

Complexité et approximation pour un problème d'ordonnancement avec tâches couplées

G. Simonin, R. Giroudeau et J.C. König

LIRMM,
161 rue Ada,
34392 Montpellier - France Cedex 5, UMR 5506
gilles.simonin@lirmm.fr

Résumé

Nous étudions un problème d'ordonnancement avec des tâches-couplées en présence d'un graphe de compatibilité sur un monoprocesseur. Dans ce cadre, nous montrerons que ce problème est \mathcal{NP} -complet.

Nous développerons également un algorithme d'approximation en $O(n^3)$ avec un ratio de $\frac{(\alpha + 6)}{6}$, avec $\alpha \in \mathcal{N}$ où α est l'intervalle de temps entre les deux sous-tâches d'une tâche-couplée.

Mots-clés : Ordonnancement, Tâche-Couplée, Compatibilité, Complexité, Approximation

1. Introduction

1.1. Présentation du problème

Dans cet article, nous étudions le problème d'ordonnancer des tâches-couplées, introduites par Shapiro [1], soumises à des contraintes de compatibilité et de précédence sur un monoprocesseur. Pour cela, un ensemble $\mathcal{A} = \{1, \dots, n\}$ de tâches-couplées est donné. Chaque tâche $A_i \in \mathcal{A}$ consiste en deux sous-tâches a_i et b_i de durée d'exécution unitaire séparées par un délai d'inactivité fixe $L_i = \alpha$ avec $\alpha \geq 1$. La seconde sous-tâche b_i doit démarrer son exécution exactement α unités de temps après que la première sous-tâche a_i ait été exécutée. Nous introduisons ensuite un graphe de compatibilité $G_c = (\mathcal{A}, E_c)$ où \mathcal{A} est l'ensemble des tâches-couplées, et où E_c représente l'ensemble des arêtes reliant deux tâches-couplées susceptibles de s'intersecter, c'est à dire de pouvoir exécuter au moins une sous-tâche d'une tâche A_i pendant le délai d'inactivité d'une autre tâche A_j .

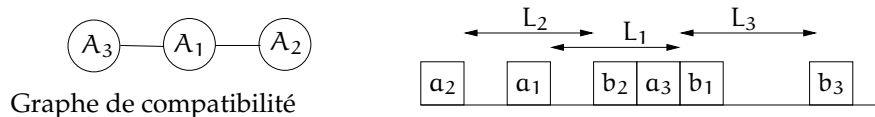


FIG. 1 – Illustration de la contrainte de compatibilité. Sur le graphe de compatibilité, A_3 est relié à A_1 et A_1 est relié à A_2 , du coup A_3 et A_1 peuvent être exécutées en parallèle, même chose pour A_1 et A_2 . Par contre il sera impossible d'exécuter en même temps A_3 et A_2 . Ainsi, pour avoir un ordonnancement optimal, nous serons obligés d'exécuter A_2 suivi de A_3 , ou l'inverse. Puisque A_1 peut s'exécuter en même temps que les deux autres, il nous suffit d'exécuter la première sous-tâche a_1 entre a_2 et b_2 , et d'exécuter la seconde sous-tâche b_1 entre a_3 et b_3 .

Nous ajoutons également un ensemble \mathcal{T} de tâches préemptives sans structure particulière de durée d'exécution p_{T_i} . De plus, nous introduisons un graphe de précédence entre l'ensemble \mathcal{A} et \mathcal{T} de telle

manière que pour une tâche A_i de \mathcal{A} , nous avons une tâche T_i de \mathcal{T} qui lui succède. Toutes les tâches dans \mathcal{A} et \mathcal{T} seront exécutées sur un monoprocesseur.

L'objectif est de minimiser la longueur de l'ordonnancement. En utilisant la notation standard introduite par Graham et al. [2], notre problème noté Π sera défini par $||\text{prec}$, tâche – couplée, $(a_i = b_i = 1, L_i = \alpha) \cup (p_{T_i}, p_{\text{mtn}}) | C_{\text{max}}$.

Nous allons mesurer l'impact de l'introduction du graphe de compatibilité sur la complexité et l'approximation pour les problèmes d'ordonnancement de tâches-couplées sur un monoprocesseur.

1.2. Motivation

L'étude du problème d'ordonner des tâches-couplées soumises à des contraintes de compatibilité est motivée par le problème d'acquisition de données par une torpille en immersion. En effet, la torpille possède des capteurs qui collectent des informations, qui sont alors traitées sur un monoprocesseur. Une sonde émet une onde qui se propage sous l'eau pour recueillir des données, appelées tâches d'acquisition. Ainsi, nous aurons deux sous-tâches, une qui envoie l'écho, l'autre qui le reçoit et un temps d'inactivité incompressible et indilatable entre les deux sous-tâches qui représente la propagation de l'écho sous l'eau. Les tâches d'acquisition peuvent être assimilées à des tâches-couplées.

Pendant ce temps d'inactivité, nous pouvons envoyer d'autres échos sur d'autres sondes afin d'employer le temps d'inactivité. Cependant, la localisation trop proche des ondes provoquent des perturbations et des interférences. Sachant que nous souhaitons traiter des informations exemptes d'erreurs, nous construisons un graphe dit de compatibilité entre les tâches. Ce graphe décrit l'ensemble des tâches pouvant potentiellement exécuter au moins une sous-tâche durant la période d'inactivité d'une autre.

Les informations récoltées via le retour de l'écho vont permettre de réaliser différents calculs traités par le monoprocesseur, par exemple au bout de dix mesures de la température de l'eau, le processeur calcule une moyenne. Ces tâches dites de traitement sont des successeurs des tâches d'acquisition. Dans la suite, nous supposons que chaque tâche d'acquisition est suivie d'une tâche de traitement.

1.3. Description du modèle

Formellement, notre problème peut-être défini comme suit :

- $t_{b_i} = t_{a_i} + 1 + \alpha, \forall A_i \in \mathcal{A}$ où t_{a_i} désigne le début d'exécution de la sous-tâche a_i de la tâche A_i
- $t_{T_i} > t_{b_i}, \forall i$ tel qu'une contrainte de précédence existe entre A_i et T_i
- $t_{a_i} \notin [t_{a_j} + 1, t_{b_j}[$ et $t_{b_i} \notin [t_{a_j} + 1, t_{b_j}[$, pour i et j tel qu'il existe une arête dans G_c entre A_i et A_j

1.4. Etat de l'art

Le problème d'ordonner un ensemble de tâches-couplées sans graphe de compatibilité a été étudié du point de vue de la complexité ([3], [4], [5]). Dans cet article nous étudions un problème dans lequel les tâches-couplées (ou tâches d'acquisition) sont soumises à des contraintes de compatibilité. A contrario, les travaux existants portent sur des contraintes de précédence entre les A_i ([3], [5], [4]). Les principaux résultats de complexité et d'approximation sont donnés par le tableau 1.

Problème	Complexité	Approximation	Référence
$a_i = b_i = L_i$	\mathcal{NP} -complet		[6]
$a_i = a, L_i = L, b_i$	\mathcal{NP} -complet		[6]
$a_i = a, L_i, b_i = b$	\mathcal{NP} -complet		[6]
$a_i = b_i = 1, L_i$	\mathcal{NP} -complet	1,75	[7]
a_i, b_i, L_i	\mathcal{NP} -complet	3,5	[6]
$a_i = a, L_i = \alpha, b_i = b$	Problème ouvert		[5]
$a_i = b_i = p, L_i = \alpha$	Polynomial		[6]

TAB. 1 – Principaux résultats de complexité et approximation

1.5. Résultats

Nous montrons tout d'abord que le problème sans graphe de compatibilité G_c connu pour être polynomial [6] devient \mathcal{NP} -complet avec la contrainte de compatibilité. Puis nous développons un algorithme d'approximation en temps polynomial basé sur un couplage maximum dans le graphe de compatibilité.

Cet article est organisé comme suit, dans la section suivante, nous allons prouver que notre problème est \mathcal{NP} -complet. Dans la dernière section, nous développerons un algorithme d'approximation en temps polynomial avec une performance de garantie inférieure à $(\frac{\alpha+6}{6})$.

2. Résultat de complexité

Remarquons que dans le cas spécial où $\alpha = 1$, le problème est polynomial. Il est suffisant de trouver un couplage maximum dans le graphe de compatibilité.

Théorème 2.1 *Le problème de décision Π est \mathcal{NP} -complet.*

Preuve

Notre approche est similaire à la preuve proposée par Lenstra et Rinnoy Kan [2] pour le problème $P|prec; P_j = 1|C_{max}$. Notre démonstration est basée sur le problème \mathcal{NP} -complet *Clique* (voir Garey et Johnson GT19 [8]) :

INSTANCE : Un graphe $G = (V, E)$ où $|V| = n$, et un entier K .

QUESTION : Peut on trouver une clique de taille K dans G ?

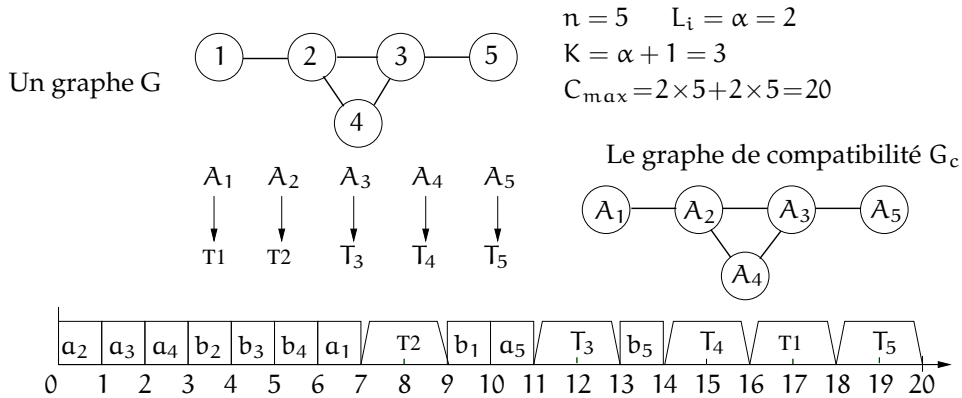


FIG. 2 – Illustration de la transformation en temps polynomial : Clique $\alpha \Pi$

Le problème où $\alpha = 2$ est ouvert. Dans toute la suite, nous considérons que $\alpha \geq 3$. Soit I^* une instance du problème Clique, nous allons construire une instance I de Π avec $C_{max} = 2n + \sum_{T_i \in \mathcal{T}} p_{T_i}$ de la manière

suivante :

Soit $G = (V, E)$ un graphe, avec $|V| = n$:

- $\forall v \in V$, une tâche d'acquisition A_v est créée, composée de deux sous-tâches a_v et b_v avec leur temps d'exécution $p_{a_v} = p_{b_v} = 1$ et avec un temps d'inactivité entre ces deux sous-tâches, de longueur $\alpha = (K - 1)$.
- Pour chaque arête $e = (v, w) \in E$, il y a une relation de compatibilité entre les deux tâches d'acquisition A_v et A_w .
- Pour chaque tâche A_v , nous créons une tâche de traitement T_v qui est son successeur.
- Chaque T_v a un temps d'exécution noté $p_{T_v} = \alpha$.

- Supposons maintenant l'existence d'une clique de longueur $K = (\alpha + 1)$ dans le graphe G . Prouvons qu'il existe alors un ordonnancement sans temps d'inactivité. Nous exécutons les $K = (\alpha + 1)$ tâches d'acquisition de la clique à $t = 0$, formant ainsi un bloc sans temps d'inactivité. Nous exécutons ensuite les tâches d'acquisition restantes A_v , puis les tâches de traitement qui remplissent les temps d'inactivités des A_v .
- Réciproquement, supposons qu'il existe un ordonnancement sans temps d'inactivité, nous allons montrer que le graphe G contient une clique de taille $K = (\alpha + 1)$. Avec les contraintes de précédence entre les tâches A_v et T_v , il est facile de voir que nous ne pouvons ordonnancer qu'une tâche A_v à $t = 0$, $\forall v \in V$. Cette tâche va créer un temps d'inactivité de taille α , sachant que nous avons un ordonnancement sans temps d'inactivité, nous sommes sûr d'avoir $(\alpha + 1)$ tâches d'acquisition exécutées les unes dans les autres.

Ainsi, nous avons $(\alpha + 1)$ tâches d'acquisition qui sont compatibles. Et dans le graphe de compatibilité G_c , nous aurons une arête entre chaque couple de ces tâches A_v . En conséquence, les tâches $A_{p_1}, A_{p_2}, \dots, A_{p_\alpha}$, associées aux sommets du graphe G , forment une clique de taille $K = (\alpha + 1)$. Ceci conclut la preuve du théorème 2.1. □

3. Algorithme d'approximation

Dans cette section, nous développerons un algorithme d'approximation de complexité polynomiale avec garantie de performance non triviale. Nous étudierons la garantie de performance classique c'est à dire le ratio, pris sur toutes les instances, entre la solution proposée par une heuristique et une solution optimale. Cet algorithme est basé sur le problème de la recherche d'un couplage maximum dans le graphe de compatibilité.

Remarque 3.1 *Notons que dans le cas où le temps d'exécution des tâches de traitement est supérieur à un ($p_{T_i} > 1, \forall i$), la somme des temps d'inactivité ne peut pas être plus élevée que lorsque les délais d'exécution des tâches de traitement est égal à un ($p_{T_i} = 1$).*

Par la suite, les tâches de traitement auront un temps d'exécution égal à $p_{T_i} = 1$.

3.1. Borne inférieure pour toute solution optimale

Nous allons proposer deux bornes inférieures. Pour la première, l'ordonnancement optimal est obtenu lorsque nous n'avons aucun temps d'inactivité. Par ailleurs, nous savons que le nombre de tâches de traitement est égal au nombre de tâches d'acquisition et que dans le pire des cas, toutes les tâches de traitement ont un temps d'exécution égal à $p_{T_i} = 1, \forall i$. Ainsi, nous avons : $C_{\max}^{\text{opt}} \geq T_{\text{seq}} = 2n + \sum_{T_i \in \mathcal{T}} p_{T_i} = 2n + n = 3n$.

De plus, en considérant un couplage maximum M dans le graphe de compatibilité de taille m , le nombre de sommets indépendants s'élèvent à $(n - 2m)$. Dans le pire des cas, l'ordonnancement optimal est forcément supérieur à l'ordonnancement des sommets indépendants suivis de la dernière tâche de traitement. Et donc nous avons $C_{\max}^{\text{opt}} \geq (n - 2m)(\alpha + 2) + 1$.

Pour notre étude, notre borne inférieure sera :

$$C_{\max}^{\text{opt}} \geq \max\{3n, (n - 2m)(\alpha + 2) + 1\}$$

3.2. Borne supérieure de l'algorithme

Algorithme 1 : Un algorithme d'approximation en temps polynomial

Données : $\mathcal{A}, \mathcal{T}, G_c, \alpha \geq 1$

Résultat : C_{max}^h

1 **début**

- Chercher un couplage maximum M dans G_c
- Pour chaque arête (i, j) du couplage maximum, les tâches d'acquisition A_i et A_j sont ordonnancées telles que $t_{a_j} = t_{a_i} + 1$
- Pour chaque sommet i restant, nous exécutons la tâche d'acquisition A_i
- Exécutons les tâches de traitement aux premiers emplacements libres en respectant les contraintes de précédence

2 **fin**

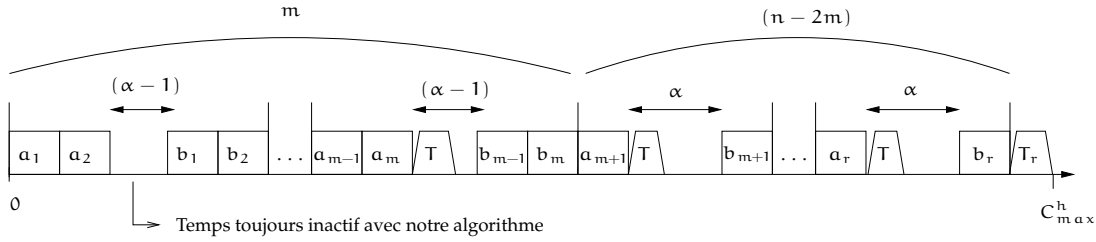


FIG. 3 – Illustration de l'algorithme d'approximation

La complexité de cet algorithme dépend surtout de celle du couplage maximum. En effet, après avoir trouvé le couplage maximum, les autres étapes de l'algorithme se font en temps linéaire. En prenant l'algorithme donné par Gabow [12], nous avons une complexité de $O(n^3)$ pour le couplage maximum. Ainsi notre algorithme s'exécutera en $O(n^3)$.

Nous allons donner quelques remarques essentielles sur la structure de l'ordonnancement donné par notre algorithme d'approximation. Nous supposons que nous avons un ordonnancement donné par l'algorithme d'approximation avec un couplage maximum de taille m .

- Dans le premier bloc créé par deux tâches d'acquisition reliées dans le couplage maximum, il existe un temps d'inactivité de taille $(\alpha - 1)$.
- Pour deux tâches M -saturées exécutées consécutivement, le temps d'inactivité de ces tâches est $(\alpha - 1)$.
- Pour chaque tâches d'acquisition M -non-saturées restantes dans le couplage maximum, le temps d'inactivité est de α .
- Considérons la dernière tâche d'acquisition notée A_r . Après son exécution, nous pouvons ordonnancer la tâche de traitement T_r (ce cas survient lorsque toutes les tâches de traitement, à l'exception de T_r , sont ordonnancées avant le temps de complétude de A_r), ou plusieurs tâches traitement (ce cas se produit lorsqu'il n'y a pas de temps d'inactivité avant le temps de complétude de A_r . Voir figure (3), pour une illustration du cas où la tâche T_r est la seule tâche de traitement exécutée après le temps de complétude de A_r).

Ainsi, le nombre de tâche de traitement exécutées après A_r est : $\max\{n - (m - 1)(\alpha - 1) - (n - 2m)\alpha, 1\}$

Finalement, notre borne supérieure sera : $C_{max}^h \leq T_{execution} + T_{temps\ d'inactivité} \leq [2n + \max\{n - n\alpha + m(\alpha + 1) + \alpha - 1, 1\}] + [\alpha(n - m) - m]$

Grâce à une étude du \max , nous pouvons étudier le comportement de la performance relative.

3.3. Performance relative

En premier lieu, nous constatons que pour $\alpha \geq 4$, il est facile de voir que $\max\{n - n\alpha + m(\alpha + 1) + \alpha - 1, 1\} = 1$ puisque par définition $m \leq \frac{n}{2}$. Ainsi $C_{max}^h \leq 2n + \alpha(n - m) - m + 1$. Du coup, dans

un première temps, le tableau suivant donne un résumé du rapport de la performance relative ρ pour $\alpha \in \{1, 2, 3\}$ où la valeur de C_{\max}^h varie selon α .

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
ρ	1	si $m \geq \frac{n}{3}$, alors $\rho \leq 1 + \frac{1}{3n}$ si $\frac{n+1}{8} \leq m < \frac{n}{3}$, alors $\rho \leq \frac{4}{3} + \frac{1}{3n}$ si $m < \frac{n+1}{8}$, alors $\rho \leq \frac{11}{6}$	si $m = \frac{n}{2}$, alors $\rho \leq 1 + \frac{2}{3n}$ si $\frac{2n+1}{10} \leq m < \frac{n}{2}$, alors $\rho \leq \frac{5}{3} + \frac{1}{3n}$ si $m < \frac{2n+1}{10}$, alors $\rho \leq \frac{21}{17}$

TAB. 2 – Performance relative pour $\alpha \in \{1, 2, 3\}$

Dans un deuxième temps, nous pouvons nous focaliser sur l'étude de $\alpha \geq 4$ où $C_{\max}^h \leq 2n + \alpha(n - m) - m + 1$. De plus, puisque $C_{\max}^{\text{opt}} \geq \max\{3n, n\alpha + 2n - 2m\alpha - 4m + 1\}$, les cas suivants doivent être considérés :

- Pour $m \in [0, \frac{n(\alpha-1)+1}{2(\alpha+2)}[$, $C_{\max}^{\text{opt}} \geq 3n$.
- Pour $m \in [\frac{n(\alpha-1)+1}{2(\alpha+2)}, \frac{n}{2}]$, $C_{\max}^{\text{opt}} \geq n\alpha + 2n - 2m\alpha - 4m + 1$.

Selon les valeurs de m , nous donnons la borne supérieure pour l'ordonnancement proposé par l'heuristique h , et la borne inférieure pour un ordonnancement optimal (voir illustration figure 4).

Notons que pour $m = 0$, $\rho = 1$ (c'est évident, puisque le graphe de compatibilité est un ensemble de tâches indépendantes), de plus pour $m = \frac{n}{2}$, $\rho = \frac{(\alpha+3)}{6}$.

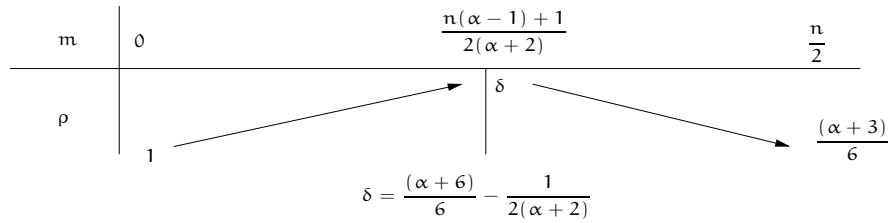


FIG. 4 – Comportement de la performance relative ρ en fonction de m

4. Conclusion et perspectives

Dans cet article, nous avons présenté un problème d'ordonnancement sur monoprocesseur avec des contraintes sur les tâches-couplées. Du côté négatif, nous avons montré que le problème est \mathcal{NP} -complet. Du côté positif, nous avons donné un algorithme d'approximation en $O(n^3)$ avec une performance relative bornée par $\rho < \frac{(\alpha+6)}{6}$, où α est le temps d'inactivité des tâches d'acquisition. La valeur de la performance relative ρ associée à l'algorithme dépend du paramètre α , qui est l'une des données du problème. Cette remarque amène une question fondamentale : «Est-ce que notre problème admet un algorithme d'approximation avec une garantie de performance égale à une valeur constante ?». Enfin, il peut être intéressant d'étendre ces résultats dans le cas de systèmes multi-processeurs, et de voir si notre problème reste \mathcal{NP} -complet.

Bibliographie

1. R.D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27 :477–481, 1980.
2. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.
3. J. Blazewicz, K.H. Ecker, T. Kis, and M. Tanas. A note on the complexity of scheduling coupled-tasks on a single processor. *Journal of the Brazilian Computer Society*, 7(3) :23–26, 2001.
4. A.J. Orman and C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72 :141–154, 1997.
5. D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled-tasks. *Mathematical Methods of Operations Research*, 59 :193–203(11), June 2004.
6. Alexander A. Ageev and Alexander V. Kononov. Approximation algorithms for scheduling problems with exact delays. In *WAOA*, pages 1–14, 2006.
7. A. A. Ageev et A. E. Baburin. Approximation algorithms for the simple and two-machine scheduling problems with exact delays. *to appear in Operations Research Letters*.
8. M.R. Garey and D.S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
9. P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., USA, 2001.
10. J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling computer and manufacturing processes*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.
11. M. Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Prentice Hall, 1995.
12. Harold N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, 23(2) :221 – 234, 1976.