SORBONNE UNIVERSITÉ
FACULTÉ DES SCIENCES ET D'INGÉNIERIE
DÉPARTEMENT D'INFORMATIQUE
2021-2022

RAPPORT DE MI-PARCOURS

Projet de Recherche

Troisième Année Licence Informatique

Comparaison d'algorithmes pour un problème d'ordonnancement sur m machines avec dates de disponibilités et queues

Réalisé par

Malik DOUFENE

Samy Mouloud NEHLIL

Encadrés par

Professeur MUNIER-KORDON A.

Table des matières

Chapitre 1

Définition du problème

1.1 Définitions et notations

On considère le problème d'ordonnancement classique *P /pmtn, ri, qi/ Cmax* défini par un ensemble de n tâches préemptives Ti de durée quelconque et des fenêtres de temps. Â toute tâche i, on associe 3 paramètres entiers strictement positifs : sa durée pi, sa date de disponibilité ri et sa queue qi. Ces tâches sont à exécuter sur m machines identiques, sachant qu'une tâche nécessite une machine durant toute son éxécution, et une machine ne peut excécuter qu'une tâche à la fois. Les tâches sont préemptives, autrement dit l'exécution de toute tâche i peut être suspendue et reprise ultérieurement

Contraintes

- La durée totale d'execution d'une tâche doit être égale à pi.
- La tâche Ti ne peut démarré qu'à partie de l'instant ri.
- Une tâche ne peut être executé simultanément sur 02 machines distinctes (Ce qui introduit une notion d'intervalle où pour tout $[U_1, U_2]$: le temps d'execution de chaque Ti dans cet intervalle $\leq U_2 - U_1$).

1.2 Problème

Un ordonnancement σ est dit réalisable si il vérifie l'ensemble des contraintes du problème. Pour toute tâche i, on note C_i^σ la date de fin d'execution de i pour l'ordonnancement σ. La durée de l'ordonnancement est définie par $D^\sigma = \max_{i \in T}(C_i^\sigma + qi)$. Le problème est de construire un ordonnancement réalisable de durée D minimale.

Ce qui signifie que le problème se divise en 02 problèmes sous-jacents :

Probléme de décision : Déterminer si il existe bien un ordonnacement réalisable quelconque pour une instance donnée.

Probléme d'optimisation : Si une instance accepte bien un ordonnacement réalisable, déterminé si il est optimal ou non, et dans le cas échéant calculer la solution optimale.

1.3 Rappels

Ce problème est résolvable de manière exacte par un algorithme polynomial qui associe un algorithme de résolution à base de flots a une recherche dichotomique.

1.3.1 les flots

Définition

Soit G un graphe orienté, S son ensemble de sommets et A son ensemble d'arcs. On défini une fonction C qui associe à chaque arc (u, v) de A une valeur positive fixe $C(u, v)$ appelé *capcité* de l'arc (u, v). De plus, on défini deux sommet spéciaux dans G : le sommet S appelé *sommet source* qui n'a aucun arc entrant, et le sommet P appelé *sommet puit* qui n'a aucun arc sortant. Le Graphe G' = (V,A,C) est appelé réseau de flots.

Exemple

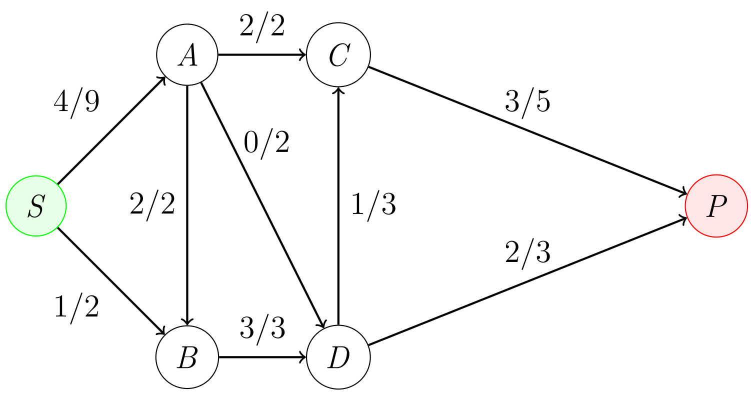

FIGURE 1.1 – Example d'un réseau de flots

Ainsi un flot dans un graphe G avec des capacités C est une application des arcs de G dans $R+$ qui respectent les contraintes suivantes :

Propriétés

Sommets non adjacent : Si $(u, v) \notin A$ alors : $C(u, v) = 0$

Capacité : pour tout $u, v : 0 \leq f(u, v) \leq C(u, v)$, la capcité d'un arc ne peut être dépassé

Conservation : si $u \neq s$ et $u \neq t$ alors :

$$\sum_{v \in S} f(u, v) = \sum_{v \in S} f(v, u) \tag{1.1}$$

C'est à dire que la somme des flots qui partent d'un sommet est égale à la somme des flots qui y entrent.

On en déduis que la somme des flots sortants de la source est aussi la somme des flots qui arrive au puit, cette somme est appelé le débit :

$$\textit{Débit de } f = \sum_{v \in S} f(s, v) = \sum_{v \in S} f(v, p) \tag{1.2}$$

1.3.2 Graphe résiduel et chemin améliorant

Soit G et un flot compatible *f*. Le *graphe résiduel* ou d'*écarts* associé, est le graphe qui modélise , sur chaque arc, l'écart entre les flot et capacité de l'arc. Il est défini de la façon suivante :

- Ses sommets sont les sommets de G.
- Il existe un arc entre u et v si $0 < f(u, v) < C(u, v)$ ou quand $f(u, v) < 0$.
  - Si $0 < f(u, v) < C(u, v)$: alors on mets un arc *(u, v)* qui indique que l'on peut encore augemanté le flot le long de cet arc (arc augmentant).
  - Si $f(u, v) < 0$ alors on mets un arc *(u, v)* qui indique que l'on peut diminué le flot entre v et u.

On dit aussi qu'un arc tel que $f(u, v) = C(u, v)$ est *saturé*. Le graphe résiduel contient donc le graphe où on enlève les arcs *saturés*.

Un *chemin améliorant* pour un flot f est juste un chemin simple de S vers P dans le graphe résiduel de G. Sa variation de flot est la valeur minimale, le long du chemin des $C(u, v) - f(u, v)$ pour les arcs augmentants et $f(u, v)$ sinon.

1.3.3 Algorithme de Ford-Fulkerson

L'algorithme de Ford-Fulkerson cherche un chemin augmentant dans le graphe résiduel. Il sature ce chemin s'il existe, sinon il retourne le flot maximum.

Après construction du graphe d'écart , un chemin de S à P est choisi s'il en existe. Sinon nous avons le flot maximal. Dans le premier cas, le nouveau flot est calculé suivant le chemin augmentant choisi dans le graphe d'écart : on rajoute le flot du chemin augmentant au flot existant si l'arc correspondant dans le graphe d'origine est dans le même sens que celui du graphe d'écart, sinon on soustrait le flot.L'algorithme s'arrête lorsqu'il n'y a plus de chemin augmentant.

Chapitre 2

Démarche et objectifs

Ce projet peut être divisé en deux grands problèmes théoriques : un problème de décision, et un problème d'optimisation, la première étape étant de déterminer si il existe un ordonnancement réalisable et si oui comment l'optimiser (C'est à dire minimiser $D = \max_{i \in T}\{Ci + qi\}$)

2.1 Problème de décision

On peut résumer les étapes de résolution du problème de décision dans les étapes suivantes :

1. Générer des instances de test aléatoires.
2. Construire le flot représentant le problème d'ordonnancement à l'aide de la transformation de Martel
3. Trouver une solution de départ réalisable à l'aide des heuristiques (Jackson's Pseudo Preemptive Scheduling).
4. Résoudre le flot maximum par un algorithme de résolution choisi (Expérimenter plusieurs algorithmes).
5. Déterminer si il existe un ordonnacement réalisable

Ce problème consiste, en premier temps à fixer une constante C qui représente la durée totale d'un ordonnancement, puis étant donné une instance de tâches T avec des ri, qi, et pi, on peut passer à une instance avec des deadlines di par la relation :

$$C = di + qi \quad donc \quad di = C - qi. \tag{2.1}$$

Une fois les deadlines des tâches spnt obtenues, on passe à la deuxième étape dans la résolution du problème de décision par un passage à une représentation en flot.

2.1.1 Génération des instances

évaluer l'efficacité de notre solution ainsi que sa complexité temporelle expérimentale, nous devons penser à une manière de générer des instances de taches en entrée de notre algorithme de décision.

— Cette génération est certes aléatoire mais doit répondre certains critères afin de donner des exemples pertinents :

  - n, m fixés tel que : m machines et n tâches
  - pi uniformes entre 1 et 10
  - Deux réels α et β fixés qui tendent vers *0*
  - les ri sont générés aléatoirement dans $\left[0, \frac{\alpha}{m} * \sum_{i \in T} pi\right]$
  - les qi sont générés aléatoirement dans $\left[0, \frac{\beta}{m} * \sum_{i \in T} pi\right]$

— Filtrer les instances dont les heuristiques retournent directement des solutions optimales et calculer le pourcentage de leurs apparitions en fonction des paramètres α, β, m, n.

— Sauvegarde toutes les instances générées dans un fichier pour les tests ultérieurs.

2.1.2 Passage vers un problème de flot

Pour cela nous allons utiliser la transformation de MARTEL qui est défnit comme suit : Soit G un graphe orienté avec une source s et un puit p.

Sommets du graphe

Soit S l'ensemble des sommets du flot G. Chaque tâche Ti est représentée par un sommet Ti.

On construit des intervalles I à partir de toutes les valeurs distinctes croissantes de $\{ri\} \cup \{di\}$ $avec\ i \in T$, où chaque valeur est une extrémité d'un intervalle, cette transformation est polynomiale, on aura au maximum *2n* intervalles qui correspond au cas où toutes les valeurs ri et di sont distinctes. Cette transformation permet aussi de s'assurer qu'une tache Ti ne peut s'exécuter que sur une seule machine au plus dans un intervalle donné. Ainsi Chaque intervalle I est représenté par un sommet de notre flot.

Arcs du graphe

Soit A l'ensemble des arcs du flot G. Il existe un arc entre la source s et chaque tache Ti. Il existe un arc entre chaque intervalle I et le puit p.

Il existe un arc entre chaque tache Ti et un intervalle Ii si et seulement si cette tache peut s'exécuter sur cet intervalle. Autrement dit, cette tâche est disponible avant le début de cet intervalle et après la fin de ce dernier, la capacité de cet arc étant la longueur de l'intervalle.

Capacité des arcs

Les arcs entre la source s et la tâches Ti ont une capacité de pi qui est la durée d'exécution de Ti (le temps cpu nécessaire pour exécuter la tache i).

La capacité de l'arc reliant une tache Ti à un intervalle Ii représente la longueur de l'intervalle.

Les arcs entre les intervalles Ii et le puit p ont une capacité de m x longueur de l'intervalle (le total de slots disponibles pour l'execution des tâches au sein de cet intervalle).

Déduction

Après cette transformation, il s'agit d'un problème de flots maximum, ou il suffit - à l'aide d'un algorithme de flots bien choisi - de résoudre le problème de flots et de répondre à ce problème de décision par oui/non tel que :
Oui : Un ordonnancement réalisable est possible pour l'instance donnée ; la condition $(\sum_{i \in T} pi \leq flot\ max)$ est remplie
Non : aucun ordonnancement réalisable n'est possible.

Exemple

$T = \{1, 2, 3\}, \quad n = 3 \quad m = 3$

	T1	*T2*	*T3*
ri	0	0	1
di	1	4	4
pi	1	2	3

FIGURE 2.1 – Exemple d'instance

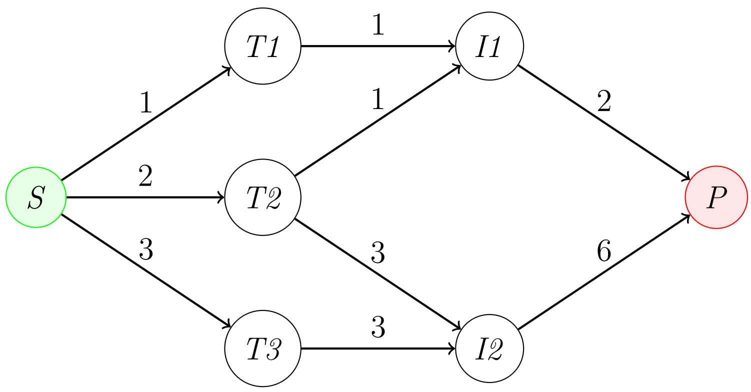

FIGURE 2.2 – Flôt obtenu après transformation

2.2 Solution de départ

Cette étape consiste à tester différentes heuristiques pour une retenir une qui, partant partant d'une instance de tâches avec des deadlines, permet de donner une solution de départ partielle pour notre flôt qui soit réalisable. Cette dernière peut être optimale comme elle peut ne pas l'être. Dans le cas échéant, l'algorithme de flot qu'on choisira devra augmenter cette solution jusqu'à trouver un flot maximum permettant de résoudre le problème de décision.

2.3 Choix d'algorithme de flot

Il s'agit dans cette étape de choisir parmi plusieurs algorithmes de recherche de flots maximum celui qui permet de résoudre le problème de décision (est plus tard le probleme d'optimisation) le plus rapidement possible. Autrement dit, il s'agit d'évaluer expérimentalement la complexité temporelle des différents algorithmes de flots sur un ensemble d'instances générées aléatoirement pour en déduire le plus performant. Tout les algorithmes que nous allons utiliser sont déja implemantés dans la bibliothéques NetworkX de Python.

2.4 Problème d'optimisation

Il s'agit dans ce problème de trouver le C minimum, ie l'ordonnancement de durée minimum pour un ensemble de test, ceci est réalisable à l'aide d'une recherche dichotomique (Binary Search).

2.5 Questions que l'on peut se poser ?

— Quelle valeur donner pour fixer C ?

— Comment générer des exemples de tests aléatoires ?

— Quelle heuristique utiliser pour une solution de départ réalisable (une solution partielle) ?

— Quel algorithme de flot maximum utiliser pour résoudre le problème de décision plus rapidement ?

— Comment résoudre le problème d'optimisation ?

Bibliographie

[1] Létocart Lucas. *Cours "Algorithmique de graphes"* . LIPN - UMR CNRS 7030. Institut Galilée, Université Paris 13. 2018/2019

[2] Cormen T. Leiserson C. Rivest R. *"Introduction à l'algorithmique" traduit par Cazin X.*. DUNOS, Paris 1994.

[3] Nicaud Cyril. *Flôts dan les graphes Cours d'algorithmique avancée*. LIGM UMR 8049, Université Paris-Est, Maren-la-vallée. 2019.