Complexité d'algorithmes exponentiels : application au domaine de l'ordonnancement

Christophe Lenté[1], Mathieu Liedloff[2], Emmanuel Néron[1], Ameur Soukhal[1], Vincent T'kindt[1]

[1] Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France.
{lente,neron,soukhal,tkindt}@univ-tours.fr.fr

[2] Université d'Orléans, Laboratoire d'Informatique Fondamentale d'Orléans, rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 2
mathieu.liedloff@univ-orleans.fr

Mots-Clés : *Complexité, Méthodes exponentielles, Ordonnancement.*

1 Introduction

Dans cette communication nous nous intéressons au calcul de bornes supérieures à la complexité dans le pire des cas de méthodes exactes exponentielles pour résoudre des problèmes d'ordonnancement $\mathcal{NP}$-difficiles. Cette problématique se pose depuis quelques années pour des problèmes d'optimisation combinatoire ([3, 4]) et renvoie à la question suivante : pour un problème d'optimisation $\mathcal{NP}$-difficile peut-on concevoir une méthode exacte pour le résoudre et établir une *bonne* borne supérieure sur sa complexité (temporelle et/ou spatiale) dans le pire des cas ? Plus précisément, la question est de savoir s'il est possible de faire mieux qu'une simple énumération exhaustive de toute les solutions. Par exemple, pour le problème du calcul d'un ensemble stable maximum de plus grande cardinalité dans un graphe, noté $MAX-IS$, l'énumération de toutes les solutions peut être réalisée en $O^*(2^n)$, *i.e.* est bornée supérieurement par une fonction $g(n) = f(n)2^n$ avec $f(n)$ un polynôme. Robson ([2]) a montré que $MAX-IS$ peut être résolu dans le pire des cas en $O^*(1.2105^n)$.

Le domaine de l'ordonnancement foisonne de problèmes $\mathcal{NP}$-difficile souvent difficiles à résoudre en pratique. D'une façon générale, un problème d'ordonnancement est défini par un ensemble de tâches à réaliser (on parle aussi de travaux constitués d'opérations) sur un ensemble de ressources (ou machines). L'objectif est alors de déterminer sur quelle(s) ressource(s) chaque tâche est traitée et à quelle date dans le but d'optimiser un ou plusieurs critères. Nous renvoyons aux ouvrages de base en ordonnancement pour plus de détails (par exemple, [1]). Notre objectif est d'étudier comment s'appliquent aux problèmes d'ordonnancement les techniques utilisées pour calculer des bornes supérieures à la complexité dans le pire des cas. Un premier travail est déjà connu dans la littérature puisque Woeginger ([3]) a montré que le problème d'ordonnancement $1|prec|\sum C_j$ peut être résolu dans le pire des cas en $O^*(2^n)$ ce qui est mieux que la complexité $O(n!)$ liée à l'énumération des solutions. Cela n'est pas le cas pour beaucoup de problèmes d'ordonnancement pour lesquels l'énumération ne peut être réalisée qu'en $O^*(n!)$ ou en $O(n^m)$ avec n le nombre de travaux et m le nombre de ressources.
Dans la suite nous nous intéressons à deux problèmes d'ordonnancement pour lesquels nous proposons des calculs de complexité dans le pire des cas en utilisant différentes méthodes.

2 Etude d'un problème de flowshop à deux machines

Soient n travaux i à ordonnancer sur deux machines. Chaque travail est d'abord traité sur la machine 1 avant d'être traité sur la machine 2. On parle d'un problème de flowshop. Tous les travaux possèdent une date de fin souhaitée commune qui est à déterminer et l'objectif est de calculer un séquencement qui minimise le nombre de travaux en retard et la valeur de cette date de fin. Nous nous intéressons plus précisément à la minimisation de la valeur de la date de fin commune sachant que le nombre de travaux en retard doit être égal à une borne ϵ donnée. Ce problème peut être montré $\mathcal{NP}$-difficile au sens faible. On peut montrer qu'un algorithme d'énumération pour ce problème existe en temps $O^*(2^n)$. Nous avons appliqué plusieurs techniques pour calculer des complexités dans le pire des cas :

1. Brancher et Réduire : nous obtenons une complexité temporelle en $O^*(1.6180^{n+\epsilon})$ qui est meilleure que l'énumération dès lors que $\epsilon < \frac{100}{227}n$.
2. Trier et Chercher : nous obtenons une complexité temporelle en $O^*(1.4142^n) < O^*(2^n)$.

3 Etude d'un problème avec travaux fixés et multicompétences

Soient n travaux à ordonnancer, chaque travail i étant défini par un intervalle de traitement $I_i = [r_i, \tilde{d}_i]$ et une compétence requise notée c_i. Pour traiter ces travaux m opérateurs sont disponibles, chaque opérateur P_j disposant d'un ensemble N_j de compétences. L'objectif est de déterminer une affectation des opérateurs sur les travaux, plus précisément : existe-t-il une affectation des opérateurs satisfaisant les contraintes d'intervalles et les compétences requises par les travaux ? Ce problème est également appelé dans la littérature *Fixed Job Scheduling Problem* ou *Interval Scheduling*. Lorsque le nombre de compétences est égal à 1, le problème peut être résolu en temps polynomial. Par contre, lorsque le nombre de compétences est quelconque le problème devient $\mathcal{NP}$-difficile. Nous notons *Enum* l'algorithme d'énumération de toutes les solutions de ce problème et montrons que sa complexité dans le pire des cas est en $O^*(m^n)$, c'est-à-dire en $O^*(2^{n \log_2(m)})$.
Plusieurs approches, exploratoires, ont été étudiées et notamment celle consistant à modéliser ce problème sous forme d'un graphe dans lequel la recherche d'une solution réalisable revient à résoudre le problème $MAX - IS$. On obtient alors une complexité dans le pire des cas en $O^*(1.3417^{n(m+1)})$, ce qui malheureusement est moins bon que $O^*(2^{n \log_2(m)})$. Néanmoins nous suspectons que la modélisation sous forme d'un graphe peut être modifiée pour conduire à une complexité temporelle en $O^*(1.3417^{nm})$ ce qui est meilleur que *Enum* dès lors que $m \in \{2, ..., 5\}$. Clairement, cette complexité est ici obtenu très simplement par réduction vers un problème connu. Il sera intéressant de voir d'autres techniques de calcul de complexité pour ce problème.

Références

[1] P. Brucker. Scheduling Algorithms. *Springer*, 5ème édition, 2007.

[2] J.M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[3] G. Woeginger. Exact Algorithms for NP-hard Problems : A Survey. *Lecture Notes in Computer Science*, 2570 :185–207, 2003.

[4] G. Woeginger. Space and Time Complexity of Exact Algorithms : Some Open Problems *Lecture Notes in Computer Science*, 3162 :281–290, 2004.