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

Table des figures

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 T de n tâches préemptives $i \in \{1, 2, ..., n\}$ de durées quelconques 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'exécution d'une tâche doit être égale à pi.
- La tâche i ne peut démarrer qu'à partir de l'instant ri.
- Une tâche ne peut être executée simultanément sur 2 machines distinctes.

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 $D^\sigma = \max_{i \in T}(C_i^\sigma + qi)$ avec $T = \{1, .., n\}$. Le problème est de construire un ordonnancement réalisable de durée D minimale.

Ce qui signifie que le problème se divise en 2 problèmes sous-jacents :

Probléme de décision : Déterminer si il existe bien un ordonnacement réalisable quelconque pour une instance donnée et un D fixée. Ce problème est résolvable par un problème de flots, en utilisant la transformation de "Martel".

Probléme d'optimisation : Si une instance accepte bien un ordonnancement réalisable, déterminé si il est optimal ou non, et dans le cas échéant calculer la solution optimale (ie. qui minimise la durée totale). Cette partie associe la problème de flot à une recherche dichotomique.

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.

Soit la fonction C appelée *fonction de capacité* qui associe à chaque arc $(u, v) \in A$ une valeur positive fixe $C(u,v)$ appelé *capcité* de l'arc (u,v). De plus, on défini deux sommets particuliers 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 $G = (S,A,C)$ est appelé *réseau de flots* ou *réseau de transport*.

Ainsi un flot dans un graphe G avec des capacités C est une application $f : A \to \mathbb{R}$ respectent les contraintes suivantes :

Propriétés des flots

- Si $(u, v) \notin A$ alors : $C(u, v) = 0$: Un arc qui ne fais pas partie de G a une capacité nulle.
- Pour tout $(u, v) \in A : 0 \leq f(u, v) \leq C(u, v)$: La quantité de flot d'un arc ne doit pas dépassé la capacité.
- Pour tout $u, v \in S : f(u, v) = -f(v, u)$.
- 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 (différent de la source et du puit) 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 la valeur du flot f :

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

La notation $|f|$ signifie la valeur du flot, et non une valeur absolue.

Exemple

En rouge les capcités des arcs en bleu les valeurs des arcs. La valeur du flot = 5.

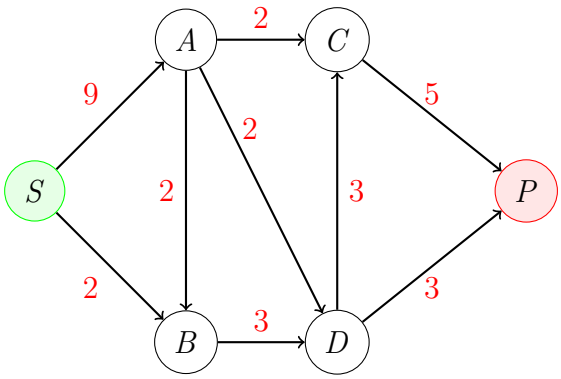

FIGURE 1.1 – Example d'un réseau de flots

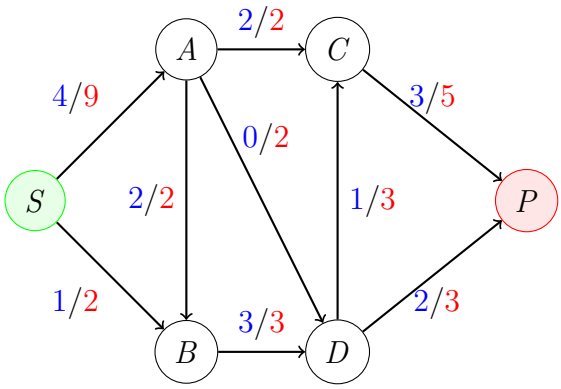

FIGURE 1.2 – Exemple d'un flot

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 le flot et la 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. 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 comme suit :

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 d'heuristiques afin d'accélerer la résolution du problème de flot, par example : Jackson's Pseudo Preemptive Scheduling [5].
4. Résoudre le flot maximum par un algorithme de résolution choisi (Expérimenter plusieurs algorithmes).
5. En déduire si il existe un ordonnacement réalisable

Ce problème consiste, en premier temps à fixer une constante D qui représente la durée totale d'un ordonnancement, puis étant donné un ensemble de tâches 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 sont 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

Pour évaluer l'efficacité de notre solution ainsi que sa complexité temporelle expérimentale, nous devons penser à une manière de générer des instances de tâches 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 (m machines et n tâches).
- pi uniformes entre 1 et 10.
- Deux réels α et $\beta \in [0, 1]$ 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.

— Sauvegarder 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 [4] résumée 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 i est représentée par un sommet Ti.

On construit des intervalles I_k avec $k \in L$ où L est l'ensemble de toutes les valeurs distinctes croissantes de $\{ri,\ i \in T\} \cup \{di,\ i \in T\}$ *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 tâche i ne peut s'exécuter que sur une seule machine au plus dans un intervalle donné. Ainsi chaque intervalle I_k 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 le sommet source S et chaque sommet Ti. Il existe un arc entre chaque intervalle I_k et le sommet puit p.

Il existe un arc entre chaque tâche Ti et un intervalle I_k si et seulement si la tâche i peut s'exécuter sur l'intervalle I_k. 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 I_k.

Capacité des arcs

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

La capacité de l'arc reliant un sommet Ti à un sommet I_k représente la longueur de l'intervalle I_k.

Les arcs entre les sommets Ii et le sommet puit p ont une capacité de m x longueur de l'intervalle I_k (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 : La condition $(\sum_{i \in T} pi = flot\ max)$ est remplie $\Rightarrow$ un ordonnancement réalisable est possible pour l'instance donnée.
Non : $\sum_{i \in T} pi > flot\ max \Rightarrow$ 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

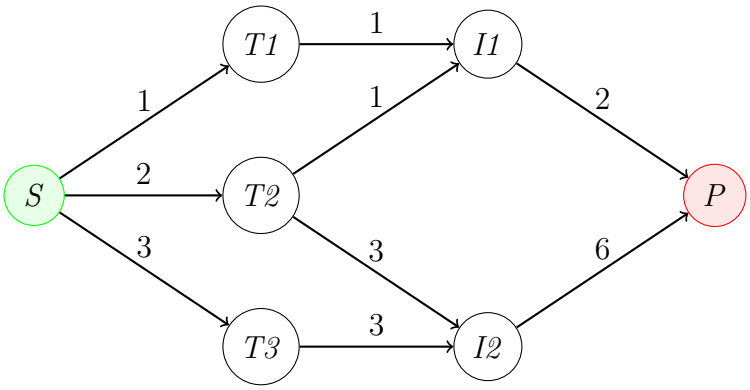

FIGURE 2.2 – Réseau de flots obtenu après transformation

2.2 Solution de départ

Cette étape consiste à tester différentes heuristiques pour une retenir une qui, 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. Nous allons appliquer l'algorithme de Ford Fulkerson à l'instance précédante.

Exemple (avec Ford-Fulkerson)

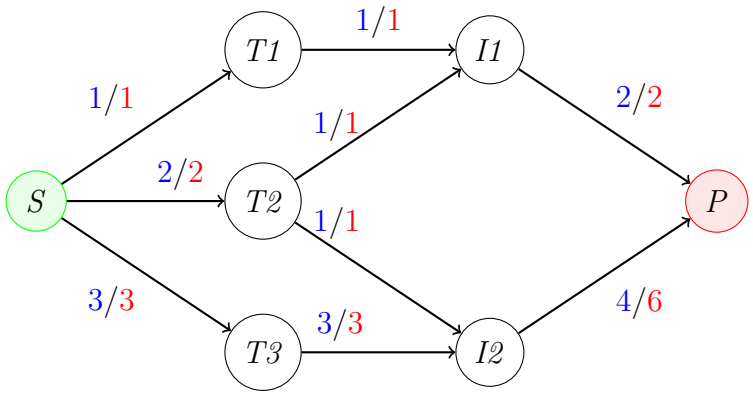

FIGURE 2.3 – Flot maximum avec la méthode de Ford-Fulkerson

On constate que $\sum_{i \in T} pi = flot\ max = 6$. Donc il existe bien un ordonnancement réalisable pour cette instance. On déduit cette ordonnancement à partir des quantités de flots sur les arcs (Ti, I_k) ; ainsi la tâche 1 va s'éxecuter entièrement dans l'intervalle I_1, la tâche 2 va éxecuter 1 unité dans l'intervalle I_1 et 1 unité dans l'intervalle I_2, et la tâche 3 va éxecuter 3 unité (entièrement) dans l'intervalle I_3.

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 Implémentation

L'ensemble des codes du projet sera réalisé en Python et la bibliothèque networkX pour la gestion des graphes.

Présentation de NetworkX

NetworkX est une bibliothèque Python pour l'étude des graphes et des réseaux. Parmi ses principales fonctionnalités qui entre dans le cadre du projet :

- Conversion de graphes depuis et vers divers formats.
- Capacité à construire des graphes aléatoires ou à les construire progressivement.
- Dessin de réseaux en 2D et en 3D.

— Implémentation de divers algorithmes de flot permettant le calcul de chemin augmentant et ainsi que du flot maximum d'un réseau de transport.

Utilisation de NetworkX

Pour ce projet, nous devrons tout d'abord générer nos exemples d'instances (voir section 2.1.1) sous forme de dictionnaires de listes, puis nous passerons ces listes dans une fonction que nous aurons créer nous même qui implémentera la transformation de Martel, afin de construire le réseau de flot correspondant à l'instance. Après application d'une heuristique préalablement choisie, nous effectuerons nos différentes experimentations sur les algorithmes de flots déja implémentés dans NetworkX, tout en respectant le format des paramètres à leur faire passer.

Exemple :
Fonction *maximum_flow* (flowG, _s, _t, capacity='capacity', flow_func=None, **kwargs)

Paramètres :

flowG : NetworkX graph

Les arêtes du graphe doivent avoir un attribut appelé "capacité". Si cet attribut n'est pas présent, on considère que la capacité de l'arête est infinie.

s : node

Sommet source du flot

t : node

Sommet puit du flot

capacity : string

Les arêtes du graphe G sont censées avoir un attribut de capacité qui indique le débit que l'arête peut supporter. Si cet attribut n'est pas présent, l'arête est considérée comme ayant une capacité infinie. Valeur par défaut : "capacity"

2.6 Questions que l'on peut se poser ?

— Quelle valeur donner pour fixer D ?
— 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 Xavier . DUNOS, Paris 1994.

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

[4] Charles Martel. *Preemptive Scheduling with Release Times, Deadlines, and Due Times.* Journal of the ACM (JACM). Volume 29, 01 juillet 1983.

[5] Carlier J. Pinson E. *Jackson's pseudo-preemptive schedule and cumulative scheduling problems.* Discrete Applied Mathematics. Volume 145, 30 Decembre 2004.