Programmation dynamique exponentielle pour des problèmes d'ordonnancement de type flowshop à 3 machines

Lei Shang, Christophe Lenté, Mathieu Liedloff, Vincent T'Kindt

Programmation dynamique exponentielle pour des problèmes d'ordonnancement de type flowshop à 3 machines

Lei Shang[1] Christophe Lenté[1] Mathieu Liedloff[2] Vincent T'Kindt[1]

[1] Université François-Rabelais de Tours, Laboratoire d'Informatique (EA 6300), ERL CNRS OC 6305, 64 avenue Jean Portalis, 37200 Tours, France
{shang,lente,tkindt}@univ-tours.fr

[2] Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France
mathieu.liedloff@univ-orleans.fr

Mots-clés : *algorithme modérément exponentiel, programmation dynamique, flowshop*

1 Introduction

De nombreux problèmes traités en recherche opérationnelle sont $\mathcal{NP}-$*difficiles*. La conséquence la plus notable est l'impossibilité de les résoudre en temps polynomial, sauf si la relation $\mathcal{P} = \mathcal{NP}$ est vérifiée. Dans ce contexte, le développement d'algorithmes super-polynomiaux est incontournable, bien que ceux-ci soient considérés comme peu efficaces face à des *grandes* instances. Le domaine de l'*algorithmique modérément exponentielle* dispose d'un ensemble d'outils pour établir et analyser des algorithmes ayant une complexité exponentielle. La monographie de Fomin et Kratsch [2] présente un ensemble de ces outils. En dépit de l'intérêt croissant des algorithmes modérément exponentiels, peu de résultats existent pour des problèmes d'ordonnancement. Pourtant l'enjeu est important : établir des complexités meilleures au pire des cas que celles obtenues naïvement par la recherche exhaustive des solutions.

Quelques résultats sur des problèmes d'ordonnancement d'atelier sont présentés dans la revue de littérature [8]. La technique *Trier & Chercher* a récemment démontrée tout son intérêt pour la résolution de problèmes d'ordonnancement. Par exemple, le problème $P_3||C_{\max}$ peut être résolu en temps $\mathcal{O}(1.7321^n)$ [7], où n est le nombre de travaux. Le problème $1|prec|\sum C_i$, est résolu par l'utilisation d'une approche de *programmation dynamique* en temps $\mathcal{O}(c^n)$ où c est une constante inférieure à 2 [1].

Dans cet article, nous traitons du problème $F3\|C_{max}$ de *flowshop* à 3 machines où nous cherchons à minimiser le *makespan*. L'algorithme le plus efficace connu pour résoudre le problème sur des instances aléatoires utilise une approche de *Séparation & Évaluation* [6]. Toutefois, sa complexité au pire des cas n'est pas meilleure que celle de l'énumération en $\mathcal{O}^*(n!)$ de l'ensemble des permutations [1]. L'approche par programmation dynamique proposée dans [3] peut résoudre ce problème en temps $\mathcal{O}((p_{max})^{2n}(m+1)^n)$. Une autre approche proposée dans [4] peut également s'appliquer et garantit une complexité au pire des cas en temps $2^{\mathcal{O}(n)} \times \|I\|^{\mathcal{O}(1)}$.

Nous proposons d'abord un algorithme modérément exponentiel pour le problème $F3\|C_{max}$. Nous montrons ensuite que notre approche peut se généraliser à d'autres problèmes de flowshop.

2 Une approche par programmation dynamique

Commençons par introduire quelques définitions. Nous posons le critère C^j_{max} comme étant égal à la date de fin du dernier travail sur la machine j, $j \in \{1,2,3\}$. Étant donnée une permutation π, nous notons $C^j_{max}(\pi)$ la date de fin sur la machine j correspondant à cet ordonnancement. Un *chemin critique*, noté CP^j, associé à C^j_{max} est défini comme une séquence d'opérations sur les j premières machines, dans laquelle il n'existe pas de temps mort entre

1. La notation communément utilisée $\mathcal{O}^*$ supprime les facteurs polynomiaux.

deux opérations adjacentes et la durée totale de ces opérations égale à C^j_{max}. Un travail est *critique* dans CP^j s'il a au moins deux opérations dans CP^j.

Nous définissons également un opérateur binaire "." qui concatène deux permutations. Notre algorithme, appelé `DP-fs` dans la suite, est basé sur le théorème 1 :

Théorème 1 *Soit S un ensemble de n travaux à ordonnancer et soit $S' \subset S$. Soient π et π' deux permutations des travaux de S' et σ est une permutation des travaux de $S \setminus S'$. On a la propriété suivante :*

$$si \begin{cases} C^2_{max}(\pi) \leq C^2_{max}(\pi') \\ C^3_{max}(\pi) \leq C^3_{max}(\pi') \end{cases} \quad alors \begin{cases} C^2_{max}(\pi.\sigma) \leq C^2_{max}(\pi'.\sigma) \\ C^3_{max}(\pi.\sigma) \leq C^3_{max}(\pi'.\sigma) \end{cases}$$

c'est-à-dire, que la permutation partielle π domine *la permutation π'.*

Il suffit de noter que $C^1_{max}(\pi) = C^1_{max}(\pi')$ et donc qu'il est toujours préférable de concaténer σ à π plutôt qu'à π', puisque π se termine plus tôt (ou en même temps) que π' sur les trois machines. Ainsi, notre algorithme `DP-fs` exploite l'idée que, lorsqu'une permutation optimale est construite à partir de permutations partielles, il est suffisant de ne considérer que les permutations partielles non dominées. Cela devient plus évident si l'on se restreint à l'espace de critères $\langle C^2_{max}, C^3_{max}\rangle$. Dès lors, les permutations partielles peuvent être représentées par des points 2D dans cet espace et les permutations non-dominées forment un *Front de Pareto* (voir la figure 1).

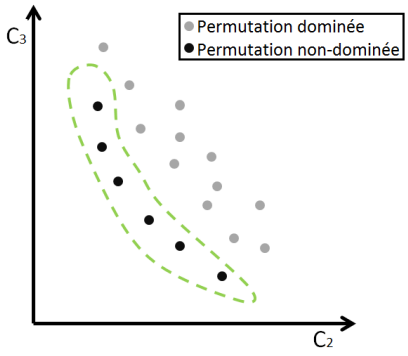


FIG. 1 – Permutations et front de Pareto d'un ensemble de travaux dans l'espace de critères.

Soit S un ensemble de travaux. Étant donné un ensemble de permutations des travaux de S, on définit le *Pareto de permutations*, noté $OptPerm(S)$, comme le sous-ensemble des permutations de S dont le vecteur de critères $\langle C^2_{max}, C^3_{max}\rangle$ n'est dominé par aucun autre vecteur correspondant à une permutation de S. Au cas où plusieurs permutations dominantes possèdent le même vecteur, une seule est retenue par $OptPerm(S)$, arbitrairement.

Notons $MinPerm(\cdot)$, la fonction qui à un ensemble de permutations d'un même ensemble S de travaux, associe $OptPerm(S)$. Désignons par $OptPerm(S, l)$, avec $l \in \{1, ..., |OptPerm(S)|\}$, la l-ième permutation de $OptPerm(S)$ (la numérotation des permutations est arbitraire).

Nous pouvons établir les deux résultats suivants :

Proposition 1 *Si $|S| = t$ alors la taille de $OptPerm(S)$ est au plus $\mathcal{O}^*(2^t)$.*

Proposition 2 *Notons P_{ij}, un entier non-négatif, représentant le temps de traitement du travail i sur la machine j. Supposons que $\forall i, j$, $P_{ij} \leq M$ pour une certaine constante M. Alors pour tout ensemble S de t travaux, le nombre de vecteurs critères $\langle C^2_{max}, C^3_{max}\rangle$ non dominés est au plus $\mathcal{O}((t+1)M) = \mathcal{O}^*(M)$.*

La proposition 1 est établie à partir du nombre maximum de valeurs différentes pour C^2_{max}, ce qui est déterminé par le nombre de chemins critiques associés. Ce nombre de chemins est au plus $t2^{t-1}$, en considérant les affectations des travaux au chemin critique sur les deux premières machines. La proposition 2 s'obtient en remarquant que la valeur maximum de C^2_{max} ne peut pas dépasser la valeur $(n+1)M$.

Théorème 2 *Étant donné un ensemble S, $OptPerm(S)$ peut être calculé par programmation dynamique par la formule de récurrence suivante :*

$$OptPerm(S) = MinPerm\Big\{OptPerm(S \setminus \{k\}, l).\{k\} : k \in S, l \in \{1, \ldots, |OptPerm(S \setminus \{k\})|\}\Big\}$$

Pour chaque valeur de k, nous concaténons le travail k à la fin de chaque permutation dans $OptPerm(S \setminus \{k\})$, puis appliquons $MinPerm$ sur les permutations construites pour trouver $OptPerm(S)$. Par la proposition 1, pour un ensemble S de t travaux, il y a au plus $\mathcal{O}^*(2^t)$ permutations différentes de $OptPerm(S \setminus \{k\}, l)$ à considérer. La fonction $MinPerm$ utilise un algorithme existant ([5]) pour déterminer les vecteurs $\langle C^2_{max}, C^3_{max}\rangle$ non-dominés, avec une complexité $\mathcal{O}(N \log N)$ pour N vecteurs. Par conséquent, le calcul de $OptPerm(S)$ donne une complexité en $\mathcal{O}^*(2^t \log 2^t) = \mathcal{O}^*(2^t)$.

L'algorithme `DP-fs` parcourt tous les sous-ensembles S des travaux du problème ; le temps total pour calculer chacun des $OptPerm(S)$ est donc

$$\sum_{t=1}^{n} \binom{n}{t} \mathcal{O}^*(2^t) = \mathcal{O}^*(3^n).$$

D'autre part, en utilisant la proposition 2, la complexité peut encore s'exprimer comme $\sum_{t=1}^{n} \binom{n}{t} \mathcal{O}^*(M) = \mathcal{O}^*(M2^n)$.

Finalement, la valeur C_{max} peut facilement être obtenue à partir de $OptPerm(S)$ avec un temps additionnel de $\mathcal{O}^*(2^n)$ (ou $\mathcal{O}^*(M)$), ce qui ne change pas la complexité établie. Notons que la complexité en espace est en $\mathcal{O}^*(3^n)$ (ou $\mathcal{O}^*(M2^n)$) si on mémorise toutes les permutations dans le *Pareto de permutations*.

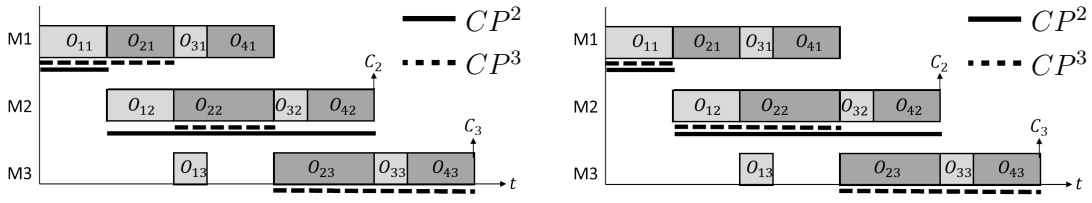
3 Généralisation

Le schéma de programmation dynamique peut être généralisé à d'autres problèmes qui présentent une condition de dominance. Néanmoins, l'analyse de complexité peut être moins simple. Dans cette section, nous traitons le problème $F3\|f_{max}$ pour démontrer la généralité de notre approche.

Soit c_i, $i \in \{1, \ldots, n\}$, la date de fin du travail i sur la machine 3. Notons $f_i(c_i)$, le coût associé à le travail i où f_i est une fonction non-décroissante. L'objectif du problème $F3\|f_{max}$ est de minimiser $f_{max} = max\{f_i(c_i)|i = 1, \ldots, n\}$. L'algorithme `DP-fs` doit être adapté pour prendre en compte le critère f_{max} dans la condition de dominance. Pour cela, nous considérons des vecteurs en dimension 3 $\langle C^2_{max}, C^3_{max}, f_{max}\rangle$. De façon similaire à l'analyse de la Proposition 1, le nombre maximum de vecteurs non-dominés $\langle C^2_{max}, C^3_{max}, f_{max}\rangle$ est borné par le nombre du vecteur $\langle C^2_{max}, C^3_{max}\rangle$. Si on considère ce dernier comme le nombre de combinaison de valeurs C^2_{max} et C^3_{max}, alors ce nombre est au plus $2^n \times 3^n = 6^n$. Néanmoins, en analysant plus finement les chemins critiques, on peut prouver que cette borne peut être améliorée. Soit CP^j_q, $q \in \{1, ..., j\}$, l'ensemble de travaux dont les opérations sur la machine q sont dans CP^j, en excluant le travail *critique* sur la machine q.

Proposition 3 *Sur la machine 1, il suffit de considérer les chemins tels que* $CP^3_1 \subseteq CP^2_1$.

Preuve : Il est évident que $CP^3_1 \subseteq CP^2_1$ ou $CP^2_1 \subseteq CP^3_1$. Dans le cas où $CP^2_1 \subsetneq CP^3_1$ (Figure 2 (a)), il existe toujours un autre choix de chemin critique pour CP^3 tel que $CP^3_1 = CP^2_1$ (Figure 2 (b)). □



(a) $CP^2_1 \subsetneq CP^3_1$ (b) Changer le choix de CP^3

FIG. 2 – Un exemple pour la Proposition 3. O_{ij} désigne l'opération du travail i sur la machine j

Avec le même raisonnement, nous établissons la Proposition 4 :

Proposition 4 *Sur la machine 2, il est suffisant de considérer deux cas :*

1. $CP_1^3 = CP_1^2$ *et* $CP_2^3 \subseteq CP_2^2$.
2. $CP_1^3 \subsetneq CP_1^2$ *et* $CP_2^3 \cap CP_2^2 = \emptyset$.

Proposition 5 *Pour un ensemble de travaux* S, $|S| = t$, *le nombre de* $\langle C_{max}^2, C_{max}^3 \rangle$ *possibles est borné par* $\mathcal{O}^*(4^t)$.

Preuve : Pour établir cette borne, choisissons d'abord les travaux critiques pour chaque chemin critique, ce qui est en temps polynomial en t. Pour les travaux restants, choisissons d'abord les i travaux qui forment CP_1^2 (et donc CP_2^2 est aussi décidé). Pour CP^3, nous considérons :

— Proposition 4, cas 1 : ces i travaux sont aussi dans CP_1^3, les $t-i$ travaux restantes peuvent être mis dans CP_2^3 ou dans CP_3^3, ce qui engendre 2^{t-i} choix ;

— Proposition 4, cas 2 : ces i travaux peuvent être mises dans CP_1^3, CP_2^3 ou CP_3^3, donc trois choix pour chacun. Les $t-i$ travaux restantes ne peuvent être mises que dans CP_3^3, puisque $CP_2^3 \cap CP_2^2 = \emptyset$. Au total cela engendre 3^i choix.

Le nombre chemins critiques est donc $\sum_{i=1}^{t} \binom{t}{i} \mathcal{O}^*(3^i + 2^{t-i}) = \mathcal{O}^*(4^t + 3^t) = \mathcal{O}^*(4^t)$. □

Avec une analyse similaire à celle pour le problème $F3\|C_{max}$, la complexité de notre algorithme pour le problème $F3\|f_{max}$ est $\sum_{t=1}^{n} \binom{n}{t} \mathcal{O}^*(4^t) = \mathcal{O}^*(5^n)$ ou bien $\sum_{t=1}^{n} \binom{n}{t} M^2 = \mathcal{O}^*(M^2 2^n)$ si $p_{ij} \leq M$, $\forall i, j$, pour un certain M. Notons que notre approche peut aussi s'appliquer au problème $F3\|\sum f_i$. Pour cela, on considère le vecteur de critères $\langle C_{max}^2, C_{max}^3, \sum f_i \rangle$ pour obtenir un algorithme en temps $\mathcal{O}^*(5^n)$. De plus, puisque l'algorithme proposé est basé sur la condition de dominance pour des fonctions objectives non-décroissantes et sur le front de Pareto de vecteurs de critères, nous pouvons l'appliquer à encore d'autres problèmes, comme $F2\|f_{max}$ et $F2\|\sum f_i$ qui peuvent être résolus en temps $\mathcal{O}^*(3^n)$.

Références

[1] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n. *Algorithmica*, 68(3) :692–714, 2014.

[2] Fedor V. Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

[3] Joaquim A.S. Gromicho, Jelke J. van Hoorn, Francisco Saldanha da Gama, and Gerrit T. Timmer. Solving the job-shop scheduling problem optimally by dynamic programming. *Computers & Operations Research*, 39(12) :2968 – 2977, 2012.

[4] Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, volume 8037 of *Lecture Notes in Computer Science*, pages 439–450. Springer Berlin Heidelberg, 2013.

[5] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P Preparata. On finding the maxima of a set of vectors. *J. acm*, 22(4) :469–476, 1975.

[6] Talel Ladhari and Mohamed Haouari. A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research*, 32(7) :1831–1847, 2005.

[7] Christophe Lenté, Mathieu Liedloff, Ameur Soukhal, and Vincent T'Kindt. On an extension of the *Sort & Search* method with application to scheduling theory. *Theoretical Computer Science*, 511 :13–22, 2013.

[8] Christophe Lenté, Mathieu Liedloff, Ameur Soukhal, and Vincent T'Kindt. Exponential algorithms for scheduling problems. Technical Report 300, Laboratoire d'Informatique, EA 6300, ERL CNRS OC 6305, Université François Rabelais, Tours, France, February 2014. `https://hal.archives-ouvertes.fr/hal-00944382`.