou, 7-8 juin)*, pages 120–131, 2004.

[31] M. Boudhar and A. Khelladi. Ordonnancement sur une machine à traitement par batch avec graphes de compatibilité. *Prépublication de la Faculté de Mathématiques (USTHB, Alger)*, 01, 2004.

[32] N. Brauner, C. Dhaenens-Flipo, M-L. Espinouse, G. Finke, and H. Gavranovic. Decomposition into parallel work phases with application to the sheet metal industry. *Proceeding of International Conference on Industrial Engineering and Production Management (IEPM'99)*, 1:389–396, 1999.

[33] P. Brucker. *Scheduling algorithms.* Springer-Verlag, Berlin, 1995.

[34] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C. Potts, T. Tautenhahn, and S. Van De Velde. Scheduling a batching machine. *Journal of Scheduling*, 1:31–54, 1998.

[35] P. Brucker and S. Knust. Complexity results of scheduling problems. *page web: http://www.mathematik.uni-osnabrueck.de/research/OR/class/*, 2000.

[36] P. Brucker and A. Kramer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90:214–226, 1996.

[37] J. Carlier. *Problèmes d'ordonnancement à contraintes de ressources.* Thèse de Doctorat, université Pierre et Marie Curie, Paris (France), 1984.

[38] J. Carlier and P. Chretienne. *Problèmes d'ordonnancement : modélisation, complexité et algorithmes.* Masson, Paris, 1988.

[39] R. Carrachan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[40] V. Chandru and S. Gupta. Managing batch processors to reduce lead time in a semiconductor packaging line. *International Journal of Production Research*, 35:611–633, 1997.

[41] V. Chandru, C.Y. Lee, and R. Uzsoy. Minimizing total completion time on a batch processing machine with job families. *Operations Research Letters*, 13:61–65, 1993.

[42] V. Chandru, C.Y. Lee, and R. Uzsoy. Minimizing total completion time on batch processing machines. *International Journal of Production Research*, 31:2097–2121, 1993.

[43] C. Chu and J.M. Proth. *L'ordonnancement et ses applications*. Masson, Paris, 1996.

[44] E.G. Coffman, A. Nozari, and M. Yannakakis. Optimal scheduling of products with two subassemblies on single machine. *Operations Research*, 37:426–436, 1989.

[45] T. Cormen, C. Leiserson, and R. Rivest. *Introduction à l'algorithmique*. Dunod, Paris, 1994.

[46] M. Demange, D. De Werra, J. Monnot, and V.T. Paschos. Weighted node coloring: when stable sets are expensive (extended abstract). *In L. Kucera (Ed.) : WG 2002, LNCS 2573, Springer-Verlag Berlin Heidelberg*, pages 114–125, 2002.

[47] X. Deng and Y. Zhang. Minimizing mean reponse time in batch processing system. *Lecture Notes in Computer Science*, 1627:231–240, 1999.

[48] V. Detry. Problème de la clique maximale dans un graphe orienté. *rapport de recherche RR-96/09, CNRS*, 1996.

[49] G. Dobson and R.S. Nambinadom. The batch loading and scheduling problem. *Research Report, Simon School of Business Administration, University of Rochester NY*, 1992.

[50] I. Duenyas and J.J. Neale. Stochastic scheduling of batch processing machine with incompatible job families. *Annals of Operations Research*, 70:191–220, 1997.

[51] L. Dupont and F. Jolai Ghazvini. Branch and bound method for minimizing mean flow time on a single batch processing machine. *International Journal of Industrial Engineering: Practice and Theory*, 4:197–203, 1997.

[52] L. Dupont and F. Jolai Ghazvini. Minimizing makespan on a single batch processing machine with non identical job sizes. In *Proceeding of the Second International Conference on Industrial Engineering and Production Management*, volume 2, pages 57–66, Lyon, France, 20-24 october 1997.

[53] P. Esquirol and P. Lopez. *L'ordonnancement*. Economica, 1999.

[54] G. Finke, M. Boudhar, H. Gavranovic, and A. Oulamara. Batch processing. In *European Chapter on Combinatorial Optimization (ECCO XIII)*, Capri (Italy), 18-20 May 2000.

[55] G. Finke, V. Jost, and M. Queyranne. Batch processing with interval graph compatibilities between tasks. *In the Second Inernational Workshop on Discrete Optimization Methods and Logistics (DOM-2004) Omsk-Irkutsk (Russia), 20–27 july 2004*.

[56] J.W. Flower, D.J. Phillips, and G.L. Hogg. Real time control of multiproduct bulk-service semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 5:158–163, 1992.

[57] S. French. *Sequencing and scheduling: An introduction to the mathematics of the job-shop.* Ellis Horwood Limited, Chichester, Englend, 1982.

[58] V. Gabrel. Scheduling jobs within time windows on identical parallel machines: New model and algorithms. *European Journal of Operational Research*, 83:320–329, 1995.

[59] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[60] H. Gavranovic. *Affectation de fréquences et conception optimale d'une ligne de production: modèles et algorithmes de résolutions.* Thèse de Doctorat, IMAG (UJF), Grenoble (France), 2002.

[61] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of chordal graph. *SIAM Journal of Computing*, 1:180–187, 1972.

[62] F. Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357–369, 1974.

[63] S. Gerke. Colouring weighted bipartite graphs with a co-site constraint. *Discrete Mathematics*, 224:125–138, 2000.

[64] F. Jolai Ghazvini. *Ordonnancement sous contrainte de groupage (machine à traitement par batch).* Thèse de Doctorat, E.N.S.G.I.(I.N.P.G.), Grenoble (France), 1998.

[65] F. Jolai Ghazvini and L. Dupont. Minimizing total completion time on a single batch processing machine. In *proceeding of the workshop on production planning and control*, pages 210–217, Mons, Belgium, 9-11 September 1996.

[66] M.C. Golumbic. The complexity of comparability graph recognition and coloring. *Computing*, 18:199–208, 1977.

[67] M.C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

[68] M.C. Golumbic. Algorithmic aspects of perfect graphs. *Annals of Discrete Mathematics*, 21:301–323, 1984.

[69] M. Gondran and M. Minoux. *Graphes et algorithmes.* Eyrolles, Paris, 1979.

[70] GOTHA. Les problèmes d'ordonnancement. *RAIRO: Recherche Opérationnelle*, 27:77–150, 1993.

[71] M. Grotschel, L. Lovasz, and A. Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

[72] H. Gurani, R. Anupindi, and R. Akella. Control of batch processing systems in semiconductor wafer fabrication facilities. *IEEE Transactions on Semiconductor Manufacturing*, 5:319–328, 1992.

[73] D.S. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, Boston, USA, 1997.

[74] D.S. Hochbaum and D. Landy. Algorithms and heuristics for scheduling semiconductor burn-in operations. *Research Report ESRC 94-8, University of California, Berkeley, USA*, 1994.

[75] D.S. Hochbaum and D. Landy. Scheduling with batching : minimizing the weighted number of tardy jobs. *Operations Research Letters*, pages 79–86, 1994.

[76] D.S. Hochbaum and D. Landy. Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations Research*, 45:874–885, 1997.

[77] H. Hoogeveen, S.L. Van De Velde, and B. Veltman. Complexity of scheduling multiprocessors tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55:259–272, 1994.

[78] H. Hoogeveen and S.V. De Velde. Scheduling by positional completion times : Analysis of a two-stage flow shop problem with a batching machine. *Mathematical Programming*, 82:273–289, 1998.

[79] J.A. Hoogeveen, J.K. Lenstra, and S.L. Van De Velde. Sequencing and scheduling : an annotated bibliography. *Memorandum COSOR 97-02, Eindhoven University of Technology*, 1997.

[80] Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5:61–65, 1986.

[81] K. Jansen, P. Scheffler, and G. Woeginger. The disjoint cliques problem. *RAIRO : Recherche Opérationnelle*, 31:45–66, 1997.

[82] C. Jordan. *Batching and scheduling : Models and methods for several problem classes.* Springer-Verlag, Berlin, 1996.

[83] B. Jurish, W. Kubiak, and J. Jozfowskaj. Algorithms for minclique scheduling problems. *Discrete Applied Mathematics*, 72:115–139, 1997.

[84] D.E. Knuth. *The art of computer programming. Volume 1: Fundamental Algorithms.* Addison–Wesley, 1968.

[85] D.E. Knuth. *The art of computer programming. Volume 3: Sorting and searching.* Addison–Wesley, 1973.

[86] W. Kubiak and F. Jolai Ghazvini. Minimizing earliness/tardiness criteria on a batch processing machine with job families. In *Second Annual International Conference on Industrial Engineering*, volume 2, pages 785–790, San Diego, California, USA, 22-15 November 1997.

[87] C.Y. Lee, L. Lei, and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997.

[88] C.Y. Lee and R. Uzsoy. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, 37:219–236, 1999.

[89] C.Y. Lee, R. Uzsoy, and L.A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40:764–775, 1992.

[90] C.L. Li and C.Y. Lee. Scheduling with agreeable release and due dates on a batch processing machine. *European Journal of Operational Research*, 96:564–569, 1997.

[91] Z. Liu and W. Yu. Scheduling one batch processor subject to job release dates. *Discrete Applied Mathematics*, 105:129–136, 2000.

[92] Z. Liu, J. Yuan, and T.C.E. Cheng. On scheduling an unbounded batch machine. *Operations Research Letters*, 31:42–48, 2003.

[93] L. Lovasz. *Combinatorial problems and exercises.* North-Holland, 1993.

[94] D.G. Luenberger. *Introduction to linear and nonlinear programming.* Addison-Wesley Publishing Company, USA, 1973.

[95] S.V. Mehta and R. Uzsoy. Minimizing total tardiness on a batch processing machine with incompatible job families. *I.I.E. Transaction on Scheduling and Logistics*, 31:165–178, 1998.

[96] P.L. Nguyen. *Planification tactique de la production: approche hiérarchisée par une classe d'entreprise de sous-traitance et application au cas d'une entreprise de laminage à froid.* thèse de Doctorat, UJF-Grenoble, France, 1997.

[97] W.J. Niehaus. *Design of maximum-cardinality and maximum weight clique heuristics with applications.* thèse de Ph.D., School of computer science, Carnegie Mellon University, Pittsburgh, PA, (USA), 1996.

[98] Noname. C clique partitioning heuristic. *page web: http://www.ececs.uc.edu/~ddel/theses/jay_new/node16.html*, 1999.

[99] A. Oulamara. *Flowshops avec détérioration des tâches et regroupement des tâches.* Thèse de Doctorat, IMAG(UJF), Grenoble (France), 2001.

[100] A. Oulamara and G. Finke. Flow shop problems with two batch processing machines. *International Journal of Mathematical Algorithms*, 2:269–287, 2001.

[101] A. Oulamara, M.Y. Kovalyov, and G. Finke. Scheduling a no-wait flow shop with unbounded batching machines. *papier soumis*, 2004.

[102] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexité.* Prentice-hall, Inc, Englewood, N.J., 1982.

[103] C. Paul. *Parcours en largeur lexicographique : un algorithme de partitionnement. Application aux graphes et généralisation.* thèse de Doctorat, université Montpellier II, France, 1998.

[104] M. Pinedo. *Scheduling theory.* John Wiley & Sons, New York, 1995.

[105] C.N. Potts and Y.K. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[106] C. Prins. *Algorithmes de graphes.* Eyrolles, Paris, 1997.

[107] J.M. Proth and N. Sauer. Homogénéisation et découpe des billettes d'aluminium (hodec). *Rapport de recherche, Projet PECHINEY, INRIA-Lorraine*, 1994.

[108] J.K. Robinson, J.W. Fowler, and J.F. Bard. The use of upstream and downstream information in scheduling semiconductor batch operations. *International Journal of Production Research*, 33:1849–1869, 1995.

[109] M. Sakarovitch. *Optimisation combinatoire: programmation discrète.* Hermann, Paris, 1984.

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

[111] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Chichester, Englend, 1999.

[112] F.D. Shallcross. A polynomial algorithm for a one machine batching problem. *Operations Research Letters*, 11:213–218, 1992.

[113] B. Simeone. Nuggets in matching theory. *Papier personnel. Department of Statistics, La Sapienza University, Rome, Italy*, pages 1–29, 2002.

[114] C.S. Sung and Y.I. Choung. Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research*, 120:559–574, 2000.

[115] C.S. Sung, Y.H. KIm, and S.H. Yoon. A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research*, 121:179–192, 2000.

[116] V.S. Tanaev, V.S. Gordan, and Y.M. Shafransky. *Single-stage systems.* Kluwer Academic Publishers, 1994.

[117] V.S. Tanaev, Y.N. Sotskov, and V.A. Strusevich. *Multi-stage systems.* Kluwer Academic Publishers, 1994.

[118] R.E. Tarjan. *Data structures and networks algorithms.* SIAM, 1983.

[119] J. Teghem. *Programmation linéaire.* Editions de l'Université de Bruxelles, 1996.

[120] R. Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32:1615–1638, 1994.

[121] R. Uzsoy. Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33:2685–2708, 1995.

[122] R. Uzsoy and Y. Yang. Minimizing total weighted completion time on a single batch processing machine. *Research Report R.R. 95*, 1995.

[123] S. Webster and K.R. Baker. Scheduling groups of jobs on a single machine. *Operations Research*, 43:692–704, 1995.

[124] N. Wirth. *Algorithm + data structures = programs.* Prentice Hall, 1976.

[125] N.H. Xuong. *Mathématique discrètes et informatique.* Masson, Paris, France, 1992.

Résumé : Le travail présenté dans cette thèse, traite le problème de l'ordonnancement de tâches indépendantes sur des machines à traitement par batch en minimisant la date de fin de traitement (le makespan). Nous supposons qu'il existe une relation de compatibilité entre les tâches, où deux tâches sont dites compatibles si elles peuvent être traitées simultanément dans un même batch. Cette relation de compatibilité est représentée 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 dernières (tâches) sont compatibles. Nous considérons plusieurs cas possibles de graphes : graphe quelconque, graphe complet, graphe biparti, graphe scindé, complémentaire d'un graphe biparti, graphe spécial, etc. La capacité de la machine à traitement par batch peut être finie (elle ne peut traiter qu'un nombre fini de tâches à la fois) ou infinie (elle peut traiter un nombre illimité de tâches à la fois). Chaque tâche a un temps de traitement et une date de disponibilité qui sont des entiers naturels. Le temps de traitement d'un batch est égal au maximum des temps de traitement des tâches appartenant à celui-ci. Toutes les tâches d'un même batch commencent leur exécution à une même date et terminent leur exécution à une même date. L'interruption des tâches n'est pas autorisée.
Nous commençons par donner une formulation mathématique de ce problème. Nous montrons que certains sous-problèmes sont difficiles à résoudre (NP-difficiles) et nous proposons plusieurs algorithmes polynomiaux exacts pour résoudre des sous-problèmes polynomiaux. Enfin, nous proposons des heuristiques et des algorithmes exacts de type Séparation et Evaluation pour résoudre le problème avec des expérimentations numériques.
Cette nouvelle combinaison de l'ordonnancement par batchs et de graphes de compatibilité définis entre les tâches apparaît comme un concept nouveau et intéressant utilisant, à la fois, la théorie de l'ordonnancement et la théorie des graphes.

Mots clés : Ordonnancement par batchs, machine à traitement par batch, graphe de compatibilité, date de fin de traitement, heuristique, séparation et évaluation.

Abstract: The work, presented in this thesis, deals with the problem of scheduling multiple independent jobs on batch processing machines in order to minimize the makespan (final completion time). We suppose that there exists a relation of compatibility between jobs, where two jobs are said to be compatible if they can be processed simultaneously in the same batch. This relation of compatibility is represented by a graph G=(V,E) where V is the set of jobs and any pair of jobs is in E if and only if they are compatible. We consider several possible cases of graphs: arbitrary graph, complete graph, bipartite graph, split graph, complementary of a bipartite graph, special graph, etc. The capacity of the batch processing machine is finite (it can process a finite number of jobs simultaneously) or infinite (it can process all jobs simultaneously). A job has a processing time and a release time that are assumed to be positive integers. The processing time of a batch is equal to the maximum processing time of any job assigned to it. All jobs in a batch start at a same date and finish at a same date. Preemption is not allowed.
We begin with a mathematical formulation of this problem. We show that several subproblems are difficult (NP-hard). We propose certain exact polynomial algorithms to solve several polynomial subproblems. Then, we propose heuristics and exact algorithms of Branch and Bound variety to solve the problem with numerical experimentations.
This new combination of batches scheduling with a compatibility graphs, defined between jobs, is an original contribution and appears to be a new and an interesting concept relating scheduling theory and graph theory.

Keywords: batch scheduling, batch processing machine, compatibility graph, makespan, heuristic, branch and bound.

ملخص : تتناول هذه الأطروحة دراسة مسألة ترتيب الأعمال المستقلة على آلة المعالجة بالدفعة بتصغير زمن نهاية المعالجة. نفرض وجود علاقة تلاؤم بين الأعمال، حيث يقال عن عملين أنهما متلائمان إذا أمكن معالجتهما في آن واحد في نفس الدفعة. علاقة التلاؤم هذه موضحة بالمنحنى البياني G=(V,E) حيث V يمثل مجموعة الأعمال، و يكون زوج من هذه الأعمال موجودا في E إذا و فقط إذا كانت هذه الأخيرة (الأعمال) متلائمة. نعتبر عدة حالات محتملة للبيانات: بيان كيفي، بيان تام، بيان ثنائي الإنقسام، بيان مجزأ، متمم لبيان ثنائي الإنقسام، بيان خاص، الخ. يمكن لسعة آلة المعالجة بالدفعة أن تكون محدودة (لا يمكها معالجة إلا عدد منته من الأعمال في آن واحد) أو غير محدودة (يمكها معالجة عدد غير منته من الأعمال في آن واحد). لكل عمل مدة معالجة و زمن توفر، و هما أعداد طبيعية. مدة معالجة الدفعة تساوي أكبر مدة معالجة للإعمال المنتمية إلى هذه الدفعة. كل الأعمال من نفس الدفعة يبدأ تنفيذها في نفس الزمن وينتهي تنفيذها في نفس الزمن. ينبغي الإشارة إلى أنه لا يسمح قطع الأعمال.
في البداية نعطي صيغة رياضية لهذه المسألة، نبرهن أن بعض المسائل الفرعية صعبة الحل (NP- صعبة) و نقترح عدة خوارزميات حدودية دقيقة لحل المسائل الفرعية الحدودية. أخيرا نقترح خوارزميات تقريبية و خوارزميات دقيقة من النوع تفرقة و تقويم لحل المسألة مع تجارب عددية.
هذا النظام الجديد للترتيب بالدفعة و بيانات التلاؤم المعرفة بين الأعمال يظهر كتصور جديد و مهم، يستعمل نظرية الترتيب و نظرية البيانات معا.

الكلمات المفاتيح : الترتيب بالدفعة، آلة المعالجة بالدفعة، بيانات التلاؤم، زمن نهاية الترتيب، خوارزميات تقريبية، تفرقة و تقويم.