Un algorithme basé sur l'Inclusion-Exclusion pour la résolution du flowshop de permutation avec précédences entre travaux

Olivier Ploton[1], Vincent T'kindt[1]

Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée (LIFAT, EA 6300), EMR CNRS 7002 ROOT, Tours, France
{olivier.ploton,vincent.tkindt}@univ-tours.fr

Mots-clés : *Ordonnancement, Inclusion-Exclusion, Flowshop, Précédences.*

1 Introduction

Nous considérons le problème d'ordonnancement du flowshop de permutation, avec n travaux et m machines. Chaque travail i se compose de m opérations $O_{i1}, \ldots, O_{im}$, qui doivent être traitées sur les machines 1 à m, dans cet ordre. Chaque opération O_{ij} a un temps de traitement p_{ij}, et chaque machine j ne peut traiter qu'une seule opération à la fois. Toutes les machines doivent traiter les opérations dans le même ordre, et un ordonnancement est essentiellement défini par cet ordre. De plus, cet ordre peut être contraint par des précédences entre les travaux.

À chaque travail i est associé un coût, obtenu par le calcul d'une fonction de coût f_i appliquée au temps d'achèvement C_{im} de la dernière opération du travail i. À partir de cela, la fonction objectif à minimiser est définie comme $f_{\max} = \max_{1 \leq i \leq n} f_i(C_{im})$. Chaque coût est supposé être régulier et polynomialement borné par rapport à son argument C_{im}, comme c'est le cas, par exemple, pour le temps d'achèvement (completion), et pour le retard algébrique (lateness) ou positif (tardiness). Ce problème est désigné par $F|prmu, prec|f_{\max}$ [2], et est fortement NP-difficile [1].

D'un point de vue théorique, nous étudions des algorithmes résolvant ce problème à l'optimalité, ainsi que leurs complexités temporelles et spatiales au pire cas. La taille d'une instance $\mathcal{I}$ est le nombre de travaux : $|\mathcal{I}| = n$. La mesure d'une instance $\mathcal{I}$ est définie comme la somme des temps de traitement : $||\mathcal{I}|| = \sum_{i,j} p_{ij}$. À notre connaissance, alors que de nombreux algorithmes spécialisés ont été décrits pour des sous-instances particulières de ce problème, le meilleur algorithme connu pour le cas général utilise la programmation dynamique classique sur des sous-ensembles de travaux [5] et s'exécute avec des complexités temporelles et spatiales en $O^*(2^n||\mathcal{I}||^m)$.

Notre contribution principale consiste en un algorithme basé sur l'Inclusion-Exclusion, qui permet, sans dégrader la complexité temporelle, de réduire la complexité spatiale par rapport à la programmation dynamique classique d'un facteur $2^{|M|}$, où $|M|$ est le nombre de travaux sans successeur dans les contraintes de précédence.

2 Inclusion-Exclusion pour le problème $F|prmu, prec|f_{\max}$

Pour résoudre le problème, il suffit de compter, pour chaque valeur d'objectif seuil ε, le nombre $N(\varepsilon)$ d'ordonnancements (semi-actifs) avec un objectif inférieur ou égal à ε. Ensuite, la valeur optimale de l'objectif est calculée par dichotomie, et on construit un ordonnancement optimal en utilisant la technique d'auto-réduction, comme décrit dans [3].

Le dénombrement $N(\varepsilon)$ se calcule en utilisant la formule d'Inclusion-Exclusion. Pour cela, on relâche le problème en autorisant les ordonnancements avec des travaux dupliqués ou absents, et pour tout ensemble de travaux J, on définit $N_J(\varepsilon)$ comme le nombre d'ordonnancements (relâchés) utilisant seulement des travaux de J. La formule stipule [3, 4] :

$$N(\varepsilon) = \sum_{J \subset \{1,\ldots,n\}} (-1)^{n-|J|}\ N_J(\varepsilon)$$

Pour calculer $N_J(\varepsilon)$, nous notons $\vec{C}$ le front temporel des dates d'achèvement des opérations d'un travail, avec une date par machine, et nous définissons $\vec{C}\bullet i$ comme les temps d'achèvement des opérations du travail i exécuté dès que possible après le front $\vec{C}$. Puisque le coût d'un travail se calcule sur le temps d'achèvement de sa dernière opération, nous abrégeons $f_i((\vec{C})_m)$ en $f_i(\vec{C})$. Nous définissons également $Pred(i)$ comme l'ensemble des prédécesseurs du travail i, et M comme l'ensemble des travaux maximaux, c'est-à-dire des travaux sans successeur.

On a alors $N_J(\varepsilon) = N_{J,\varepsilon}[\emptyset, \vec{0}, n]$ où $N_{J,\varepsilon}[S, \vec{C}, \ell]$ est le nombre d'ordonnancements relâchés composés de ℓ travaux de J, d'objectif au plus ε, ordonnancés à partir de $\vec{C}$, après que $(n-\ell)$ travaux ont été placés, y compris tous les travaux de S, chacun une fois, où S ne contient que des travaux non maximaux. Nous le calculons en utilisant ce schéma de programmation dynamique :

$$\begin{aligned} N_{J,\varepsilon}[S, \vec{C}, 0] &= \begin{cases} 1 & \text{si } S{=}\{1,\ldots,n\}\backslash M \\ 0 & \text{sinon} \end{cases} \\ N_{J,\varepsilon}[S, \vec{C}, \ell] &= \sum_{\substack{i \in J \\ i \notin S \\ Pred(i) \subset S \\ f_i(\vec{C}\bullet i) \leq \varepsilon}} N_{J,\varepsilon}[S \cup \{i\}\backslash M, \vec{C}\bullet i, \ell{-}1] \qquad \forall \ell{>}0 \end{aligned}$$

Nous obtenons les résultats de complexité suivants. La proposition 1 s'applique dans le cas général, et la proposition 2 s'applique lorsque les précédences sont des chaînes.

Proposition 1 *Pour une instance $\mathcal{I}$ du problème $F|prmu, prec|f_{\max}$, avec $|M|$ travaux sans successeurs, l'algorithme calcule une solution optimale en temps $O^*(2^n||\mathcal{I}||^m)$ et espace $O^*(2^{n-|M|}||\mathcal{I}||^m)$.*

Proposition 2 *Pour une instance $\mathcal{I}$ du problème $F|prmu, chains|f_{\max}$, avec c chaînes de longueurs $\ell_1, \ldots, \ell_c$, l'algorithme calcule une solution optimale en temps $O^*(2^c \ell_1 \cdots \ell_c||\mathcal{I}||^m)$ et espace $O^*(\ell_1 \cdots \ell_c||\mathcal{I}||^m)$.*

Références

[1] M.R. Garey, D.S. Johnson and R. Sethi (1976). *The complexity of flowshop and jobshop scheduling.* Mathematics of Operations Research, 1(2) : 117–129.

[2] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan (1979). *Optimization and approximation in deterministic sequencing and scheduling : a survey.* Annals of discrete mathematics, 5(2) : 287–326.

[3] F.V. Fomin and D. Kratsch (2010). *Exact Exponential Algorithms.* Springer.

[4] J. Nederlof (2008). *Inclusion exclusion for hard problems.* Master Thesis, Utrecht University.

[5] G. J. Woeginger (2003). *Exact Algorithms for NP-Hard Problems : A Survey.* Combinatorial Optimization – Eureka, You Shrink ! 185–207